Osijek, 2022.

ELPROS
REGIONALNI CENTAR KOMPETENTNOSTI
ELEKTROTEHNIČKA I PROMETNA ŠKOLA OSIJEK

FRONTEND
PROGRAMER

Priručnik

ESF UČINKOVITI LJUDSKI POTENCIJALI
EUROPSKI STRUKTURNI I INVESTICIJSKI FONDOVI

Pozadina naslovnice: Freepik.com

ELPROS
REGIONALNI CENTAR KOMPETENTNOSTI
ELEKTROTEHNIČKA I PROMETNA ŠKOLA OSIJEK

FRONTEND PROGRAMER

Priručnik

Algebra d.o.o.

Osijek, 2022.

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.

Sadržaj:

1. Suvremene *web*-tehnologije

U OVOM POGLAVLJU NAUČIT ĆETE:

- analizirati rad *web*-preglednika i *web*-pretraživača
- analizirati karakteristike tehnologije na strani klijenta u odnosu na tehnologije na strani poslužitelja
- osmisliti informacijsku strukturu *web*-sjedišta
- kreirati wireframe dijagram
- analizirati način rada sustava za verzioniranje koda
- kreirati repozitorij za vlastite projekte.

1.1. Suvremene *web*-tehnologije

Informacijsko-komunikacijske tehnologije povezuju u jednu cjelinu skup tehnologija koje uključuju komunikacijsku i mrežnu opremu, hardver i softver. IKT sustav sastoji se od više podsustava, a cilj mu je omogućiti komunikaciju između različitih komunikacijskih uređaja.

Web- (engl. *World Wide Web*) tehnologije mogu se promatrati kroz dva dijela:

- razvojne *web*-tehnologije prije pojave interneta
- suvremene *web*-tehnologije.

Suvremene *web*-tehnologije započele su svoj razvoj pojavom interneta, početkom 1980-ih godina. Pojavom interneta došlo je do potrebe i za uređivanjem web-stranica te tehnologija koje to omogućuju.

U razvoju *web*-sadržaja postoje dva glavna dijela razvoja: *frontend* i *backend*. *Frontend* se odvija na strani korisnika, daje vizualni prikaz sadržaja i omogućuje interakciju s korisnikom, dok *backend* predstavlja serverski dio, čija je uloga komunikacija s bazom podataka na način da zahtjev korisnika prosljeđuje serveru.

Web-stranice hipertekstni su dokumenti koji mogu sadržavati tekstualne, slikovne, video i zvučne sadržaje. Za izradu *web*-stranica koristi se označni jezik HTML-a (engl. *HyperText Markup Language*). Njegova je uloga označavanje sadržaja na *web*-stranici kako bi ih *web*-preglednici lakše mogli interpretirati te mogli prikazati na sličan način na različitim računalima i različitim *web*-preglednicima. *Web*-sjedište (engl. *web sites*) čini skup međusobno povezanih *web*-stranica. Sadržaji na *web*-stranici mogu biti poveznice (engl. *links*) na druge sadržaje unutar samog *web*-sjedišta ili izvan njega.

Kako bi *web*-sjedište bilo javno dostupno, potrebno ga je smjestiti na odabrani *web*-poslužitelj (engl. *server*). Zakup prostora na *web*-poslužitelju ili oblaku (engl. *cloud*) omogućuje *hosting* usluga koju je potrebno zakupiti kod davatelja internetskih usluga. *Hosting* podrazumijeva i neke dodatne usluge kao što su administriranje baze podataka i tehničku podršku.

Za pregled *web*-stranica koriste se *web*-preglednici (engl. *web browser*), dok se za pretraživanje sadržaja koriste *web*-pretraživači.
Zadaća *web*-pretraživača pronalazak je informacija pohranjenih na drugim *web*-mjestima.

Web-pretraživači jesu:

- Google
- Yahoo!
- Bing
- Baidu
- Ask
- AOL…

Web-preglednici preuzimaju sadržaje s *web*-servera i interpretirajući HTML oznake prilagođavaju ga za prikaz na zaslonu. *Web*-preglednici besplatni su programi koji se mogu prilagoditi zahtjevu korisnika.

Web-preglednici jesu:

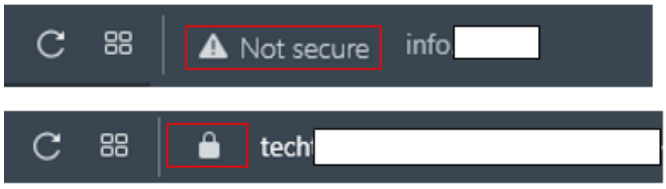
- Google Chrome
- Mozilla Firefox
- Safari
- Opera
- Microsoft Edge…

Slika 1.1. Logotipi nekih web-preglednika

Dva su glavna protokola pomoću kojih *web*-preglednik omogućuje pregled *web*-stranica:

- HTTP (engl. *Hypertext Transfer Protocol*) i
- URL (engl. *Uniform Resource Locator*).

HTTP protokol jest protokol koji omogućuje prijenos *web*-stranica između korisnika i poslužitelja. HTTP protokol čini skup pravila koja omogućuju *web*-pregledniku slanje zahtjeva prema *web*-poslužitelju za određenom *web*-stranicom. Nakon obrade poslanog zahtjeva *web*-poslužitelj traženu *web*-stranicu isporučuje korisniku ako su uvjeti ispunjeni. Zahtjev klijenta definira željenu akciju (GET, POST…), adresu *web*-dokumenta, verziju HTTP protokola te klijentski definirane parametre. HTTP poslužitelj ima standardni port 80. Sigurna inačica HTTP protokola jest HTTPS (engl. *Hypertext Transfer Protocol Secure*) protokol. Prepoznaje se po ikoni lokota u adresnoj traci.



Slika 1.2. Izgled HTTPS protokola u adresnoj traci

Transportni sigurnosni protokol SSL/TLS djeluje u okviru HTTP protokola i ima ulogu zaštite podataka pri njihovoj razmjeni između klijenta i poslužitelja.

SSL/TLS protokol koristi se prilikom prijave korisnika, *web*-trgovine, internetskog bankarstva i slično. Prilikom upisivanja podataka u *web*-forme podaci se kriptiraju pomoću SSL/TLS protokola. SSL certifikat instalira se na poslužitelju kako bi osigurao sigurnost *web*-stranica.

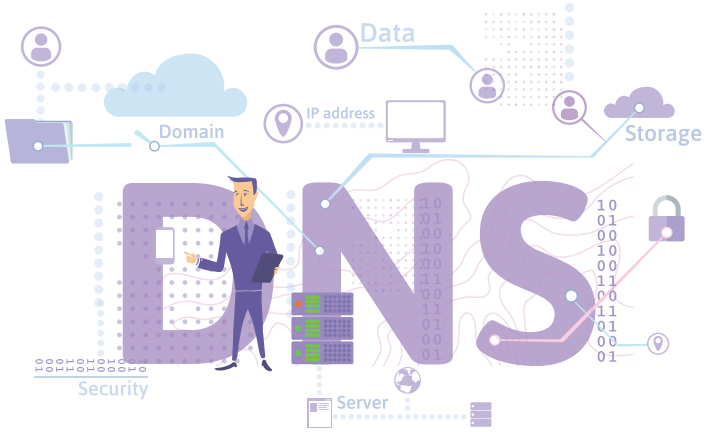
URL (engl. *Uniform Resource Locator*) predstavlja jedinstvenu adresu koja opisuje gdje se određeni sadržaj nalazi, a sastoji od protokola za pristup, naziva poslužitelja i naziva dokumenta.

URL adrese imaju opći oblik:

protokol://ime_poslužitelja/naziv_dokumenta

http://rck.elpros.net/partneri/

Sadržaj na koji upućuje URL može biti *web*-stranica, slika ili neka datoteka koja se nalazi na određenom poslužitelju.



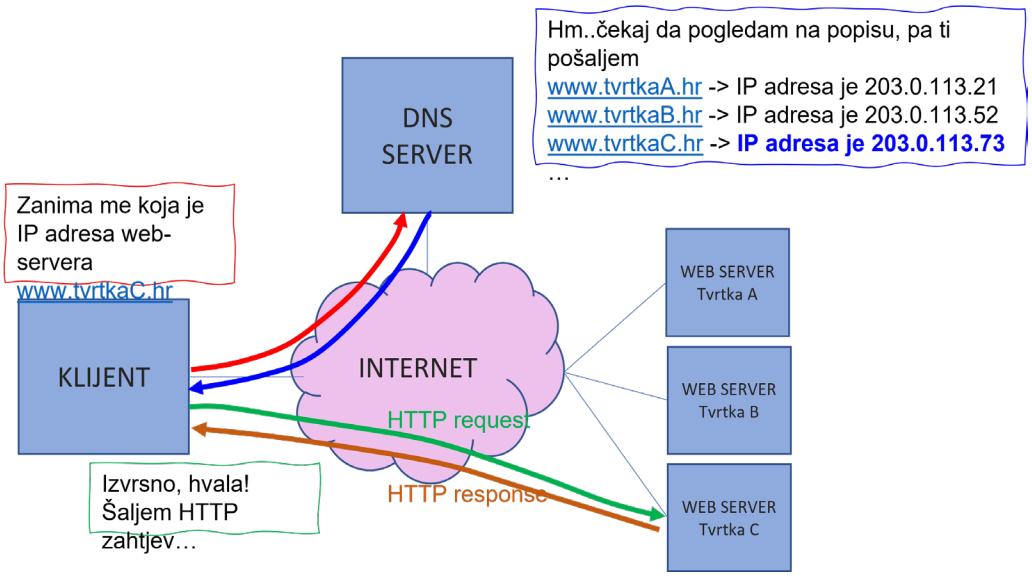
Slika 1.3. Princip rada DNS servera

Komunikacija se u pozadini zapravo odvija putem IP adrese, čiji je zapis teško pamtiti. IP (engl. *Internet Protocol*) adresa 32 bitni je binarni broj koji se zapisuje najčešće u njegovoj decimalnoj protuvrijednosti. Odnosno 32 bita dijelimo na četiri broja od 8 bitova.

11000000 10101000 00000001 00001111

192 . 168 . 1 . 15

Iz tog se razloga web-adrese pamte i zapisuju u tekstualnom obliku, tzv. simboličkim adresama. Za prevođenje IP adresa u simbolički zapis i obrnuto koristi se DNS sustav (engl. *Domain Name System*). On omogućuje upisivanje naziva domene (engl. *Domain Name*) umjesto IP brojčane adrese.



Slika 1.4. Princip obrade zahtjeva korisnika

Web-poslužitelj na računalo korisnika sprema male tekstualne datoteke, tzv. kolačiće (engl. *cookies*). Oni najčešće sadrže informacije o stranicama koje korisnik posjećuje, korisničke podatke za prijavu. Uloga kolačića poboljšati je iskustvo korisnika. Korisnik unutar svog web-preglednika može definirati postavke za kolačiće, blokirati ih ili izbrisati. Neke *web*-stranice bez kolačića neće se učitati ili će biti djelomično funkcionalne. Pregledavanje *web*-stranica u anonimnom prozoru sprečava pohranu kolačića na računalo korisnika.

Kolačiće je, ovisno o izvoru, moguće podijeliti na:

- kolačiće prve strane (engl. *first party cookie*), koji dolaze s *web*-stranica koje korisnik posjećuje
- kolačiće treće strane (engl. *third party cookie*), koji dolaze s drugih mrežni stranica s ciljem prikupljanja podataka u marketinške svrhe o navikama korisnika.

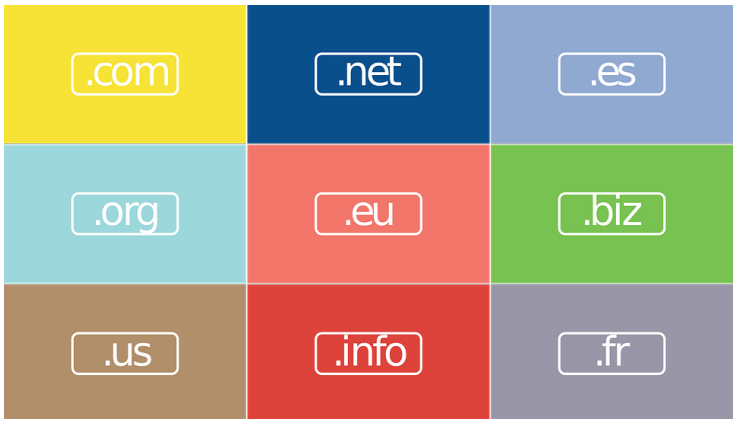
Prema vremenu trajanja pohrane kolačići prve vrste mogu se podijeliti na privremene i trajne. Privremeni traju dok je preglednik otvoren. Nakon zatvaranja preglednika ko-

lačići se brišu. Primjer takvih kolačića košarice su u *web*-trgovinama. Trajni kolačići mogu imati datum isteka i u tom slučaju traju do tog datuma ili mogu ostati na korisničkom računalu dok ih korisnik ne ukloni.

Domena je nužan element za objavu *web*-sjedišta, a predstavlja jedinstveno ime za *web*-adresu.

Domene mogu biti:

- internacionalne (generičke) domene (.com, .org, .net, .edu...) povezane s djelatnošću
- nacionalne, odnosno državne domene (.hr, .uk, .ba, .de...).



Izvor: Pixabay.com

Slika 1.5. Domene

Republika Hrvatska ima nacionalnu domenu .hr, a upravljanje .hr domenom u službi je CARNeta. Pravo na registraciju jedne besplatne .hr domene imaju svi građani RH, bilo da su fizičke ili pravne osobe.

1.2. Organizacija arhitekture klijent/poslužitelj

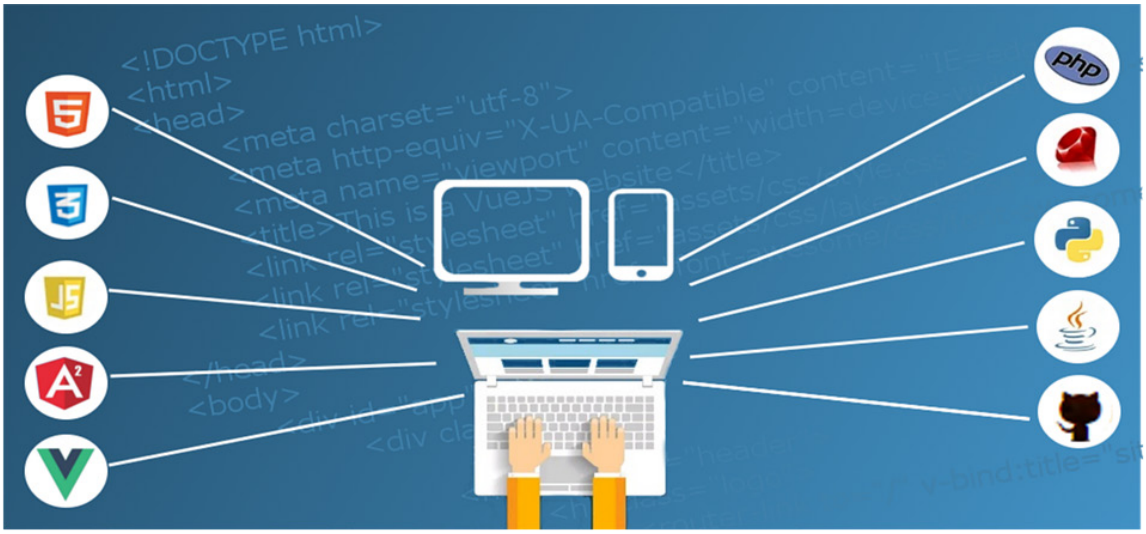
Prilikom izrade *web*-sjedišta koriste se različite tehnologije na strani klijenta i na strani poslužitelja.

Tehnologije koje se koriste za izradu web-stranica na strani klijenta (engl. Frontend) jesu:

- HTML – označni jezik koji daje strukturu *web*-dokumentu
- CSS – stilski jezik koji služi za definiranje stilova *web*-dokumenta
- Java Script – skriptni jezik koji daje dinamičnost *web*-dokumentu i omogućuje interakciju s korisnikom. Može se koristiti i na klijentskoj i na poslužiteljskoj strani
- različite biblioteke i razvojni okviri za napredniji rad kao npr. jQuery, React, Bootstrap, SASS…

Tehnologije koje se koriste pri izradi web-stranica na strani poslužitelja (engl. Backend) jesu:

- Java, PHP, C#, Perl, Ruby, Python – programski jezici na strani poslužitelja
- SQL – pristup bazi podataka
- različite biblioteke i razvojni okviri – npr. Laravel, ASP.NET…



Izvor: Pixabay.com

Slika 1.6. Simbolički prikaz *frontend* i *backend* tehnologija

Na strani poslužitelja programiranje omogućuje prikupljanje podataka putem obrazaca, obradu zahtjeva korisnika te pristup bazi podataka i manipulaciju podacima unutar nje. Na poslužitelju kreiraju se skripte koje služe za dohvat podataka iz baze. Na zahtjev korisnika dohvaćeni podaci s poslužitelja obrađuju se i šalju klijentskoj strani.

Programski jezici na strani poslužitelja mogu omogućiti sljedeće:

- interaktivnost *web*-obrazaca
- kreiranje korisničkih sesija i upravljanje njima
- praćenje aktivnosti korisnika
- spremanje postavki korisnika pomoću kolačića
- konekciju na bazu podataka i manipulaciju podacima unutar baze
- provjeru i potvrdu korisničkih podataka prilikom prijave.

Formate prijenosa podataka na internetu dijelimo na:

- XML (engl. *eXtensible Markup Language*) i
- JSON (engl. *JavaScript Object Notation*).

Slika 1.7. Ikone XML i JSON datoteka

XML je označni jezik koji služi samo za označavanje podataka. Njegova je uloga pojednostavljivanje, tj. odvajanje podataka od njihove prezentacije i omogućivanje brze razmjene podataka internetom te neovisnost o pregledniku, platformi, aplikaciji i uređaju koji ih prikazuju. XML pohranjuje podatke u tekstualnom formatu, čime omogućuje neovisnost o softveru ili hardveru pri pohrani, dijeljenju i prijenosu podataka. Princip zapisa vrlo je jednostavan, odgovarajući sadržaj uokviruje se oznakama koje ga opisuju i imaju poznato, odnosno lako razumljivo značenje. Format oznaka u XML-u vrlo je sličan formatu oznaka u HTML jeziku.

JSON je jednostavan tekstualni standard, zasnovan na objektima JavaScripta, a omogućuje razmjenu i prijenos podataka između klijentske i poslužiteljske strane. JSON podatke lagano je iščitati bez dodatnih programskih rješenja, što znači da ne ovisi o drugim jezicima i može se koristiti samostalno. Većina programskih jezika podržava funkcije za generiranje i prosljeđivanje podataka JSON formata. Nas-

tao je iz potrebe za protokolom koji će omogućiti komunikaciju poslužitelja i *web*-preglednika u stvarnom vremenu, bez upotrebe i instalacije različitih dodataka. Datoteke ovog formata imaju nastavak .JSON. JSON sintaksa izvedena je iz sintakse jezika JavaScripta.

1.3. Organizacija *web*-sjedišta

Osmišljavanje strukture *web*-sjedišta proces je kojim se prolazi prilikom samog razvoja web-sjedišta. Ovisno o složenosti izrade, za određivanje koliko će dugo proces izrade trajati potrebno je provesti analizu i definirati zahtjeve korisnika u s njim.

Koraci u razvoju web-sjedišta jesu:

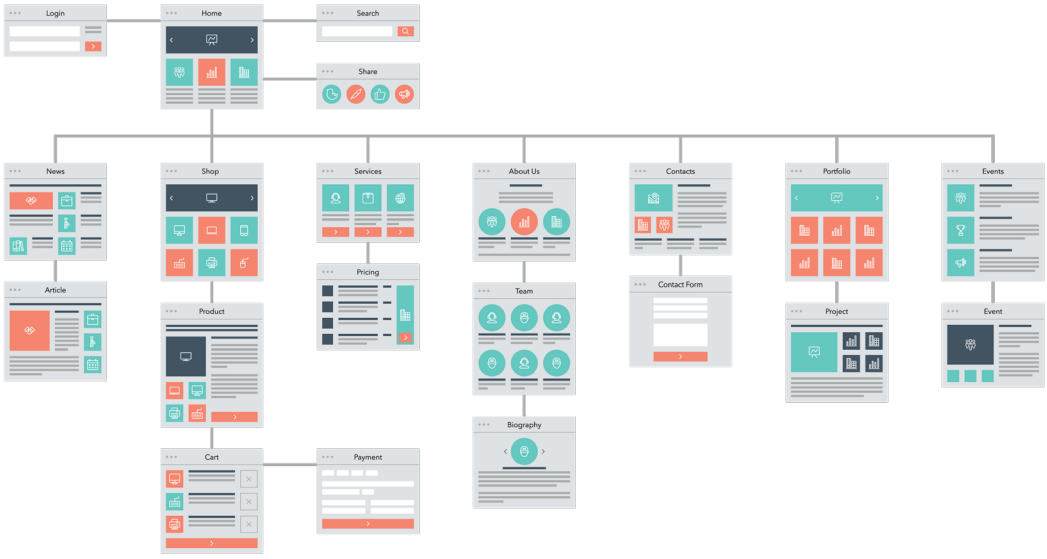
- **Prikupljanje informacija o korisničkom zahtjevu** – definiranje svrhe *web*-stranice i ciljane publike.
- **Plan izrade** – potrebno je unaprijed definirati vrstu *web*-stranice kao i vrstu sadržaja kojima će se stranica puniti. U ovoj fazi dobro je provesti istraživanje i analizu tržišta kako bi se vidjelo koliko stranica sa sličnim sadržajima postoji.
- **Dizajniranje** – izrada žičnog okvira (engl. *wireframe*) za stranicu, a nakon toga i korisničkog sučelja prilagođenog grupi korisnika koje će sadržaj te stranice zanimati.
- **Razvoj i programiranje** – u ovu se fazu kreće nakon što je korisnik odobrio dizajn same stranice. Kreira se pripadajuća baza podataka, doda ju se novih sadržaji na *web*-stranicu te dinamičnost u vidu animacija i događaja koji će privući korisnika da posjeti stranicu. U ovoj fazi potrebno je dogovoriti i hosting *web*-rješenja, a posebnu pozornost treba posvetiti SEO optimizaciji kako bi *web*-stranica što bolje prolazila na *web*-tražilicama.
- **Testiranje i isporuka** – u ovoj se fazi testiraju sve funkcionalnosti *web*-stranice te točnost objavljenih sadržaja. Ako je testiranje prošlo sve potrebne zahtjeve, potrebno je objaviti *web*-stranicu na poslužitelju.
- **Praćenje i održavanje** – ova faza podrazumijeva dodavanje novih sadržaja, otklanjanje pogrešaka, ažuriranje baze podataka te dodatnu SEO optimizaciju.

Vizualni prikaz *web*-stranice može se prikazati izradom *wireframe* dijagrama, koji ima ulogu prikazati funkcionalnost i organizaciju *web*-sjedišta. Iako ne definira njen konačni izgled, dobar je početak jer prikazuju grubu strukturu *web*-stranice. Pri izradi *wireframe* dijagrama treba izbjegavati pogreške kao što su previše informacija i detalja na *web*-stranici, prevelika posvećenost dizajnu umjesto rasporedu sadržaja.

Izvor: Pixabay.com

Slika 1.8. Skiciranje wireframe dijagrama

Informacijska arhitektura (eng. *Information Architecture*) važan je element kod izrade i održavanja *web*-sjedišta. Ona prikazuje njegovu složenost i način navigacije, odnosno kretanja korisnika kroz objavljene sadržaje. Informacijska struktura svakog *web*-sjedišta obuhvaća njegove sadržaje, navigaciju, organizaciju i preglednost sadržaja te sljedivost istih. Tu spadaju navigacija, organizacija i označavanje sadržaja, pretraživanje i slično. Cilj informacijske strukture organizirati je strukturu *web*-sjedišta kako bi korisniku bila što preglednija i razumljivija te da način pronalaska određenih sadržaja bude jasno naznačen.



Slika 1.9. Izgled strukture web-sjedišta

1.4. Verzioniranje koda (GIT)

Sastavni dio razvojnog procesa i obavezni alat za praćenje promjena programskog koda jesu alati za verzioniranje koda. Alati za verzioniranje koda koriste se za detaljno bilježenje i praćenje promjena u projektu, odnosno programskim datotekama, koje se događaju tijekom vremena i pri izradi novih verzija. Oni omogućuju pozivanje prethodnih verzija, bilo verzija programskog koda bilo drugih vrsta datoteka na računalu.

Slika 1.10. Logo GitHub

Verzioniranje koda (engl. *code versioning*) proces je u kojem se svakoj novoj verziji dodjeljuje jedinstveno ime ili broj. Tijekom životnog vijeka (engl. *Life cycle*) programa stvara se više verzija zbog nadogradnje i njegova razvoja, a verzioniranje omogućuje praćenje tih promjena. Na početku životnog vijeka programa stvaranja različitih verzija uglavnom su namijenjena dodavanju novih funkcionalnosti, a kasnije doradi istih.

Važni pojmovi u verzioniranju koda:

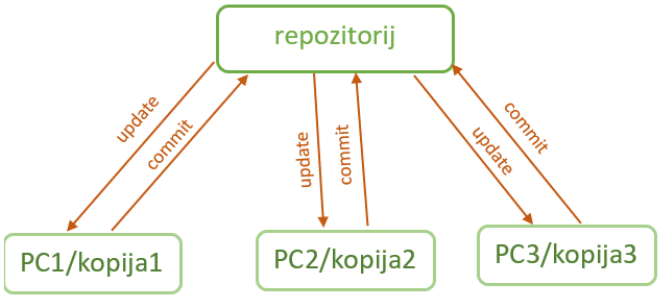
- **Commit** – predstavlja svaku zabilježenu promjenu.
- **Glavna grana** (engl. *Master branch*) **projekta** – jedinstvena i sve izmjene kreću od nje i na kraju se sve ostale grane stope s njom.
- **Grana** (engl. *Branch*) – zasebna grana, neovisna o ostalim granama, a znatno olakšava tijek razvoja projekta. Može biti kopija glavne grane ili neovisna o njoj. Omogućuju paralelan razvoj programskog koda i nesmetan rad više programera.
- **GIT povijest** (engl. history) – predstavlja povijest verzija, odnosno niz commitova.
- **GIT repozitorij** – projekti koji koriste GIT.
- **Sustavi za repozitorije** – GitHub, Bitbucket, GitLab i drugi.

Sustavi za kontrolu verzije (engl. *Version Control System* – VCS) omogućuju suradnju na projektima za više programera. Povijest repozitorija pokazuje koje promjene uključuje pojedini *commit*, zašto su i kada one napravljene te tko je autor istih. Moguć je privremeni i trajni povratak na ranije verzije. Povijest jednog repozitorija grana se na

puno mjesta, ovisi o potrebama i zahtjevima. Sve grane na kraju se spajaju s glavnom granom.

Postoje dvije kategorije sustava za verzioniranje koda:

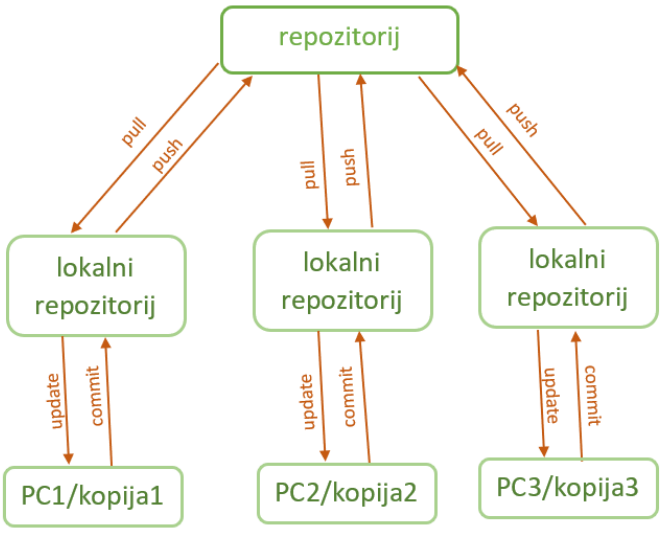
- ***centralizirani sustav*** – jedna centralna kopija i omogućene direktne izmjene na njoj
- ***distribuirani sustav*** – dijeljenje kopija na centralni repozitorij i skup lokalnih skladišta.



Slika 1.11. Shematski prikaz centraliziranog sustava

Svaki programer može ažurirati svoju kopiju s podacima s centralnog repozitorija ili može mijenjati podatke ili pohranjivati na repozitorij. Svaka operacija izvodi se izravno na repozitoriju.

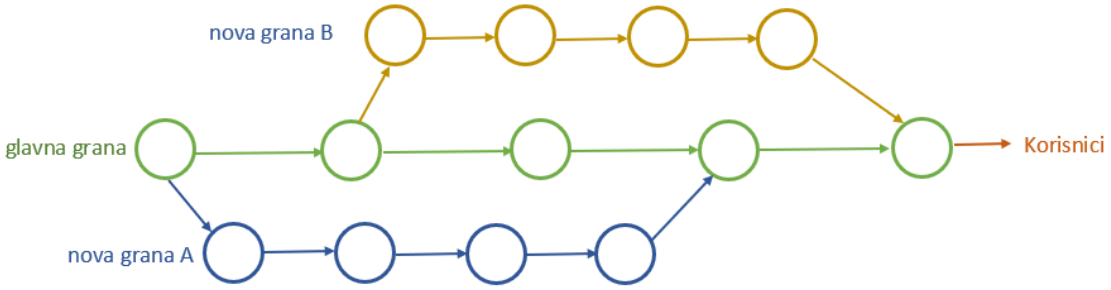
Kod distribuiranog sustava svaki programer samostalno održava lokalni repozitorij, koji je zapravo kopija središnjeg repozitorija na njegovu računalu. Oni mogu raditi i ažurirati promjene na svoj lokalni repozitorij neovisno o drugima. Svoj lokalni repozitorij mogu ažurirati novim podacima s centralnog repozitorija pomoću naredbe pull requests i prebaciti sve promjene s lokalnog na centralni repozitorij naredbom *push*.



Slika 1.12. Shematski prikaz distribuiranog sustava

Pull requests omogućit će pregled promjena unutar koda. Jedan od članova tima može otvoriti *pull request* i omogućiti drugima da pregledavaju i komentiraju promjene koje su nastale.

Grananje GIT repozitorija omogućuje nesmetan rad više ljudi na istom projektu. Nove grane kreiraju se kada se žele napraviti veći zahtjevi nad tim repozitorijem. Glavna grana predstavlja trenutno stanje projekta, a nove grane omogućuju rad na projektu bez promjene inicijalnog stanja. Kad je rad na novim granama završen i postignute su željene promjene, nove (vlastite) grane vraćaju se ponovo na glavnu granu.



Slika 1.13. Primjer grananja unutar projekta

Opći oblik GIT naredbe:

```
git
```

- GIT instalacija može se preuzeti na Git - Downloads (git-scm.com)

Konfiguracija GIT-a radi se pomoću naredbe `git config.`

Postavke GIT-a mogu biti lokalne ili globalne. Lokalne postavke vezane su uz samo jedan projekt, dok se globalne vežu uz korisnika računala.

Za postavljanje globalnih postavki korist se naredba:

```
git config  --global <naziv> <vrijednost>
```

Kako bi se moglo pratiti tko je unio izmjene u kodu (*commitove*), potrebno je podesiti korisničke podatke, ime i e-mail adresu.

To se postiže naredbama:

```
git config --global user.name “Ana Rako”
git config --global user.email “ana.rako@elpros.hr”
```

Pregled najčešće korištenih GIT naredbi i njihovo značenje nalazi se u tablici ispod.

GIT naredbe	Značenje
`git help <naredba>`	Naredba za pregled svih naredbi i opcija koje se mogu koristiti.
`git init`	Naredba za inicijalizaciju lokalnog GIT repozitorija.
`git clone`	Naredba za kreiranje lokalne kopije repozitorija.
`git status`	Naredba za provjeru stanja.
`git add`	Naredba za dodavanje datoteka na GIT repozitorij.
`git rm`	Naredba za brisanje datoteke ili mape s repozitorija.
`git commit –m`	Naredba za lokalno spremanje trenutne verzije. m – commit mesage
`git branch, git branch –a`	Naredba za izlistavanje grana.
`git branch <ime grane>`	Naredba za kreiranje vlastite grane repozitorija.
`git checkout`	Naredba za promjenu grane ili vraćanje na prijašnje stanje unutar trenutne grane.
`git checkout <ime grane>`	Naredba za prebacivanje na novu granu.
`git checkout –`	Naredba za odbacivanje zadnje grane.
`git merge <ime grane>`	Naredba za spajanje dvije grane. Najčešće vlastite grane s glavnom granom.
`git push`	Naredba za pohranu svih promjena na udaljeni repozitorij.
`git pull`	Naredba za povlačenje promjena na lokalni repozitorij.
`git log`	Naredba za pregled promjena.

Tablica 1.1. Pregled najčešće korištenih GIT naredbi

Pitanja za ponavljanje

1. Što je www?
2. Koja je razlika između *web*-preglednika i *web*-pretraživača?
3. Nabroji internet protokole i njihovu ulogu.
4. Koje se tehnologije koriste na strani klijenta?
5. Koje se tehnologije koriste na strani poslužitelja?
6. Kakva je razlika između XML i JSON formata podataka?
7. Nabroji i opiši korake u razvoju web-sjedišta.
8. Što je *wireframe* dijagram?
9. Koja je prednost verzioniranja koda?
10. Objasni razliku između centraliziranog i distribuiranog sustava verzioniranja koda.
11. Što je GIT?
12. Kako se konfigurira repozitorij?
13. Koje su prednosti stvaranja novih grana unutar GIT repozitorija?

2. Strukturiranje *web*-stranica prezentacijskim jezikom HTML (engl. *HyperText Markup Language*)

U OVOM POGLAVLJU NAUČIT ĆETE:

- organizirati strukturu *web*-sjedišta
- kreirati *web*-stranicu primjenjujući HTML strukturu dokumenta
- ugraditi tablice, liste, obrasce u *web*-stranicu
- ugraditi grafičke datoteke i multimedijske sadržaje u *web*-stranicu.

2.1. Sintaksa prezentacijskog jezika za izradu *web*-stranica HTML *(engl. HyperText Markup Language)*

Sadržaj *web*-stranice oblikuje se HTML jezikom. Njegova osnovna zadaća uputiti je *web*-pregledniku kako interpretirati hipertekstualni dokument. HTML datoteke imaju nastavak .html ili .htm. To su tekstualne datoteke koje pomoću oznaka (engl. *tags*) određuju kako će neki sadržaj izgledati u *web*-pregledniku.

Slika 2.1. Logotip HTML5

Trenutno korištena verzija je HTML5, čiju većinu značajki podržava većina *web*-preglednika. HTML5 koristi semantičke oznake koje olakšavaju izradu *web*-stranica. *Web*-preglednik prilikom učitavanja *web*-stranice oslanja se na oznake elemenata i na taj način razlikuje sadržaje.

Svi kodovi iz ovog priručnika bit će pisani u programu Visual Studio Code.

Slika 2.2. Logotip Visual Studio Code

2.1.1. Struktura dokumenta izrađenog prezentacijskim jezikom za izradu *web*-stranica HTML

Nazivi HTML elemenata navode se unutar para znakova <>.

Primjer:

```
<h1> Naslov teksta </h1>
```

Opći oblik HTML elemenata ima izgled:

```
<naziv_oznake>sadržaj_elementa</naziv_oznake>
```

Postoje i neuparane oznake koje se koriste za označavanje elemenata koje nemaju sadržaj. One nemaju završnu oznaku nego se zatvaraju dodavanjem kose crte (/).

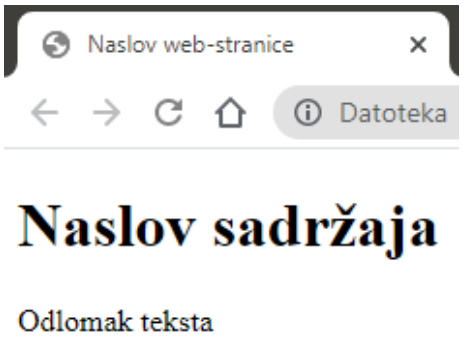
Svakoj HTML oznaci možemo dodati i svojstva kojima će se definirati neka obilježja tog elementa. Unutar navodnika navodimo vrijednosti svojstva HTML elementa.

Osnovna struktura HTML dokumenta:

```
<head>
            <title> Naslov web-stranice </title>
</head>
<body>
      <h1>Naslov sadržaja</h1>
      <p> Odlomak teksta</p>
</body>
tml>
```

Slika 2.3. Osnovna struktura HTML dokumenta

```
<!DOCTYPE html>
<html lang="hr">
<head>
    <meta charset="UTF-8">
    <title>Naslov web-stranice</title>
</head>
<body>
    <h1>Naslov sadržaja</h1>
    <p>Odlomak teksta</p>
</body>
</html>
```



Slika 2.4. Primjer osnovnog HTML dokumenta

Na početku HTML dokumenta nalazi se deklaracija <!DOCTYPE html> koja govori da je riječ o HTML5 i na taj način omogućuje *web*-preglednicima ispravno očitavanje stranice.

Značenje oznaka:

- **<html> </html>** – označavaju početak i kraj primjene HTML-a
- **<head> </head>** – zaglavlje stranice
- **<title> </title>** – naslov stranice koji se prikazuje na kartici web-preglednika.
- **<body> </body>** – tijelo stranice koje sadrži cjelokupan njen sadržaj.

Podrazumijevani stilovi *web*-preglednika jesu:

- font – Times New Roman
- veličina fonta – 12 px
- boja pozadine – bijela
- boja teksta – crna.

Zato je potrebno stilove *web*-stranice urediti stilskim jezikom CSS-om.

HTML oznake mogu biti ugniježđene, odnosno biti jedna unutar druge i zbog toga moramo paziti na redoslijed zatvaranja oznaka. Oznaka koja se otvara zadnja, zatvara se prva.

Komentari

Dijelove *web*-stranice koje ne želimo da se prikazuju u *web*-pregledniku stavljamo u komentar. Najčešće ih dodajemo radi lakšeg snalaženja unutar koda ukoliko više ljudi radi na istoj *web*-stranici. Komentar može biti unutar jednog ili više redova.

Sintaksa:

<!-- tekst komentara -->

```
<body>
    <!--Komentar. Sadržaj komentara se ne prikazuje u web-pregledniku-->
    <h1>Naslov sadržaja</h1>
    <p>Odlomak teksta</p>
</body>
```

Slika 2.5. Komentar unutar HTML koda

- HTML je ograničen na konačan broj oznaka (tagova) definiran pojedinim inačicama. HTML5 ima svoje oznake, ali podržava većinu oznaka HTML4.

2.1.2. Metaelementi

Unutar **<head>** oznake smještaju se metaelementi. Uloga im je optimizacija *web*-stranice.

Metaelementi:

- **Meta Content Type** – za uključivanje kodnih stranica hrvatskog jezika (utf-8)
- **Meta Content Language** – pomaže robotima u pozadini *web*-tražilica za određivanje jezika *web*-stranice
- **Meta Description** – dodaje se kratki opis stranice koji će se prikazivati na tražilicama ispod naziva *web*-stranice
- **Meta Keywords** – navode se odabrane ključne riječi vezane za sadržaj *web*-stranice
- **Meta Viewport** – koristi se za prilagodljivost stranice različitim veličinama zaslona.

```
<head>
    <meta charset="UTF-8">
    <meta content="hr-HR">
    <meta name="description" content="Priručnik za polaznike">
    <meta name="keywords" content="HTML; CSS; JavaScript, SASS">
    <meta name="author" content="Anica Leventić">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Naslov web-stranice</title>
</head>
```

Slika 2.6. Metaoznake

2.1.3. Pravila sintakse

Pravila kojih se treba pridržavati prilikom upotrebe HTML oznaka jesu:

- Svi HTML elementi i njihova svojstva pišu se malim slovima.
- Unutar dvostrukih navodnika navode se vrijednosti svojstva.
- Svi HTML elementi moraju biti zatvoreni.

Opći oblik HTML elemenata:

<naziv_oznake>sadržaj_elementa</naziv_oznake>

Opći oblik upotrebe svojstva:

<oznaka svojstvo1="vrijednost1" svojstvo2="vrijednost2"...> **sadržaj** </oznaka>

Svojstvo se smješta unutar početne oznake HTML elementa. Svakom svojstvu dodjeljuje se naziv i pridružuje njegova vrijednost koja se navodi unutar para dvostrukih navodnika. Neki element može imati više svojstava i u tom slučaju navodimo ih jedan iza drugog u proizvoljnom redoslijedu. Svojstvima se koristimo kako bi određenom HTML elementu osigurali dodatne informacije.

2.1.4. HTML oznake

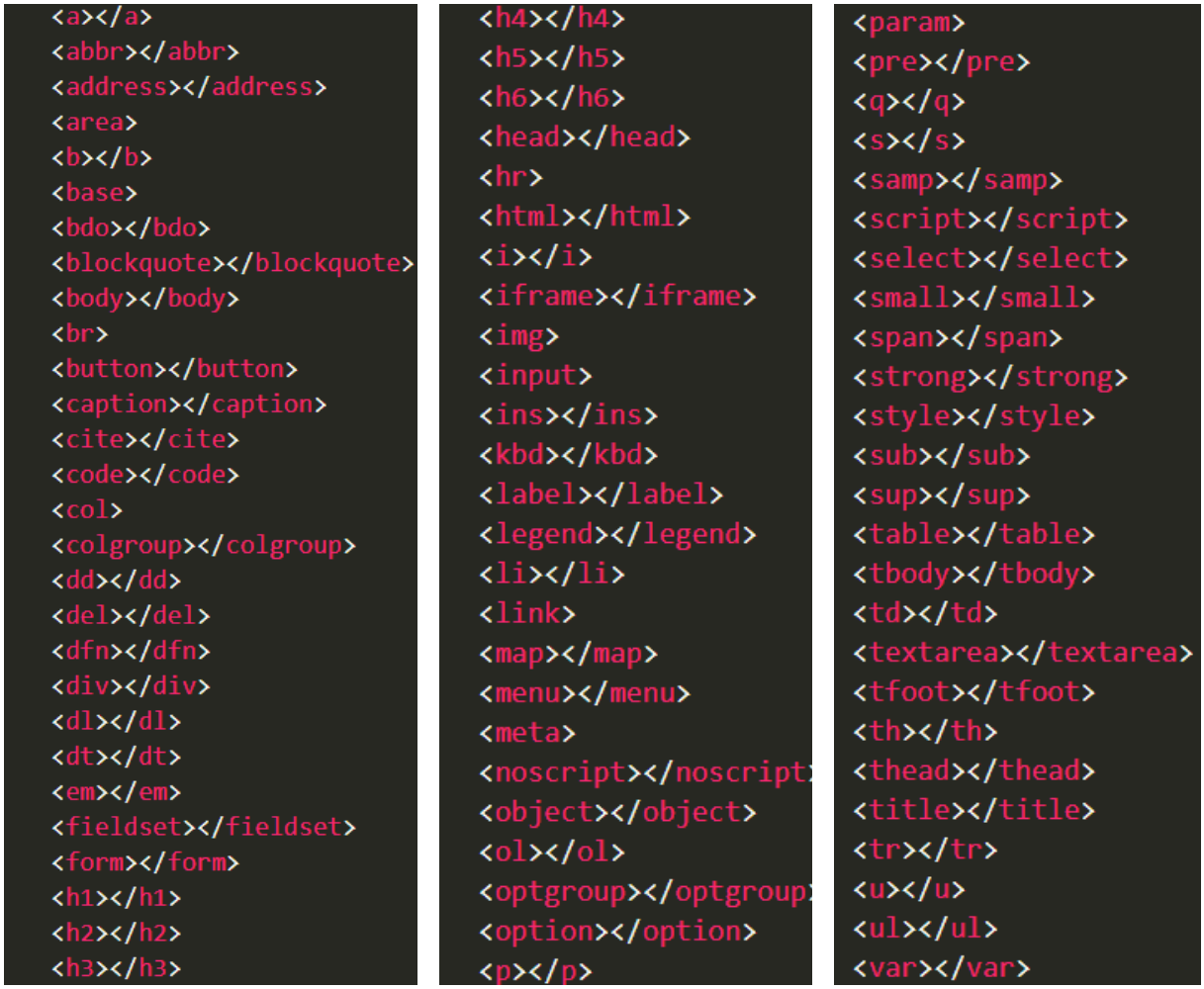
HTML5 inačica ima svoje elemente, ali podržava i većinu elemenata iz HTML4.

HTML5 oznake:

```
<article></article>
<aside></aside>
<audio></audio>
<bdi></bdi>
<canvas></canvas>
<data></data>
<datalist></datalist>
<details></details>
<dialog></dialog>
<figcaption></figcaption>
<figure></figure>
<footer></footer>
<header></header>
```

```
<header></header>
<main></main>
<mark></mark>
<meter></meter>
<nav></nav>
<progress>
<rp></rp>
<rt></rt>
<ruby></ruby>
<section></section>
<summary></summary>
<svg></svg>
<time></time>
<video></video>
```

Slika 2.7. HTML5 oznake



Slika 2.8. HTML4 oznake podržane u HTML5

- HTML5 ne podržava oznake: <acronym>, <applet>, <basefont>, <big>, , <dir>, <font>, <frame>, <frameset>, <isindex>, <strike>, <tt>.

- Elemente <u> i <s> treba izbjegavati, iako su podržane u HTML5, jer imaju drugačije značenje nego u HTML4, pa ih preglednici često interpretiraju pogrešno.

2.2. Rad s tekstom

2.1.2. Oblikovanje teksta *web*-stranice oznakama prezentacijskog jezika HTML

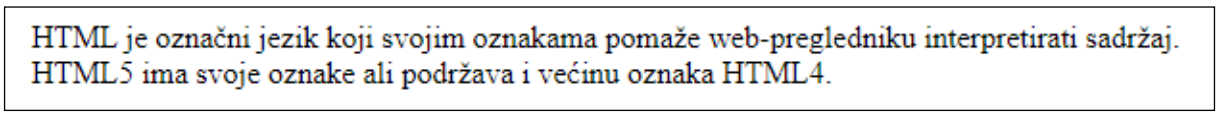
HTML je prezentacijski jezik koji svojim oznakama pomaže *web*-pregledniku interpretirati sadržaj. U radu s tekstom nije važno kako smo zapisali i uredili tekst unutar HTML dokumenta, na prikazu u *web*-pregledniku neće imati utjecaja.

```
<p>
    HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj.
    HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.
</p>
```

Slika 2.9. Izgled teksta

Ukoliko želimo prelazak u novi red, koristimo oznaku **
** (engl. *break row*). Ova je oznaka tzv. neparna oznaka. Neparne oznake imaju samo ili otvorenu ili zatvorenu oznaku i navode se s kosom crtom na kraju oznake. HTML5 dopušta pisanje neparne oznake bez navođenja kose crte.

```
<p>
    HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj. <br>
    HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.
</p>
```

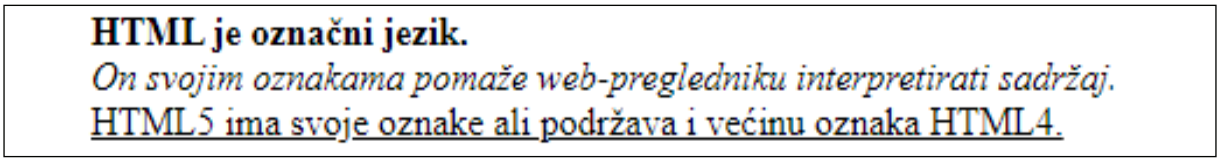


Slika 2.10. Oznaka break row

Za dodatno isticanje teksta možemo se koristiti oznakama:

- bold **** podebljani tekst
- italic **** ukošeni tekst
- **** naglašeni tekst
- **<small></small>** mala slova u tekstu
- **** indeks
- **** eksponent
- **<ins></ins>** umetnuti tekst
- **<del></del>** obrisani tekst
- **<mark></mark>** posebno označeni ili istaknuti tekst.

```
<p>
    <b>HTML je označni jezik.</b> <br>
    <i> On svojim oznakama pomaže web-pregledniku interpretirati sadržaj. </i> <br>
    <u> HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.</u>
</p>
```



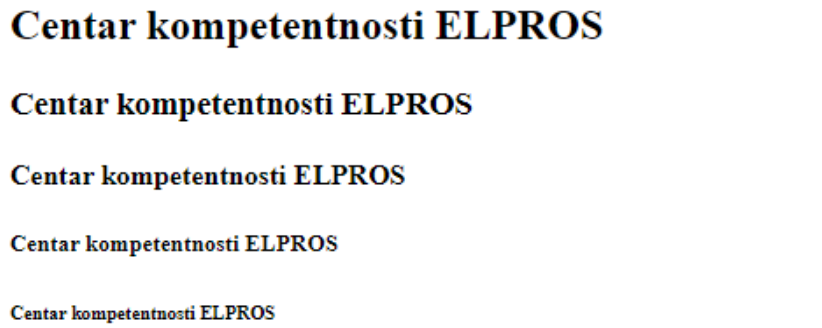
Slika 2.11. Isticanje teksta

2.2.2. Oblikovanje naslova *web*-stranice oznakama prezentacijskog jezika HTML

Naslove (engl. *heading*) oblikujemo oznakama od **<h1>** do **<h6>**, gdje je h1 najveći font naslova prema h6 koji je najmanji. Svaki naslov počinje u novom redu. Preglednici ih interpretiraju tako da oko njih dodaju još prostora. Naslovima možemo dodijeliti svojstvo za poravnanje teksta *align*, s vrijednostima:

- left – lijevo poravnanje
- right – desno poravnanje
- center – centriranje sadržaja.

```
<body>
    <h1>Centar kompetentnosti ELPROS</h1>
    <h2>Centar kompetentnosti ELPROS</h2>
    <h3>Centar kompetentnosti ELPROS</h3>
    <h4>Centar kompetentnosti ELPROS</h4>
    <h5>Centar kompetentnosti ELPROS</h5>
    <h6>Centar kompetentnosti ELPROS</h6>
</body>
```



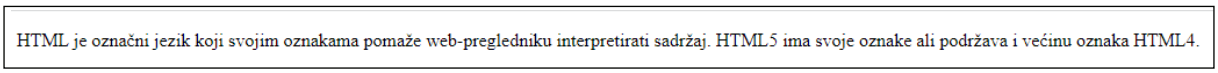
Slika 2.12. Razine naslova

2.2.3. Oblikovanje odlomaka *web*-stranice oznakama prezentacijskog jezika HTML

Koristeći se oznakama za naslove i odlomke postižemo drugačiji prikaz teksta u *web*-pregledniku.

Za kreiranje odlomaka (engl. *paragraph*) koristi se oznaka ** .**

```
<p>
    HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj.
    HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.
</p>
```



Slika 2.13. Izgled teksta u pregledniku

Sadržaj svakog odlomka uvijek započinje u novom redu.

```
<body>
    <p> HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj.</p>
    <p>HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.</p>
</body>
```

HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj.

HTML5 ima svoje oznake ali podržava i većinu oznaka HTML4.

Slika 2.14. Kreiranje odlomaka

Svojstvo **align** služi za pozicioniranje odlomka, a vrijednosti koje može poprimiti jesu:

- left – lijevo poravnanje
- right – desno poravnanje
- center – centriranje sadržaja.

```
<body>
    <p algin="center"> HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj.</p>
    <p>HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.</p>
</body>
```

HTML je označni jezik koji svojim oznakama pomaže web-pregledniku interpretirati sadržaj.

HTML5 ima svoje oznake ali podržava i većinu oznaka HTML4.

Slika 2.15. Centriranje teksta upotrebom svojstva align

Druga oznaka za odlomak koju možemo koristiti jest oznaka **<pre></pre>**. Ovom oznakom zadržavamo predefinirana svojstva teksta iz HTML dokumenta.

```
<body>
    <pre> HTML je označni jezik.
                On svojim oznakama pomaže web-pregledniku interpretirati sadržaj.
                    HTML5 ima svoje oznake, ali podržava i većinu oznaka HTML4.
    </pre>
</body>
```

```
HTML je označni jezik.
            On svojim oznakama pomaže web-pregledniku interpretirati sadržaj.
                        HTML5 ima svoje oznake ali podržava i većinu oznaka HTML4.
```

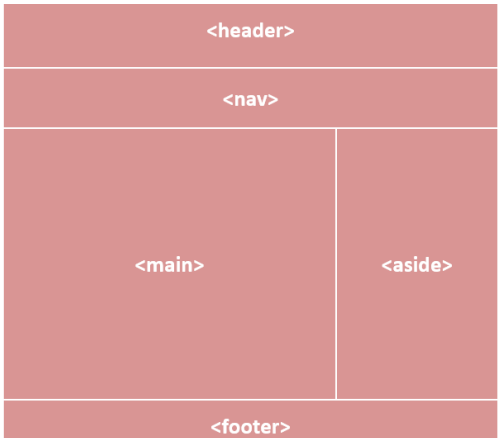
Slika 2.16. Prikaz predefiniranog teksta

2.3. Strukturni elementi za označavanje sadržaja *web*-stranice

Strukturne elemente dijelimo u dvije kategorije semantički i nesemantički. Uloga strukturnih elemenata na stranici pomoć je *web*-pregledniku da jasno razlikuje sadržaj stranice. Strukturni elementi mogu se koristiti više puta ako je potrebno. Semantički strukturni elementi koriste parne oznake.

Semantički strukturni elementi:

- **<header></header>** – zaglavlje stranice ili zaglavlje određenog dijela stranice
- **<nav></nav>** – navigacija stranice, koristi se za izradu glavne navigacije na stranici.
- **<main></main>** – glavni dio stranice i njegov sadržaj treba biti jedinstven
- **<section></section>** – proizvoljna sekcija u dokumentu, rabi se za kreiranje tematskih sadržaja povezanih s ostatkom sadržaja na stranici
- **<article></article>** – članak, cjelovit neovisan dio sadržaja stranice
- **<aside></aside>** – sadržaj „sa strane” (engl. *sidebar*) stranice, cjelovit sadržaj koji nije izravno povezan sa sadržajem stranice, ali daje dodatne informacije o njemu
- **<footer></footer>** – podnožje stranice ili odabranog dijela stranice
- **<details></details>** – služi za definiranje sadržaja koji može biti prikazan ili skriven
- **<summary></summary>** – povezan je s <details> elementima jer za njih definira vidljivi dio
- **<figure></figure>** – služi za definiranja sadržaja poput ilustracija, dijagrama, fotografija, kodova i slično
- **<figcaption></figcaption>** – služi za označavanje naslova elementa <figure>



Slika 2.17. Primjer rasporeda semantičkih elemenata

- **<time></time>** – služi za definiranje datuma i vremena.

Nesemantički strukturni elementi:

- ** ** – blok-element, upotrebljava se za grupiranje dijelova sadržaja koje želimo urediti CSS-om
- **** – linijski element, upotrebljavamo ga kada želimo istaknuti neki dio teksta unutar odlomka.

Elementima i dodajemo oznaku klase ili identifikatora za povezivanje s CSS-om ili JavaScriptom.

Slika 2.18. Ilustracija blok- i linijskih elemenata

Blok-elementi slažu se jedan ispod drugoga prilikom kreiranja stranice. Oni zauzimaju cijelu širinu prikaza u pregledniku ako im nije zadana širina. Sadržaji blok-elemenata uvijek se ispisuju u novom redu. HTML blok-elementi primjerice su ** , <h1></h1>...<h6></h6>, <table></table>...**

Blok-elementi unutar sebe mogu sadržavati druge blok- ili linijske elemente.

Linijski elementi zauzimaju onoliko mjesta koliko je potrebno da stane njihov sadržaj. Oni se ispisuju u istom retku. HTML linijski elementi su primjerice **<a>, <img>, …** Linijski elementi unutar sebe ne mogu sadržavati druge elemente.

2.4. Povezivanje dokumenata na mrežnim stranicama

Važan su dio svake web-stranice poveznice (engl. *links*). Poveznice označavamo HTML oznakom **<a>**, koja dolazi od engleske riječi *anchor* (sidro). Poveznice nam služe za međusobno povezivanje HTML dokumenata.

Opći oblik:

```
<a href="URL stranice">Tekst poveznice</a>
```

Poveznice mogu povezivati:

- međusobno stranice unutar istog web-sjedišta, primjerice s glavnom stranicom
- na neku vanjsku stranicu
- na adresu elektroničke pošte
- na broj telefona
- na neko mjesto na istoj stranici (sidro), primjerice na vrh stranice.

Za stvaranje sidra, odnosno oznake mjesta na vrhu stranice na koje će se moći povezati unutar *web*-dokumenta koristi se svojstvo **id**. Svojstvu id dodjeljujemo ime sidra.

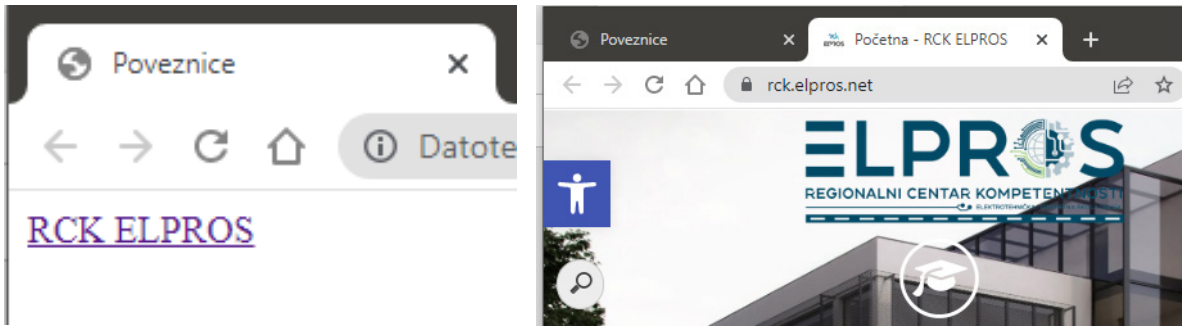
```
<!DOCTYPE html>
<html>
<head>
    <title>Poveznice</title>
</head>
<body>
<h3 id="vrh">Vrste poveznica</h3>
<a href="index.html"></a>
<a href="http://rck.elpros.net" target="_blank">RCK ELPROS</a>
<a href="mailto:ime.prezime@elpros.net">Kontak</a>
<a href="tel:+385400100">Telefon</a>
<a href="#vrh">Vrh stranice</a>
</body>
</html>
```

Slika 2.19. Vrste poveznica

Mailto i tel posebne su vrste linkova, njih pokrećemo vanjskim aplikacijama za elektroničku poštu ili poziv.

Svojstvom **target** koristimo se za definiranje gdje će se poveznica otvoriti. Tako za otvaranje poveznice u novoj kartici svojstvu **target** dodjeljujemo vrijednost **"_blank"**

```
href="http://rck.elpros.net" target="_blank">RCK ELPROS</
```



Slika 2.20. Otvaranje poveznice u novom prozoru

Ukoliko se svojstvu **target** dodijeli vrijednost **"_self"**, stranica će se otvorit u trenutno otvorenom prozoru preglednika. Kada se svojstvu **target** dodijeli vrijednost **"_parent"**, stranica će se otvoriti u prozoru roditeljskog odnosno nadređenog elementa.

Kad je vrijednost svojstva **target** jednaka **"_top"**, stranica će se otvoriti u trenutnom prozoru zanemarujući okvire.

Poveznice možemo koristiti za izradu navigacije među stranicama. Uobičajeno je da se prilikom izrade navigacije *web*-sjedišta koriste nenumerirani popisi koji se smještaju unutar semantičkog elementa **<nav>** s kojima ćete se upoznati u idućim poglavljima.

```
<h2>RCK Elpros</h2>
<a href="index.html">Naslovna</a><br>
<a href="programi.html">Programi obrazovanja</a><br>
<a href="galerija.html">Galerija slika</a><br>
<a href="kontakt.html">Kontakt</a><br>
```

RCK Elpros

Naslovna
Programi obrazovanja
Galerija slika
Kontakt

Slika 2.21. Povezivanje stranica

2.5. Ugradnja grafičkih elemenata u *web*-stranice

2.5.1. Formati slikovnih sadržaja za ugradnju na *web*-stranice

Današnji *web* nezamisliv je bez grafičkih sadržaja. Grafiku dijelimo na vektorsku i rastersku s obzirom na način generiranja grafike. Grafičke datoteke dolaze u raznim formatima. Format utječe na njihovu kvalitetu i veličinu. Rasterska slika ima svoje izvorne dimenzije u kojima ima najbolji prikaz slike. One se mogu smanjiti ili povećati, ali pritom gube se informacije ili detalji, što dovodi do nepravilnosti slike. Rasterski su formati GIF, JPEG, PNG, BMP, TIFF i ostali. Vektorski formati imaju prednost što se mogu smanjivati i povećavati bez promjene u kvaliteti. SVG je najpoznatiji vektorski format.

Za sve multimedijske formate, pa tako i slike, vrijedi pravilo da je kvaliteta proporcionalna veličini datoteke.
Grafičke datoteke prikladne za ugradnju u web-stranice jesu:

- **.jpeg** – pri sažimanju ne gubi na kvaliteti, nedostatak je nepodržavanje prozirnosti
- **.gif** – koristi se kod izrade kratkih animiranih sadržaja
- **.png** – pri sažimanju ne gubi na kvaliteti, podržava djelomičnu prozirnost
- **.svg** – datoteke su male veličine
- **.webp** – format koji podržava i kompresiju s gubitkom podataka i kompresiju bez gubitka podataka i pogodan je za sve vrste grafičkih sadržaja.

2.5.2. Umetanje slika u *web*-stranice pomoću oznaka <img> i <picture>

U HTML dokument slike ugrađujemo navođenjem relativne ili apsolutne adrese slikovne datoteke. Apsolutna adresa sadrži URL adresu slike na nekoj vanjskoj mrežnoj stranici. U tom slučaju nemamo nadzor nad datotekama jer se ne nalaze u našoj mrežnoj mapi. Apsolutna adresa može sadržavati i adresu do neke datoteke na lokalnom računalu. Problem nastaje kod prelaska s poslužitelja na poslužitelj.

```
<img src="https://rck.elpros.net/wp-content/uploads/2021/09/ELPROS_RCK_logo3.png">
```

Slika 2.22. Primjeri navođenja apsolutne adrese

Relativna adresa kraća je zapisom i povezuje HTML dokument sa slikovnim sadržajem. Može biti zadana na različite načine, ovisno o tome gdje se slika nalazi u odnosu na HTML dokument. Prilikom seljenja sadržaja s jednog servera na drugi ne treba se mijenjati.

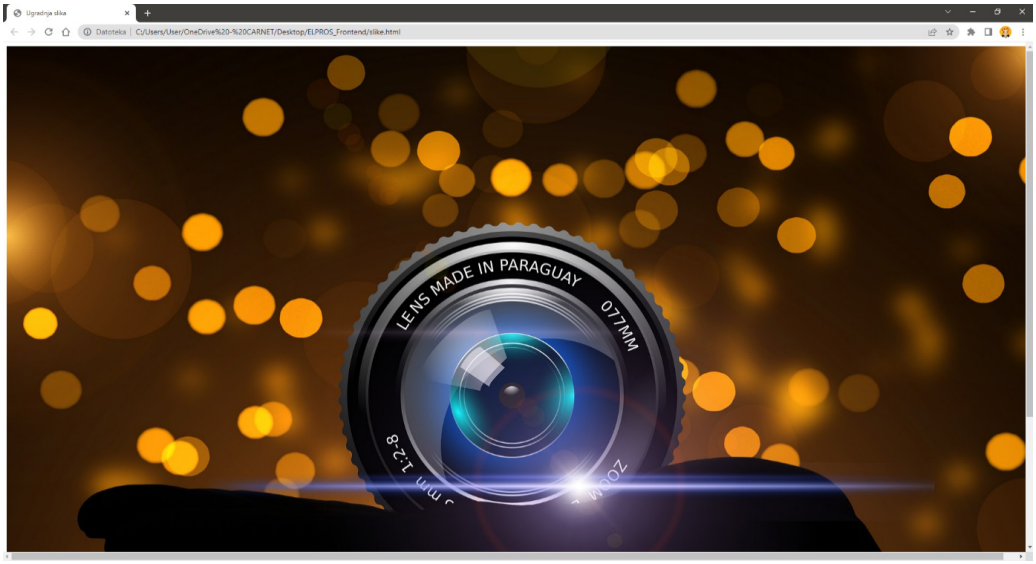
Za ugradnju slika koristimo se elementima **<img>** i **<picture>.**
Svojstva koja se mogu definirati jesu:

- **src** – navodi se naziv i lokacija slike
- **alt** – naziv sadržaja
- **width** – definiranje širine slike
- **height** – definiranje visine slike.

```
<img src="Mediji/slika3.jpg" alt="FOTO">
<img src="../Mediji/slika3.jpg" alt="FOTO">
<img src="../../Mediji/slika3.jpg" alt="FOTO">
```

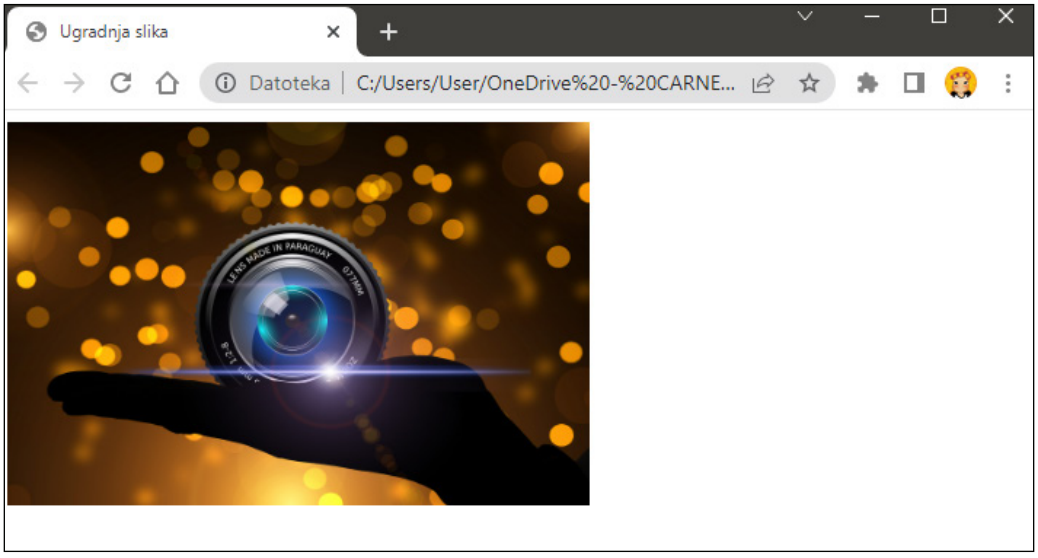
Slika 2.23. Primjeri navođenja relativne adrese

Svojstvo **alt** sadrži neki zamjenski tekst koji će se učitati ako se slika ispravno ne prikazuje. Za učitavanje slike *web*-preglednik zahtijeva samo svojstvo **src** iako su po HTML standardima sva svojstva obavezna. Ukoliko ne definiramo visinu i širinu slike, ona će se ugraditi u HTML dokument sa svojim stvarnim dimenzijama.




```
<img src="slika3.jpg" alt="FOTO">
```

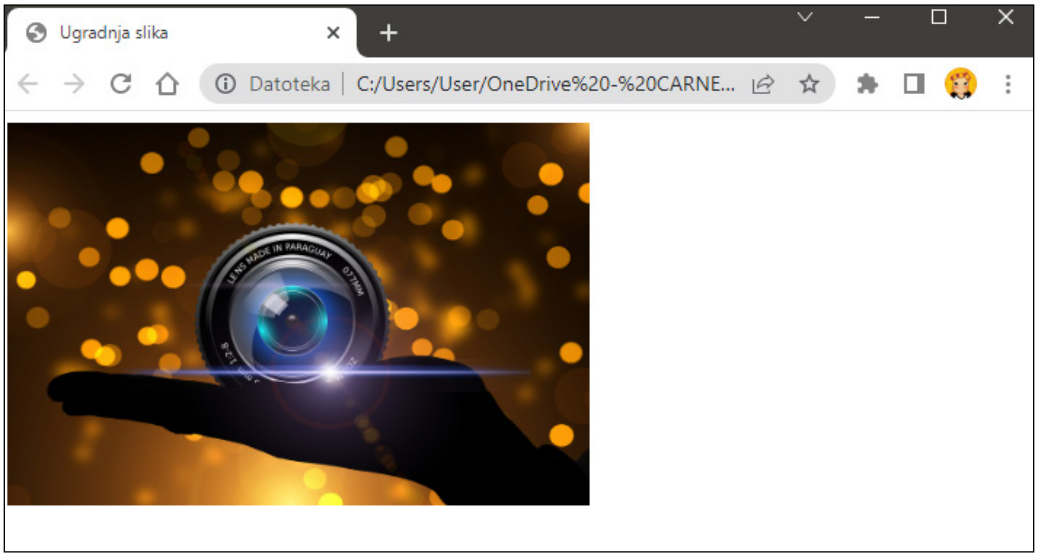
Slika 2.24. Ugradnja slike s originalnim dimenzijama




```
<img src="slika3.jpg" width="400" height="300" alt="FOTO">
```

Slika 2.25. Ugradnja slike s definiranjem širine i visine

Ukoliko se slici odredi samo jedna dimenzija, visina ili širina, druga će se vrijednost odrediti proporcionalno. Uočite da je izgled u pregledniku jednak za oba primjera prikazana na slikama.




```
<img src="slika3.jpg" width="400" alt="FOTO">
```

Slika 2.26. Ugradnja slike s definiranjem jedne dimenzije

Slika može biti i poveznica na neki drugi sadržaj ili *web*-stranicu. To se postiže tako da se unutar oznake **<a>** smjesti oznaka **<img>** s odgovarajućim svojstvima. Klikom na sliku otvara se vanjska stranica čija je URL adresa navedena kao vrijednost svojstva **href**.

```
<a href="https://rck.elpros.net">
    <img src="slika3.jpg" alt="RCK Elpros" width="400" height="300">
</a>
```

Slika 2.27. Slika kao poveznica

Element **<picture>** omogućuje veću fleksibilnost pri ugradnji slika jer je moguće definirati više istih slika, ali različitih dimenzija. Prilikom učitavanja *web*-stranice preglednik učitava onu datoteku koja mu najviše odgovara. Na taj način se osigurava i responzivnost prilikom učitavanja sadržaja na različitim veličinama uređaja jer preglednik uzima prvi sadržaj *source* elementa koji odgovara određenom zaslonu.

Za poravnanje slike koristi se svojstvo align. Vrijednosti koje može poprimiti jesu: **left, right, center, top, middle i bottom**. Njihovo značenje može se vidjeti u tablici ispod.

Vrijednost svojstva	Značenje
`left`	Poravnava sliku desno i tekst se omota oko nje.
`right`	Poravnava sliku desno i tekst se omota oko nje.
`center`	Poravnava se sredina slike s donjim rubom teksta.
`top`	Poravnava se gornji rub slike s gornjim rubom teksta.
`middle`	Poravnava se sredina slike sa sredinom teksta.
`bottom`	Poravnava se donji rub slike s donjim rubom teksta.

Za definiranje razmaka između teksta i slike koriste se svojstva:

- **vspace** – za definiranje vertikalnog razmaka; odnosi se na marginu ispod i iznad slike
- **hspace** – za definiranje horizontalnog razmaka; odnosi se na marginu lijevo i desno od slike. Vrijednosti svojstava vspace i hspace zadaju se u pikselima.

```
<img src="Elpros.jpg" width="400" height="300" alt="FOTO" align="left" vspace="10" hspace="20">
```

Slika 2.28. Lijevo poravnanje slike

```
<img src="Elpros.jpg" width="400" height="300" alt="FOTO" align="right" vspace="10" hspace="20">
```

Slika 2.29. Desno poravnanje slike

Unutar **<picture>** elementa moguće je ugraditi i **<img>** element te jedan ili više **<source>** elemenata.

Svojstva definirana za <picture> element jesu:

- **srcset** – sadrži putanju do ugrađene slike
- **media** – sadrži dimenzije odabranog uređaja
- **width** – širina elementa
- **height** – visina elementa.

```
<picture>
    <source media="(min-width:1200px)" srcset="Mediji/slika3_1">
    <source media="(min-width:650px)" srcset="Mediji/slika3_2">
    <img src="Mediji/slika3.jpg" alt="FOTO">
</picture>
```

Slika 2.30. Ugradnja slike s elementom **<picture>**

2.5.3. Ugradnja vektorske grafike u *web*-stranice pomoću <svg> (engl. *Scalable Vector Graphics*) elemenata

Format .svg spada u vektorsku grafiku, veličine datoteka male su i pogodan je za indeksiranje sadržaja za *web*-pretraživače. Element **<svg></svg>** koristimo za iscrtavanje grafike u pregledniku. Pomoću njega mogu se iscrtavati krugovi, pravokutnici, kvadrati i slično. Direktnim iscrtavanjem u pregledniku smanjuje se količina podataka pri učitavanju stranice i stranica se brže učitava. Bez obzira na veličinu zaslona na kojem se učitava .svg slika ne gubi na kvaliteti.

Unutar <svg> </svg> elementa možemo iscrtavati sljedeće elemente:

- **<circle/>** – krug
- **<ellipse/>** – elipsu
- **<rect/>** – pravokutnik
- **<line/>** – liniju
- **<polyline/>** – spojene linije
- **<polygon/>** – poligon.

Svaki je od navedenih elemenata objekt s pripadajućim svojstvima, čije se vrijednosti zadaju prilikom crtanja. Ponašaju se kao i ostali HTML elementi, ali za dobivanje očekivanih rezultata navedene oznake moraju se staviti unutar pripadajuće oznake.

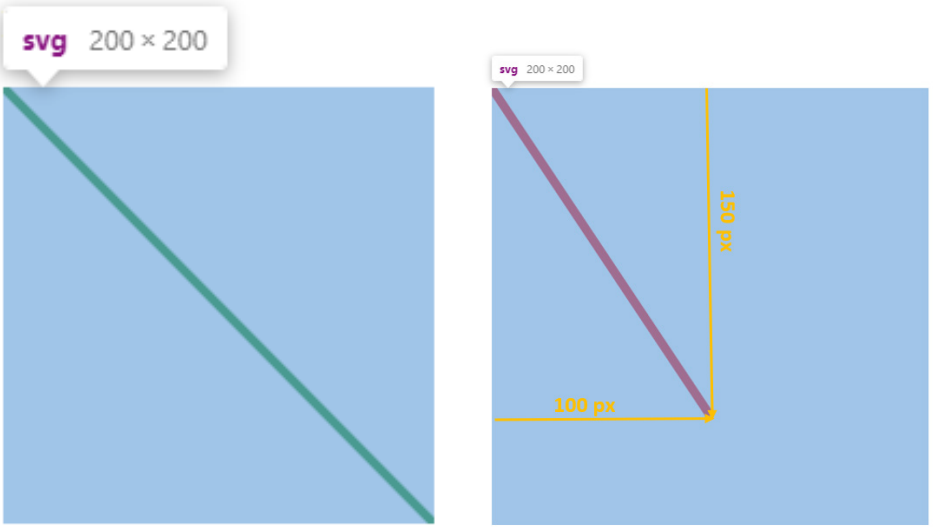
Pri iscrtavanju elemenata u *web*-pregledniku vrlo je važno poznavanje koordinatnog sustava:

- gornji lijevi kut ima koordinate x = 0, y = 0
- vrijednost x koordinate povećava se vodoravno
- vrijednost y koordinate se povećava okomito.

Isto vrijedi za prostor koji HTML element zauzima.

Oznake **<svg> </svg>** parne su, dok su oznake za crtanje unutar nje neparne.

Za crtanje linije koristimo neparnu oznaku *line* **<line>**. Unutar nje navode se koordinate početne (x_1,y_1) i završne točke (x_2,y_2) te svojstvo **stroke** za definiranje boje linije i **stroke-width** za definiranje debljine linije. Koordinate točke govore o udaljenosti te točke od gornjeg lijevog kuta.

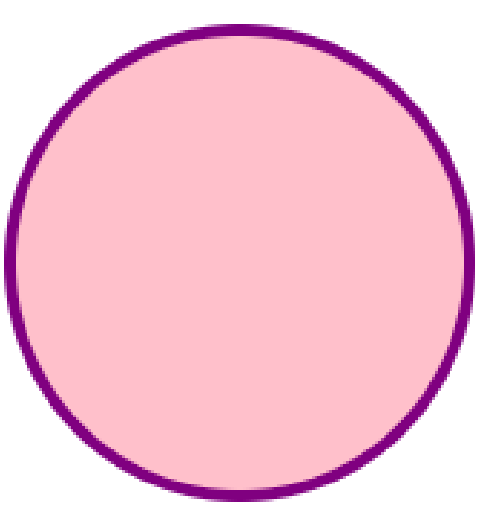


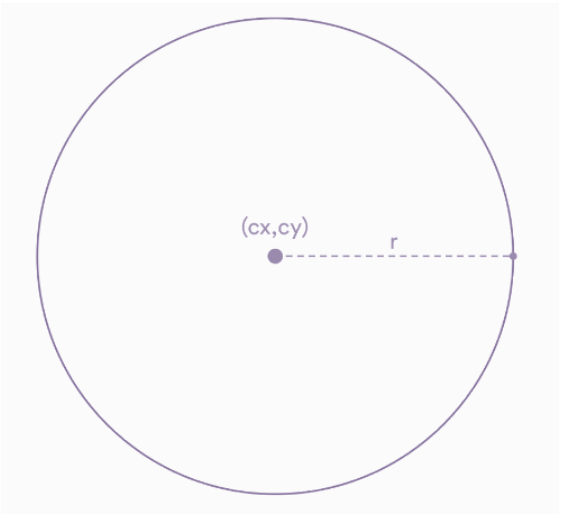

```
<h3>Linija</h3>
<svg width="200" height="200">
    <line x1="0" y1="0" x2="200" y2="200" stroke="green" stroke-width="4" />
</svg>
```

```
<svg width="200" height="200">
    <line x1="0" y1="0" x2="100" y2="150" stroke="red" stroke-width="4" />
</svg>
```

Slika 2.31. Crtanje linije

Na slici 2.27. prikazana je usporedba crtanja dviju linija unutar **svg** elementa iste veličine. Obje linije imaju koordinate početne točke (0,0) i kreću iz gornjeg lijevog kuta elementa čije su dimenzije 200x200 px. Završna točka zelene linije ima koordinate (200,200), što znači da završava u donjem lijevom kutu elementa. Crvena linija ima koordinate završne točke (100,150), što znači x-pomak u lijevo za 100 px i y-pomak prema dolje za 150 px.



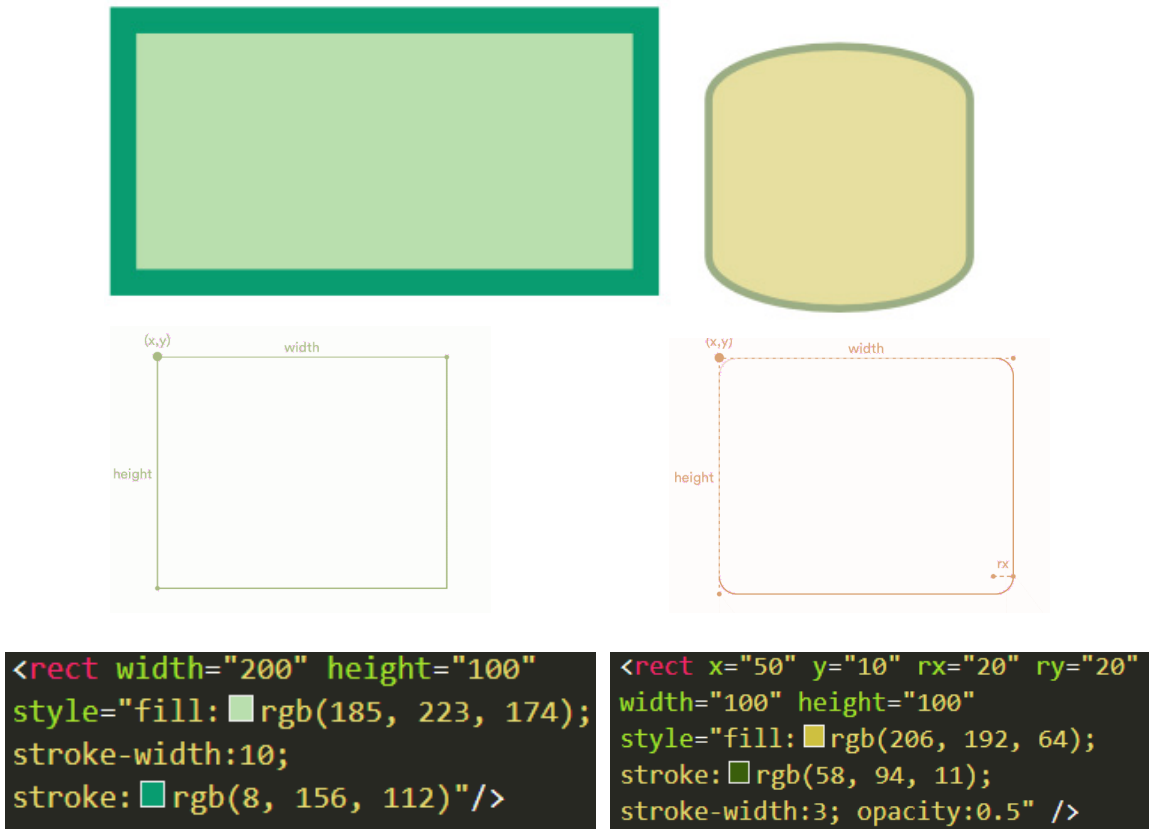



```
<h3>Krug</h3>
<svg width="200" height="200">
    <circle cx="100" cy="100" r="80" stroke="purple" stroke-width="4" fill="pink" />
</svg>
```

Slika 2.32. Crtanje kruga

Za crtanje kruga definiraju se:

- udaljenost koordinata središta kružnice (cy,cy) od gornjeg lijevog kuta <svg> slike
- polumjer r
- stroke – definira boja linije
- stroke-width – debljina linije
- fill – ispuna objekta, odnosno kruga.



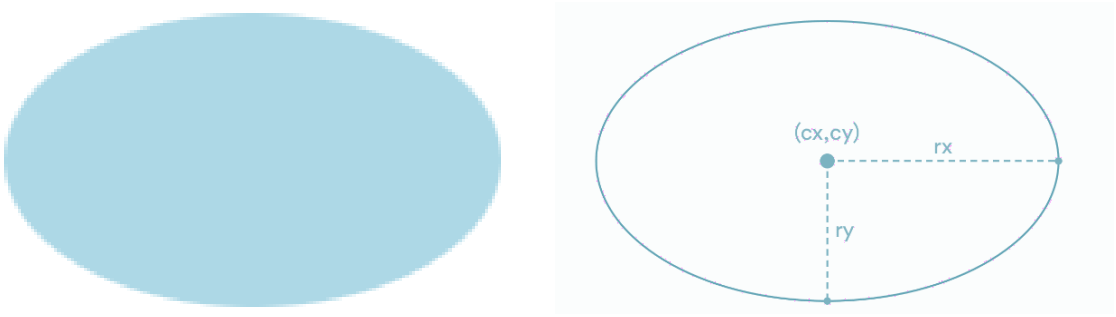
Slika 2.33. Crtanje pravokutnika

Za crtanje zelenog pravokutnika definiraju se:

- **širina i visina elementa <react/>** svojstvima **width** i **hight**
- boja ispune svojstvom **fill**
- boja linije svojstvom **stroke**
- debljina linije svojstvom **stroke-width**
- prozirnost boje obojenog područja svojstvom **opacity.**

Za crtanje žutog pravokutnika sa zaobljenim rubovima definiraju se svojstva:

- x i y udaljenost gornjeg lijevog kuta pravokutnika od gornjeg lijevog kuta **<svg>** slike
- širina i visina elementa **<react/>** svojstvima **width** i **hight**
- boja ispune svojstvom **fill**
- boja linije svojstvom **stroke**
- debljina linije svojstvom **stroke-width**
- rx i ry određuju zaobljenost ruba na x odnosno y stranici svakoga kuta.




```
<h3>Elipsa</h3>
<svg width="200" height="200">
    <ellipse cx="100" cy="100" rx="75" ry="45" fill="lightblue" />
</svg>
```

Slika 2.34. Crtanje elipse

Za crtanje elipse definiraju se svojstva:

- boja ispune svojstvom **fill**
- cx i cy označavaju x i y udaljenost središta elipse od gornjeg lijevog kuta <svg> slike
- rx i ry određuju radijus elipse.

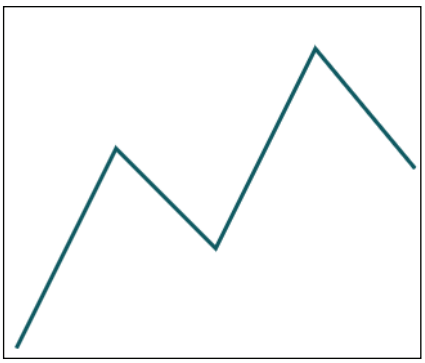

```
<h3>Poligon</h3>
<svg width="200" height="200">
    <polygon points="100,20 180,120 30,160" stroke="pink" stroke-width="3" fill="lightblue" />
</svg>
```

Slika 2.35. Crtanje poligona

Za crtanje poligona definiraju se svojstva:

- boja linije **stroke**
- debljina linije svojstvom **stroke-width**
- ispuna poligona svojstvom **fill**
- koordinate vrhova poligona svojstvom **points**; navode se redom parovi koordinata prve, druge pa treće točke odvojene razmakom.

Za crtanje elementom **<polyline/>** koristi se svojstvo **points**, čije su vrijednosti koordinate točaka koje čine povezane linije.



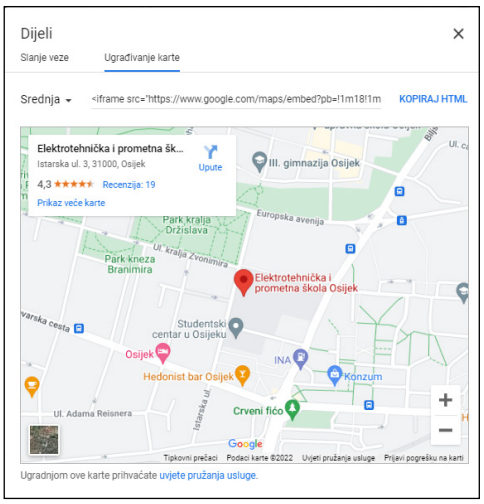
```
<h3>Spojene linije</h3>
<svg width="200" height="200">
    <polyline points="0,200 50,100 100,150 150,50 200,110"
    stroke="rgb(15, 89, 97)" stroke-width="2" fill="none" />
    </svg>
```

Slika 2.36. Crtanje spojenih linija

Za crtanje spojenih linija definiraju se svojstva:

- boja ispune svojstvom **fill**
- boja linije **stroke**
- debljina linije svojstvom **stroke-width**
- koordinate spojnih točaka svojstvom **points**; navode se redom parovi koordinata točaka odvojeni razmakom.

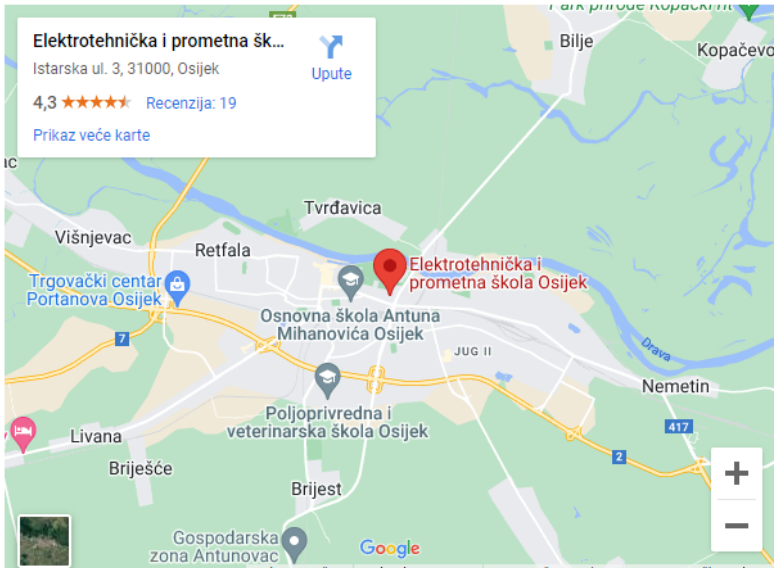
2.5.4. Ugradnja mape u *web*-stranice pomoću <iframe>



Često se na mrežnim stranicama ugrađuju mape lokacije radi lakšeg pronalaženja. Mapa se može ugraditi pomoću elementa **<iframe>**.

Kopira se HTML kod i ugradi na odgovarajuće mjesto na *web*-stranici. Izgled same mape nakon ugradnje vidljiv je na slici.

Slika 2.37. Ugradnja mape preko Google mapsa




```
<iframe src="https://www.google.com/maps/embed?pb=!1m18!1m12!1m3!1d698.4163285590121!2d18.
693937094079033!3d45.55705786678222!2m3!1f0!2f0!3f0!3m2!1i1024!2i768!4f13.
1!3m3!1m2!1s0x475ce7b9feb3702d%3A0xea9859cc2089e027!2sElektrotehni%C4%8Dka%20i%20prometna%
20%C5%A1kola%20Osijek!5e0!3m2!1shr!2shr!4v1655829386801!5m2!1shr!2shr" width="600"
```

Slika 2.38. Ugradnja mape elementom **<iframe>**

Za definiranje umetnutog okvira **<iframe>** koriste se svojstva:

- **style** za uređivanje stilova, **border=0** za uklanjanje obruba koji po podrazumijevanim postavkama dolazi uz element **<iframe>**
- **allowfullscreen** određuje je li dopušten prikaz na cijelom zaslonu ili ne
- **loading** definira kako će web-preglednik učitati okvir.

Vrijednosti svojstva loading mogu biti:

- **eager** – učitava se odmah bez obzira na to je li izvan vidljivog područja. Ovo je zadana vrijednost preglednika
- **lazy** – odgađa učitavanje okvira dok ne bude u području vidljivog područja.

2.6. Ugradnja multimedijskih sadržaji u *web*-stranice

Kombiniranje različitih vrsta sadržaja kao što su tekst, slika, zvuk, animacija i video zovemo multimedijom. Upotreba multimedije na web-stranicama učinila ih je dostupnijima korisnicima i omogućila njihovo lakše korištenje.

2.6.1. Priprema multimedijskih sadržaja za primjenu na *web*-stranicama

HTML5 omogućio je ugradnju multimedijskih elemenata bez dodatnih dodataka (engl. *plug-in*), tako da je novim HTML elementima omogućio reprodukciju zvuka i videa u preglednicima. Multimedijske datoteke zahtjevne su i dolaze u različitim formatima, stoga ih je potrebno prilagoditi za upotrebu na webu.

Formati podržani HTML5 standardom	
Audio	Video
.mp3	.mp4
.wav	.webm
.ogg	.ogg

Tablica 2.2. Podržani video i audioformati

U HTML dokument ugrađuju se s lokacije gdje su smješteni. Unutar odabranog HTML elementa navodimo apsolutnu ili relativnu putanje dokumenta.

2.6.2. Ugradnja videosadržaja u *web*-stranice

Za ugradnju videa koristi se element **<video>**, koji predstavlja spremnik za videosadržaj. Nemaju svi preglednici istu podršku za videoformate, pa je preporuka koristiti mogućnost pozivanja datoteka iz više izvora u više različitih formata. Za to koristimo element **<source**>.

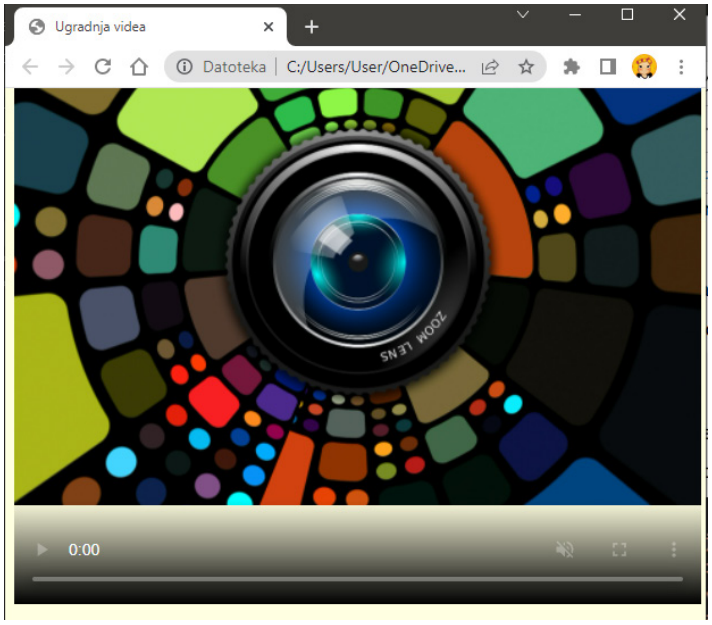
Kod <source> elementa koristimo se svojstvima:

- **src** – omogućuje navođenje putanje
- **type** – omogućuje navođenje tipa video datoteke.

Element <video> ima svojstva:

- **width, height** – pomoću njih određujemo dimenzije videa i na taj način sprečavamo moguće treperenje prilikom učitavanja *web*-stranice. Vrijednost izražavamo u pikselima
- **autoplay** – omogućuje automatsku reprodukciju videa
- **controls** – omogućuje prikaz kontrola reprodukcije videa
- **muted** – omogućuje automatsko stišavanje zvuka
- **loop** – automatsko ponavljanje videa kad dođe do kraja (beskonačna petlja)
- **poster** – daje mogućnost postavljanja slike koja se prikazuje prije pokretanja video zapisa.

```
<video  width="800" height="600" controls
muted autoplay loop poster="../Mediji/Slika1.jpg">
    <source src="../Mediji/video.mp4" type="video/mp4">
    <source src="../Mediji/video.webm" type="video/webm">
    <source src="../Mediji/video.ogg" type="video/ogg">
</video>
```



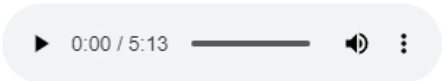
Slika 2.39. Ugradnja videa

Za ugradnju i dijeljenje videosadržaja s javnih servisa kao što su YouTube, Vimeo i slično koristimo se elementom **<iframe>**. Svojstvo **src** sadrži ID video zapisa, a kopiramo ga desnim klikom na željeni video i biramo opciju kopiraj ugrađeni kod. Primjenjujemo ista svojstva kao kod elementa **<video>**.

2.6.3. Ugradnja audiosadržaja u *web*-stranice

Za ugradnju zvučnih datoteka u *web*-dokument koristimo element **<audio>.**

Jednako kao i kod ugradnje videa, moguće je pozvati datoteke iz različitih izvora upotrebom elementa **<source>** sa svojstvima **src** i **type**. Preglednik će preskočiti formate koje ne može učitati.




```
<audio controls loop>
    <source src="../Mediji/pjesma.mp3" type="audio/mp3">
    <source src="../Mediji/pjesma.ogg" type="audio/ogg">
    <source src="../Mediji/pjesma.wav" type="audio/wav">
</audio>
```

Slika 2.40. Ugradnja audiodatoteke

2.6.4. Vrste licenci multimedijskih sadržaja dijeljenih na internetu

Autorsko pravo je pravo autora da za svoje autorsko djelo traži naknadu ili postavi uvjete dijeljenja i korištenja. Uvjeti pod kojima se neki sadržaj može koristiti i na koji način opisani su licencom.

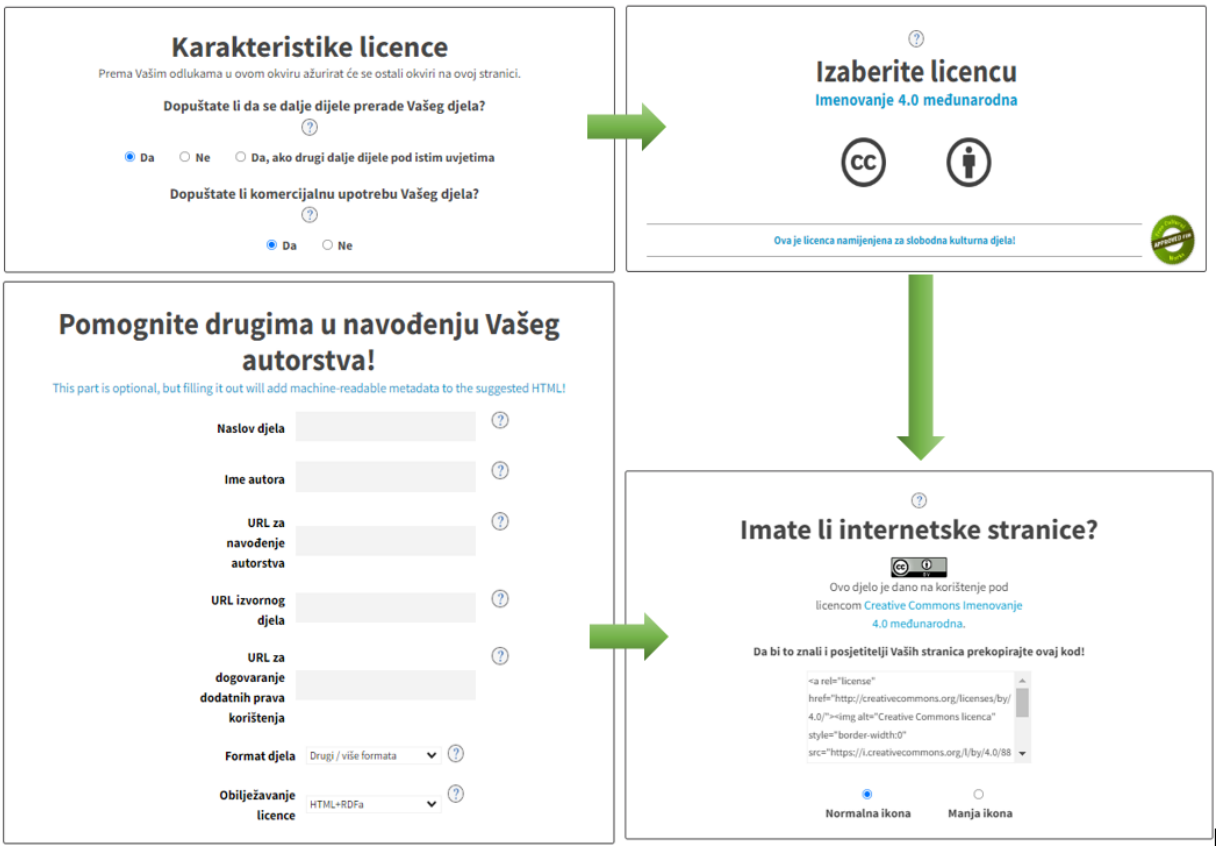
Pri korištenju sadržaja preuzetih s interneta potrebno je provjeriti uvjete pod kojima su objavljeni. Podrazumijeva se da su svi objavljeni sadržaji automatski zaštićeni, osim ako nije drugačije navedeno. Sadržaji kao što su slike, tekstovi, video i audiozapisi mogu se koristiti uz odobrenje autora ili spadaju u sadržaje koje je dozvoljeno koristiti.

Sadržaji označeni Creative Commons licencom mogu se koristiti pod određenim uvjetima. Ponekad je dovoljno navođenje autora ili objava uratka pod istom licencom i slično. Na slici 2.41 mogu se vidjeti oznake licenci i što je obveza pri objavljivanju sadržaja na koji se odnosi.

Licenca	Kratica	Opis
	CC BY	Obveza imenovanja autora djela
	CC BY-SA	Obveza imenovanja autora djela te pravo korisnika da djelo dijeli pod istim uvjetima
	CC BY-ND	Obveza imenovanja autora djela te pravo korisnika da djelo dijeli bez ikakvih promjena
	CC BY-NC	Obveza imenovanja autora djela te pravo korisnika da se djelom koristi u nekomercijalne svrhe
	CC BY-NC-SA	Obveza imenovanja autora djela te pravo korisnika da se djelom koristi u nekomercijalne svrhe i da ga dijeli pod istim uvjetima
	CC BY-NC-ND	Obveza imenovanja autora djela te pravo korisnika da se djelom koristi u nekomercijalne svrhe bez ikakvih promjena

Slika 2.41. Vrste licenci (preuzeto s https://creativecommons.org/licenses)

Ukoliko se želi postaviti licenca na fotografiju, video, web-stranicu ili bilo koji drugi sadržaj, potrebno je kreirati licencu za svoj rad. Na stranici https://creativecommons.org/choose/ definiraju se karakteristike i vrsta licence te način dijeljenja i objavljivanja sadržaja.



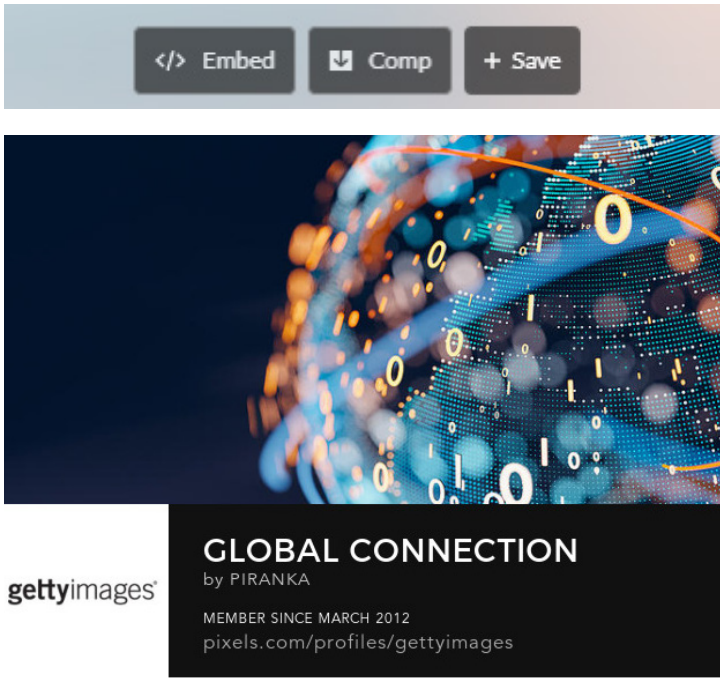
Slika 2.42. Proces kreiranja i postavljanja CC licence

Nakon definiranja uvjeta korištenja i izbora vrste licence na *web*-stranicu ugrađuje se kod kreirane licence. Klikom na poveznicu otvara se stranica s opisom licence.

```
<a rel="license" href="http://creativecommons.org/licenses/by/4.0/"><img alt="Creative
Commons licenca" style="border-width:0" src="https://i.creativecommons.org/l/by/4.0/88x31.
png" /></a><br />Ovo djelo je dano na korištenje pod licencom <a rel="license"
href="http://creativecommons.org/licenses/by/4.0/">Creative Commons Imenovanje 4.0
međunarodna</a>.
```

Ovo djelo je dano na korištenje pod licencom Creative Commons Imenovanje 4.0 međunarodna.

Slika 2.43. Prikaz u pregledniku postavljene CC licence



Slika 2.44. Ugradnja fotografije s prikazom autora i izvora

Fotografije je moguće ugraditi na *web*-stranicu i s prikazom izvora i autora fotografije. Na internetu se pronađe fotografija koja ima opciju ugrađivanja u HTML kod, embed.

Klikom na gumb embed generira se HTML kod koji se ugrađuje u HTML dokument. Ova mogućnost vidljiva je samo kod onoga sadržaja koji je javno dostupan.

Neke od stranica za slobodno preuzimanje sadržaja:

- https://pixabay.com/
- https://www.stockvault.net/
- https://stocksnap.io/
- https://unsplash.com/
- https://burst.shopify.com/
- https://www.publicdomainpictures.net/en/

2.7. Ugradnja popisa (lista) na *web*-stranicama

Popisi (liste) služe nam za oblikovanje tekstualnih sadržaja na stranici.
U HTML-u razlikujemo tri vrste popisa:

- **numerirani ili poredani** **
** (engl. *ordered list*) – ispred stavki popisa (engl. *list item*) dodaje se redni broj ili slovo.
  Elementi popisa imaju oznaku **
**
- **nenumerirani ili neporedani** **
** (engl. *unordered list*) – ispred stavki se dodaje grafička oznaka. Elementi popisa imaju oznaku **
**
- **definicijski** **<dl></dl>** (engl. *definition list*) – omogućuju kreiranje popisa bez grafičkih oznaka i rednih brojeva. On sadrži pojam **<dt></dt>** (engl. *definition term*) za definiranje naslova i definiciju stavki unutar popisa **<dd></dd>** (engl. *definition description*).

Sve oznake za kreiranje popisa i njegovih stavki imaju parne oznake, znači da imaju i početnu i završnu oznaku.

```
h3>Programi obrazovanja</h3>
   <ol>
      <li>Frontend</li>
      <li>Backend</li>
      <li>DevOps</li>
      <li>Quality Assurance</li>
   </ol>
```

Programi obrazovanja

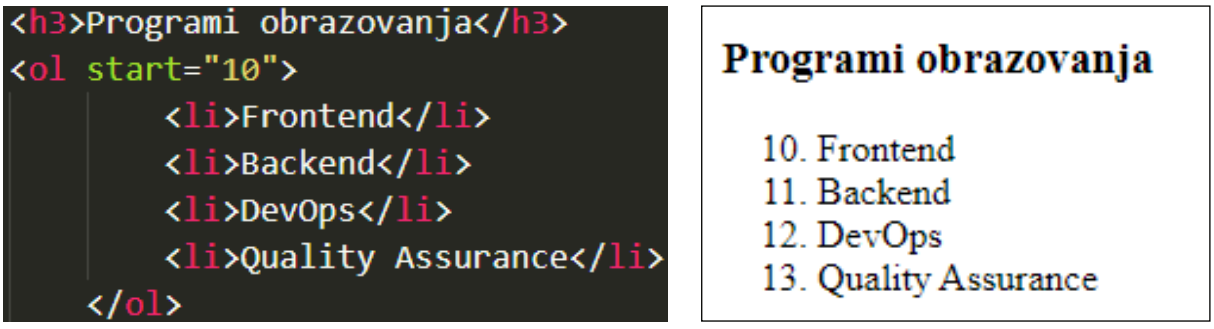
1. Frontend
2. Backend
3. DevOps
4. Quality Assurance

Slika 2.45. Numerirani popis

Zadano je oblikovanje numeriranih popisa arapskim bojevima. Način numeriranja možemo promijeniti pomoću svojstva **type**.

Slika 2.46. Promjena načina numeriranja pomoću svojstva type

Upotrebom svojstva start popis možemo započeti od nekog drugog broja umjesto broja jedan.



Slika 2.47. Primjena svojstva start

Zadano oblikovanje nenumeriranih popisa ispunjeni je crnim kružić (engl. disc), a može se promijeniti upotrebom svojstva **type**.

```
<h3>Programi obrazovanja</h3>
    <ul>
        <li>Frontend</li>
        <li>Backend</li>
        <li>DevOps</li>
        <li>Quality Assurance</li>
    </ul>
```

Programi obrazovanja

- Frontend
- Backend
- DevOps
- Quality Assurance

Slika 2.48. Nenumerirani popis

Svojstvo type može poprimiti vrijednosti: **disc**, **circle**, **square** i **none**.

```
<h3>Programi obrazovanja</h3>
    <ul type="square">
        <li>Frontend</li>
        <li>Backend</li>
        <li>DevOps</li>
        <li>Quality Assurance</li>
    </ul>
```

Programi obrazovanja

- Frontend
- Backend
- DevOps
- Quality Assurance

Slika 2.49. Primjer promjene grafičke oznake svojstvom type

```
<h3>Programi obrazovanja</h3>
<ol >
    <li>Frontend</li>
<ul>
    <li>WEb tehnologije</li>
    <li>HTML</li>
    <li>CSS</li>
</ul>
    <li>Backend</li>
<ul>
    <li>My SQL</li>
    <li>C#</li>
</ul>
    <li>Quality Assurance</li>
<ul>
    <li>Metode testiranja softvera</li>
    <li>Pristupi testiranju softvera</li>
</ul>
</ol>
```

Popise možemo kreirati unutar drugih popisa i na taj način kreirati ugniježđene popise.

Programi obrazovanja

1. Frontend
    - Web tehnologije
    - HTML
    - CSS
2. Backend
    - My SQL
    - C#
3. Quality Assurance
    - Metode testiranja softvera
    - Pristupi testiranju softvera

Slika 2.50. Primjer ugniježđenog popisa

Za tekst koji treba strukturirano formatiranje koriste se definicijski popisi.

```
<h3>Programi obrazovanja</h3>
<dl>
        <dt>Frontend</dt>
            <dd>Suvremene web tehnologije</dd>
            <dd>HTML</dd>
            <dd>CSS</dd>
            <dd>JS</dd>
        <dt>Backend</dt>
            <dd>MySQL</dd>
            <dd>C#</dd>
        <dt>DevOps</dt>
            <dd>Aplikativa rješenja</dd>
            <dd>Infrastrukturna rješenja</dd>
        <dt>Quality Assurance</dt>
            <dd>Metode testiranja softvera </dd>
            <dd>Pristupi testiranju softvera </dd>
</dt>
</dl>
```

Programi obrazovanja

Frontend
- Suvremene web tehnologije
- HTML
- CSS
- JS

Backend
- MySQL
- C#

DevOps
- Aplikativa rješenja
- Infrastrukturna rješenja

Quality Assurance
- Metode testiranja softvera
- Pristupi testiranju softvera

Slika 2.51. Primjer definicijskog popisa

2.8. Ugradnja tablica u *web*-stranice

Tablice omogućuju pregledniji prikaz podataka na *web*-stranici. Podaci se grupiraju u retke i stupce. Koriste se za prikaz različitih izračuna, statističkih podataka, rezultata i slično.

U HTML-u tablica ima tri osnovna elementa:

- granice – predstavlja rubove tablice
- ćelija – pojedinačni element tablice
- raspon ćelija – obuhvaća više ćelija.

HTML elementi za kreiranje tablica jesu:

- **<table></table>** – definira granice tablice
- **<tr></tr>** – definira redak tablice – (eng. *table row*).
  Uvijek je smješten unutar elementa <table>
- **<td></td>** – sadržaj ćelije tablice (eng. *table data*), uvijek se smješta unutar elementa <tr>
- **<thead></thead>** – zaglavlje
- **<th></th>** – zaglavlje tablice – podebljano i centrirano
- **<caption></caption>** – naslov ili opis tablice
- **<tbody></tbody>** – glavni dio tablice.

Svojstva u radu s tablicama:

- **border** – određuje debljinu ruba tablice, zadaje se u pikselima
- **rowspan** – omogućuje spajanje ćelija unutar nekog retka
- **colspan** – omogućuje spajanje ćelija unutar nekog stupca
- **cellspacing** – određuje razmak između ćelija
- **cellpading** – određuje udaljenost sadržaja unutar ćelije od samog ruba ćelije.

Sadržaji unutar tablice uvijek se ispisuju red po red. Širina stupaca definirana je sadrža-

jem unutar ćelija. Svaki stupac širine je koliko je potrebno da sav sadržaj u njegovim ćelijama bude vidljiv. Zbroj širina svih stupaca čini ukupnu širinu tablice ukoliko nismo drugačije definirali širinu tablice. U pregledniku se prikazuje tablica bez obruba. Obrub tablice definira se svojstvom **border**.

```
<table border="1">
    <thead>
        <caption>Popis programa obrazovanja</caption>
        <tr>
            <th>Naziv programa</th>
            <th>Skupne konzultacije</th>
            <th>Individualne konzultacije</th>
            <th>Praktična nastava</th>
            <th>Ukupan broj sati</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Frontend</td>
            <td>53</td>
            <td>24</td>
            <td>178</td>
            <td>255</td>
        </tr>
        <tr>
            <td>Backend</td>
            <td>50</td>
            <td>22</td>
            <td>183</td>
            <td>255</td>
        </tr>
        <tr>
            <td>Quality Assurance</td>
            <td>37</td>
            <td>19</td>
            <td>194</td>
            <td>250</td>
        </tr>
        <tr>
            <td>DevOps</td>
            <td>70</td>
            <td>36</td>
            <td>144</td>
            <td>250</td>
        </tr>
    </tbody>
</table>
```

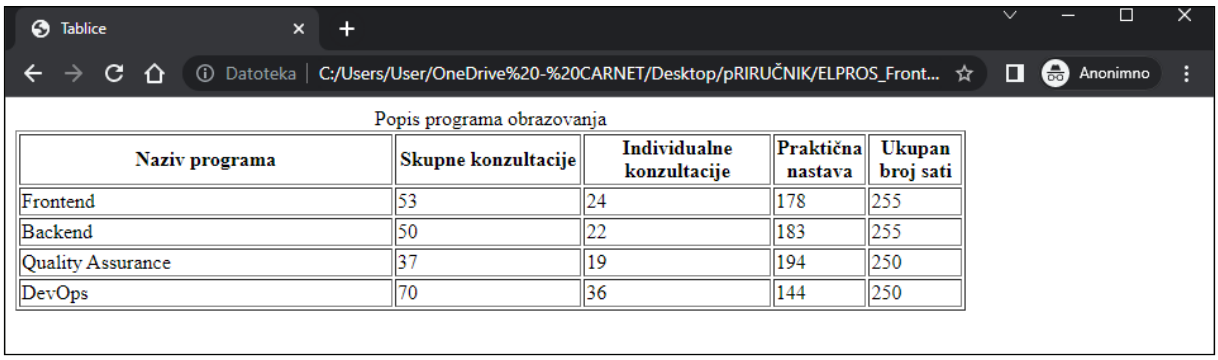
Popis programa obrazovanja

Naziv programa	Skupne konzultacije	Individualne konzultacije	Praktična nastava	Ukupan broj sati
Frontend	53	24	178	255
Backend	50	22	183	255
Quality Assurance	37	19	194	250
DevOps	70	36	144	250

Slika 2.52. Kreiranje tablice

Za definiranje točno određene dimenzije tablice koristimo svojstvo **width**, koje može biti izraženo u pikselima ili u postocima u odnosu na element u kojem se tablica nalazi. Pomoću istog svojstva možemo definirati i širinu ćelije. Po istom principu za određivanje visine redaka ili tablice koristimo se svojstvom **height**.

```
<table width="80%" border="1">
    <thead>
        <caption>Popis programa obrazovanja</caption>
        <tr>
            <th width="50%">Naziv programa</th>
            <th width="10%">Skupne konzultacije</th>
            <th width="20%">Individualne konzultacije</th>
            <th width="10%">Praktična nastava</th>
            <th width="10%">Ukupan broj sati</th>
        </tr>
    </thead>
```



Popis programa obrazovanja

Naziv programa	Skupne konzultacije	Individualne konzultacije	Praktična nastava	Ukupan broj sati
Frontend	53	24	178	255
Backend	50	22	183	255
Quality Assurance	37	19	194	250
DevOps	70	36	144	250

Slika 2.53. Definiranje širine tablice

Svojstvo **width** za tablicu postavljeno je na 80 %, što znači da tablica u pregledniku zauzima 80 % širine prozora. Širina prvog stupca iznosi 50 % širine tablice (od 80 %),

drugi, četvrti i peti stupac su na 10 %, dok je treći na 20 % od širine tablice.

Svojstva **rowspan** i **colspan** koriste se za spajanje ćelija unutar nekog retka odnosno stupca. Njihova vrijednost cijeli je broj i predstavlja broj ćelija koje želimo spojiti. Najčešće ih primjenjujemo kod kreiranja složenih tablica. Svojstvom **cellpadding** određuje se koliko će sadržaj ćelije biti odmaknut od ruba ćelije, dok svojstvom **cellspacing** određujemo razmak između ćelija.

```
<table border="1" cellpadding="5" cellspacing="0.5">
    <caption>Program obrazovanja sa sadržajima i brojem sati</caption>
    <thead>
        <tr>
            <th rowspan="2">Naziv programa</th>
            <th rowspan="2">Sadržaji</th>
            <th colspan="3">Broj sati</th>
        </tr>
        <tr>
            <th>Teorija</th>
            <th>Vježbe</th>
            <th>Ukupno</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td rowspan="8">Frontend programer</td>
            <td>Suvremene web tehnologije</td>
            <td>5</td>
            <td>10</td>
            <td>15</td>
        </tr>
        <tr>
            <td>Strukturiranje web-stranica prezentacijskim jezikom HTML </td>
            <td>10</td>
            <td>20</td>
            <td>30</td>
        </tr>
        <tr>
            <td>Vizualno oblikovanje web-stranice  stilskim jezikom CSS </td>
            <td>10</td>
            <td>30</td>
            <td>40</td>
        </tr>
        <tr>
```

```
            <td>Izrada dinamičkih web-stranica ugradnjom skripti klijentskog skriptnog jezika JavaScript </td>
            <td>20</td>
            <td>40</td>
            <td>60</td>
        </tr>
        <tr>
            <td>Stilski jezik za vizualno oblikovanje web-stranice SASS</td>
            <td>5</td>
            <td>10</td>
            <td>15</td>
        </tr>
        <tr>
            <td>Razvojni okvir za strukturiranje i oblikovanje izgleda web-stranice - Bootstrap </td>
            <td>10</td>
            <td>20</td>
            <td>30</td>
        </tr>
        <tr>
            <td>JavaScript razvojni okvir - React </td>
            <td>12</td>
            <td>48</td>
            <td>60</td>
        </tr>
        <tr>
            <td>Zaštita na radu</td>
            <td>5</td>
            <td>0</td>
            <td>5</td>
        </tr>
        <tr>
            <td colspan="2">Ukupno sati</td>
            <td>77</td>
            <td>178</td>
            <td>255</td>
        </tr>
    </tbody>
</table>
```

Program obrazovanja sa sadržajima i brojem sati

Naziv programa	Sadržaji	Broj sati		
		Teorija	Vježbe	Ukupno
Frontend programer	Suvremene web tehnologije	5	10	15
	Strukturiranje web-stranica prezentacijskim jezikom HTML	10	20	30
	Vizualno oblikovanje web-stranice stilskim jezikom CSS	10	30	40
	Izrada dinamičkih web-stranica ugradnjom skripti klijentskog skriptnog jezika JavaScript	20	40	60
	Stilski jezik za vizualno oblikovanje web-stranice SASS	5	10	15
	Razvojni okvir za strukturiranje i oblikovanje izgleda web-stranice - Bootstrap	10	20	30
	JavaScript razvojni okvir - React	12	48	60
	Zaštita na radu	5	0	5
Ukupno sati		77	178	255

Slika 2.54. Izrada složene tablice

2.9. Ugradnja obrazaca u *web*-stranice

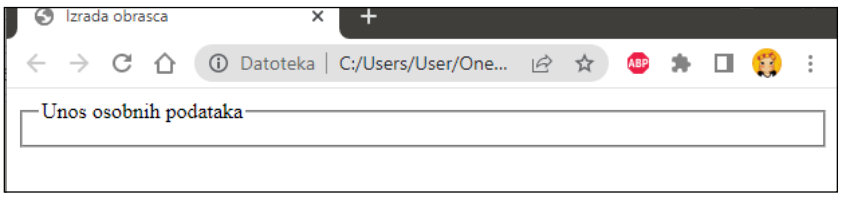
Obrazac (engl. *form*) dio je *web*-stranice pomoću kojeg prikupljamo podatke od korisnika. Obrasci se sastoje od polja koja treba popuniti i gumba kojim se pokreću neke radnje. Svakom polju obrasca može se unaprijed dodijeliti određena vrijednost. Svaki obrazac treba imati gumb za slanje podataka, *submit*. Obrazac se najčešće koristi za registraciju korisnika, odabir ponuđenih stavki, prijava na neki *web*-portal ili *web*-servis, slanje upita i slično. U pravilu obrasci se povezuju s bazom podataka u koju se pohranjuju prikupljeni podaci. Rad s tim podacima izvršava se pomoću skripti, pomoću kojih se podaci dohvaćaju iz baze i šalju serveru na obradu.

Za izradu i uređivanje elemenata obrasca koristimo HTML oznaku **<form></form>**.

Izrada obrasca na *web*-stranici postiže se HTML elementima:

- **<form></form>** – kreiranje obrasca, odnosno govori pregledniku gdje je početak i kraj obrasca
- **<fieldset></fieldset>** – grupiranje polja za unos dodavanjem okvira
- **<legend></legend>** – imenovanje grupe podataka
- **<input></input>** – omogućuje odabir različitih tipova podataka
- **<textarea></textera>** – omogućuje unos više linija teksta. Pomoću svojstava rows i cols može se odrediti proizvoljni broj redaka i stupaca
- **<select></select>** – sadrži dodatan element **<option></option>** pomoću čijeg se atributa **value** mogu postavljati nove vrijednosti
- **<label></label>** – oznaka za opisivanje elementa unosa u polje.

```
<form>
    <fieldset>
        <legend>Unos osobnih podataka</legend>
          </fieldset>
</form>
```



Slika 2.55.
Dodavanje okvira i opisa na elemente obrasca

Oznaka <input></input> može imati sljedeće vrijednosti atributa type:

- **text** – za unos teksta u jednoj liniji
- **radio** – odabir jedne od ponuđenih mogućnosti
- **checkbox** – odabir više ponuđenih mogućnosti
- **file** – omogućuje slanje dokumenta
- **submit** – omogućuje slanje prikupljenih podataka klikom na gumb
- **reset** – briše sve unesene podatke iz obrasca
- **password** – omogućuje unos šifre bez prikaza na zaslonu, uneseni znak vidljiv je kao točka
- **email** – unos adrese elektroničke pošte
- **number** – klikom se odabire broj iz ponuđenog raspona
- **time** – odabir vremena
- **tel** – unos telefonskog broja
- **url** – unos poveznice
- **color** – odabir boje
- **range** – izbor brojčanog raspona
- **button** – potvrdu unosa.

Dodatni atributi elementa <input></input> elementa jesu:

- **required** – unos u polje obavezan je
- **placeholder** – sugestija korisniku što treba upisati.

```
<legend>Unos osobnih podataka</legend>
<label>Ime: </label>
<input type="text" name="ime" placeholder="Ana" required>
<br><br>
<label>Prezime: </label>
<input type="text" name="prezime" placeholder="Anić" required>
<br><br>
<label>Godina rođenja: </label>
<input type="number" name="godina" placeholder="2000" min="1900" max="2022" required>
```

Slika 2.56. Primjena atributa placeholder i required

Svojstvo **name** označava naziv polja (okvira) za unos podataka i šalje se zajedno s unesenim podacima na poslužitelj.

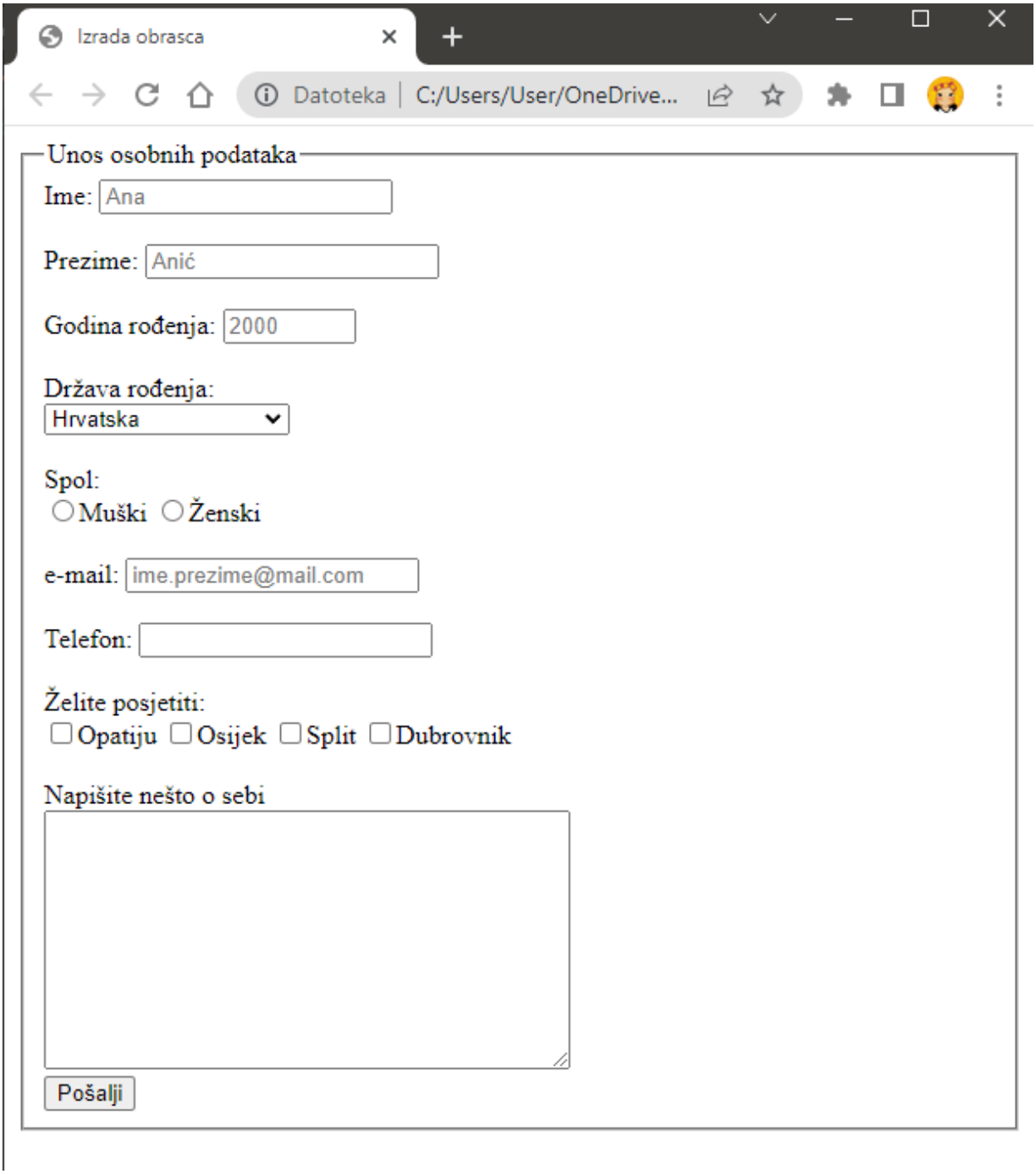
Provjeru unesenih podataka obrasca na strani korisnika možemo postići:

- upotrebom atributa **reqiured** da bi se osigurala popunjenost polja
- upotrebom atributa **type** kojim određujemo jesu li uneseni podaci broj, e-mail, tekst i slično
- upotrebom atributa **min** i **max** provjeravamo najmanju i najveću unesenu vrijednost.

```
<form>
    <fieldset>
        <legend>Unos osobnih podataka</legend>
        <label>Ime: </label>
        <input type="text" name="ime" placeholder="Ana" required>
        <br><br>
        <label>Prezime: </label>
        <input type="text" name="prezime" placeholder="Anić" required>
        <br><br>
        <label>Godina rođenja: </label>
        <input type="number" name="godina" placeholder="2000" min="1900" max="2022" required>
        <br><br>
        <label>Država rođenja: </label>
        <br>
        <select name="drzava">
        <option value="rh">Hrvatska</option>
        <option value="bih">Bosna i Hercegovina</option>
        <option value="slo">Slovenija</option>
        <option value="aut">Austrija</option>
        <option value="ger">Njemačka</option>
        </select>
        <br><br>
        <label>Spol: </label><br>
        <input type="radio" name="spol" value="muško"><label>Muški</label>
        <input type="radio" name="spol " value="žensko"><label>Ženski</label>
        <br><br>
        <label>e-mail: </label>
        <input type=" email " name="e-mail" placeholder="ime.prezime@mail.com" required>
        <br><br>
        <label>Telefon: </label>
        <input type="tel " name="tel" required>
        <br><br>
        <label>Želite posjetiti:</label>
        <br>
        <input type="checkbox" name="izbor1" value="Opatija"><label>Opatiju</label>
        <input type="checkbox" name="izbor2" value="Osijek"><label>Osijek</label>
        <input type="checkbox" name="izbor3" value="Split"><label>Split</label>
        <input type="checkbox" name="izbor4" value="Dubrovnik"><label>Dubrovnik</label>
        <br><br>
        <label>Napišite nešto o sebi</label>
        <br>
        <textarea name="opis" cols="40" rows="10" id="tekst"></textarea>
        <br>
        <input type="submit" name="unos" value="Pošalji">
    </fieldset>
```

Slika 2.57. HTML kod za kreiranje obrasca

Svojstvo **value** vrijednost je koja se šalje na obradu na poslužitelj, ne ispisuje se. Svojstvo **value** definira ulazne podatke, odnosno zadaje vrijednost potvrdnog okvira. Svojstvo **name** predstavlja naziv polja za unos podataka i trebalo bi biti jedinstveno.



Slika 2.58. Izgled obrasca u pregledniku

Pitanja za ponavljanje

1. Što je HTML?
2. Navedite HTML oznake za naslove.
3. Kako kreiramo odlomke?
4. Po čemu se razlikuju semantički i nesemantički strukturni elementi?
5. Kako se kreiraju poveznice?
6. Kako se kreira navigacija web-sjedišta?
7. Koje HTML elemente koristimo za ugradnju multimedije?
8. Nabrojite semantičke strukturne elemente.
9. Koje vrste listi razlikujemo u HTML-u?
10. Koje HTML oznake koristimo za kreiranje tablice?
11. Nabrojite oznake za kreiranje obrasca.
12. Koje atribute koristimo za provjeru popunjenosti polja?

Literatura:

Danijela Ivanović-Ižaković, Anica Leventić, Dinka Šafar Đerki. Mrežna sjedišta i baze podataka, Školska knjiga, Zagreb, 2021. https://www.e-sfera.hr/prelistaj-udzbenik/c5c80429-fea4-49e6-9707-7e4abda1502e (8.12.2021.)

3. Vizualno oblikovanje *web*-stranice stilskim jezikom CSS *(engl. Cascading Style Sheets)*

U OVOM POGLAVLJU NAUČIT ĆETE:

- povezivati HTML dokumente za strukturiranje web-stranice i dokumente s listom stilova (CSS)
- izraditi listu stilova
- uređivati sadržaj web-stranice primjenom liste stilova
- raspoređivati elemente web-stranice primjenom pravila liste stilova.

3.1. Sintaksa stilskog jezika CSS (engl. *Cascading Style Sheets*)

Za uređivanje stilova web-stranica koristi se stilski jezik CSS (engl. Cascading Style Sheets). Za definiranje sadržaja i same strukture na web-stranici koristimo se HTML oznakama. Stilove određenog HTML elementa oblikujemo pomoću CSS pravila. CSS pravila možemo primijeniti ili samo na jedan element, na više elementa ili na sve elemente. Ona HTML elementima daju određeni vizualni izgled prilikom prikazivanja u pregledniku. Organizaciju sadržaja na web-stranici definiramo pozicioniranjem određenih elemenata ili spremnika.

3.1.1. Povezivanje dokumenata za strukturiranje *web*-stranice (HTML) i dokumenta s listom stilova (CSS)

CSS liste stilova možemo primijeniti na:

- jedan element – linijski predlošci (engl. *inline*)
- jedan *web*-dokument (*web*-stranicu) – umetnuti predlošci (interni)
- više *web*-dokumenata (*web*-stranica), vanjski (eksterni) CSS.

Kod linijskog CSS-a stilovi se pišu unutar samog HTML elementa. Nedostatak ovog načina ograničeno je djelovanje svojstva na samo jedan element. Zbog toga je rijetko u upotrebi.

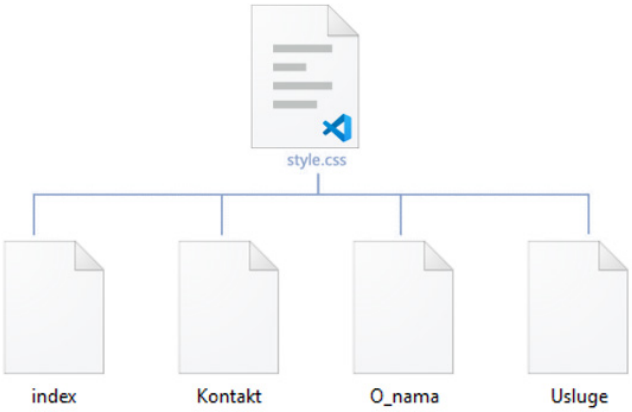
```
<h1 style="color: cadetblue;">Regionalni centar kmpetentnosti</h1>
    <p style="color: chocolate">
        Elektrotehnička i prometna škola Osijek (ELPROS) u srpnju 2018.
        od MZO-a imenovana je Regionalnim centrom kompetentnosti u
        elektrotehnici i informacijsko-komunikacijskim tehnologijama.
    </p>
```

Slika 3.1. Primjena *inline* stila

Kod unutarnjeg CSS-a lista stilova piše se unutar zaglavlja između oznake **<style></style>.**

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.
    0">
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Unutarnji stil</title>
    <style>
    h1 {
        color: cadetblue;
    }
    p{
        color:chocolate;
    }
    </style>
</head>
<body>
    <h1 >Regionalni centar kmpetentnosti</h1>
        <p>
        Elektrotehnička i prometna škola Osijek (ELPROS) u srpnju
        2018. od MZO-a imenovana je Regionalnim centrom kompetentnosti
        u elektrotehnici i informacijsko-komunikacijskim tehnologijama.
        </p>
</body>
</html>
```

Slika 3.2. Primjena unutarnjeg stila

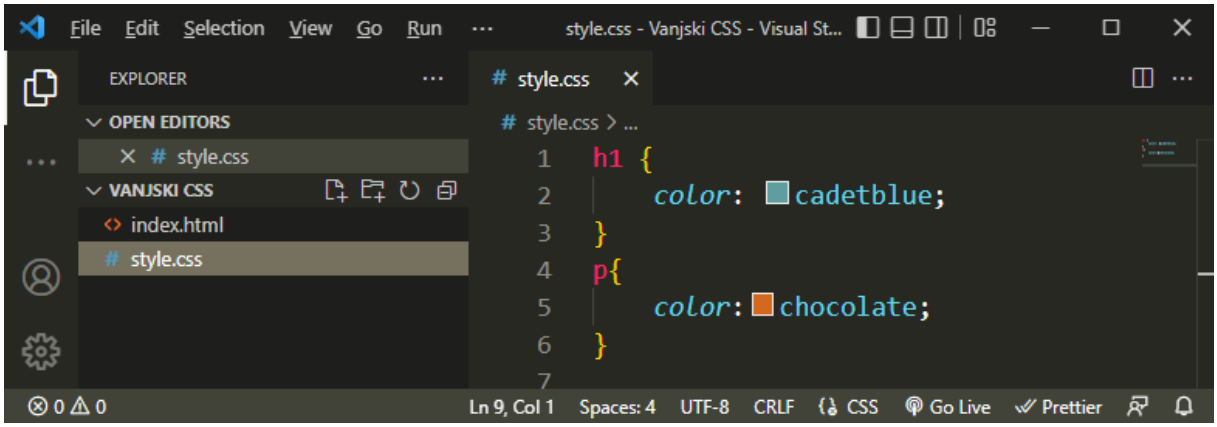


Kod vanjskog CSS-a svi stilovi pišu se unutar jednog dokumenta i preko elementa **<link>** povezuju s HTML dokumentima.

Slika 3.3. Shematski prikaz djelovanja vanjskog CSS-a

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="style.css">
    <title>Vanjski stil</title>
</head>
<body>
    <h1>Regionalni centar kmpetentnosti</h1>

        <p>
            Elektrotehnička i prometna škola Osijek (ELPROS) u srpnju 2018.
            od MZO-a imenovana je Regionalnim centrom kompetentnosti u
            elektrotehnici i informacijsko-komunikacijskim tehnologijama.
        </p>
</body>
</html>
```

Slika 3.4. Primjena vanjskog stila



Slika 3.5. Sadržaj i smještanje datoteke style.css

Na slici 3.5. vidimo listu stilova vanjskog CSS-a. Datoteka style.css smještena je u istu mapu kao i HTML datoteka.

3.1.2. Sintaksa liste stilova

Svako CSS pravilo sastoji se od:

- selektora
- deklaracije.

Pomoću selektora dohvaćamo HTML element na koji će se primijeniti lista stilova. Deklaraciju čini svojstvo i njegova vrijednost. Broj deklariranih svojstava unutar bloka nije ograničen. Više stilova može se pridružiti jednom selektoru, odnosno HTML elementu.

Opći oblik:

```
selektor{
svojstvo_1: vrijednost;
svojstvo_2: vrijednost;
.
.
.
svojstvo_n: vrijednost;
 }
```

3.1.3. Vrste selektora u listama stilova

Selektor možemo podijeliti na:

- univerzalni selektor (*) – odabir svih HTML elemenata web-stranice
- selektor elementa (p, h1, table, body, nav i slično)
- klasni selektor (.class)
- identifikator elementa (#id)
- selektor atributa
- pseudoklase i pseudoelementi
- selektor potomka (engl. *descendant*).

Univerzalni selektor koristimo ako želimo svim elementima *web*-stranice dodijeliti neko univerzalno svojstvo. Ako više selektora djeluje na jedan element, svojstva univerzalnog selektora imaju prioritet.

```
* {
    font-family: 'Open Sans', sans-serif;
    font-size: 1.2vw;
}
```

Slika 3.6. Primjer univerzalnog selektora

Selektori s nazivom elementa najjednostavniji su oblik selektora za primjenu liste stilova. Svi HTML elementi s odabranom oznakom na *web*-sjedištu će imati deklarirane stilove.

```
p,h1{
    color: yellow;
    font-size: 1vh;
}
```

Slika 3.7. Primjer selektora elementa

```
body{
  background-color: lightblue;
 }
```

Slika 3.8. Primjena grupiranja selektora

Ako se ista lista stilova primjenjuje na više selektora, možemo ih navesti jedan iza drugog odvojene zarezom.

Klasne selektore koristimo kada se lista stilova primjenjuje na točno određene HTML elemente. U tom slučaju određeni element označavamo kao klasu koristeći atribut **class**. Prilikom dohvaćanja klase ispred njezina naziva stavlja se točka. Lista stilova primijenit će se samo na elemente koji imaju dodijeljenu tu klasu. Više HTML elemenata može imati klasu istog naziva.

```
<nav>
    <ul>
        <li class="gumb"><a href="index.html">Naslovna</a></li>
        <li class="gumb"><a href="Onama.html">O nama</a></li>
        <li class="gumb"><a href="programi.html">Programi obrazovanja</a></li>
        <li class="gumb"><a href="Kontakt.html">Kontakt</a></li>
    </ul>
</nav>
```

Slika 3.9. Definiranje klase gumb

```
.gumb {
    background-color: #4529c0;
}
```

Slika 3.10. Dohvaćanje klase i deklaracija stilova

Identifikatorom elementa (#id) označava se HTML element za koji se želi postaviti jedinstvena lista stilova. U jednom dokumentu moguć je samo jedan id selektor tog imena.

```
<div id="naslov">
    <h1>Programi obrazovanja</h1>
</div>
```

```
#naslov {
    color: #49488b;
}
```

Slika 3.11. Postavljanje identifikatora elementa

Selektor atributa koristi se za stiliziranje HTML elemenata koji imaju specifične atribute ili vrijednosti atributa. Primjer gumb za predaju podataka iz obrasca.

```
<form>
    <label>e-mail:</label>
    <input type="email" name="email"
           placeholder="Unesite svoju e-mail adresu"><br/>
    <input type="submit" value="Pošalji!">
</form>
```

Slika 3.12. Kreiranje HTML obrasca

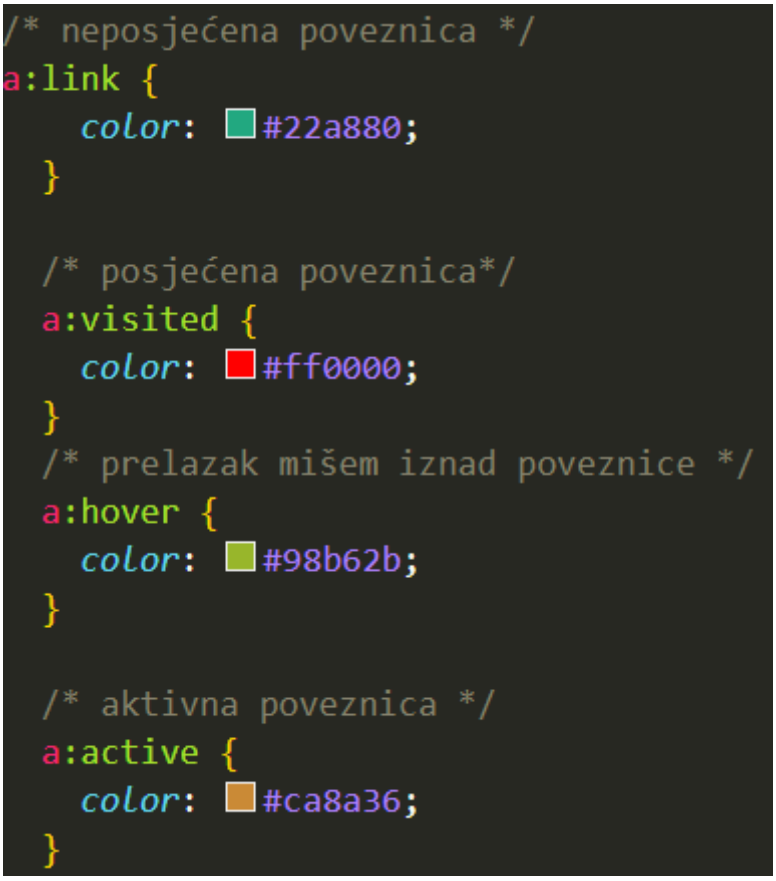
```
input[type="submit"] {
    background-color: #4b39b3;
    color: #e1e6d5;
}
```

Slika 3.13. Selektor atributa

Pseudoklase i pseudoelementi omogućuju primjenu liste stilova na neke selektore ovisno o djelovanju korisnika. Često se koriste kod poveznica i definiraju njihova stanja kad korisnik pređe mišem preko njih ili klikom na njih. Mogu se definirati različite liste stilova u odnosu na izvedenu radnju. Primjerice, moguće je definirati različite stilove za posjećene i neposjećene stranice, promjenu boje ili izgleda prilikom prelaska mišem preko određenog elementa i slično.

Pseudoklase su:

- **:first-child**
- **:last-child**
- **:nth-child()**
- **:first-of-type**
- **:last-of-type**
- **:nth-last-child().**



Slika 3.14. Primjer pseudoklasa

Pseudoelementi omogućuju odabir jednog dijela unutar odabranog HTML elementa. Primjerice, prvo slovo ili prva rečenica unutar nekog odlomka i slično.

Sintaksa za pseudoelemente:

```
selector:pseudoelement {
svojstvo: vrijednost;
}
```

Pseudoelementima služimo se kako bismo postavili sadržaj iza ili ispred sadržaja elementa. Pomoću njih možemo urediti samo dio sadržaja nekog elementa.

Pseudoelementi su:

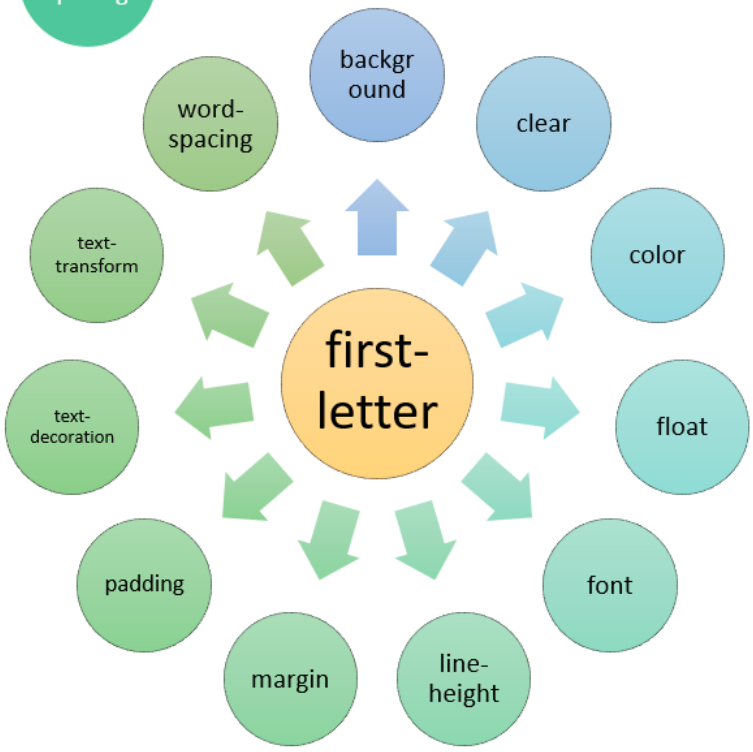
- **::after**
- **::before**
- **::first-letter**
- **::first-line**
- **::selection.**

```
{
  font-size: 12px;
  }
:first-line {
  font-size: 14px;
  color: #7a1976;
  }
```

Slika 3.15. Primjer pseudoelementa

Iako odlomak ima veličinu fonta 12 px, na prvu liniju teksta primijenit će se definirana lista stilova, odnosno veličina fonta 14 px i boja čija je vrijednost #7a1976.

Slika 3.16. Prikaz svojstava koja mogu biti dodijeljena pseudoelementu **first-line**



Slika 3.17. Prikaz svojstava koja mogu biti dodijeljena pseudoelementu **first-letter**

Tijekom pisanja CSS pravila jako je važno da pomoću selektora dohvatimo onaj HTML element koji želimo stilski urediti. Često se jedno pravilo može primijeniti na više elemenata ili se više pravila primjenjuje na jedan element. U tom se slučaju CSS lista stilova izvršava redom kako ih preglednik čita. Za više definiranih stilova na nekom elementu primijenit će se zadnji navedeni stil.

!important pravilo nam omogućuje da sami kreiramo svoju listu stilova, odnosno damo veću važnost nekom svojstvu od uobičajenog.

```
<p>Pravilo !important</p>
<p class="tekst1">Pravilo !important</p>
<p id="tekst2">Pravilo !important</p>
```

```
#tekst2 {
    background-color: chocolate;
  }
.tekst1 {
    background-color: #98b62b;
  }
p {
    background-color: darkcyan !important;
  }
```

Slika 3.18. Primjer primjene **!important** pravila

Bez obzira na mjesto gdje je svojstvo **!important** navedeno, ono se uvijek primjenjuje i izvršava bez obzira na sva ostala definirana svojstva za taj element.

Slika 3.19. Izgled u pregledniku prije i nakon primjene pravila **!important**

Selektor potomka (engl. *descendant*) koristi se za dohvat ugniježđenih elemenata.

```
nav ul {
    font-family: Arial, Verdana;
    font-size: 14px;
    }
```

Slika 3.20. Primjer selektora potomka

3.1.4. Mjerne jedinice vrijednosti u listi stilova

CSS svojstva imaju svoju vrijednost koja se definira prilikom pisanja CSS pravila. Ukoliko vrijednost svojstva nije pravilno navedena, preglednik ga neće primijeniti.

Mjerne jedinice dijelimo na:

- **apsolutne – njihova je vrijednost točno određena:**
  - ***px*** – (engl. *pixel*) – broj zaslonskih točaka
  - ***pt*** – (engl. *point*) – broj točaka – veličine fonta
- **relativne – vrijednost ovisi o drugom elementu ili prozoru preglednika jer se određuje u odnosu prema njima**:
  - ***em*** (engl. *element*) – relativna jedinica za veličinu fonta (1 em = 12 pt = 16 px = 100%)
  - ***rem*** (engl. *root element*) – veličina se određuje prema body elementu
  - ***%*** – postotak se određuje u odnosu prema roditeljskom elementu, a ne prozoru preglednika
  - ***vw*** (engl. *viewport width*) – širina prikaza (1 vw = 1% širine prozora)
  - ***vh*** (engl. *viewport height*) – visina prikaza (1 vh = 1% visine prikaza)
- **posebne – koriste se za samo određena svojstva tranzicija**
  - ***s*** – sekunde za određivanja vremena trajanja tranzicije
  - ***ms*** – milisekunde za preciznije određivanje vremenskog trajanja prijelaza
  - ***deg*** – stupnjevi kao mjerna jedinica za rotaciju.
  - Pozitivne vrijednosti određuju smjer rotacije u smjeru kazaljke na satu, a negativne u suprotnom smjeru.

3.2. Uređivanje sadržaja *web*-stranice primjenom liste stilova

Sadržaj *web*-stranice stilizira se primjenom određenih CSS pravila. CSS pravila mogu se dodijeliti jednom HTML elementu ili skupini njih. Upotrebom univerzalnog selektora lista stilova može se dodijeliti svim HTML elementima. Svi HTML elementi imaju pravokutni oblik prikaza u *web*-pregledniku.

3.2.1. Uređivanje pozadine *web*-stranice primjenom pravila liste stilova

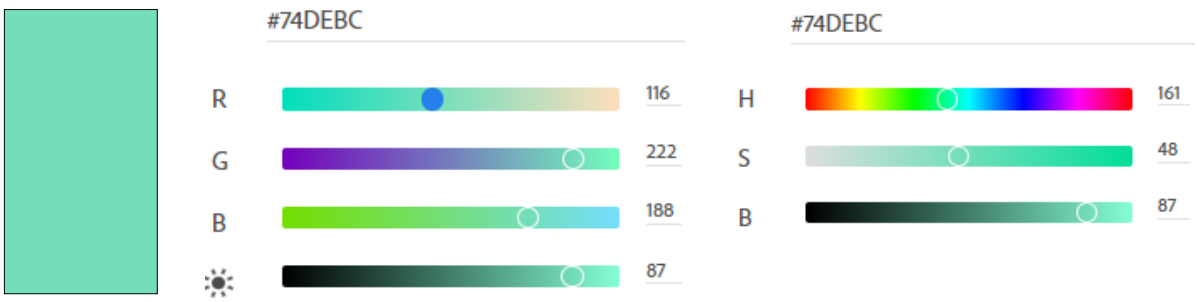
Pozadina stranice, odnosno HTML elementa može biti ispunjena bojom, gradijentom ili pozadinskom slikom.

Svojstva kojim se može definirati pozadinska lista stilova:

- **background-color** – boja pozadine
- **background-image** – slika kao pozadina
- **background-repeat** – ponavljanje pozadinske slike
- **background-position** – pozicija pozadinske slike
- **background-size** – veličina pozadinske slike.

Boje se u kodu mogu zadati na više načina:

- imenom boje
- RGB zapisom – rgb(116, 222, 188)
- HSL/HSB zapisom – hsl(161%, 62%, 66%)
- heksadecimalni zapis – #74debc

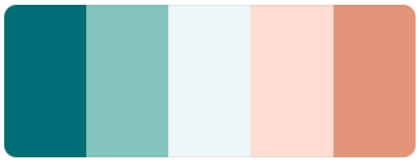


Slika 3.21. Primjer generiranja boje

Heksadecimalni format zapisa sadrži udio crvene, zelene i plave boje u pojedinoj nijansi, oblik zapisa #RRGGBB. To su heksadecimalni cijeli brojevi između 00 i FF koji određuju intenzitet boje. HSL/HSB (engl. *hue, saturation, lightness/brightnes*) format boje određuju tri komponente: ton boje, zasićenost i svjetlina.

Palete boja mogu se preuzeti s različitih generatora za boje:

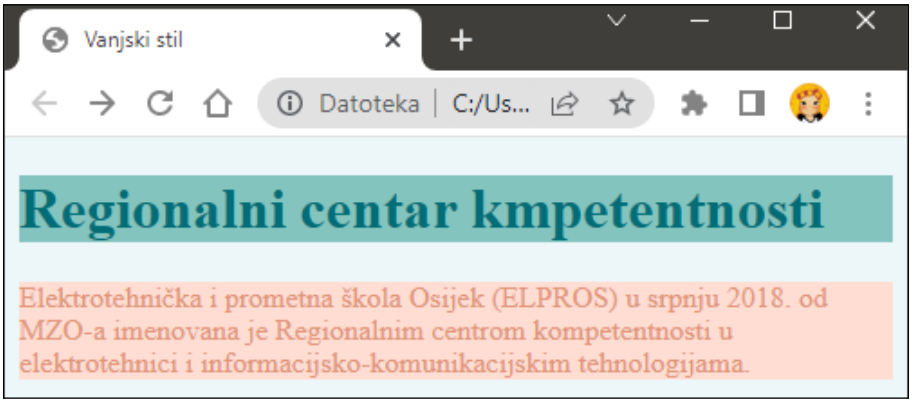
- https://coolors.co/palettes/trending
- https://www.canva.com/colors/color-palette-generator/
- https://colors.dopely.top/palette-generator/unsaved-palette
- https://www.w3schools.com/colors/colors_picker.asp



Slika 3.22. Odabrana paleta boja

```
h1 {
    color: #006d77;
    background-color: #83c5be ;
}
p{
    color: #e29578;
    background-color: #ffddd2;
}
body
{
    background-color: #edf6f9;
}
```

Slika 3.23. Oblikovanje pozadine odabranom paletom bojom



Slika 3.24. Izgled u pregledniku

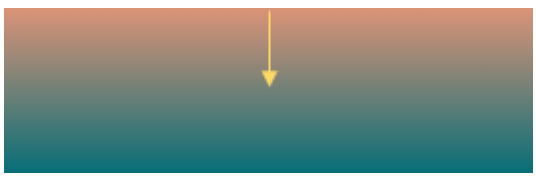
Pozadina stranice ili elementa može se urediti i prijelazima i pomoću funkcije **linear-gradient()**. Argumenti ove funkcije jesu oznaka za smjer, naziv boja s kojima radimo prijelaz. Za stvaranje prijelaza funkcijom **linear-gradient()** moraju se navesti najmanje dvije boje. Kao treći argument može se dodati i smjer prijelaza. Smjer prijelaza obavezno se navodi prije imena boja.

Slika 3.25. Funkcija linear-gradient(#e29578, #006d77);

Ova funkcija može imati više smjerova prijelaza iz boje u boju.
Prikaz smjerova prijelaza i njihov izgled nalazei se u tablici ispod.

Opći oblik zapisa:

```
background-image: linear-gradient(smjer, boja1, boja2);
```

Smjer prijelaza	Izgled u pregledniku
to right	
to top	
to bottom	

to left	
to top left	
to bottom right	
to bottom left	
to top right	

Tablica 3. 1. Prikaz djelovanja funkcije **linear-gradient()**

Kod postavljanja pozadinske slike potrebno je paziti na njene dimenzije i način prikazivanja.

Pozadinska slika postavlja se pomoću svojstva **background-image**. Vrijednost svojstva **background-image** navodi se unutar url-a, kojim se navodi putanja do slike. Svojstvom **background-size** određuje se veličina slike u pozadini.

Vrijednosti svojstva background-size jesu:

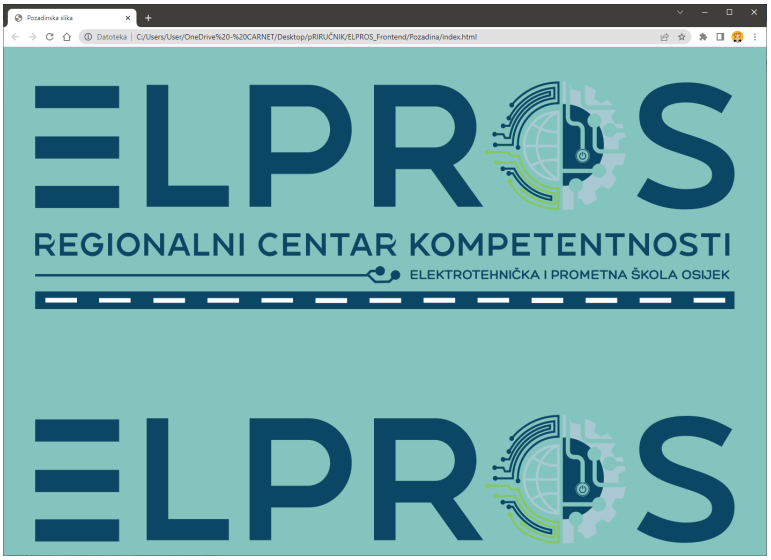
- **cover** – prilagodba slike veličini elementa po dužem rubu, može biti odsječena slika
- **contain** – prilagodba slike veličini elementa po kraćem rubu, vidljiva

cijela slika

- **mjerne jedinice** – px, em, vw, vh
- **postotak** – slika zauzima određeni postotak od veličine elementa
- **auto** – veličina slike računa se proporcionalno prema zadanoj jednoj dimenziji
- **inherit** – veličina slike nasljeđuje se od roditeljskog elementa.

```
body
{
        background-color: #83c5be;
        background-image: url("elpros.png");
}
```

Slika 3.26. Postavljanje pozadinske slike



Slika 3.27. Izgled pozadinske slike u pregledniku

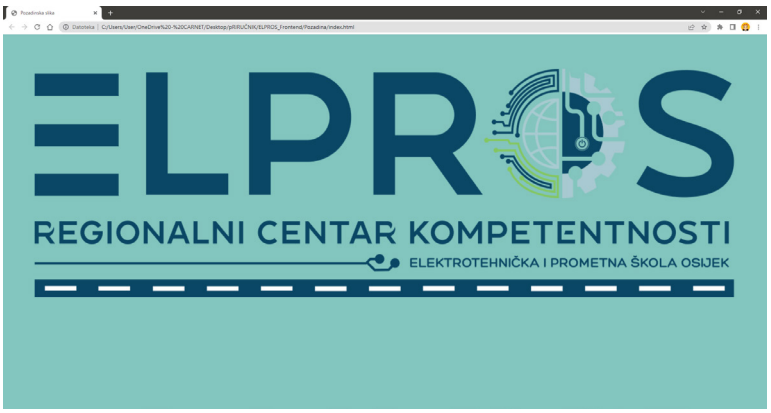
Slika se postavlja vodoravno i okomito sve dok ne popuni element.
Svojstvo **background-repeat** omogućuje promjenu načina ponavljanja slike.

Svojstvo background-repeat može poprimiti vrijednosti:

- **repeat** – ponavljanje slike po osi x i osi y
- **no-repeat** – bez ponavljanja slike
- **repeat-x** – ponavljanje po x osi
- **repeat-y** – ponavljanje po y osi
- **space** – ponavljanje slike s jednakim razmacima
- **round** – smanjene slike za ponavljanje.

```
body
{
        background-color: #83c5be;
        background-image: url("elpros.png");
        background-size: cover;
        background-repeat: no-repeat;
}
```

Slika 3.28. Bez ponavljanja slike

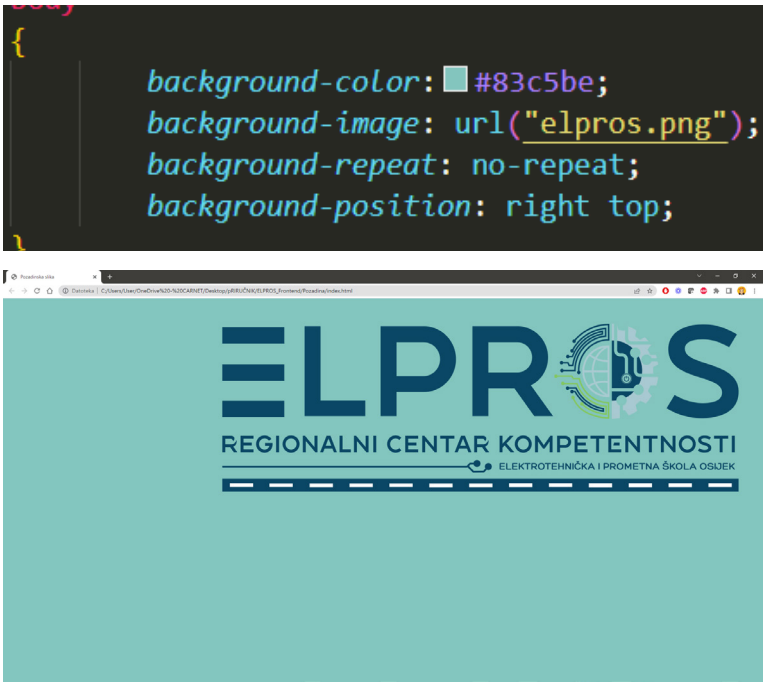


Slika 3.29. Izgled u pregledniku no-repeat

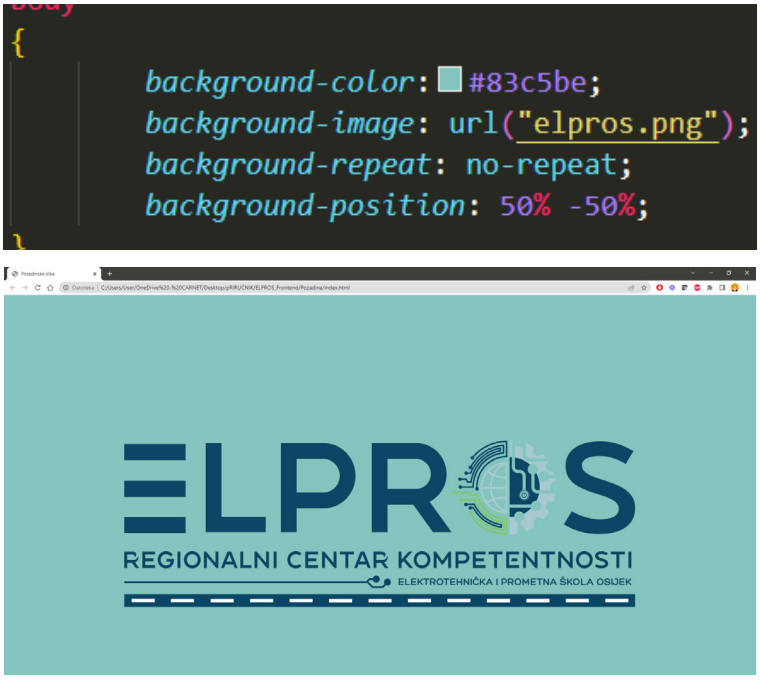
Pomoću svojstva **background-position** definiraju se početne koordinate slike. Početne pozicije po podrazumijevanim su postavkama gornji lijevi kut, odnosno pozicija (0,0).

Svojstvo background-position može poprimiti vrijednosti:

- **left top**
- **left bottom**
- **left center**
- **right top**
- **right bottom**
- **right center**
- **center top**
- **center bottom**
- **center center.**



Slika 3.31. Izgled primjene svojstva **background-position** u pregledniku



Slika 3.33. Vrijednost svojstva **background-position** izražena u % prikaz u pregledniku

Svojstvom **background-attachment** koristimo se za određivanje ponašanja pozadinske slika pri radu s kliznom trakom. Vrijednosti koje može poprimiti dane su u tablici ispod.

Vrijednost	Opis
`scroll`	Pozadinska slika pomiče se sa stranicom.
`fixed`	Pozadinska slika ne pomiče se sa stranicom, fiksna je.
`local`	Pozadinska slika pomiče se s određenim elementom.
`inherit`	Vrijednost će se naslijediti od roditelja.

Tablica 3. Vrijednosti svojstva background-attachment

Pozadinska svojstva možemo navesti unutar jedne deklaracije:

```
background: color image repeat position/size;
```

Važno je da svojstvo **background-size** bude iza svojstva **background-position** i odvojeno kosom crtom. Ostala svojstva navodimo proizvoljnim redoslijedom.

Moguće je postaviti i više pozadinskih slika navodeći ih jednu iza druge.

```
background: url("elpros.png"), url("logo.png");
```

3.2.2. Oblikovanje teksta *web*-stranice primjenom pravila liste stilova

Tekstovi su važan dio svake *web*-stranice. Poželjno je da budu pisani jednostavno i pravopisno ispravno. Pri kreiranju svake stranice kreće se od tekstualnih elemenata, koje je potrebno stilski urediti. Tekstualni sadržaji u web-dokumentu jesu naslovi i podnaslovi, odlomci i popisi, poveznice, izbornici i slično. Tekstualni sadržaji na web-stranici zauzimaju više od 80 % svih sadržaja.

Svojstva koja se odnose na font jesu:

- **font-family** – vrste fontova: `serif, sans-serif, monospace, script, display,fantasy`
- **font-size** – veličina fonta: numeričke mjerne jedinice jesu px, pt, em, rem, %, vw, vh; a opisne su xx-large, x-large, large, larger, medium, small, smaller, x-small i xx-small
- **font-style** – stil fonta: vrijednosti `italic, oblique, normal i inherit`
- **font-weight** – debljina fonta: vrijednosti normal, `bold, bolder, lighter i inherit` te numeričke vrijednosti
- l**ine-height** – prored: vrijednosti `normal, px, em, %, broj i inherit`
- **font-variant** – omogućuje ispis velikih slova smanjenim fontom. Vrijednosti su `normal` i `small-caps`
- **font – zapis** istodobne upotrebe više gore navedenih svojstava. Važan je redoslijed navođenja.

```
font: style variant weight size/line-height family
font: italic 12pt / 1.5 “Times New Roman”, serif
```

Najčešće su u upotrebi Serif i Sans Serif fontovi.
Razlikuju se po tome imaju li ili nemaju ukrase (Serife) na rubovima slova. Serif fontovi imaju ukrase na rubovima, dok Sans Serif nemaju.

Slika 3.34. Primjeri San Serif i Serif fonta

- Google fontove možete preuzeti na linku: Browse Fonts - Google Fonts

Za uključivanje *web*-fontova potrebno je koristiti se posebnim pravilom **@font-face.**

```
@font-face {
  font-family: "OpenSans";
  src: url("/fonts/OpenSans-Regular.woff");
  src: url("/fonts/OpenSans-Regular.ttf");
  font-weight: normal;
  font-style: normal;
}
```

Slika 3.35. Primjer upotrebe @font-face pravila

Moguće je postaviti više formata fontova na poslužitelj radi maksimalne kompatibilnosti s različitim preglednicima. Preporučeni su formati: .WOFF, .WOFF2 i TTF/OTF. Kod formiranja pravila potrebno je prvo navesti obitelj fontova, a zatim putanju do fonta.

Slika 3.36. Odabir fonta za web-stranicu

Google fontovi mogu se uključiti u *web*-dokument pomoću **@import** pravila. Na početku CSS dokumenta navodi se i pravilo **@charset**, koje omogućuje postavljanje

kodnih stranica za posebne vrste znakova. Ima istu ulogu kao <meta charset = “UTF–8”> u HTML-u.

```
t "UTF-8";
url('https://fonts.googleapis.com/css2?family=Open+Sans:wght@300&display=
```

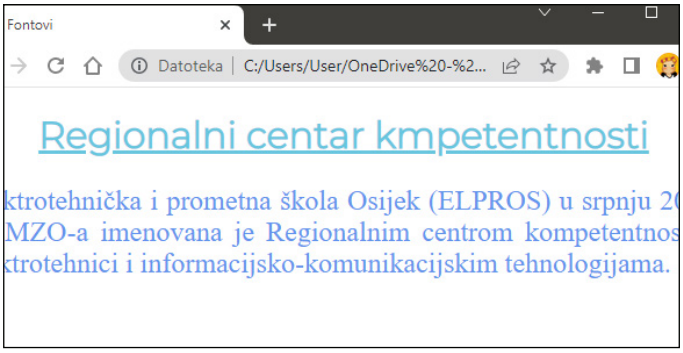
CSS pravilo za navođenje obitelji fontova jest:

```
font-family: 'Open Sans', sans-serif;
```

Svojstva kojima se koristimo za stiliziranje teksta jesu:

- **color** – boja teksta: vrijednosti: naziv boje, hex ili rgb oznaka
- **text-align** – poravnanje: vrijednosti `left, right, justify i center`
- **text-decoration** – dodavanje linije: vrijednosti `underline, overline, line–through, none`
- **text-indent** – uvlačenje teksta
- **text-trnansform** – velika i mala slova: vrijednosti `uppercase, lowercase, capitalize i none`
- **letter-spacing** – razmaci između znakova: sve mjerne jedinice osim %
- **word-spacing** – razmaci između riječi: sve mjerne jedinice osim %
- **text-shadow** – dodavanje sjene: vrijednosti `h–sadow, v–shadow, blur–radius, color.`

```
h1 {
    font-family: Montserrat;
    color: #68c6df;
    text-align: center;
    vertical-align: middle;
    text-decoration: underline;
}

p{
    font-size: 1.5em;
    text-align: justify;
    color: cornflowerblue;
}
```



Slika 2.37. Primjer primjene svojstava za rad s tekstom

Svojstvom **text-shadow** dodajemo sjenu tekstu.

Vrijednost ovog svojstva može imati četiri argumenta:

- boju sjene
- horizontalni pomak
- vertikalni pomak
- radijusa zamućenja.

Tekstu se može zadati i više sjena koje pri deklaraciji odvajamo zarezom.

CSS svojstva	Izgled teksta u pregledniku
`color: navy;` `text-shadow: 5px 5px orange;`	
`text-shadow: 3px 3px 3px gray;`	
`color: white;` `text-shadow: 0 0 10px gray;`	

`color: rgb(221, 226, 216);`
`font-size: 100px;`
`text-shadow:`
`0 5px rgba(120, 122, 14, 0.4),`
`0 10px rgba(32, 44, 150, 0.4),`
`0 15px rgba(165, 25, 25, 0.4);`
	

Tablica 3.2. Različiti primjeri upotrebe svojstva **text-shadow**

3.2.3. Oblikovanje poveznica *web*-stranice primjenom pravila liste stilova

Poveznice na *web*-stranici jako su važan element. One služe za interakciju s korisnikom, pa ih je potrebno istaknuti radi preglednosti i lakšeg kretanja kroz navigaciju.

Stanja poveznica definiramo pseudoklasama, koje omogućuju odabiranje elementa. Preglednici imaju zadane postavke za sve poveznice, ali često ih je potrebno mijenjati.

Poveznice mogu biti u različitim stanjima. Pseudoklase kreiraju se na način da se selektoru elementa pridružuje dvotočka (:) za označavanje stanja poveznice:

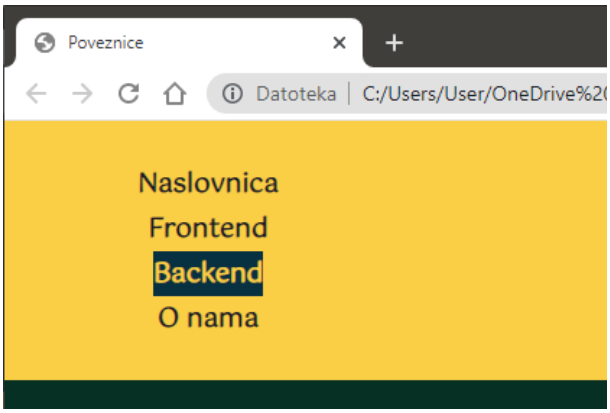
- **a:link** – neposjećena
- **a:visited** – posjećena
- **a:hover** – prelazak miša preko poveznice
- **a:active** – aktivne pov eznice.

Poveznica istovremeno može biti u više različitih stanja. Potrebno je paziti na redoslijed njihova postavljanja kako bi se poveznica mogla pokrenuti.

Pravilan redoslijed:

- **a:hover** mora doći iza **a:link** i **a:visited**
- **a:active** mora doći iza **a:hover** kako bi bilo aktivno.

```
li {
    list-style-type: none;
    width: 20vw;
}

a {
    text-decoration: none;
    color: #1D1C26;
    font-weight: bold;
}
a:hover {
    background-color: #073040;
    color: #FBCF45
}
a:visited{
    color: #1e7c9a;
}
a:active{
 background-color: #1e7c9a;;
 color: brown;
}
```

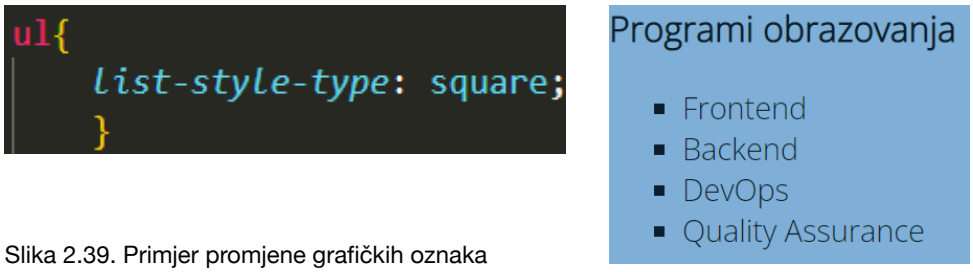


Slika 2.38. Primjer promjene pseudoklasa

3.2.4. Oblikovanje popisa (listi) *web*-stranice primjenom pravila liste stilova

Popisi nam služe za nabrajanje, a najčešće ih koristimo kod izrade izbornika. Izgled popisa definiran je postavkama preglednika, ali ako to želimo promijeniti, definiramo za njih liste stilova. Numerirani popisi imaju zadano oblikovanje arapskim bojevima, dok nenumerirani popisi imaju zadano oblikovanje ispunjenim kružićem (engl. *disc*).

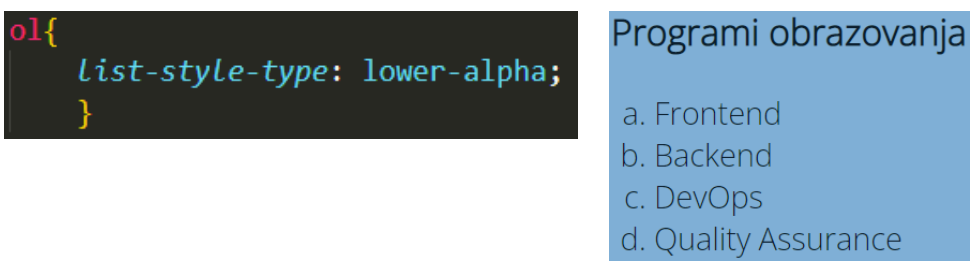
Promjena grafičkih oznaka kod nenumeriranih popisa može se postići pomoću svojstva **list-style-type**.
Vrijednosti koje može poprimiti jesu: `circle, disc, square, none`.



Slika 2.39. Primjer promjene grafičkih oznaka

Svojstvo i vrijednost	Opis
`list-style-type:disc;`	Oznaka je ispunjeni krug.
`list-style-type:circle;`	Oznaka je kružić.
`list-style-type:square;`	Oznaka je kvadrat.
`list-style-type:none;`	Bez grafičke oznake.
`list-style-image: url("naziv.format");`	Oznaka je slika.

Tablica 3.3. Oblikovanje oznaka nenumeriranih popisa



Slika 2.40. Primjer promjene označavanja

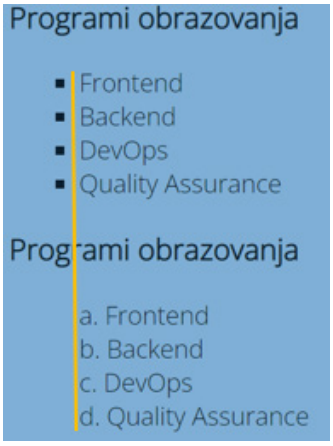
Svojstvo i vrijednost	Opis
`list-style-type:upper-roman;`	označavanje velikim rimskim brojevima
`list-style-type:lower-roman;`	označavanje malim rimskim brojevima
`list-style-type:upper-alpha;`	označavanje velikim slovima abecede
`list-style-type:lower-alpha;`	označavanje malim slovima abecede
`list-style-type:decimal-leading-zero`	označavanje brojevima s 0 ispred (01, 02…)
`list-style-type:decimal;`	označavanje brojevima
`list-style-type:lower-greek;`	označavanje malim grčkim alfabetom
`list-style-type:none;`	bez oznake numeriranja

Tablica 3.4. Oblikovanje oznaka numeriranih popisa

Numerirani popisi imaju zadano oblikovanje arapskim bojevima, ali promjenom vrijednosti svojstva **list-style-type** mijenja se i način označavanja. U tablici se nalazi popis vrijednosti i njihovo značenje.

Svojstvom **list-style-position** oznake se mogu pozicionirati u odnosu na tekst. Vrijednosti ovog svojstva jesu `inside` ili `outside`.

```
ul{
    list-style-type: square;
    list-style-position: outside;
    }
ol{
    list-style-type: lower-alpha;
    list-style-position: inside;
    }
```



Slika 2.41. Primjer pozicioniranja popisa

3.2.5. Oblikovanje tablica na *web*-stranici primjenom pravila liste stilova

U pregledniku podrazumijevani prikaz tablica bez rubova je i s podebljanim tekstom zaglavlja. Listu stilova definira se dohvaćanjem pojedinog elementa tablice. Izradit ćemo listu stilova za složenu tablicu kreiranu u poglavlju 2. HTML (Slika 2.53.).

```
table,
th,
td {
    border: 2px solid #15801e;
    border-collapse: collapse;
    padding: 5px 10px;
}
```

Slika 3.42. Stilovi obruba

Svojstvom **border** definirani su debljina, boja i obrub tablice. Svojstvo **border-collapse** uklanja dvostruke obrube ćelija i tablice. Linije obruba mogu biti različite ovisno o vrijednostima koje mogu poprimiti. Svojstvom **padding** postiže se odmak od rubova ćelije.

Širinu i visinu tablice definiramo svojstvima **width** i **height**. Postavljane širine na 80 % znači da će tablica zauzeti 80 % prikaza u prozoru.

```
table {
    width: 80%;
    margin-left: 10vw;
}
```

Slika 3.43. Svojstva tablice

Širinu tablice moguće je izraziti i relativnom mjernom jedinicom vw (width: 80vw), koja će dati prilagodljiv prikaz tablice. 80vw znači da tablica zauzima 80 % širine prikaza bez obzira na veličinu prozora.

```css
tr {
    height: 5vh;
    vertical-align: middle;
    background-color: #e4d555;
}

td,
th {
    padding: 5px 10px;
    text-align: center;
}
```

Slika 3.44. Uređivanje ćelija

Boja pozadine i font definiraju se preko elementa tablice. Svojstvo **vertical-align** koristi se za vertikalno poravnanje sadržaja u ćeliji, a može poprimiti vrijednosti `middle, top i bottom.`

Naziv tablice dohvaća se pomoću elementa **caption**. Može se pozicionirati na tri načina koristeći svojstvo **caption-side**:

- **top** – iznad tablice
- **bottom** – ispod tablice
- **inherit** – pozicija se preuzima od roditeljskog elementa.

```css
caption {
    font-family: 'Open Sans', sans-serif;
    caption-side: bottom;
    text-align: center;
    font-weight: bold;
}
```

Slika 3.45. Lista stilova naziva tablice

Naziv programa	Sadržaji	Broj sati		
		Teorija	Vježbe	Ukupno
Frontend programer	Suvremene web tehnologije	5	10	15
	Strukturiranje web-stranica prezentacijski jezikom HTML	10	20	30
	Vizualno oblikovanje web-stranice stilskim jezikom CSS	10	30	40
	Izrada dinamičkih web-stranica ugradnjom skripti klijentskog skriptnog jezika JavaScript	20	40	60
	Stilski jezik za vizualno oblikovanje web-stranice SASS	5	10	15
	Razvojni okvir za strukturiranje i oblikovanje izgleda web-stranice - Bootstrap	10	20	30
	JavaScript razvojni okvir - React	12	48	60
	Zaštita na radu	5	0	5
Ukupno sati		77	178	255

Program obrazovanja sa sadržajima i brojem sati

Slika 3.46. Izgled tablice u pregledniku

Naglašavanje dijelova tablice promjenom boje dok prelazimo mišem iznad ćelije ili retka može se postići pseudo klasom **hover**.

```
tr:hover {
    background-color: #366e26;
    color: #d1cf46;
}
```

Slika 3.47. Pseudoklasa hover

Naziv programa	Sadržaji	Broj sati		
		Teorija	Vježbe	Ukupno
Frontend programer	Suvremene web tehnologije	5	10	15
	Strukturiranje web-stranica prezentacijski jezikom HTML	10	20	30
	Vizualno oblikovanje web-stranice stilskim jezikom CSS	10	30	40
	Izrada dinamičkih web-stranica ugradnjom skripti klijentskog skriptnog jezika JavaScript	20	40	60
	Stilski jezik za vizualno oblikovanje web-stranice SASS	5	10	15
	Razvojni okvir za strukturiranje i oblikovanje izgleda web-stranice - Bootstrap	10	20	30
	JavaScript razvojni okvir - React	12	48	60
	Zaštita na radu	5	0	5
Ukupno sati		77	178	255

Program obrazovanja sa sadržajima i brojem sati

Slika 3.48. Izgled djelovanja pseudoklase hover u pregledniku

3.2.6. Ugradnja animacija u *web*-stranicu primjenom liste stilova

Animacije u CSS-u kreiraju se tako da se u određenom vremenskom periodu mijenjaju vrijednosti odabranih svojstava.
Dva svojstva pomoću kojih je to moguće napraviti jesu:

- **transition**
- **animation.**

CSS stilovima postupno se mijenja vrijednost određenih svojstava, a kontrola animacije postiže se pomoću ključnih okvira (engl. *keyframes*). Animacija se sastoji od liste stilova koji je opisuju i niza okvira koji određuju njena međustanja.

Broj promjena stilova nije ograničen. Nisu sva CSS svojstva pogodna za izradu animacija. CSS animacija niz je slika određenog objekta koji svojim rasporedom stvaraju privid pokreta.

Vrijeme prikaza dijelova animacije definira se postotkom i sve vrijednosti unutar ključnog okvira moraju biti poredane uzlazno. Ako dođe do preklapanja vrijednosti, koristi se zadnje navedeno. Osim ključnog okvira pri izradi CSS animacija za promjenu elementa iz jednog stanja u drugo koriste se prijelazi (tranzicije).

Pokretanje animacije razlikuje se kod ova dva svojstva jer se kod svojstva **animation** animacije mogu pokretati i automatski pri njihovu učitavanju, dok kod svojstva **transition** to nije moguće.

Osim toga, tranzicije se mogu pokrenuti samo jednom, dok je animacije moguće ponavljati određeni broj puta. Kod tranzicija svojstvo se prebacuje iz jednog stanja u drugo, odnosno animacija se odvija samo pomoću definiranja početnih i krajnjih vrijednosti nekog CSS svojstva.

Svojstva tranzicija:

- **transition-duration** – vrijeme trajanja izraženo u sekundama
- **transition-timing-function** – tempo prijelaza
- **transition-delay** – odgoda početka tranzicije
- **transition-property** – svojstvo na koje se tranzicija primjenjuje.

Sva ova svojstva možemo navesti i u skraćenom obliku:

```
transition: [property] [duration] [timing-function] [delay];
transition: font-size 2s linear 500ms;
```

Kao vrijednost se zadaje kombinacija vrijednosti kojima se definira animacija za jedno navedeno svojstvo. Prva vremenska oznaka, **duration**, označava trajanje, dok druga označava odgađanje tranzicije. Nije obavezno navođenje svih vrijednosti, ali poželjno je uvijek uz naziv svojstva navesti i duljinu vremenskog trajanja. Tempo prijelaza definira se svojstvom **transition-timing-function**. Podrazumijevana vrijednost **timing-function** jest **ease 0s**.

Vrijednosti svojstva vezana su uz Bezierovu krivulju **cubic-bezier(x1,y1,x2,y2)**. Vrijednosti koje može poprimiti ovo svojstvo nalaze se u tablici ispod.

Vrijednost	Kontrolne točke Bezierove fukcije
`ease`	`cubic-bezier(0.25,0.1,0.25,1)`
`linear`	`cubic-bezier(0, 0,1,1)`
`ease-in`	`cubic-bezier(0.42,0,1,1)`
`ease-out`	`cubic-bezier(0,0,0.58,1)`
`ease-in-out`	`cubic-bezier(0.42,0,0.58,1)`
`step-start`	`steps(1,start)`
`step-end`	`steps(1,end)`
`cubic-bezier(n,n,n,n)`	
`steps(int,start|end)`	

Tablica 3.5. Kontrolne točke Bezierove funkcije

Za realizaciju tranzicije potrebno je odrediti:

- početnu vrijednost
- završnu vrijednost
- duljinu trajanja tranzicije
- okidač.

Za kompatibilnost u svim podržanim preglednicima potrebni su prefiksi:

```
.primjer {
    -webkit-transition-timing-function: ease-in-out;
    -moz-transition-timing-function: ease-in-out;
    -o-transition-timing-function: ease-in-out;
    transition-timing-function: ease-in-out;
}
```

Slika 3.49. Prefiksi za različite web-preglednike

Transformacije omogućuju elementima promjenu oblika, veličine i pozicije. Potrebno je razlikovati 2D i 3D transformacije. Transformacije se deklariraju kao funkcije unutar svojstva **transform**.

Opći oblik deklaracije:

```
selektor{
transform: funkcija();
 }
```

Transformirani element ponaša se kao da je pozicioniran relativno. Element se ne mijenja i zadržava svoj položaj, a mijenja se njegova slika i tako je stvoren privid pomaka.

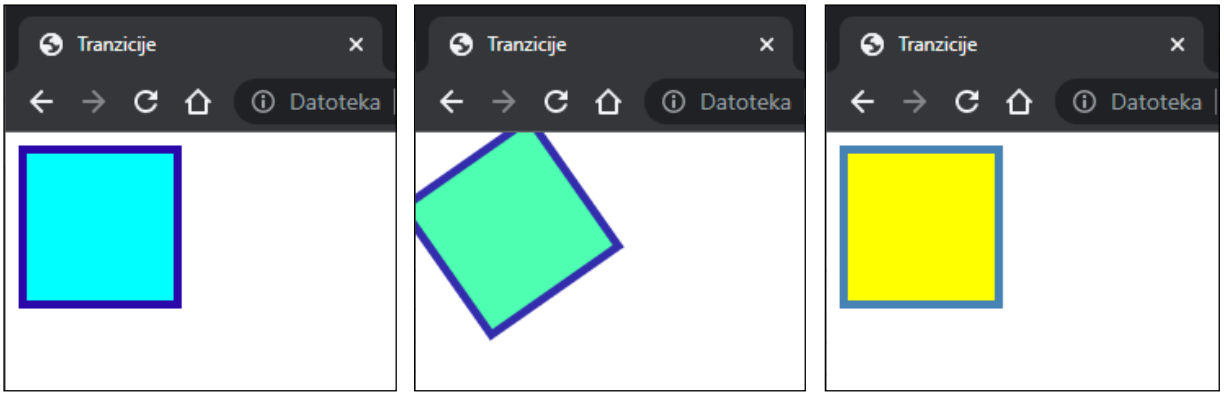
Metode svojstva transform kod 2D transformacija mogu biti:

- **translate()** – pomak elementa s početnog položaja
- **rotate()** – rotacija elementa
- **scale()** – povećava ili smanjuje element
- **skew()** – promjena kuta
- **matrix()** – kombinira sve ranije navedene metode u jednu.

Transform svojstvo se koristi u kombinaciji s tranzicijama i animacijama.

```html
<!DOCTYPE html>
<html>
  <head>
    <title>Tranzicije</title>
    <link rel="stylesheet" href="style.css">
  </head>
  <body>
    <div class="kutija"></div>
  </body>
</html>
```

```css
.kutija {
    background-color: aqua;
    border: 5px solid rgb(43, 8, 170);
    height: 100px;
    width: 100px;
    box-sizing: border-box;
    transition: all 1s ease-in-out, height 0ms;
  }
  .kutija:hover {
    background-color: yellow;
    border-color: steelblue;
    transform: rotate(180deg);
    height: 100px;
  }
```



Slika 3.50. Primjer tranzicije

3D tranzicije dodaju treću dimenziju na zadani element. Element se rotira oko svoje x-osi ili y-osi. Za 3D rotiranje objekta koristi se svojstvo **transform** u kombinaciji s metodama **rotateX()** ili **rotateY()**. Metode **rotateX()** ili **rotateY()** imaju jedan argument, a to je za koliko će se stupnjeva okrenuti oko osi. Nekoliko primjena **transform** metoda nalazi se u tablici ispod.

Izgled u pregledniku	Metode
	`transform: rotate(-25deg);`
	`transform: rotate(-15deg);` `transform-origin: left center;`
	`transform: translate(20px,-20px);`
	`transform: skewX(-45deg);`
	`transform: skale(0.5);`

Tablica 3.6. Metode transform svojstva

- Sve metode za transformacije pogledajte na stranici: https://codepen.io/vineethtrv/pen/XKKEgM

Za razliku od tranzicija, gdje se određuje početna i krajnja vrijednost određenog svojstva, kod svojstva **animation** pomoću ključnog okvira **@keyframe** može se definirati neograničen broj međustanja.

Svojstva koja se koriste prilikom izrade animacija jesu:

- **animation-name** – ime keyframe pravila
- **animation-duration** – trajanje animacije
- **animation-timing-function** – funkcija za određivanje brzine izvršavanja
- **animation-delay** – vrijeme odgode prijelaza
- **animation-direction** – smjer animacije
- **animation-iteration-count** – broj ponavljanja animacije
- **animation-fill-mode** – određuje koje se vrijednosti postavljaju prije ili poslije animacije.

Opći oblik deklaracije:

```
@keyframes {
animation-name {
Keyframe (%){ svojstvo:vrijednost; }}}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <link rel="stylesheet" type="text/css" href="style.css">

    <title>Animacija</title>
</head>
<body>
    <div class="lopta animacija"></div>
</body>
</html>
```

```
.lopta{
width: 100px;
height: 100px;
background-color: #83c5be;
border: 5px dotted #006d77;
border-radius: 80px;
}
```

```
.animacija {
    animation-name: animacija;
    animation-duration: 5s;
    animation-iteration-count:infinite;
    position:absolute;
    }
```

Slika 3.51. Definiranje animacije i objekta za animiranje

```css
@keyframes animacija {
    0% {opacity: 0; background: #e29578; left:0;}
    25% {opacity: 1; background: #83c5be; }
    50% {transform:rotate(0.5turn); left:80%; background: #e29578;}
    75% {opacity: 1; background: #ffddd2;}
    100% {opacity: 0; background: #fff; left:0; transform:rotate(0);}
}
```

Slika 3.52. Definiranje ključnih okvira za animaciju

Slika 3.53. Nekoliko međustanja animacije u pregledniku

U prikazanom primjeru pozadinska boja elementa mijenjat će se postepeno. Umjesto postotaka može se koristiti i ključna riječ **from** za početni kadar i **to** za završni kadar.

Opći oblik deklaracije:

```
@keyframes {
from
{. . .}
to
{. . .}
}
```

- Popis CSS svojstava koja je moguće animirati nalazi se na stranicama: CSS Animations (w3schools.com) ili Animatable CSS properties - CSS: Cascading Style Sheets | MDN (mozilla.org)

3.3. Oblikovanje i pozicioniranje HTML elemenata na web-stranici

Pozicioniranje elemenata na web-stranici jako je zahtjevan i važan zadatak. Poznato je da se HTML elementi redaju jedan iza drugog kako se navedeni u HTML dokumentu. Razlika između blok i linijskih elemenata u prostoru je koji zauzimaju.

Blok-elementi slažu se jedan ispod drugoga i zauzimaju cijelu širinu prikaza u pregledniku ako im nije zadana širina. Linijski elementi redaju se jedan pored drugog u istom redu sve dok ima raspoloživog prostora. Oni zauzimaju onoliko mjesta koliko je potrebno da stane njihov sadržaj. Visina svih elemenata ovisi o količini sadržaja unutar njih.

Primjenom pravila pozicioniranja moguće je razmjestiti HTML elemente na *web*-stranici prema željenom rasporedu.

Elemente je moguće pozicionirati na više načina:

- primjenom modela okvira
- primjenom svojstva **position**
- primjenom svojstva **display**.

3.3.1. Primjena modela okvira (engl. box model) za oblikovanje sadržaja *web*-stranice

HTML elementi prikazuju se u pregledniku u obliku pravokutnika. HTML elementi međusobno se dodiruju, zato je potrebno među njima definirati razmake. To postižemo određivanjem margina nekog elementa. Linijskim elementima postavlja se samo lijeva i desna margina. Vrijednosti koje margina može poprimiti izražene su u mjernim jedinicama mm, cm, px, em ili %.

Margina međusobno odvaja HTML elemente, odnosno definira prostor oko njih. Odmak ili **padding** definira razmak između sadržaja nekog HTML elementa i njegova ruba.

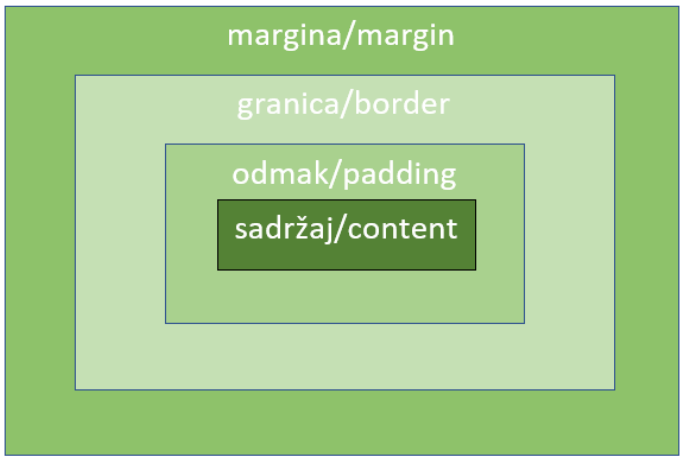
Oba navedena svojstva mogu poprimiti vrijednosti:

- top
- right
- bottom
- left.

Redoslijed navođenja vrijednosti jest: ***top – right – bottom – left***.

```
margin: 2px 10px 15px 5px;
padding: 10px 20px 5px 5px;
```

Slika 3.54. Redoslijed navođenja vrijednosti



Slika 3.55. Model okvira (engl. box model)

Svojstvo **min-width** i svojstvo **max-width** omogućuju definiranje minimalne i maksimalne širine i visine elementa. Minimalna može biti 0, a maksimalna none.

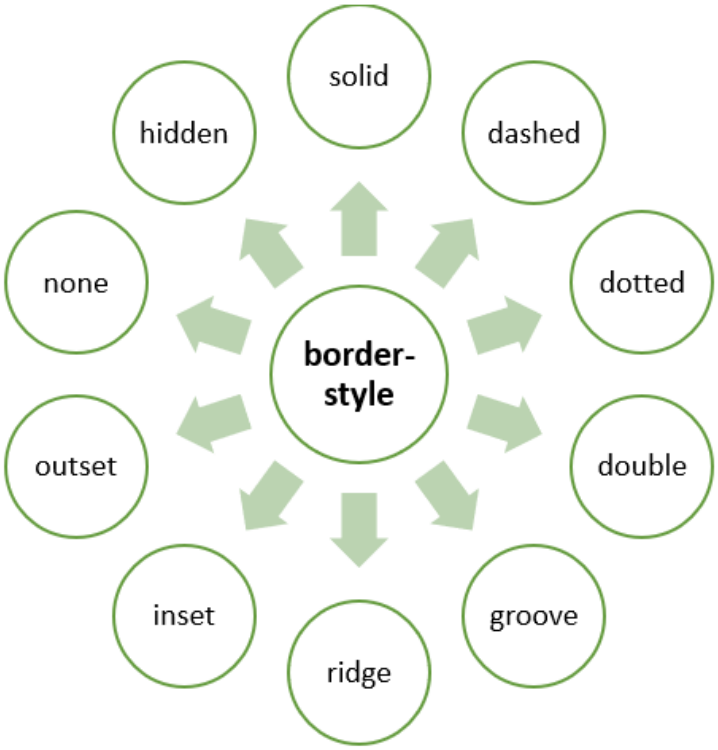
Rubovi HTML elementa (engl. *borders*) mogu se iscrtati HTML elementima. Svojstvo **border** može imati tri argumenta:

- debljina obruba
- stil linije obruba
- boja linije obruba.

Svojstva obruba pomoću kojih je moguće definirati im stilove:

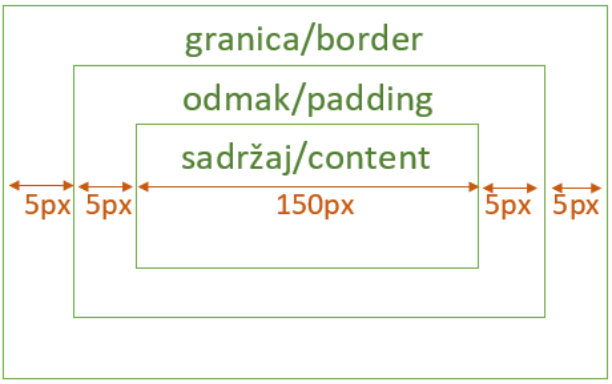
- **border-width** – debljina linije: vrijednosti `thin`, `medium`, `thick` ili debljina izražena u px
- **border-style** – `solid`, `dashed`, `dotted`, `inset`, `outset`, `ridge`, `groove`, `double`, `hiden` i `none`
- **border-color** – boja okvira: vrijednosti naziv boje, rgb ili hex oznaka boje
- **border-radius** – zaobljenost rubova.

Moguće je navesti više vrijednosti za obrub. Ukoliko nije zadana, njezina je vrijednost none.



Slika 3.56. Stilovi linija svojstva **border**

Računanje dimenzija elementa ovisi o dimenzijama svih njegovih komponenti. Primjerice, ako imamo element div čija je širina 170 px s obrubom i padingom od po 5px. Vizualni prikaz takvog elementa vidimo na slici 3.56.



Slika 3.57. Dimenzije elementa (border-box)

Širina elementa = 5px + 5px + 150px + 5px + 5px = 170px.

Širina elementa sa svim njegovim sastavnim elementima je 170px. Prostor za sam sadržaj širine je 150 px jer se ukupna širina umanjuje za širinu odmaka (2 x 5 px) i širinu granice (2 x 5 px). Isti princip primjenjuje se i na visinu elementa.

Margine nisu uključene u dimenziju elementa jer one su omotač oko elementa i odvajaju ga od drugih HTML elemenata.

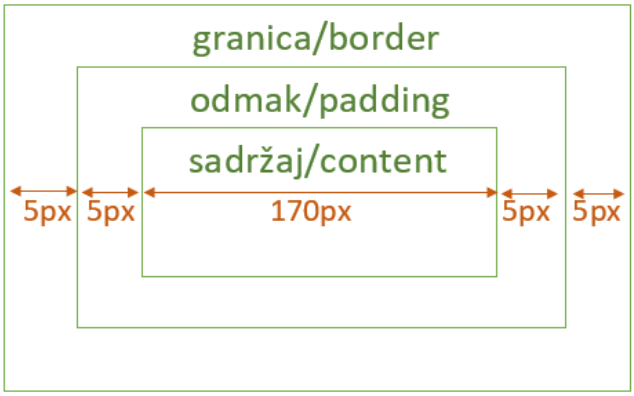
Upotrebom svojstva **box-sizing** može se promijeniti način računanja dimenzija elementa.

Svojstvo box-sizing ima sljedeće vrijednosti:

- **content-box** – visina i širina odnose se samo na sadržaj
- **border-box** – visina i širina odnose se na širinu sadržaja + padding + border.

Za naredbu **box-sizing: content-box;** širina samog sadržaja će biti 170 px i na tu dimenziju dodat će se ostali elementi.

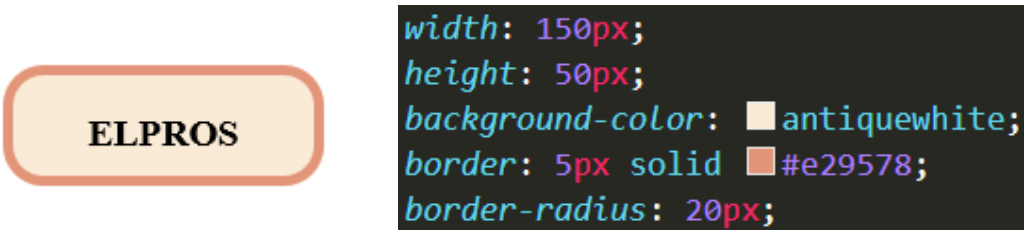
Širina elementa = 5px + 5px + 170px + 5px + 5px = 190px.



Slika 3.58. Dimenzije elementa (content-box)

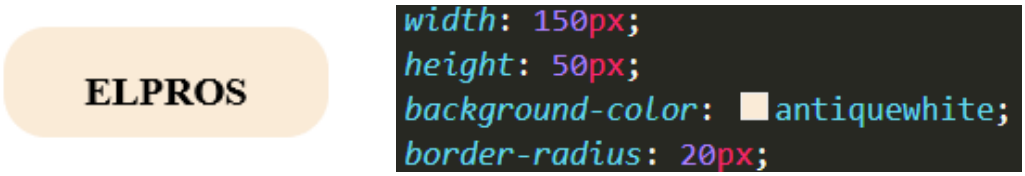
Preporuka je koristiti **box-sizing: border-box**; jer je lakši rad s elementima koji imaju ta svojstva. Mali je nedostatak nepodržanost u starijim verzijama web-preglednika. Dosta preglednika već koristi **box-sizing: border-box;** za neke elemente obrasca.

Svojstvo **border-radius** koristi se za definiranje zaobljenih rubova HTML elementa. Početna vrijednost ovog svojstva je 0. Vrijednost se zadaje mjernim jedinicama px, em, rem, pt ili postocima.



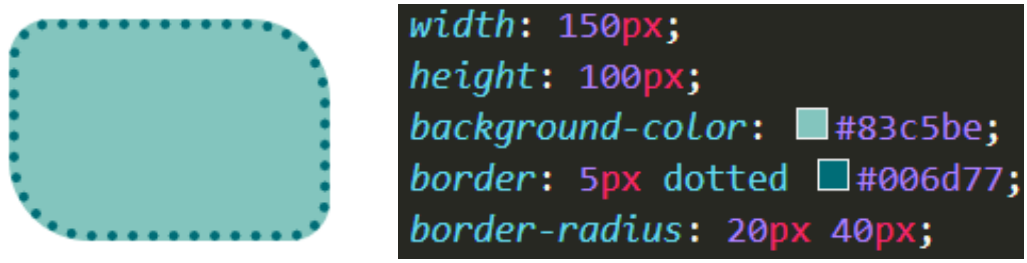
Slika 3.59. Primjena svojstva border-radius

Kod postavljanja svojstva **border-radius** treba paziti jer bez obzira na to je li definirano svojstvo **border**, na pozadinu elementa djelovat će svojstvo **border-radius**.



Slika 3.60. Djelovanje svojstva border-radius bez svojstva border

Moguće je definirati i različite radijuse zaobljenja koji će se primijeniti na neki element. Isto tako različita border svojstva moguće je postaviti i na sliku.



Slika 3.61. Definiranje različitih načina zaobljenja

```
width: 150px;
height: 100px;
background-color: #83c5be;
border-radius: 20px 40px;
```

Slika 3.62. Postavljanje različitih border svojstava na sliku

3.3.2. Oblikovanje prikaza elemenata na *web*-stranici primjenom svojstva display

Svojstvo display igra važnu ulogu u kontroliranju i raspoređivanju sadržaja na *web*-stranici. HTML elementi imaju predefinirane vrijednosti za svojstvo display:

- blok-elementi – protežu se cijelom širinom zaslona
- inline – zauzimaju širinu svog sadržaja
- none.

Predefinirana svojstva HTML elemenata može se promijeniti svojstvom display.

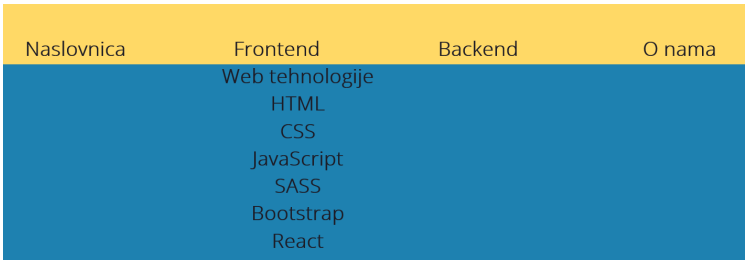
Vrijednosti svojstva mogu biti:

- **display:block** – element s ovim svojstvom postaje blok-element
- **display-inline** – element s ovim svojstvom postaje linijski
- **display:list-item** – element se ponaša kao numerirana ili nenumerirana lista, svaku stavku stavlja u novi element
- **display:none** – skrivanje elementa.

```
<nav>
<ul>
    <li><a href="#">Naslovnica</a></li>
    <li><a href="#">Frontend</a>
        <ul>
          <li><a href="web.html">Web tehnologije</a></li>
          <li><a href="html.html">HTML</a></li>
          <li><a href="css.html">CSS</a></li>
          <li><a href="javascript.html">JavaScript</a></li>
          <li><a href="sass.html">SASS</a></li>
          <li><a href="bootstrap.html">Bootstrap</a></li>
          <li><a href="react.html">React</a></li>
        </ul>
    <li><a href="#">Backend</a></li>
    <li><a href="#">O nama</a></li>
</ul>
</nav>
```

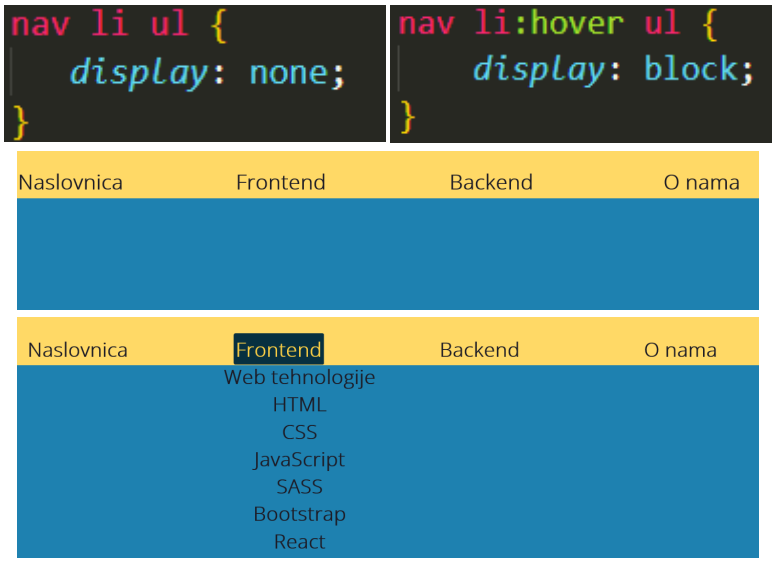
Slika 3.63. Izbornik s podizbornikom prije i nakon stiliziranja

Nakon primjene svojstva **display:block** izbornik postaje blok-element i zauzima vodoravni položaj.



Slika 3.64. Izgled izbornika nakon primjene svojstva display:block

Još je jedan problem kod izbornika na slici 64 a to što je sadržaj podizbornika stalno vidljiv, što nije poželjno. Za skrivanje podizbornika koristi se svojstvo **display:none.**

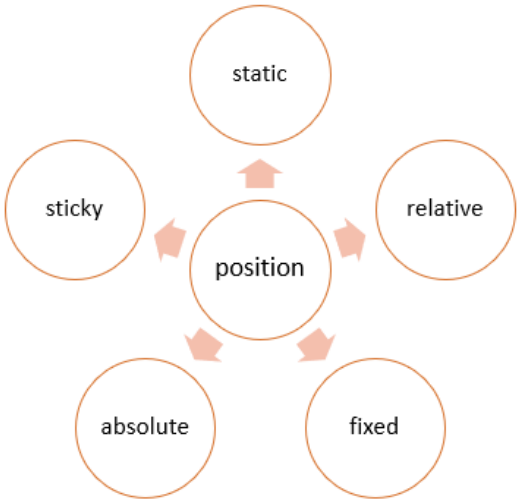


Slika 3.65. Izgled izbornika nakon upotrebe svojstva display

Skrivene stavke pokazat će se tek nakon prelaska mišem preko gumba Frontend.

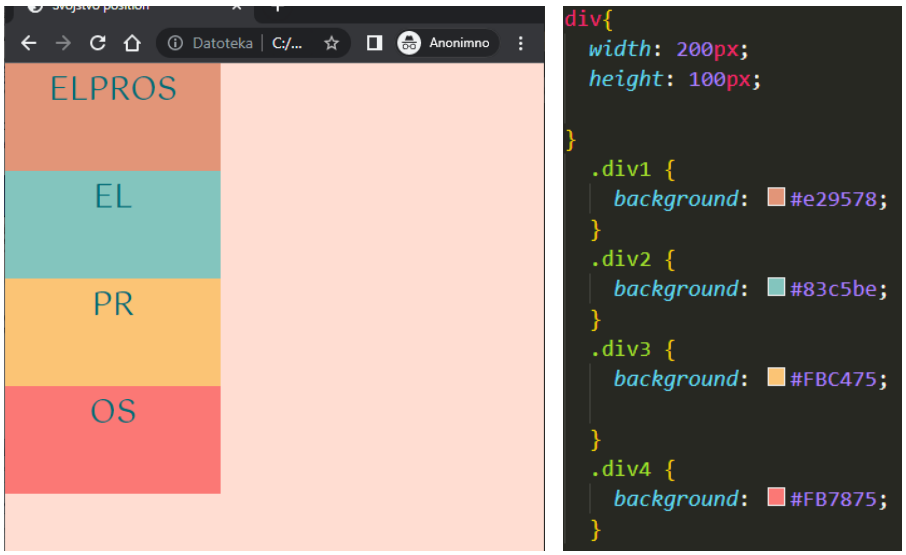
3.3.3. Pozicioniranje HTML elemenata na *web*-stranici primjenom svojstva position

Jedno od ključnih svojstava za pozicioniranje sadržaja na *web*-stranici jest i svojstvo **position**. HTML elementi u pregledniku se redaju redom kojim su navedeni. Za promjenu rasporeda i drugačije pozicioniranje elemenata koristi se svojstvo **position**. Ponekad dolazi do preklapanja elemenata prilikom pozicioniranja. Koji element ide u prvi plan definira se upotrebom z-indeksa. Z-indeks je pozitivan ili negativan cijeli broj. Definira se prema tome koji element dolazi preko drugog elementa. Elementu koji se postavlja u prvi plan postavlja se dovoljno velik z-indeks. Element s najmanjim z-indeksom bit će u pozadini. Podrazumijevana vrijednost z-indeksa je 0.

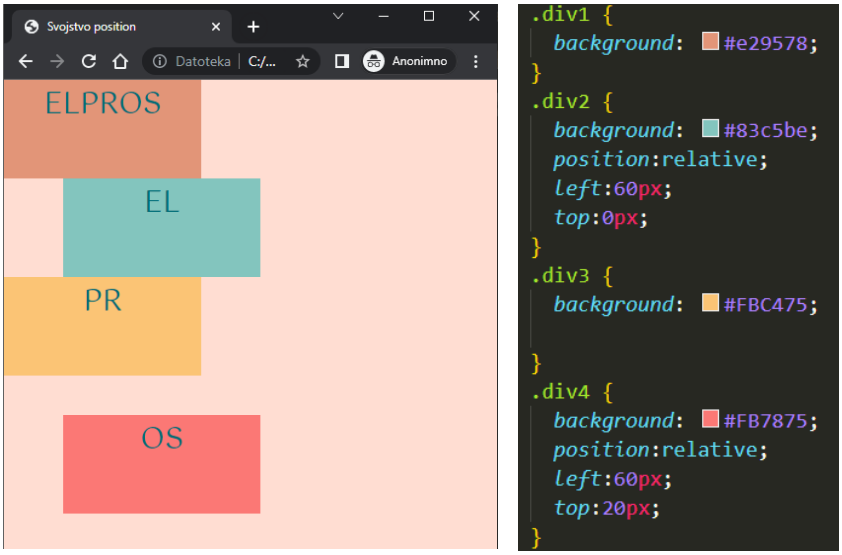


Slika 3.66. Vrijednosti svojstva position

Svojstvo **position:static** rijetko se definira jer su svi HTML elementi već automatski definirani kao **static**. Iako se navede svojstvo static, neće biti promjene pozicije elemenata jer na njih ne utječu svojstva **top, bottom, left** i **right**. Koristi se samo kad želimo isključiti druge vrijednosti ovog svojstva.



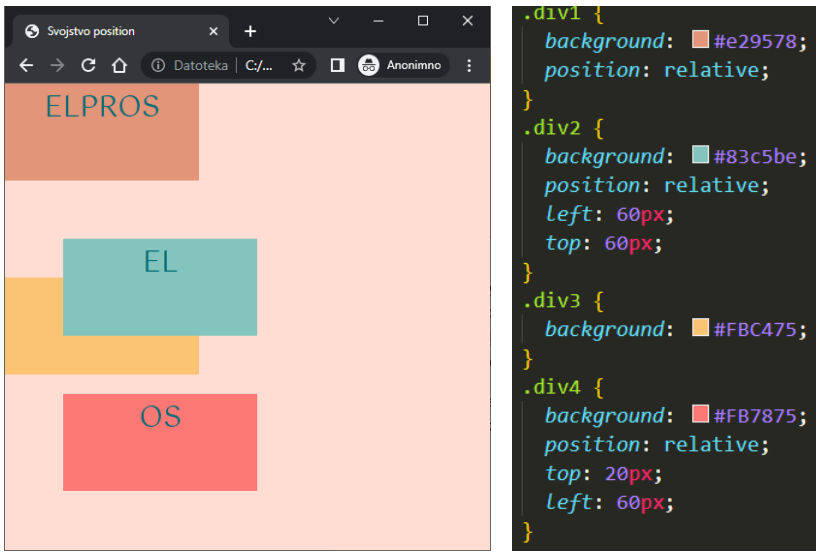
Slika 3.67. Svojstvo **position:static**



Slika 3.68. Svojstvo **position:relative**

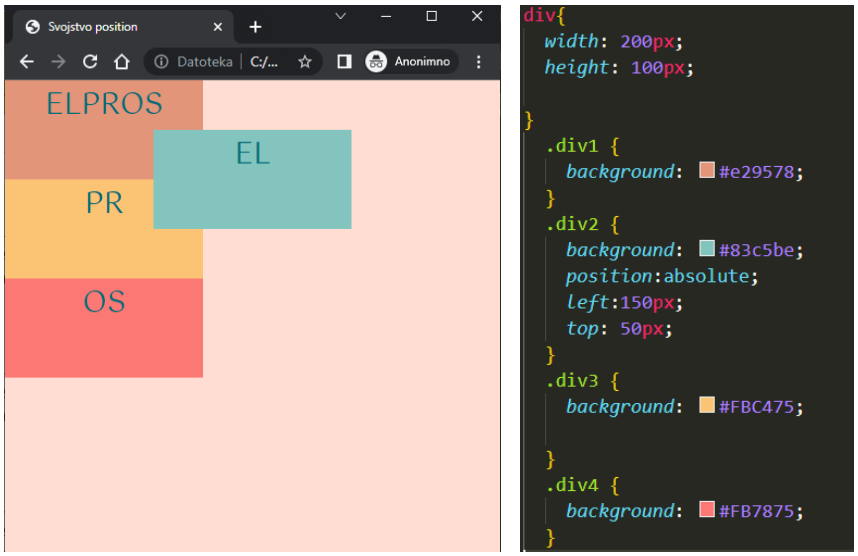
Svojstvo **position:relative** pozicionira element za zadane vrijednosti u odnosu na njegov početni položaj. Na slici drugi element (EL) pomaknuo se sa svoje pozicije za 60 px udesno. Prema dolje nije bilo pomaka jer je svojstvu top dodijeljena vrijednost 0. Četvrti element (OS) imao je pomak za 60 px udesno i 20 px prema dolje sa svoje početne pozicije.

Ako elementu EL dva dodamo pomak i prema dolje za 60 px, on će se pomaknuti iz svoje početne pozicije za 60 px udesno i 60 px prema dolje i prijeći preko elementa koji se nalazi iza njega, odnosno elementa PR.



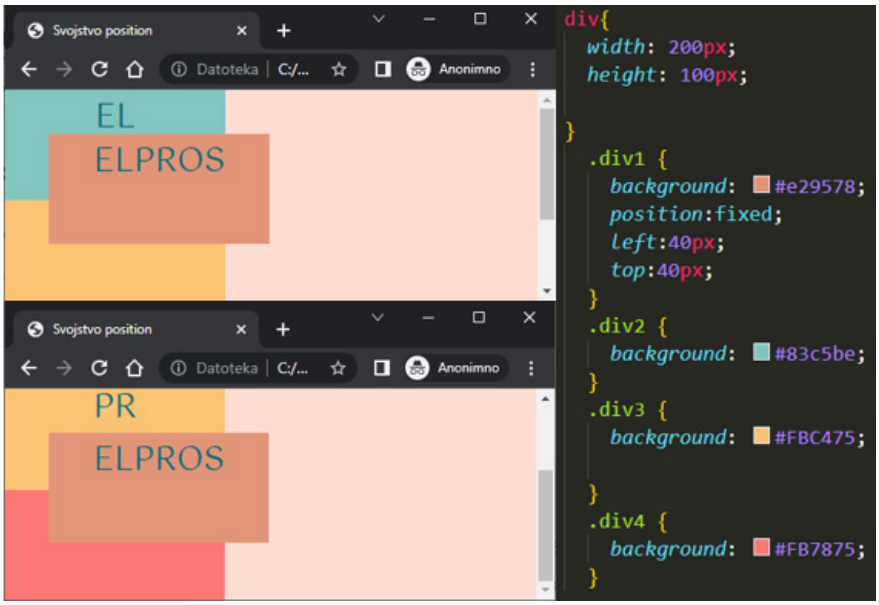
Slika 3.69. Preklapanje elemenata

Vidljivo je da nije došlo do automatskog pomaka trećeg elementa kako bi ustupio mjesto drugom elementu, nego je došlo do preklapanja elemenata.



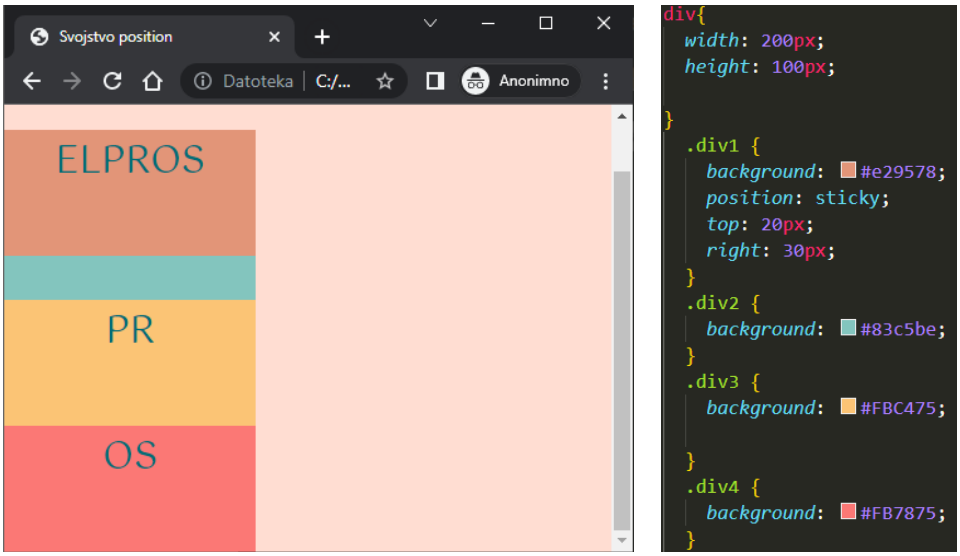
Slika 3.70. Svojstvo **position:absolute**

Svojstvo **position:absolute** pozicionira element u odnosu na roditeljski element; ako ne postoji roditeljski element, veže se na <body> element. Element (EL) koristi tijelo dokumenta i kod skrolanja se miče uzduž stranice.



Slika 3.71. Svojstvo **position:fixed**

Element ELPROS ostaje uvijek na istoj poziciji bez obzira na pomicanje, odnosno skrolanje stranice jer je pozicioniran relativno prema vidljivom području (engl. *view-port*).



Slika 3.72. Svojstvo **position:sticky**

Svojstvo **position:sticky** kombinacija je relativnog i fiksnog. Pomoću njega postavljamo odabrane elemente na stranici tako da uvijek budu vidljivi. Na slici je vidljivo da je element ELPROS ostao na svojoj poziciji, dok su ostali elementi išli ispod njega prilikom skrolanja.

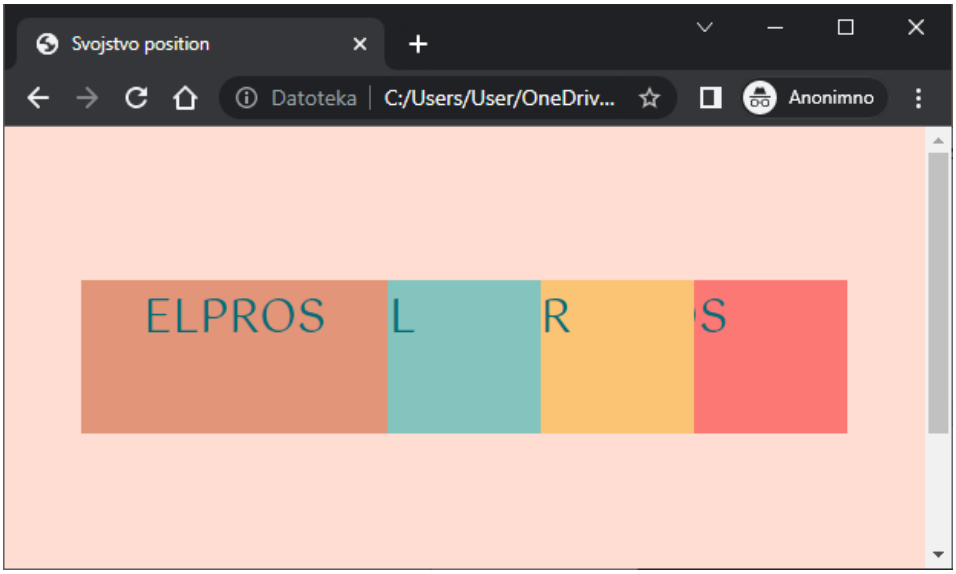
Svojstvo **position** omogućuje i pozicioniranje elemenata jedan iznad drugoga, kao što je vidljivo u primjeru ranije (Slika 3.68.). Za redoslijed slaganja koristi se svojstvo z-indeks, koje određuje koji element ide u prednji plan, a koji u pozadinu.

Želimo li odrediti koji će element biti iznad ili ispod, koristimo se svojstvom z-indeks. Onaj element koji ima manji z-indeks bit će pozicioniran ispod. Element s najvećim z-indexom dolazi u prvi plan. Zbog toga, ako se želi određeni element postaviti u prvi plan, vrijednost njegova z-indexa postavlja se na dovoljno velik broj kako bi se to osiguralo. Vrijednosti koje se dodjeljuju z-indeksu proizvoljne su.

```css
.div1 {
  background: #e29578;
  position: relative;
  top: 100px;
  left: 50px;
  z-index: 999;
}
.div2 {
  background: #83c5be;
  position: relative;
  left: 150px;
  z-index: 5;
}
```

```css
.div3 {
  background: #FBC475;
  position: relative;
  top: -100px;
  left: 250px;
}
.div4 {
  background: #FB7875;
  position: relative;
  top: -200px;
  left: 350px;
  z-index: -5;
}
```

Slika 3.73. Djelovanje z-indeksa



Slika 3.74. Izgled elemenata u pregledniku nakon djelovanja z-indeksa

3.4. Raspoređivanje elemenata web-stranice primjenom pravila liste stilova

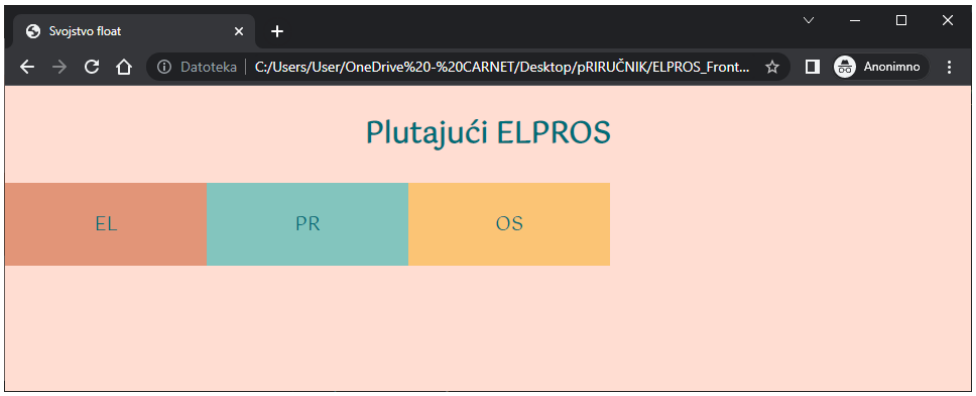
3.4.1. Raspoređivanje elemenata web-stranice primjenom svojstva float

Svojstvo **float** jedno je od osnovnih svojstava za kreiranje rasporeda na web-stranici. Ovim svojstvom ne određuje se određena pozicija HTML elementa, nego im se dopušta da sami odrede svoj položaj. Vrijednosti koje ovo svojstvo može poprimiti su **left, right, none** i **inherit**. Ovisno o tim vrijednostima element koji nije pozicioniran plutat će u tom smjeru. Elementi na kojima se primjenjuje svojstvo **float** ponašaju se kao blok-elementi, odnosno kao da im je dodijeljeno svojstvo **display:block.**

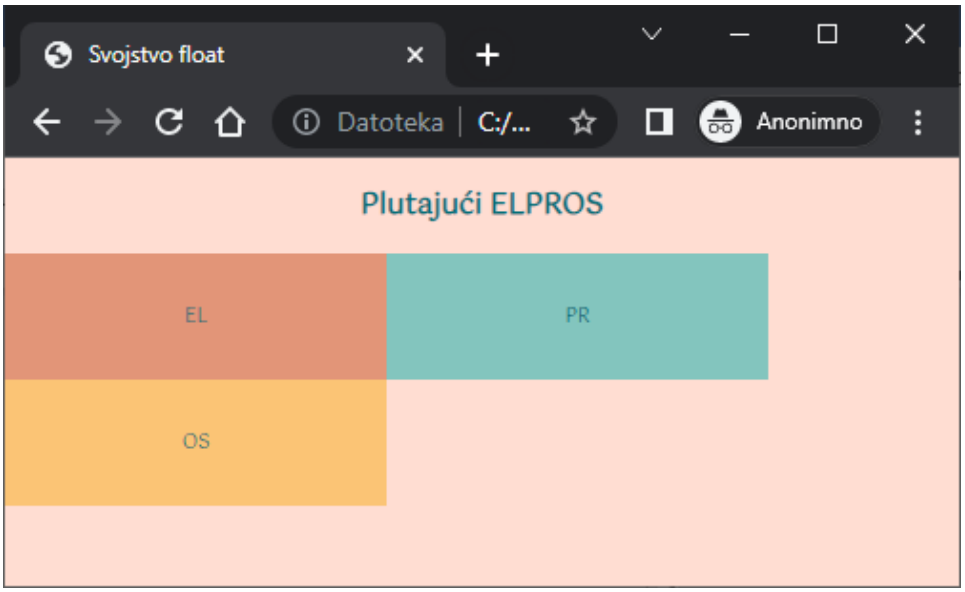
Kad definiramo svojstvo **float: left** nekom elementu, on će plutati od svog položaja prema lijevom dijelu svog spremnika sve dok ne nađe svoje mjesto. Svi ostali slagat će se jedan pored drugog dok ima raspoloživog prostora. Kada ne nađe mjesto pored, plutajući se spušta prema dolje i opet pluta na lijevu stranu spremnika. Element sa svojstvom **float:right** po istom principu pluta prema desnoj strani spremnika. Ako je svojstvo **float** s vrijednosti none odijeljeno nekom elementu, taj element ne pluta. Smješta se na svoju podrazumijevanu poziciju. Vrijednost **inherit** znači da element nasljeđuje svojstva pozicioniranja od roditeljskog elementa. Svojstvo float može poslužiti i za omatanje teksta oko slike na web-stranici. Sadržaje u HTML-u često smještamo unutar elemenata. Po podrazumijevanim svojstvima oni se smještaju jedan iznad drugog, a upotrebom svojstva float može se definirati da plutaju usporedno.

```
div {
    float: left;
    padding: 25px;
  }

  .div1 {
      width: 150px;
    background: #e29578;
  }
```

```
  .div2 {
      width: 150px;
    background: #83c5be;
  }

  .div3 {
    width: 150px;;
    background: #FBC475;
  }
```



Slika 3.75. Djelovanje svojstva **float:left**



Slika 3.76. Prelazak u novi raspoloživi dio spremnika

Ukoliko ne želimo da se neki elementi smještaju uz prethodne, dodijelit ćemo im svojstvo **clear**. Ovo svojstvo određuje koji elementi mogu plutati uz obrisani element. Clear se najčešće koristi nakon primjene svojstva **float**. Nakon brisanja svojstva **float** s vrijednosti **left** potrebno je na lijevoj strani primijeniti i svojstvo **clear**. Obrisani elementi na stranici se smještaju ispod plutajućih elemenata. Vrijednosti svojstva **clear** određuju stranu plutanja.

Vrijednosti svojstva clear jesu:

- **none** – plutajući elementi na obje strane spremnika
- **left** – na lijevoj strani nisu dopušteni plutajući elementi

- **right** – na desnoj strani nisu dopušteni plutajući elementi
- **both** – plutajući elementi nisu dozvoljeni ni s jedne strane spremnika
- **inherit** – vrijednost se nasljeđuje od roditeljskog elementa.

Svojstvo **flat** često se koristi za međusobni položaj teksta i slike. Svojstvom **float** slika je plutajući element koja se smjestila uz lijevi rub spremnika. Svojstvo **padding** definiralo je odmak između teksta i slike.

```
img{
    float: left;
    margin: 0 15px;
    width: 250px;
    border: #006d77 solid 1px;
}
```

Slika 3.77. Djelovanje svojstva float na položaj teksta i slike

3.4.2. Raspoređivanje elemenata web-stranice primjenom svojstva flexbox

Svojstvo **flexbox** (engl. *flexible box layout module*) omogućuje fleksibilan raspored elemenata na *web*-stranici i na taj način rješava pozicioniranje elemenata pri različitim prikazima. Elementi smješteni unutar spremnika mogu mijenjati veličinu i položaj unutar raspoloživog prostora spremnika.

Flex layout raspoređuje elemente jednodimenzionalno, ili u retke ili u stupce, definiranjem fleksibilnoga spremnika (roditeljski element) dodjeljujući svojstvu **display** vrijednost **flex**. Elementi unutar fleksibilnog spremnika postaju fleksibilni.

Raspored flexbox ima dva tipa elemenata:

- **flex container** – fleksibilni spremnik koji sadrži jedan ili više elemenata
- **flex items** – fleksibilni elementi unutar fleksibilnoga spremnika.

Elementi unutar spremnika mogu biti smješteni ili vodoravno ili okomito.

Uz pomoć flexboxa moguće je:

- automatsko skaliranje elemenata, odnosno promijeniti im visinu ili širinu tako da popune raspoloživi prostor spremnika
- automatski smanjiti ili povećati elemente kako bi se uklopili u spremnik i spriječili prelijevanje
- promijeniti redoslijed rasporeda elemenata
- lakše rukovanje vodoravnim ili okomitim poravnanjem.

Fleksibilni elementi osim svojih svojstava mogu naslijediti svojstva od roditelja, odnos-

Svojstva koja se primjenjuju na fleksibilni spremnik (engl. *flex container*):

- `align-content`
- `align-items`
- `display`
- `flex-direction`
- `flex-flow`
- `flex-wrap`
- `justify-content.`

Svojstva koja se primjenjuju na fleksibilne elemente (engl. *flex items*)

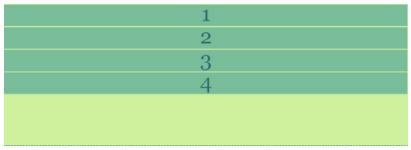
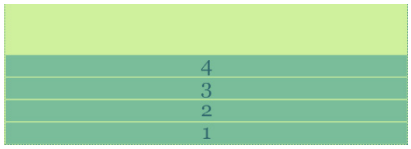
- `align-self`
- `flex`
- `flex-basis`
- `flex-grow`
- `flex-shrink`
- `order.`

no fleksibilnog spremnika.

Kako bi neki spremnik postao fleksibilni postavlja se:

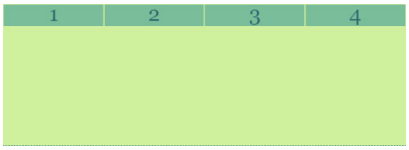
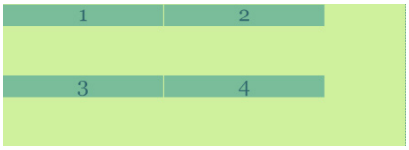
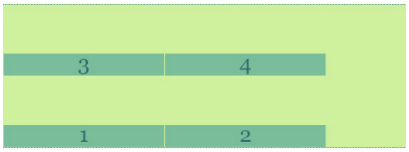
- **za blok-spremnik:** `display: flex;`
- **inline-spremnik:** `display: inline-flex.`

Smjer slaganje fleksibilnih elemenata unutar fleksibilnog spremnika definira se svojstvom **flex-direction**.

Prikaz slaganja elemenata	Vrijednosti svojstva flex-direction
	`flex-direction: row;`
	`flex-direction: row-reverse;`
	`flex-direction: column;`
	`flex-direction: column-reverse;`

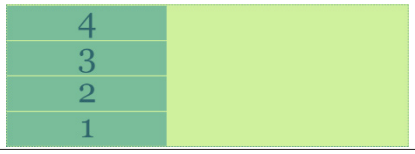
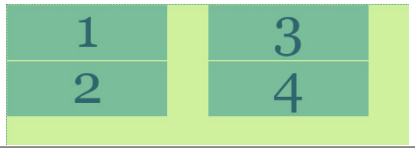
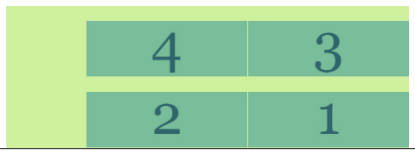
Tablica 3.7. Svojstvo **flex-direction**

Svojstvo **flex-wrap** omogućuje slaganje elemenata u više redaka ili stupaca. Kada se popuni prvi redak odnosno stupac, sadržaj se slaže u sljedeći.

Prikaz slaganja elemenata	Vrijednosti svojstva flex-wrap
	`flex-wrap: nowrap;`
	`flex-wrap: wrap;`
	`flex-wrap: wrap-reverse;`

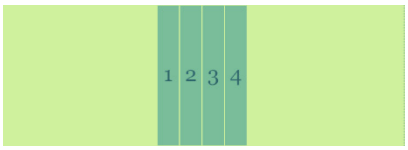
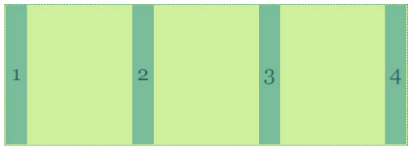
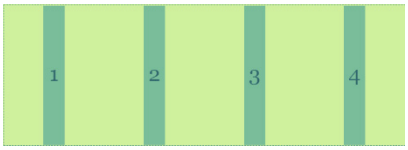
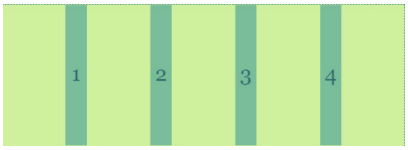
Tablica 3.8. Svojstvo **flex-wrap**

Svojstvo **flex-flow** jest skraćivanje zapisivanja svojstava **flex-wrap** i **flex-direction**.

Prikaz slaganja elemenata	Vrijednosti svojstva flex-flow
	`flex-flow: row nowrap;`
	`flex-flow: column-reverse;`
	`flex-flow: column wrap;`
	`flex-flow: row-reverse wrap-reverse;`

Tablica 3.8. Svojstvo **flex-flow**

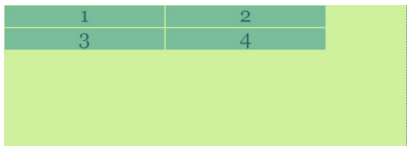
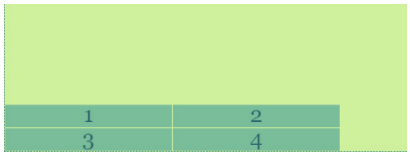
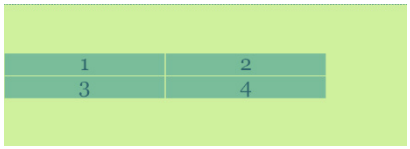
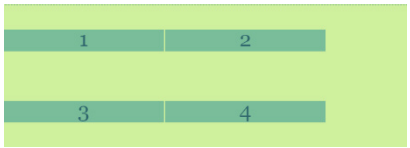
Svojstvo **justify-content** koristi se za vodoravno poravnanje fleksibilnih elemenata unutar fleksibilnoga spremnika.

Prikaz slaganja elemenata	Vrijednosti svojstva justify-content
	`justify-content: flex-start;`
	`justify-content: flex-end;`
	`justify-content: center;`
	`justify-content: space-between;`
	`justify-content: space-around;`
	`justify-content: space-evenly;`

Tablica 3.9. Svojstvo justify-content

Svojstvom **align-items** definiraju se poravnanja fleksibilnih elemenata prema zadanom smjeru slaganja, ili vodoravno (*row*) ili okomito (*column*). Kod vodoravnog slaganja koristi se okomito poravnanje elementa. Po istom principu poravnavaju se elementi kod okomitog slaganja. Svojstvo **align-content** definira okomito poravnanje

elemenata fleksibilnoga spremnika. U tablici prikazano je djelovanje svojstva **align-items** uz svojstvo **flex-wrap: wrap.**

Prikaz slaganja elemenata	Vrijednosti svojstva align-content
	`align-content: flex-start;`
	`align-content: flex-end;`
	`align-content: center;`
	`align-content: space-between;`
	`align-content: space-around;`
	`align-content: stretch;`

Tablica 3.10. Svojstvo align-content

Definirajući fleksibilan raspored elementi se prilagođavaju mijenjajući svoju širinu odnosno visinu kako bi ispunili raspoloživi prostor unutar fleksibilnog spremnika. Fleksibilni spremnik raspodjeljuje slobodan prostor svojim elementima kako bi se spremnik popunio ili skuplja elemente kako bi spriječio prelijevanje.

Element unutar fleksibilnog kontejnera nefleksibilan je ako su njegove vrijednosti svojstava **flex-grow** i **flex-shrink** jednake nuli, dok je u suprotnom fleksibilan.

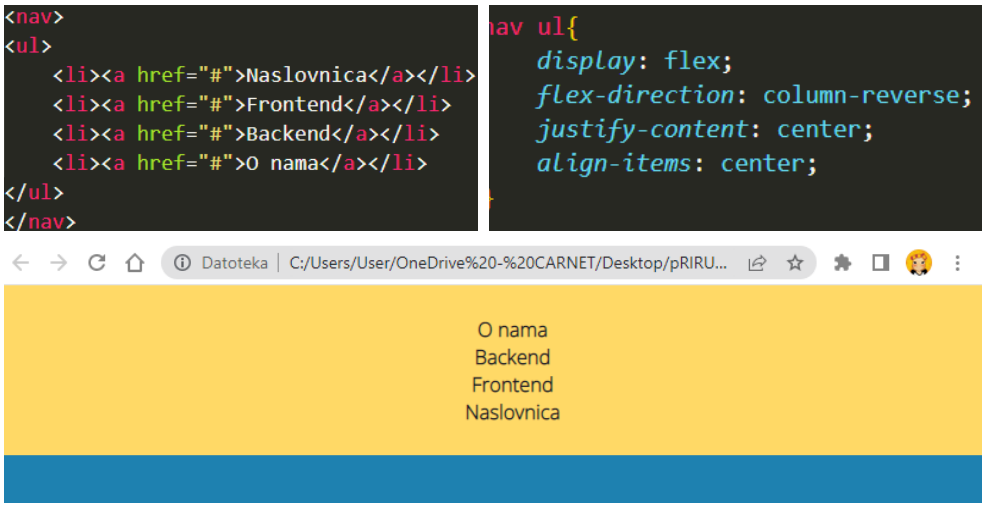
Prikaz slaganja elemenata	Vrijednosti svojstva flex-grow i flex-shrink
	`flex-grow: 0;`
	`flex-grow: 1;`
	`flex-shrink: 0;`
	`flex-shrink: 1;`

Tablica 3.11. Svojstvo flex-shrink

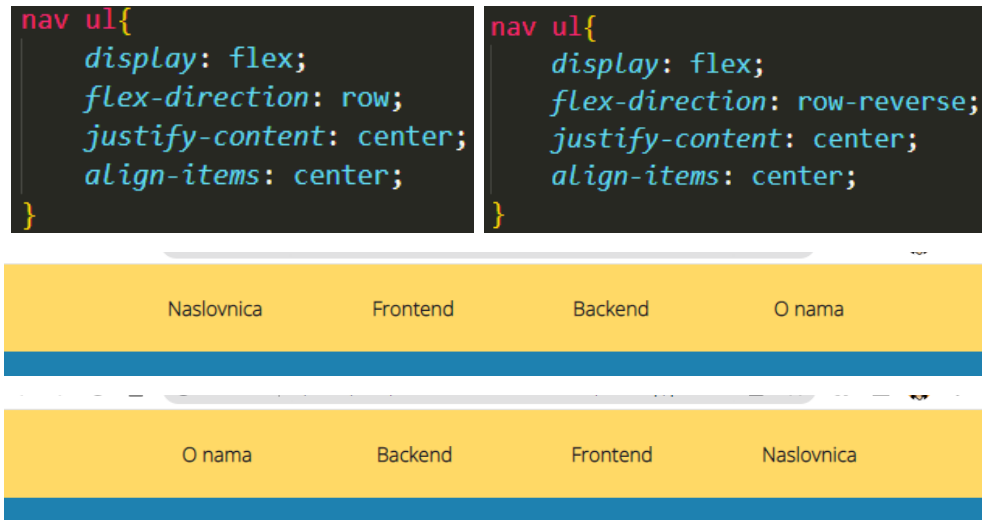
Primjer:

Na primjeru navigacije demonstrirat ćemo neko od svojstava fleksibilnog spremnika. Navigacijska lista predstavlja fleksibilni spremnik jer smo joj dodijelili svojstvo **display:flex**. Lista predstavlja fleksibilni spremnik, a članovi liste (*list item*) predstavljaju njene fleksibilne elemente.

Slika 3.78. Okomit smjer slaganja elemenata



Slika 3.79. Obrnut okomit smjer slaganja elemenata



Slika 3.80. Vodoravan i obrnut vodoravan smjer slaganja elemenata

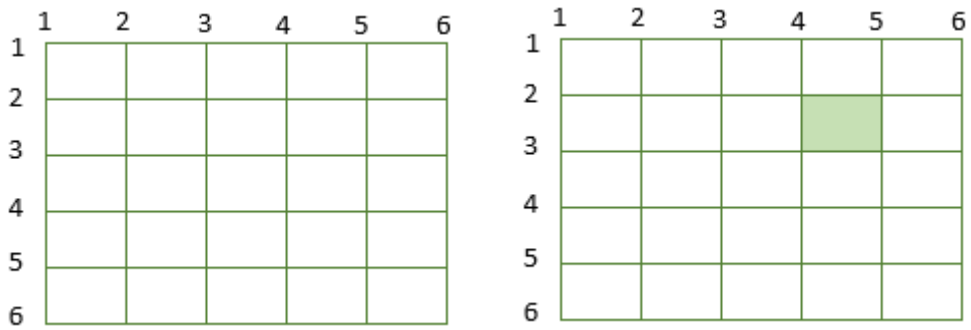
3.4.3.. Raspoređivanje elemenata *web*-stranice primjenom svojstva grid

Za dvodimenzionalno raspoređivanje elemenata na *web*-stranici koristimo CSS grid. Sadržaj stranice pomoću grid mreže raspoređuje se u retke i stupce. Ako se zadaju određene dimenzije za visinu i širinu redaka, onda će HTML elementi, bez obzira na sadržaj, biti poslagani unutar njih.

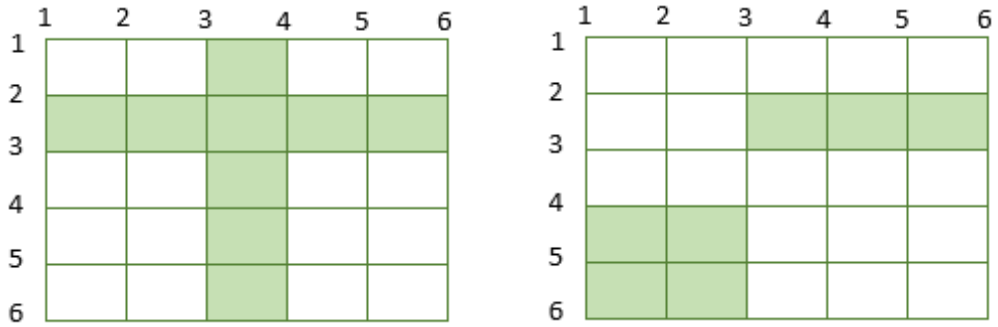
Kako bi se omogućio grid raspored svojstvu **display** dodjeljuje se vrijednost **grid.** Odnosno svojstvo display definira spremnik unutar kojeg se stvara grid. Spremnik može biti bilo koji HTML element. Grid svojstvo omogućuje fleksibilno pozicioniranje elemenata na *web*-stranici prema željenom rasporedu. Broj stupaca u **grid-view** načinu oblikovanja jest 12 i zauzimaju 100 % širine zaslona. Ovako definirane mreže omogućuju bolju kontrolu. Prilikom kreiranja grid mreže potrebno je svim HTML elementima dodijeliti svojstvo **box-sizing:border-box** koristeći univerzalni selektor.

Osnovni elementi grid spremnika jesu:

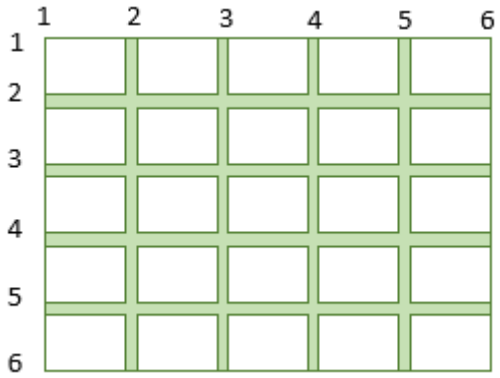
- **grid-item** – element unutar grid spremnika
- **grid-line** – linije koje razdvajaju retke i stupce grida
- **grid-cell** – ćelija unutar grid spremnika
- **grid-track** – prostor između linija grida, cijeli redak ili stupac
- **grid-area** – više ćelija grida pravokutnog oblika, zauzima jednu ili više njih
- **grid-gap** – prostor između **grid-track** elemenata.



Slika 3.81. Svojstva **grid-line** i **grid-cell**



Slika 3.82. Svojstva **grid-track** i **grid-area**

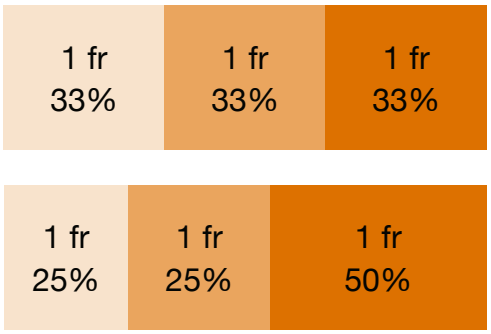


Slika 3.83. Svojstva **grid-gap**

Za definiranje grid stupaca ili redaka koriste se različite mjerne jedinice:

- auto – automatsko popunjavanje
- px, vh, vw, em, %
- frakcija (fr).

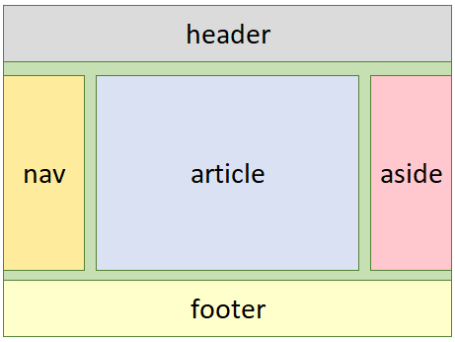
Frakcija je mjerna jedinica koja predstavlja dio raspoloživog prostora u retku ili stupcu. Fleksibilna je i omogućuje zadržavanje rasporeda elemenata na stranici bez obzira na promjenu veličine prozora preglednika.



Slika 3.84. Zadavanje širine stupaca pomoću frakcije

Grid možemo kreirati stvaranjem stupaca na različite načine:

- **grid-template-columns:** 1fr 1fr; – 3 stupca od kojih svaki zauzima 1/3 raspoložive širine
- **repeat** (3, 1fr) – isto značenje kao prethodna naredba
- **grid-template-columns:** 33% 33% 33% – 3 stupca širine 33 % ukupne širine grid rasporeda.



Slika 3.85. Primjer grid rasporeda

Kreirano je devet div elemenata i raspoređeno u mrežu 3x3. Stupci su širine jedne frakcije, a redci na fiksnu visinu od 100 px.

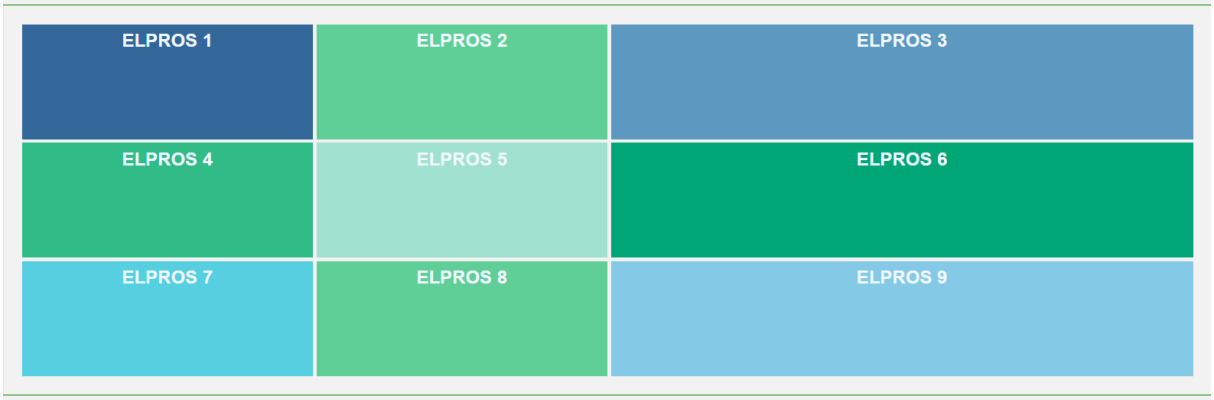
```
<div class="grid">
  <div class="elpros1">ELPROS 1</div>
  <div class="elpros2">ELPROS 2</div>
  <div class="elpros3">ELPROS 3</div>
  <div class="elpros4">ELPROS 4</div>
  <div class="elpros5">ELPROS 5</div>
  <div class="elpros6">ELPROS 6</div>
  <div class="elpros7">ELPROS 7</div>
  <div class="elpros8">ELPROS 8</div>
  <div class="elpros9">ELPROS 9</div>
</div>
```

```
.grid{
  margin: 1em;
  padding: 1em;
  border: 1px solid green;
  display: grid;
  grid-template-columns: 1fr 1fr 1fr;
  grid-template-rows: 100px 100px 100px;
  grid-gap: 3px;
}
```

Slika 3.86. Grid mreža 3x3

Grid mreža (engl. *grid layout*) omogućuje raspored elemenata u retke i stupce. Vidljivo je da se nakon definiranja širine redaka i visine stupaca elementi raspoređuju prema kreiranoj mreži. Svojstvo **grid-gap** definira razmak između elemenata.

```
grid-template-columns: 1fr 1fr 2fr;
```



Slika 3.87. Grid s različitim omjerom stupaca

Širina stupaca definira se pomoću frakcija u omjeru 1fr : 1fr : 2fr. Prva dva stupca zauzimaju po 25 % širine, a treći stupac zauzima 50 %.

Kreiranje drugačije mreže s istim brojem elemenata dat će drugačiji raspored. Imamo devet elemenata i mrežu od četiri stupca i tri retka. Elementi će se redom smjestiti jedan iza drugog, a preostali dio grida ostat će prazan.

```
grid-template-columns: repeat(4, 1fr);
grid-template-rows: 100px 100px 100px;
grid-gap: 3px;
```

Slika 3.88. Smještanje elemenata unutar kreirane grid mreže 4x3

```
grid-template-columns: repeat(2, 1fr);
grid-template-rows: 100px 100px 100px;
grid-gap: 3px;
```

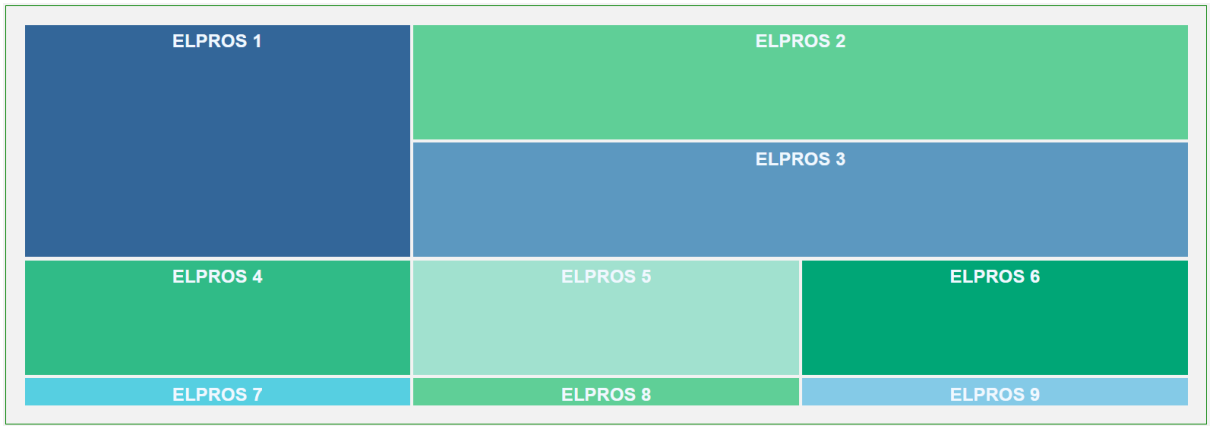
Slika 3.89. Smještanje elemenata unutar kreirane grid mreže 2x3

Ukoliko želimo rasporediti elemente unutar grida pomoću linija grida, treba paziti da 3 stupca ili retka određuju 4 linija grida.

```
.elpros1 {
  background-color: #336699;
  grid-row: 1/3;
}
```

```
.elpros2{
  background-color: #5FCF97;
  grid-column: 2/4;
}
```

```
.elpros3 {
  background-color: #5C98C0;
  grid-column: 2/4;
}
```



Slika 3.90. Kreiranje rasporeda pomoću linije grida

Prva tri elementa raspoređena su upotrebom linije grida:

```
grid-row: 1/3
```

Element 1 počinje smještanje od prve linije retka i završava s linijom 3.

```
grid-column: 2/4
```

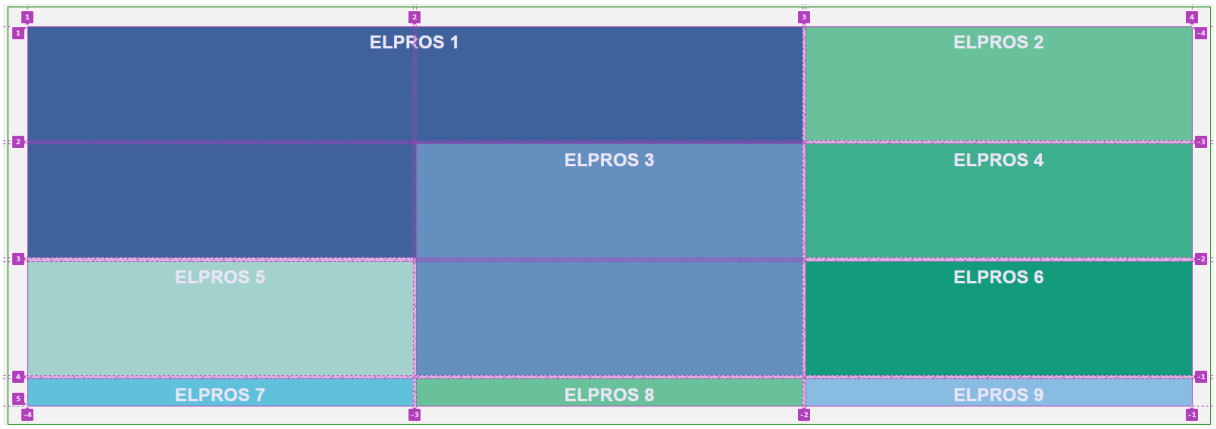
Element 2 i element 3 počinju smještanje od druge linije stupca i završavaju s linijom 4.

Uočava se pomicanje i skupljanje ostalih elemenata kako bi se oslobodio prostor za ovako definiran raspored elemenata.

Ovakvim raspoređivanjem elemenata može doći do preklapanja elemenata. U prednji plan dolazi element s većim z-indeksom.

```
.elpros1 {
  background-color: #336699;
  grid-row: 1/3;
  grid-column: 1/3;
}
```

```
.elpros3 {
  background-color: #5C98C0;
  grid-column: 2/3;
  grid-row: 2/4;
}
```



Slika 3.91. Djelovanje z-indeksa na raspored elemenata

Isto tako pomoću linija grida možemo premještati elemente definirajući njihovu željenu lokaciju. Ostali elementi mreže pomaknut će se i smještati iza tog elementa.

```
.elpros9{
  background-color: #84CAE7;
  grid-row: 2/3;
  grid-column: 2/3;
}
```

Slika 3.92. Premještanje elemenata

Deveti element premješten je na lokaciju petog elementa. Peti element pozicionirao se odmah iza njega.

Grid mrežu može se izraditi i na drugačiji način definirajući područje **grid, grid-template.**

Predložak mreže definira se navođenjem naziva područja kao što su zaglavlje, podnožje, navigacija, članak i slično. Ponavljanje naziva područja znači da sadržaj obuhvaća više ćelija.

Vrijednosti su:

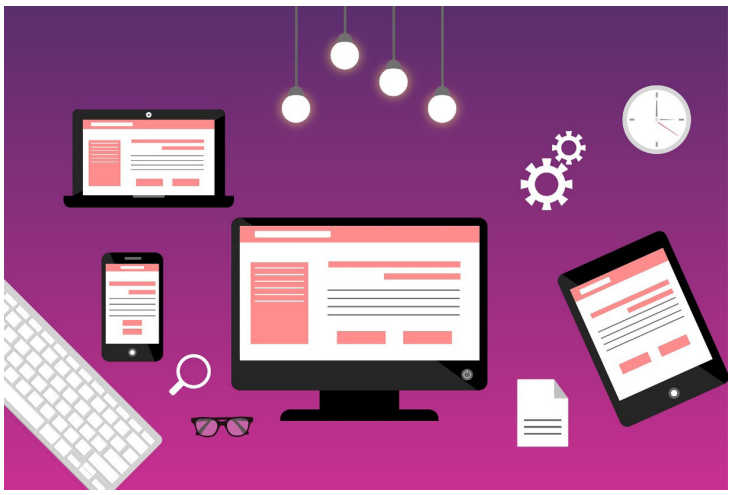
- naziv područja grid
- točka koja označava praznu ćeliju mreže
- none – područja mreže nisu definirana.

```
  display: grid;
grid-template:
    "header header header header" 2em
    "nav main main aside" 10vh
    "footer footer footer footer"2em
    /repeat(1fr);
}
```

Slika 3.93. Kreiranje rasporeda navođenjem imena područja

3.5. Prilagodljivost prikaza *web*-dokumenata s obzirom na širinu prikaza

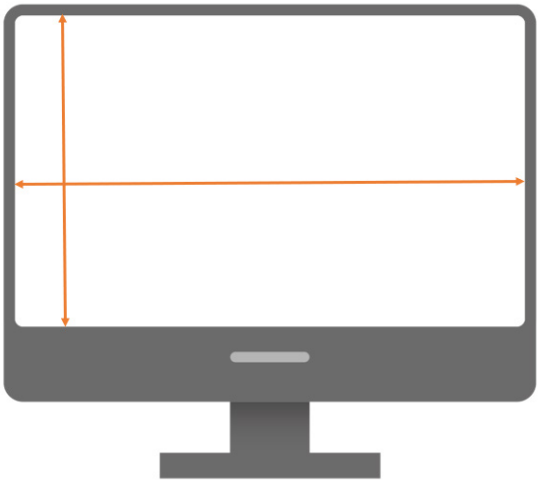
Prilagodljivost web-dokumenta različitim veličinama zaslona na različitim uređajima važno je svojstvo pri izradi web-stranica. Prilagodljive (engl. responsive) web-stranice automatski prilagođavaju svoje sadržaje veličini zaslona na kojima se prikazuju.



Izvor: Pixabay.com

Slika 3.94. Prikaz prilagodljivih web-stranica na različitim uređajima

Prilikom promjene prikaza dolazi i do promjene rasporeda elemenata na stranici ovisno o veličini i rezoluciji zaslona uređaja. Prilikom izrade prilagodljivog dizajna koriste se flexbox i grid raspored elemenata te medija upiti (engl. *media queries*). Važnu ulogu imaju i prilagodljive slike na stranici, kao i videozapisi.



Slika 3.95. Dimenzije vidljivog područja

3.5.1. Prilagodljivost *web*-dokumenta ovisno o širini prikaza

Vidljivo područje (engl. *viewport*) prilagodljivih *web*-stranica ovisi o uređaju na kojem se učitava i njegove se dimenzije mijenjaju ovisno o veličini zaslona uređaja. Vidljivo područje predstavlja dio zaslona uređaja na kojem se učitava sadržaj *web*-stranica.

Metaelementom *viewport* određuju se inicijalne vrijednosti vidljivog područja te način kontrole dimenzija stranice (skaliranje) prilikom prikaza u *web*-pregledniku.

```
eta name="viewport" content="width=device-width, initial-scale=1.0
```

Za vidljivo područje (engl. *viewport*) moguće je definirati:

- **device-width** – postavljanje širine prikaza da prati širinu zaslona (200 – 10000 px ili device-width)
- **device-height** – postavljanje visine prikaza da prati visinu zaslona (223 – 10000 px ili device-height)
- **initial-scale** – postavljanje prve razine zumiranja učitavanja stranice (0.1 – 1)
- **minimum-scale, maximum-scale i user-scalable** – svojstva za kontrole korisničkog zumiranja (0.1 – 10)
- **userscalable(“yes”/”no”)** – uključivanje ili isključivanje zumiranja.

Važno je pri izradi responzivnog *web*-dokumenta koristiti se i relativnim mjernim jedinicama kao što su:

- **ViewWidth (vw)** – predstavlja 1 % širine vidljivog područja
- **ViewHeight (vh)** – predstavlja 1 % visine vidljivog područja
- **vmin** – predstavlja 1 % najmanje dimenzije vidljivog područja
- **vmax** – predstavlja 1 % najveće dimenzije vidljivog područja.

3.5.2. Medijski upiti u listama stilova za upravljanje dimenzijama sadržaja ovisno o veličini prikaza (engl. *Media Queries*)

Medijski upiti (engl. *Media Queries*) omogućuju prilagodbu sadržaja različitim rezolucijama i primjenu različitih CSS stilova. Vezani su uz uređaj koji prikazuje sadržaj učitane *web*-stranice. Oni se odnose na medijske postavke izlaznog uređaja, kao što su dimenzije zaslona, rezolucija, orijentacija prikaza i slično. Medijski upiti omogućuju izradu pravila vezanih za karakteristike određenog uređaja na kojem se sadržaj učitava.

Vrste medija (engl. *media type*) za primjenu svojstva media jesu:

- **all** – upit je za sve uređaje
- **screen** – upit za različite zaslone računala, tableta, mobitela
- **print** – upit za prikaz i ispis na pisaču
- **projection** – upit za prikaz na projektoru
- **speech** – upit za čitače zaslona
- **tv** – upit za televizore.

Svojstvo media može poprimiti vrijednosti:

- **width** – širina prikaza: max-width i min-width za postavljanje maksimalne i minimalne širine prikaza
- **height** – visina prikaza: max-height i min-height za postavljanje maksimalne i minimalne visine prikaza
- **device-width** – širina zaslona
- **device-height** – visina zaslona
- **aspect-ratio** – omjer širine i visine prikaza izlaznog uređaja
- **device-aspect-ratio** – omjer širine i visine uređaja
- **orientation** – pejzažna ili portretna orijentacija zaslona uređaja
- **resolution** – rezolucija zaslona izlaznog uređaja
- **color** – broj bitova po boji, kada nema boja poprima vrijednost 0.

Točke loma (engl. *breakpoints*) prikazane su u tablici:

Vrijednost	Opis
0 – 480 px	manji mobilni uređaji
481 – 768 px	veći mobilni uređaji i tableti
769 – 1279 px	prijenosna računala / manji zasloni na stolnim računalima / pejzažna orijentacija na tabletima
1280 px i više	veći zasloni na stolnim računalima

Tablica 3.12. Točke loma (engl. breakpoints)

Poželjno je prilagoditi *web*-stranicu za mobilne uređaje zbog njihove široke upotrebe.

Pravilo dobre prakse jest izrada liste stilova prvo za manje zaslone, odnosno **mobile first** pristup.

Izvor: Pixabay.com

Slika 3.96. Shematski prikaz za ***mobile first*** pristup

Izvor: Pixabay.com

Slika 3.97. Shematski prikaz za ***desktop first*** pristup

Ovisno o odabiru pristupa, *mobile first* ili *desktop first*, slaže se i poredak medijskih upita. *Mobile first* pristup kreće s izradom liste stilova za male zaslone, a nakon njih za veće zaslone. Kod *desktop first* pristupa radi se suprotno, odnosno prvo se rade liste stilova za velike zaslone, a nakon toga liste stilova za manje zaslone.

Medijske upite moguće je postaviti unutar HTML elemenata <link>. Mogu se povezati s različitim CSS stilovima za uređaje različitih širina zaslona.

```
<link rel="stylesheet" media="all" href="../css/normal.css">
<link rel="stylesheet" media="screen and (min-width: 1025px)" href="../css/veliki.css">
<link rel="stylesheet" media="screen and (max-width: 768px)" href="../css/mali.css">
```

Slika 3.98. Povezivanje s različitim CSS dokumentima ovisno o veličini zaslona

Ako se medijski upiti postavljaju unutar CSS dokumenta, navodi se ključna riječ **@media** iza koje se unutar vitičastih zagrada { } navodi lista CSS stilova.

```
/*mobilni uređaji širine zaslona do 320 px, portret orjentacija*/
@media only screen
and (max-width: 320px)
{
/* CSS lista stilova */
}

/*mobilni uređaji širine zaslona između 321 px i 767 px, pejzaž orjentacija*/
@media only screen
and (min-width: 321px)
and (max-width: 767px)
{
/* CSS lista stilova */
}
```

Slika 3.99. Primjer postavljanja medijskih upita za mobilne uređaje

```
/* Tableti, širine zaslona između 768 px i 1024 px, portret orjentacija */
@media only screen
and (min-device-width: 768px)
and (max-device-width: 1024px)
and (orientation: portrait)
{
/* CSS lista stilova */
}

/* Tableti, širine zaslona između 768 px i 1024 px, pejzaž orjentacija */
@media only screen
and (min-device-width: 768px)
and (max-device-width: 1024px)
and (orientation: landscape)
{
/* CSS lista stilova */
}
```

Slika 3.100. Primjer postavljanja medijskih upita za tablete

```
/* Stolna računala s širinom zaslona većim od 1280 px */
@media only screen
and (min-width: 1280px)
{
/* CSS lista stilova */
}
```

Slika 3.101. Primjer postavljanja medijskih upita za stolna računala

Općenito, medijski upiti sastoje se od ključne riječi **@media**, zatim navođenja vrste medija i uvjeta, odnosno jedne ili više karakteristika izlaznog uređaja. Kod kreiranja složenih medijskih upita, pri navođenju više različitih uvjeta koristimo se logičkim operatorima *only, and, or* ili *not*.

U prethodnim primjerima vidljivo je kako se **operator *and*** koristi za grupiranje vrste medija s karakteristikama medija ili kombiniranje više medijskih karakteristika u jedan medijski upit.

Operator *not* koristi se za negiranje cijelog medijskog upita, odnosno mijenja njegovo normalno značenje. Operator *not* ne može se koristiti za negiranje pojedinačnog uvjeta, već samo za cijeli medijski upit.

Operator *only* sprečava starije preglednike da primjenjuju stilove, odnosno skriva upite od njih. Kod upotrebe operatora *not* ili *only* nužno je navođenje tipa medija.

Ukoliko u medijskom upitu nije naveden tip medija po podrazumijevanim postavkama primijenit će se tip *all*.

Ukoliko je kombinirano više medijskih upita upotrebom operatora *or* ili odvajanjem zarezima, omogućit će se primjena iste liste stilova u različitim situacijama.

```
@media (min-height: 768px), screen and (orientation: portrait)
{
    /* CSS lista stilova */
}
```

Slika 3.102. Upotreba operatora *or*

Lista stilova primijenit će se na uređajima s minimalnom visinom 768 px ili kad je uređaj u portretnom načinu prikaza. Upit će se primjerice izvršiti i na pisaču s visinom stranice 800 px jer je prvi uvjet istinit, ali isto tako i na mobitelu koji je u portretnom načinu prikaza bez obzira na to je li njegova rezolucija ispod ili iznad 768 px jer će u tom slučaju drugi uvjet biti istinit.

3.5.3. Prilagodljivost slika u ovisnosti o širini prikaza

Za prilagodljivost *web*-stranice važna je i prilagodljivost slika. Ona se postiže definiranjem širine za sve slike na 100 % dostupne širine roditeljskog elementa i automatsku prilagodbu visine. Na taj će se način zadržati proporcije dimenzija slike.

```
img{
    width: 100%;
    height: auto;
}
```

Slika 3.103. CSS stilovi za prilagodljive slike

Ovako definirana svojstva dopustit će skaliranje slike i na veće dimenzije od stvarnih dimenzija slike, što bi moglo rezultirati njenom lošom rezolucijom. Zbog toga je poželjnije koristiti se svojstvom max-width, koje će dopustiti smanjivanje slike, ali prilikom njezina povećanja ići će do maksimalnih dimenzija izvorne slike.

```
img{
    max-width: 100%;
    height: auto;
}
```

Slika 3.104. Svojstvo *max-width*

Nije preporučljivo postavljati fiksne dimenzije slika jer se u tom slučaju ne prilagođavaju zaslonima na kojima se prikazuju.

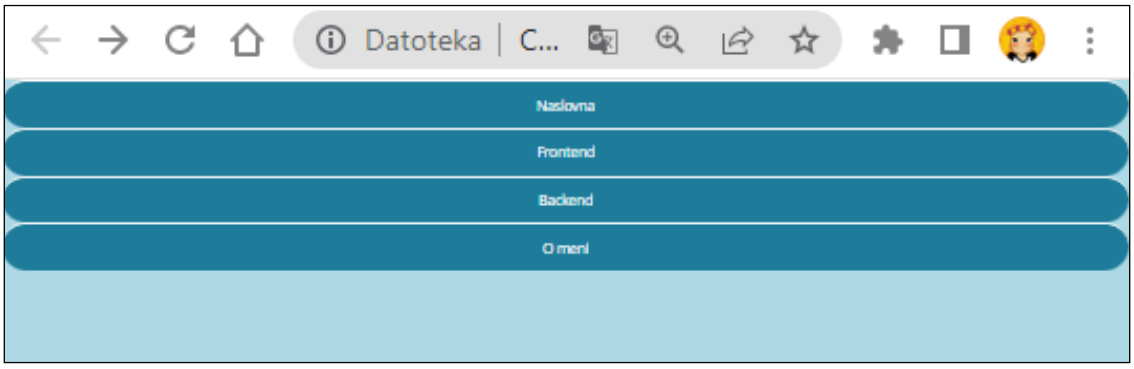
3.5.4. Izrada složenih izbornika oblikovanjem prikaza pravilima liste stilova

Kod izrade složenih izbornika koriste se ugniježđeni popisi za definiranje stavki izbornika.

```html
<nav>
  <ul>
    <li><a href="index.html">Naslovna</a></li>
    </li>
    <li><a href="#">Frontend</a>
      <ul>
        <li><a href="web.html">Web tehnologije</a></li>
        <li><a href="html.html">HTML</a></li>
        <li><a href="css.html">CSS</a></li>
        <li><a href="javascript.html">JavaScript</a></li>
        <li><a href="sass.html">SASS</a></li>
        <li><a href="bootstrap.html">Bootstrap</a></li>
        <li><a href="react.html">React</a></li>
      </ul>
    </li>
      <li><a href="#">Backend</a>
        <ul>
          <li><a href="web.html">Web tehnologije</a></li>
          <li><a href="mysql.html">MySQL</a></li>
          <li><a href="csharp.html">C#</a></li>
          <li><a href="dotnet.html">.NET</a></li>
          <li><a href="webapi.html">Web API</a></li>
          <li><a href="testiranje.html">Testiranje koda</a></li>
        </ul>
    </li>
        <li><a href="about.html">O meni</a>
  </ul>
</nav>
```

Slika 3.105. HTML kod izbornika

Slika 3.106. Izgled izbornika u pregledniku



Slika 3.107. Izgled izbornika na zaslonima manjima od 600 px

```css
nav ul {
    font-family: Arial, Verdana;
    font-size: 14px;
    margin: 0;
    padding: 0;
    list-style: none;
}
nav ul li {
    display: block;
    position: relative;
    float: left;
}
```

Slika 3.108. CSS stilovi vanjske liste

Nakon definiranja stilova za vanjsku listu potrebno je sakriti stavke unutarnje liste kako ne bi uvijek bile vidljive. Prelaskom miša preko odabranog gumba glavnog izbornika stavke podizbornika postat će vidljive.

```css
nav li ul {
    display: none;
}
```

Slika 3.109. Skrivanje stavki unutarnje liste

```css
nav ul li a {
    display: block;
    text-decoration: none;
    text-align: center;
    color: #ffffff;
    border-top: 1px solid #ffffff;
    padding: 5px 15px 5px 15px;
    border-radius: 10px;
    background: #1e7c9a;
    margin-left: 1px;
    width: 400px;
}
```

Slika 3.110. CSS stilovi za oblikovanje izgleda gumba

Prilikom dohvaćanja elemenata izbornika koristimo se selektorom potomaka (engl. *descendant*), koji nam omogućuje dohvaćanje elemenata smještenih unutar drugih elemenata. Na njih primjenjujemo željene liste stilova.

Za definiranje stanja poveznica koristimo se pseudoklasama **hover** i **visited**.

```css
nav li:hover a {
  background: #66d0eb;
}
nav li:hover li a:hover {
    background: #1e7c9a;
}
```

Slika 3.111. Primjer upotrebe pseudoklase hover

Pitanja za ponavljanje

1. Koja je uloga CSS-a na stranici?
2. Nabrojite vrste CSS-a.
3. Koji su nedostaci linijskog i unutarnjeg CSS-a?
4. Koje su prednosti vanjskog CSS-a?
5. Koja je razlika između blok- i linijskih elemenata?
6. Zašto koristimo selektore?
7. Nabrojite vrste selektora!
8. Koja je razlika između ugnježđivanja i grupiranja selektora?
9. Koja sve svojstva možemo koristiti za oblikovanje teksta i fonta?
10. Čemu služe pojedina svojstva?
11. Na koje načine možemo urediti pozadinu stranice korištenjem CSS-a?
12. Objasnite na koji se način mogu urediti poveznice i definirati njihova stanja.
13. Što su pseudoklase, a što pseudoelementi?
14. Kojim svojstvima uređujemo popise?

15. Kako se može promijeniti način označavanja?

16. Koja se svojstva mogu koristiti za uređivanja tablice?

17. Nabrojite dijelove box modela.

18. Na koji način određujemo širinu i visinu nekog elementa?

19. Koja je razlika između svojstva width, min-width i max-width?

20. Koja je razlika između svojstava height, max-height i min-height?

21. Čemu služi svojstvo position i koje vrijednosti može poprimiti?

22. Navedite i usporedite metode pozicioniranja.

23. Koja je uloga svojstva float? Koje vrijednosti može poprimiti?

24. Što omogućuje svojstvo display?

25. Što je flexbox? Koje su njegove prednosti?

26. Koji su elementi rasporeda grid?

27. Što je prilagodljivost (responzivnost) stranice?

28. Koja je razlika između mobile first i desktop first pristupa?

29. Što su medijski upiti?

30. Kako se definiraju medijski upiti?

4. Izrada dinamičkih *web*-stranica ugradnjom skripti klijentskog skriptnog jezika JavaScript

U OVOM POGLAVLJU NAUČIT ĆETE:

- Što je JavaScript?
- Koja su mu svojstva?
- Kako koristiti JavaScript za interakciju s korisnicima?

4.1. Uvod u skriptni jezik JavaScript

Slika 4.1. Logotip JavaScripta

JavaScript je skriptni jezik koji omogućuje dodavanje **interaktivnih sadržaja** za primjenu na mrežnim stranicama. Kompatibilan je sa svim preglednicima. Ugrađuje se ili poziva u HTML. Dodavanjem JavaScript koda omogućuje se promjena izgleda dokumenta, od teksta i boje, pa do promjena opcija dostupnih u padajućem izborniku i slično. JavaScript je **klijentski** (engl. *client-side*) jezik, što znači da se sve operacije koje se izvode događaju na klijentskom računalu.

U novije vrijeme primjena JavaScripta proširila se i na razvoj poslužiteljskih servisa poput **API-a** (engl. *Application Programming Interface*) i sl.

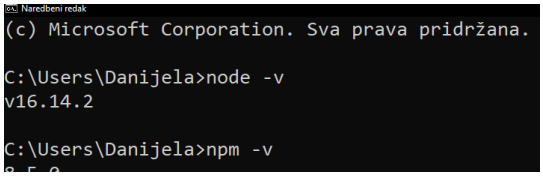
Danas je JavaScript pod standardom **ECMAScript** jedan od najpopularnijih i najkorištenijih programskih jezika u kojem su napisane brojne moderne *web*- i mobilne aplikacije. Standard se svake godine nadograđuje novim funkcionalnostima i unapređuje se dizajn.

Godine 2009. pojavio se **Node.js – JavaScript runtime okruženje**, koje omogućuje izvođenje JavaScript koda izvan preglednika. Node.js omogućuje programerima pisanje JavaScripta koda na strani poslužitelja, kao i izvođenje skripti na strani poslužitelja, što omogućuje stvaranje dinamičkog sadržaja mrežne stranice prije nego što se stranica pošalje u korisnikov preglednik. Node.js predstavlja paradigmu *JavaScript posvuda,* koja objedinjuje razvoj *web*-aplikacija oko jednog programskog jezika, umjesto različitih jezika za skripte na strani poslužitelja i klijenta.

Godine 2010. pojavljuje se i upravitelj paketima pod nazivom **npm**, koji je olakšao programerima objavljivanje i preuzimanje dijelova otvorenog koda za Node.js. Danas npm sadrži mnoštvo paketa otvorenog koda te značajno olakšava razvoj klijentskih i poslužiteljskih aplikacija.

- Prije početka rada s Node.js potrebno je preuzeti datoteke ovisno o korištenom operacijskom sustavu i instalirati ga.

  https://nodejs.org/en/download/



Nakon odabira lokacije za instalaciju dovoljno je pratiti korake instalacije. Po završetku instalacije potrebno je potvrditi instalaciju pomoću naredbenog retka. Unosimo node -v i npm -v da provjerimo koje su verzije ovih aplikacija instalirane.

Za pisanje programskog koda JavaScripta, Node.js i primjenu NPM paketa koristit ćemo **Visual Studio Code.**

4.1.1. Svojstva skriptnih jezika

Skriptni jezici imaju izrazito **malu brzinu izvođenja.**

Dijele se u dvije skupine:

- **poslužiteljski** (engl. server side), primjerice Node.js, Python, PHP, ASP, Java, C++
- **klijentski** (engl. client side) primjerice JavaScript, AJAX.

Svojstva JavaScripta:

- objektno orijentiran
- izvodi se na strani klijenta
- omogućuje kontrolu izgleda i sadržaja stranice
- omogućuje kreiranje ponašanja preglednika, primjerice otvaranje dodatnih prozora (engl. *popup boxes*)
- omogućuje validaciju unosa korisnika
- rad mu se temelji na događajima (engl. *events*).

4.1.2. Sintaksa i ugradnja skripte u HTML dokument

Skripta JavaScripta može biti ugrađena u HTML ili vanjska.

Ugrađena skripta

```
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Ugrađena skripta</title>
8      <script type="text/javascript">
9          alert("Hello World!");
10     </script>
11 </head>
12 <body>
13
14 </body>
15 </html>
```

Slika 4.2. Primjer skripte ugrađene u HTML

Za pisanje ugrađene skripte koristimo oznake **<script>…</script>**. Budući da postoji više različitih klijentskih skriptnih jezika, potrebno je specificirati o kojem se jeziku radi, pa dodajemo atribut **type=”text/javascript”**.

Skripte možemo postaviti u **<head>…</head>** ili **<body>…</body>** ili na oba mjesta.

Vanjska skripta

Želimo li da se kod iz skripte izvodi na više različitih stranica bez da na svakoj pišemo skriptu, kreirat ćemo vanjsku skriptu koju onda **uključujemo** u HTML dokument stranice.

Možemo je uključiti u zaglavlju ili na dnu oznake **<body>…</body>.**

Opća sintaksa jest:

<script src=”imeSkripte.js”></script>.

Budući da datoteka sadrži nastavak .js, preglednik automatski prepoznaje tip skripte, pa ga nije potrebno specificirati.

- Neki antivirusni programi (primjerice Bitdefender) blokiraju izvođenje vanjske -.js skripte, pa je potrebno omogućiti njihovo izvođenje u postavkama preglednika.

Ukoliko se skripta ne nalazi u mapi u kojoj se nalazi HTML datoteka u koju je uključujemo, potrebno je navesti **putanju do skripte**.

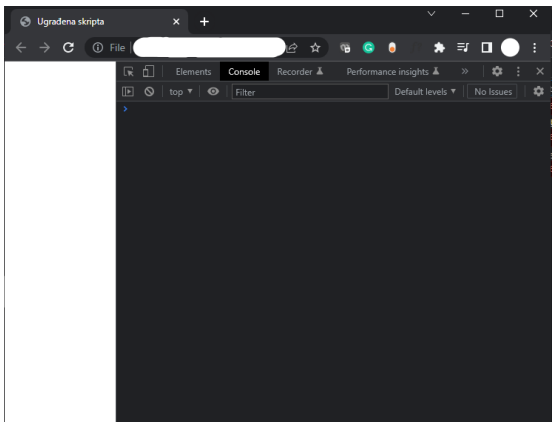
Možemo uključivati i skripte s vanjskih mrežnih sjedišta. Tada je potrebno unutar atributa **src** navesti puni **URL** skripte.

- U JavaScriptu sve naredbe programskog koda završavaju s ;.

Slika 4.3. Primjer vanjske skripte

JavaScript može prikazati podatke na različite načine:

- ispis u HTML element pomoću **innerHTML**
- ispis u HTML dokument pomoću **document.write()**
- ispis u pop up prozor pomoću **window.alert()**
- ispis u razvojnu konzolu preglednika pomoću **console.log().**



Chrome DevTools konzola pomaže nam pratiti ponašanje koda. Otvaramo je desnim klikom u prozoru preglednika i biramo **Provjeri** (engl. *inspect*). Ako postoje problemi s izvršavanjem, ovdje nam se ispisuju pogreške, a možemo i tijekom kodiranja ispisivati podatke u nju primjenom metode **console.log().**

4.1.3. Varijable i konstante u skriptnom jeziku JavaScript

Varijable su spremnici za pohranu podataka. Svaka varijabla mora imati jedinstveno ime – **identifikator**. Za kreiranje imena varijabli možemo koristiti:

- slova engleske abecede a-z i A-Z, znamenke 0 – 9, donju crtu _ i oznaku dolar $
- ime mora započinjati slovom ili _ ili $
- JS razlikuje velika i mala slova
- ključne riječi ne smijemo koristiti kao ime varijable (break, delete, case, do,

if, switch, var, catch, else, in, this, void, continue, new, null, for, with, de fault, NaN, Number, String …).

- JavaScript je jezik koji razlikuje velika i mala slova:
  mojaVarijabla != mojavarijabla
- Dobra je praksa varijablama davati imena koja ih dobro opisuju, primjerice:
  ime, prezime, dob
- U JavaScriptu imena varijabli obično pišemo upotrebom načela camelCase
  godisnjiDohodak, brojDana

Za deklaraciju varijabli i konstanti koristimo ključne riječi **var, let i const.**

Deklaraciju **var** koristimo za globalno dostupne varijable i lokalne varijable unutar tijela funkcije (dostupne samo unutar te funkcije). **Vrijednost** im se tijekom izvođenja programa **može mijenjati**.

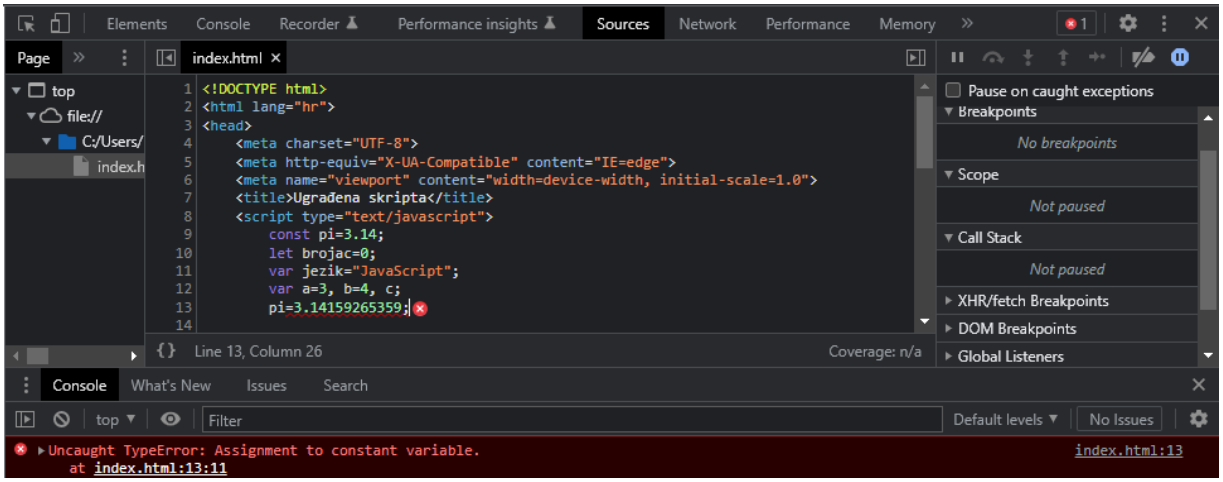
Deklaraciju **let** koristimo unutar određenog bloka, a njihova vrijednost **nije dostupna** izvan tog bloka. **Vrijednost** im se tijekom izvođenja programa **može mijenjati**. Često ih koristimo kao iteracijske varijable kod programskih petlji.

Deklaracija **const** ima svojstva jednaka deklaraciji let, no obavezno ju je inicijalizirati. Inicijalizacija podrazumijeva pridruživanje vrijednosti deklariranoj konstanti. Jednom dodijeljena vrijednost **ne može se mijenjati**. (Pogledaj sliku 4.4.) Koristimo ih kad deklariramo novo polje, objekt, funkciju i RegExp.

var	let	const
Doseg je globalan ili unutar bloka funkcije.	Doseg je unutar bloka u kojemu je definirana.	Doseg je unutar bloka u kojemu je definirana.
Moguće je mijenjati vrijednost i redeklarirati je.	Moguće je mijenjati vrijednost, ali ne i redeklarirati je unutar istog bloka.	Nije moguće mijenjati vrijednost i redeklarirati je jer ima konstantnu referencu. Redaklaracija podrazumijeva ponovno deklariranje konstante istog imena.
Moguća je deklaracija bez inicijalizacije vrijednosti.	Moguća je deklaracija bez inicijalizacije vrijednosti.	Nije moguća deklaracija bez inicijalizacije vrijednosti.
Moguće joj je pristupiti bez inicijalizacije, a zadana vrijednost je undefined.	Nije moguće pristupiti joj bez inicijalizacije, te u tom slučaju vraća error.	Nije moguće pristupiti joj bez inicijalizacije i nije ju moguće deklarirati bez inicijalizacije.

Tablica 4.1. Razlike između var, let i const

- Preporuka je koristiti deklaraciju **let** i **const** jer se primjenom var pri promjeni vrijednosti varijable unutar bloka mijenja vrijednost varijable izvan bloka.
- Prije prve upotrebe varijable potrebno ju je kreirati, odnosno **deklarirati i inicijalizirati** je, odnosno dodijeliti joj vrijednost.
- Dobra je praksa deklarirati sve varijable na početku skripte.



Slika 4.4. Primjer upotrebe ključnih riječi var, let i const i pogreška pri pokušaju izmjene vrijednosti konstante

Osnovni elementi sintakse JavaScripta jesu:

- **vrijednosti** – varijable i literali
- **operatori** – aritmetički, logički, relacijski…
- **izrazi** – kombinacija vrijednosti i operatora
- **ključne riječi** – let, var, const, return, function, new…
- **komentari** – za objašnjavanje ili onemogućavanje izvršavanja dijelova koda
- **uvlake i razmaci** – za bolju preglednost koda
- **zagrade**

  ( ) - definiranje parametara funkcije, testiranje uvjeta i definiranje redoslijeda izvođenja

  [ ] – definiranje polja

  { } – za blokove koda kod grananja, programskih petlji, objekata i funkcija

  < > – za ugnježđivanje oznaka HTML-a.

4.1.4. Važnost i načini komentiranja programskog koda u JavaScriptu

```
//komentar u jednom retku
/* Komentar
kroz
više
redaka*/
```

Dva su načina pisanja komentara u JavaScriptu, u jednom retku ili kroz više redaka:

Slika 4.5. Primjer pisanja komentara u JavaScriptu

Komentiranje programskog koda predstavlja jednu od poželjnih aktivnosti prilikom izrade programskih rješenja. Komentari služe lakšem razumijevanju značenja dijelova koda, kao i obavljenih operacija u svrhu lakšeg održavanja.

Komentare možemo koristiti i kad ne želimo da se izvrši određeni dio programskog koda.

Pojedinačni tipovi podataka (primitivni) u JavaScriptu mogu biti:

- **brojčani**
- **tekstualni** – znakovni nizovi (stringovi)
- **Booleani** (mogu poprimiti samo jednu od dvije vrijednosti true ili false)
- **symbol**
- **bigint.**

Varijable mogu poprimiti i **vrijednosti**:

- **undefined** – označava vrijednost varijabli koje nisu inicijalizirane ili objekata ili elemenata niza koji ne postoje. Ovu vrijednost dobit ćemo i pri izvršenju funkcije koja ne vraća nikakvu vrijednost. Razlikujemo je od *not defined*, koja označava da deklariranoj varijabli nije pridružena vrijednost.
- **null** – označava odsutnost vrijednosti, u JavaScriptu poprima vrijednost un defined
- **NaN (engl. *Not a Number*)** – doslovno označava vrijednost koja nije broj (primjerice kvocijent stringa i broja). Svaka operacija s NaN vraća NaN, a ne predstavlja nikakvu vrijednost.

Varijable su u JavaScriptu **dinamičke**, što znači da istoj varijabli možemo dodijeliti vrijednost broja, stringa ili Booleana.

Kolekcije podataka u JavaScriptu:

- **polja**
- **objekti.**

Tip	Objašnjenje	Primjer
znakovni niz (string)	Tekst odnosno znak ili niz znakova omeđen s ‘ ‘ ,” ” ili ```	let ime=”JavaScript”;
brojčani (number)	Broj koji nije omeđen s ‘ ‘ ili ” ” Integer – cjelobrojna vrijednost (bez decimalnih vrijednosti) Float – broj s decimalnom vrijednosti, obično zauzima više memorijskog prostora. Decimalni broj automatski se konvertira u cijeli ako su znamenke iza decimalne točke nule. Svi su brojevi u JavaScriptu floating point tip broja i pohranjuju se u 64-bitnom zapisu.	let ES=6;
Boolean	Mogu poprimiti samo jednu od dvije vrijednosti true ili false	let JavaScript = true;
symbol	Za razliku od ostalih jednostavnih tipova ne koriste se literali, nego se definira pomoću metode Symbol(), koja može, a i ne mora primati parametre. Definira novu jedinstvenu vrijednost pri svakom pozivu.	let ime = Symbol(‘Ana’);
bigint	Predstavlja cijele brojeve veće od 2^{53}-1. Vrijednosti varijable na kraju dodamo oznaku n ili primjenjujemo metodu BigInt().	const velikiBroj=9007199254740991n; const josVeci=BigInt(9007199254740992);
niz (array)	Struktura koja omogućuje pohranu većeg broja vrijednosti različitih tipova.	let ELPROS=[‘RCK ELPROS’, ‘Istarska’, 3, 31000, ‘Osijek’];

objekt	Objekt u JavaScriptu može biti bilo što, ali obično sadrži više vrijednosti	Sve prijašnje primjere možemo smatrati objektima, kao primjerice i:
		let naslov=document.querySelector(‘h1’);
		const person={
		ime: “Ana”,
		prezime: “Anić”,
		dob: 22
		};

typeof	Operator koji nam omogućuje određivanje tipa varijabli	const ime = ‘Ana’;
		typeof(ime); //vraća “string”
		const dob = 22;
		typeof(dob); // “number”
		const zensko = true;
		typeof(zensko); // “boolean”
		const adresa = null;
		typeof(adresa); // “object”

Tablica 4.2. Tipovi podataka u JavaScriptu

4.2. Osnovni koncepti skriptnog jezika JavaScript

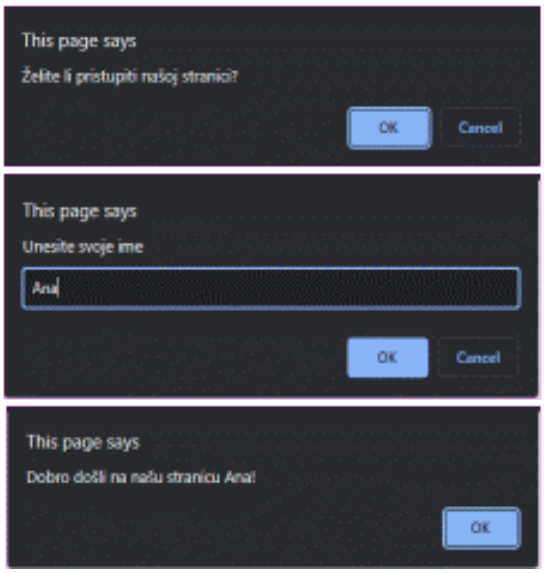
4.2.1. Interakcija s korisnikom i konverzija tipova podataka u JavaScriptu

JavaScript nam omogućuje interakciju s korisnicima. Reakcija programa ovisi o informaciji dobivenoj od korisnika.

Funkcije korisničkog sučelja za interakciju s korisnikom:

- **alert** – jednostavan pop up prozor sa sadržajem ili bez njega koji uvijek dolazi s gumbom *U redu (OK)*. Služi za prikazivanje poruke i pauziranje izvršavanja skripte dok se ne pritisne navedeni gumb
- **prompt** – služi za prikupljanje informacija od korisnika. Korisniku se prikazuje tekst i omogućuje unos u polje za unos. Osim toga ima dva gumba *OK* i *Cancel*
- **confirm** – korisniku se prikazuje tekst i omogućuje odabir jedne od ponuđenih opcija: *OK* ili *Cancel.*

```
let punoljetni=confirm('Želite li pristupiti našoj stranici?');
let ime=prompt('Unesite svoje ime');
alert('Dobro došli na našu stranicu '+ime+'!');
```



Slika 4.6. Primjena funkcija za interakciju s korisnikom

Osim putem navedenih metoda podatke od korisnika možemo prikupiti u različitim elementima na samoj stranici (engl. *input*). O oblikovanju elemenata forme i validaciji korisnikova unosa bit će riječi kasnije.

JavaScript **implicitno (automatski)** obavlja konverzije tipova podataka ovisno o kontekstu u kojemu se podatak koristi te o operatoru koji se primjenjuje, ali nam omogućuje i da varijablama **eksplicitno (manualno)** promijenimo tip.

Metoda	Opis	Primjer

Number()	- uzima znakovni niz i vraća broj
	- nizovi koji sadrže brojeve poput „3.14" pretvaraju se u brojeve
	- prazni nizovi pretvaraju se u 0
	- sve ostalo se pretvara u **NaN** (engl. *Not a Number*)
		Number('123') // vraća broj 123
		Number('123') === 123 //true
		Number("unicorn") // NaN
		Number(undefined) // NaN

parseFloat()	- uzima znakovni niz i vraća decimalni broj	parseFloat(10); // vraća 10
		parseFloat("10"); // 10
		parseFloat("10.33"); // 10.33
		parseFloat("34 45 66");// 34
		parseFloat("Dob: 40");//NaN

parseInt()	- uzima znakovni niz i vraća cijeli broj	parseInt(10); // vraća 10
		parseInt("10"); // 10
		parseInt("10.33"); // 10
		parseInt("34 45 66");// 34
		parseInt("Dob: 40");//NaN

String() i .toString()	- uzima broj i pretvara ga u znakovni niz	String("12345"); // vraća 12345
		String(12345); // vraća 12345

.toFixed(n)	- uzima broj i vraća znakovni niz zapisan s zadanim brojem (n) decimala	let broj=5.56789; let n=broj.toFixed();//6
.toPrecision(n)	- uzima broj i vraća znakovni niz zapisan zadanim brojem (n) znamenki	let broj=5.56789; let n=broj.toPrecision(1);//5.6

Tablica 4.3. Funkcije za pretvorbu tipova podataka u JavaScriptu

4.2.2. Operatori i metode za rad s brojevima i znakovnim nizovima u JavaScriptu

U JavaScriptu razlikujemo više vrsta operatora.

Aritmetički operatori

Operator	Opis	Primjeri

+	binarni operator za zbrajanje dviju vrijednosti	let a=5, b=2;
		let c=a+b; ↳c poprima vrijednost 7
		let d='11', e=45;
		let f=c+d; ↳ f poprima vrijednost 1145
		let ime='Ana', prezime='Anić';
		let imePrezime=ime+' '+prezime ↳
		imePrezime poprima vrijednost 'Ana Anić

-	binarni operator za oduzimanje dviju vrijednosti	let a=5, b=2; let c=a - b; ↳c poprima vrijednost 3

*	binarni operator za množenje dviju vrijednosti	let a=5, b=2;
		let c=a * b; ↳ c poprima vrijednost 10
		let d='11', e=45;
		let f=c*d; ↳ f poprima vrijednost 495

**	binarni operator za potenciranje	et a=5, b=2; let c=a ** b; ↳ c poprima vrijednost 25
/	binarni operator za dijeljenje dviju vrijednosti	let a=5, b=2; let c=a / b; ↳ c poprima vrijednost 2.5
%	binarni operator – modul (vraća ostatak cjelobrojnog dijeljenja dva broja)	let a=5, b=2; let c=a % b; ↳c poprima vrijednost 1
++	unarni operator koji djeluje na jednu varijablu, a povećava njezinu trenutnu vrijednost za 1 razlikujemo primjerice ++i i i++: ++i poveća vrijednost varijable i za 1 i vraća uvećanu vrijednost i++ poveća vrijednost varijable i za 1, ali vraća prethodnu vrijednost	let a=5; a++; ↳ a poprima vrijednost 6 a++ skraćeni je zapis izraza a=a+1;
- -	varijablu, a umanjuje njezinu trenutnu vrijednost za 1 razlikujemo primjerice --i i i--: --i umanji vrijednost varijable i za 1 i vraća umanjenu vrijednost i-- umanji vrijednost varijable i za 1, ali vraća prethodnu vrijednost	let a=5; a--; ↳ a poprima vrijednost 6 a-- skraćeni je zapis izraza a=a-1;

Tablica 4.4. Aritmetički operatori u JavaScriptu

Operatori za dodjelu vrijednosti

Operator	Opis	Primjeri	Zamjenjuje
=	varijabli dodjeljuje određenu vrijednost	let a=5, ime='Ana'; a poprima vrijednost 5 ime poprima vrijednost 'Ana'	
+ =	vrijednost varijable s lijeve strane operatora povećava za vrijednost s desne strane	let a=5, b=2; a+=b; ↳ a poprima vrijednost 7	a=a+b;
- =	vrijednost varijable s lijeve strane operatora umanjuje za vrijednost s desne strane	let a=5, b=2; a-=b; ↳ a poprima vrijednost 3	a=a-b;
* =	vrijednost varijable s lijeve strane operatora množi s vrijednošću s desne strane	let a=5, b=2; a*=b; ↳ a poprima vrijednost 10	a=a*b;
/ =	vrijednost varijable s lijeve strane operatora dijeli s vrijednošću s desne strane	let a=5, b=2; a/=b; ↳ a poprima vrijednost 2.5	a=a/b;
% =	vraća vrijednost ostatka cjelobrojnog dijeljenja vrijednosti s lijeve strane s vrijednošću s desne strane	let a=5, b=2; c=a % b; ↳ c poprima vrijednost 1	a=a%b;
** =	vrijednost varijable s lijeve strane operatora potencira s vrijednošću s desne strane	let a=5; a**=2; ↳ a poprima vrijednost 25	a=a**2;

Tablica 4.5. Aritmetički operatori u JavaScriptu

Operatori usporedbe

Operator	Opis	Primjeri za slučaj a=5;
= =	služi za provjeru jesu li vrijednosti s lijeve i desne strane jednake vrijednosti	a==8 // *false* a==5 // *true* a=='5' // *true*
= = =	služi za provjeru jesu li vrijednosti s lijeve i desne strane jednake vrijednosti i jednakog tipa	a===5 // *true* a==='5' // *false*
!=	služi za provjeru jesu li vrijednosti s lijeve i desne strane različite vrijednosti	a!=8 // *true*

Operator	Opis	Primjeri za slučaj a=5;
!= =	služi za provjeru jesu li vrijednosti s lijeve i desne strane različite vrijednosti i različitog tipa; ako uspoređuje stringove, konvertira u broj pa uspoređuje jednu po jednu znamenku	a!==8 // *true* a!=='5' // *true* a!==5 // *false*
>	služi za provjeru je li vrijednosti s lijeve strane veća od vrijednosti s desne strane	a>8 // *false*
<	služi za provjeru je li vrijednosti s lijeve strane manja od vrijednosti s desne strane	a<8 //*true*
>=	služi za provjeru je li vrijednosti s lijeve strane veća ili jednaka od vrijednosti s desne strane	a>=8 //*false*
<=	služi za provjeru je li vrijednosti s lijeve strane manja ili jednaka od vrijednosti s desne strane	a<=8 //*true*

?	uvjetni (engl. *ternary*) operator	let dob=22;
		let voznja=(dob>=18) ? "automobil" : "bicikl";
		//vraća automobil

Tablica 4.6. Operatori usporedbe u JavaScriptu

Logički operatori

Operator	Opis	Primjeri
&&	logičko I, vraća *true* ako su oba uvjeta zadovoljena (*true*), inače vraća *false*	let a=5, b=2; (a>2)&&(b<2) vraća *false*
||	logičko ILI, vraća *true* ako je bilo koji uvjet zadovoljen, inače vraća *false*	let a=5, b=2; (a>2)||(b<2) vraća *true*
!	logičko ne, vraća *false* ako je uvjet zadovoljen, u suprotnom vraća *true*	let a=5, b=2; !(a>2) vraća *false* !(b<2) vraća *true*

Tablica 4.6. Operatori usporedbe u JavaScriptu

Prioritet izvršavanja operatora

Prioritet	Operatori
1.	Aritmetički: ++, --, *, **, /, %, +, -
2.	Operatori usporedbe: <, <=, ==, ===, !=, !==, >=,>
3.	Logički: !,&&,||

Tablica 4.7. Prioritet izvršavanja operatora

Metode za rad s tekstom

Svojstva/metode	Opis	Primjeri
.length	svojstvo koje vraća broja znakova u stringu	let jezik = "JavaScript"; jezik.length;//vraća 10
charAt()	metoda koja vraća znak na navedenom indeksu	let jezik=”JavaScript”; jezik.charAt(1); //vraća a
trim()	metoda koja uklanja praznine na početku i kraju stringa	let jezik = " JavaScript "; jezik.trim();//vraća JavaScript
indexOf()	metoda koja vraća indeks stringa unutar stringa	let stih=”Crvena ruža, žuti leptir”; stih.indexOf(”žuti”); //vraća 13
split()	metoda koja pretvara string u niz	let tekst=“Lako je naučiti JavaScript“; tekst.spilt(” ”); //vraća Lako,je,naučiti,JavaScript
replace()	metoda koja vraća novi string sa zamijenjenim izrazom	let stih=“Crvena ruža, žuti leptir“; stih.replace(/žuti/g,“plavi“); //vraća Crvena ruža, plavi leptir

Svojstva/metode	Opis	Primjeri
toUpperCase()	metoda koja pretvara sva slova u stringu u velika	let jezik = "JavaScript"; jezik.toUpperCase();//vraća JAVASCRIPT
toLowerCase()	metoda koja pretvara sva slova u stringu u mala	et jezik="JavaScript"; jezik.toLowerCase();//vraća javascript
search()	metoda koja traži da li se izraz nalazi unutar znakovnog niza – vraća indeks podudaranja (-1 ako ga nema)	let tekst="Lako je naučiti JavaScript"; tekst.search(/je/i);//vraća 5

concat()	metoda koja povezuje više nizova u jedan	let ime="Ana";
		let prezime="Anić";
		ime.concat(prezime);//AnaAnić
		ime.concat(" ",prezime); // Ana Anić

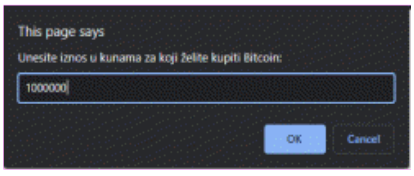
substring()	metoda koja vraća dio znakovnog niza između dvaju indeksa	let jezik = "JavaScript"; jezik.substring(4,9);//vraća Script

Tablica 4.8. Dio svojstva/metoda za rad sa znakovnim nizovima u JavaScriptu

- Neke metode za rad sa stringovima mijenjaju izvorni string, a neke vraćaju novi string.

U sljedećem primjeru od korisnika preko prompta preuzimamo podatak. Kako bismo osigurali da JavaScript podatak koji je preuzet neće promatrati kao string, pomoću metode **parseFloat()** pretvaramo ga u broj kako bismo ga u sljedećem koraku mogli dijeliti s vrijednošću tečaja Bitcoina. Metodom **.toFixed**(2) dobiveni iznos zaokružujemo na dva decimalna mjesta.

```
index.html > ...
1  <!DOCTYPE html>
2  <html lang="en">
3  <head>
4      <meta charset="UTF-8">
5      <meta http-equiv="X-UA-Compatible" content="IE=edge">
6      <meta name="viewport" content="width=device-width, initial-scale=1.0">
7      <title>Coin shop</title>
8  </head>
9  <body>
10     <script type="text/javascript">
11         let n=prompt("Unesite iznos u kunama za koji želite kupiti Bitcoin: ");
12         n=parseFloat(n);
13         let BTC=n/203654.30;
14         document.write("Za željeni iznos dobiti ćete "+BTC+" BTC.<br>");
15         document.write("Odnosno "+BTC.toFixed(2)+" BTC.");
16     </script>
17 </body>
18 </html>
```



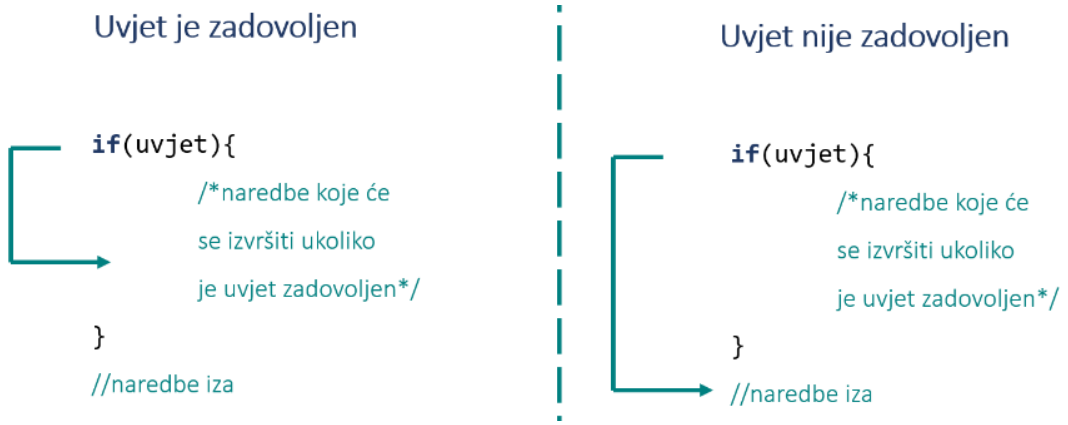
Za željeni iznos dobiti ćete 4.910281786340873 BTC.
Odnosno 4.91 BTC.

Slika 4.7. Primjer jednostavne komunikacije s korisnikom uz upotrebu osnovnih operatora i metoda za rad s brojevima

4.2.3. Uvjetno grananje i petlje u skriptnom jeziku JavaScript

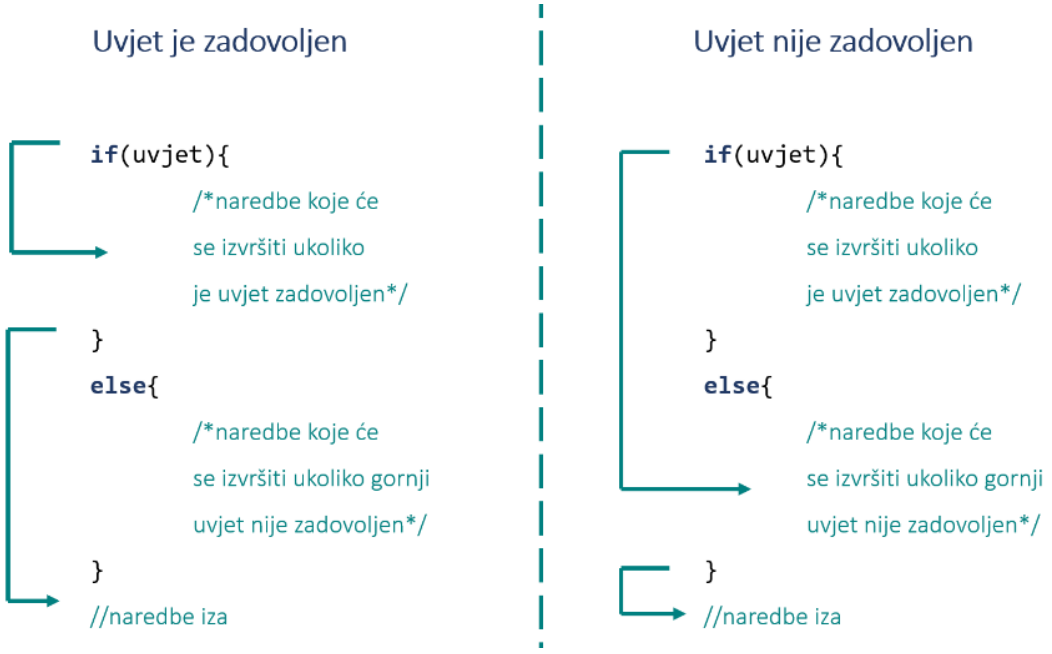
Kod **uvjetnog grananja** izvršava se ili preskače blok naredbi ovisno o istinitosti zadanog uvjeta.

Razlikujemo:



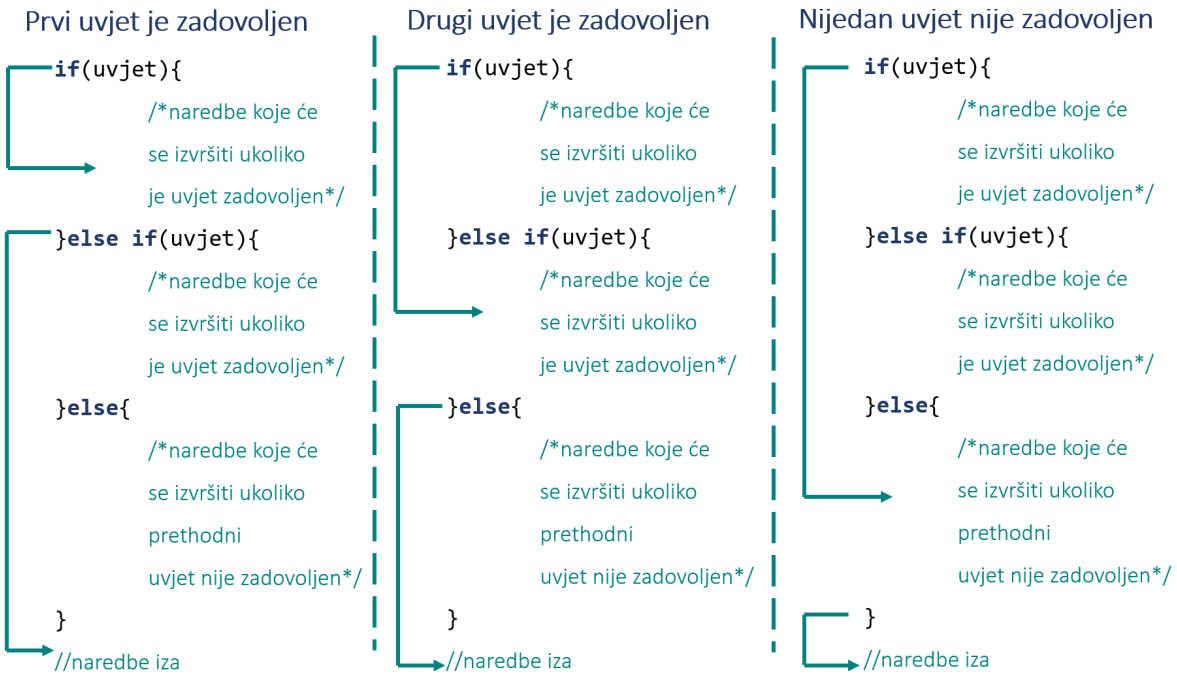
ako je uvjet zadovoljen onda se izvrši blok naredbi unutar { },
inače se izvršavanje nastavlja iza { }

Slika 4.8. Rad s naredbom if



else – za **if** vrijedi ranije navedeno, a **else** nam daje alternativu za slučaj da uvjet nije zadovoljen

Slika 4.9. Rad s naredbama if else



if else if else … – koristimo dodatne else if ako želimo provjeriti dodatne uvjete.

Slika 4.10. Rad s naredbama if else if else

U sljedećem primjeru možemo vidjeti primjenu grananja if else if else za provjeru je li broj koji je unio korisnik jednak nuli ili negativan, odnosno pozitivan. Podatke od korisnika preuzimamo pomoću prompta, a ispis rezultata vršimo na samoj stranici

index.html > ...

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Uvjetno grananje</title>
</head>
<body>
    <script type="text/javascript">
    // Provjera je li broj negativan, nula ili pozitivan
        const broj = prompt("Unesite broj: ");
    // Prvojera je li broj već od 0 (pozitivan)
        if (broj > 0) {
            document.write("Broj je pozitivan.");
        }
    // check if number is 0
        else if (number == 0) {
            document.write("Broj je 0");
        }
    // Ako broj nije niti veći od 0 niti 0
    else {
        document.write("Broj je negativan");
    }
    </script>
</body>
</html>
```

Slika 4.11. Rad s naredbama if else if else

Switch case sastoji se od niza slučajeva (*case*), od kojih svaki završava ključnom riječi break, koja osigurava izlazak iz tijela (bloka) naredbe switch.

```
switch(varijabla/izraz){
case 1:
        /*naredbe koje će se izvršiti ukoliko je uvjet zadovoljen*/
break;
case 2:
        /*naredbe koje će se izvršiti ukoliko je uvjet zadovoljen*/
break;
case 3:
        /*naredbe koje će se izvršiti ukoliko je uvjet zadovoljen*/
break;
…
default:
        /*naredbe koje će se izvršiti ukoliko se ne izvrši niti jedan od slučajeva*/
break;
```

Slika 4.12. Rad sa switch case

U sljedećem primjeru možemo vidjeti primjenu grananja switch *case* za izradu jednostavnog kalkulatora. Podatke od korisnika preuzimamo pomoću prompta, no ispis je ovaj puta u konzoli preglednika. U tu svrhu koristimo metodu

console.log().

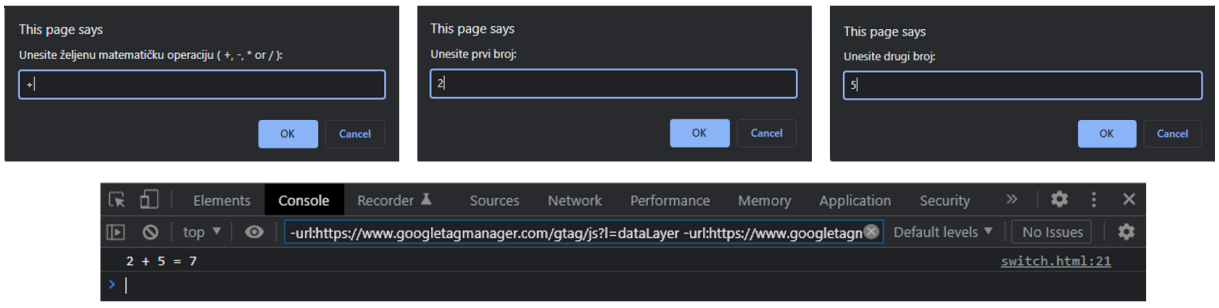
Možemo uočiti način uključivanja varijabli JavaScripta u ispis primjenom sintakse

${imeVarijable}

što nam znatno olakšava ispis. Da bismo mogli koristiti uključivanje vrijednosti varijabli na ovaj način, potrebno je koristiti tzv. **backtick** ``.

```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Klakulator</title>
</head>
<body>
    <script type="text/javascript">
        // program za jednostavan kalkulator
    let rezultat;
// unos operatora
const operator = prompt('Unesite željenu matematičku operaciju ( +, -, * or / ): ');
// unos brojčanih vrijednosti
const broj1 = parseFloat(prompt('Unesite prvi broj: '));
const broj2 = parseFloat(prompt('Unesite drugi broj: '));
switch(operator) {
    case '+':
        rezultat = broj1 + broj2;
        console.log(`${broj1} + ${broj2} = ${rezultat}`);
        break;
    case '-':
        rezultat = broj1 - broj2;
        console.log(`${broj1} - ${broj2} = ${rezultat}`);
        break;
    case '*':
        result = broj1 * broj2;
        console.log(`${broj1} * ${broj2} = ${rezultat}`);
        break;
    case '/':
        rezultat = broj1 / broj2;
        console.log(`${broj1} / ${broj2} = ${rezultat}`);
        break;

    default:
        console.log('Krivi operator');
        break;
}
    </script>
</body>
</html>
```

Slika 4.13. Primjer programskog koda koji koristi switch case



Slika 4.14. Primjer izvođenja programskog koda koji koristi switch case i ispisa na konzoli preglednika

Programske petlje omogućuju nam izvršavanje skupa naredbi ili funkcija uzastopno dok god je testirani uvjet zadovoljen. U JavaScriptu možemo koristiti programske petlje:

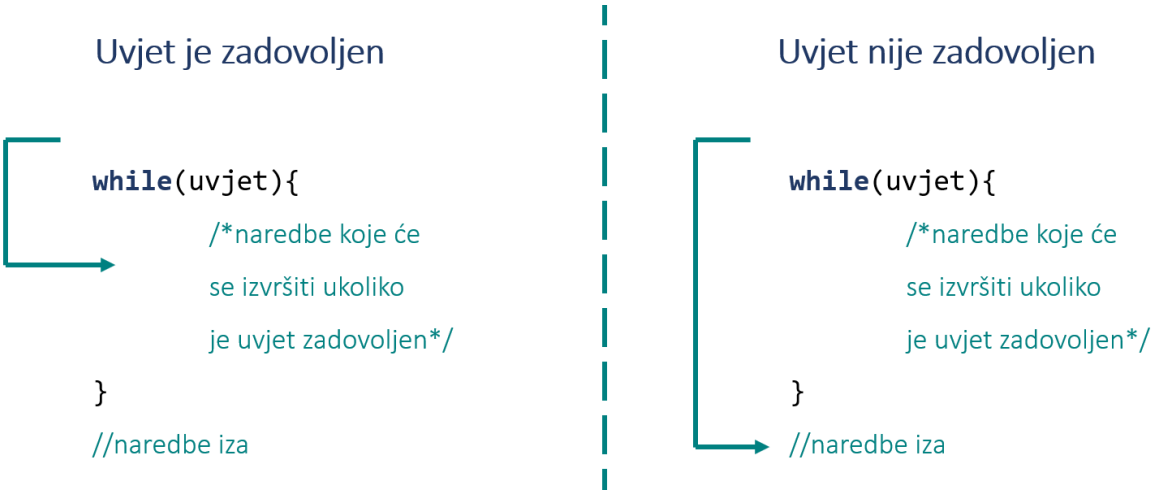
- **for**
- **while**
- **do while.**

Razlikujemo dvije vrste petlji:

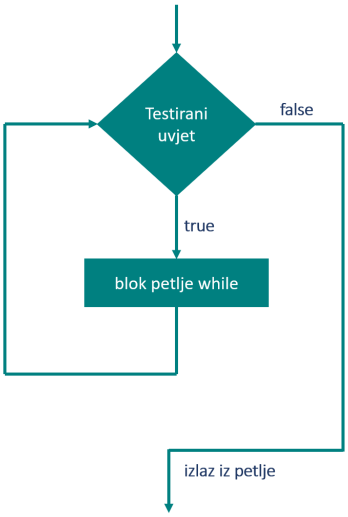
- **petlje kod kojih se uvjet provjerava na početku**, prije izvođenja naredbu unutar tijela- bloka petlje (for i while)
- **petlje kod kojih se uvjet provjerava na kraju**, nakon izvođenja naredbi unutar bloka petlje (do while), što osigurava izvođenje naredbi unutar bloka petlje najmanje jednom bez obzira na zadovoljenost testiranog uvjeta.

Programska petlja while

Kod programske petlje while **uvjet se testira na samom početku petlje**. Ukoliko je uvjet zadovoljen, izvode se naredbe unutar bloka petlje; ako nije, program se nastavlja izvoditi naredbama nakon bloka petlje.



Slika 4.15. Rad s programskom petljom *while*

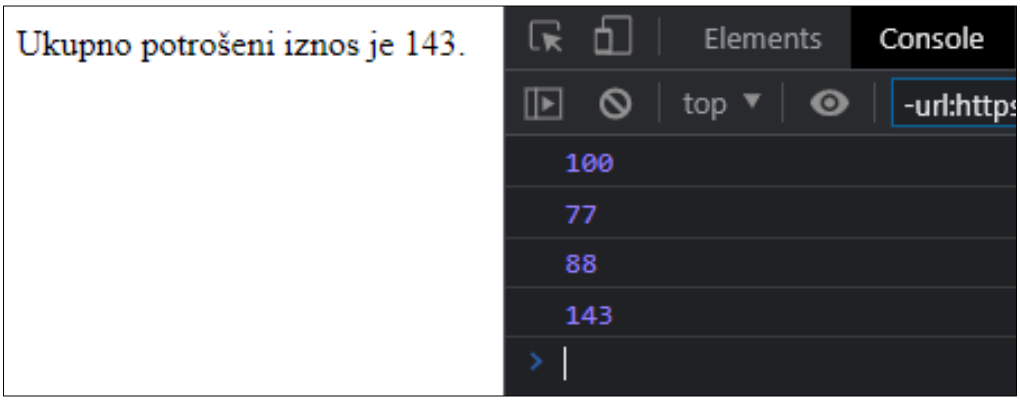


Slika 4.16. Blok-dijagram izvođenja petlje while

U sljedećem primjeru korisnik unosi cijene artikala koje želi kupiti. Za završetak kupovine unosi 0. U konzoli ispisujemo trenutno stanje sume košarice. Dodavanjem negativnog predznaka iznosu možemo uklanjati artikle iz košarice. Ispis u konzoli preglednika služi nam za provjeru rada petlje. Po završetku ukupnu sumu ispisujemo na stranici.

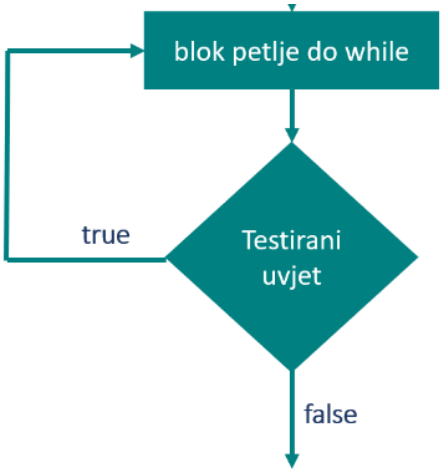
```
//početno inicijaliziranje sume u 0
let suma = 0;
// Inicijalni unos korisnika
let broj = parseFloat(prompt('Unesite cijenu prvog artikla: '));
while(broj != 0) {
// zbrajanje unosa
suma += broj;
// ponovljeni unos (izvršava se samo ako je prethodni unos bio različit od 0)
broj = parseFloat(prompt('Unesite cijenu slijedećeg artikla: '));
//ispis trenutne vrijednosti sume u konzoli preglednika
console.log(suma);
}
// ispis završne sume
document.write(`Ukupno potrošeni iznos je ${suma}.`);
```

Slika 4.17. Primjer upotrebe programske petlje *while*



Slika 4.18. Primjer ispisa rezultata izvođenja programske petlje *while* u konzoli preglednika i na samoj stranici

Programska petlja do while



Kod programske petlje do while **uvjet se testira na kraju petlje**, tako da se naredbe unutar bloka petlje uvijek izvrše **najmanje jednom**. Ukoliko je uvjet zadovoljen, izvode se naredbe unutar bloka petlje još jednom, a ako nije, program se nastavlja izvoditi naredbama nakon bloka petlje.

Slika 4.19. Blok-dijagram izvođenja petlje do while

Prethodni primjer prilagodit ćemo za programsku petlju do while. Možemo uočiti smanjenje broja linija koda jer nije potreban inicijalni unos korisnika budući da se njegova vrijednost sada provjerava na kraju izvršenja petlje.

```
//početno inicijaliziranje sume u 0
let suma = 0;
do{
// unos korisnika
broj = parseFloat(prompt('Unesite cijenu artikla dodanog u košaricu: '));
// zbrajanje unosa
suma += broj;
//ispis trenutne vrijednosti sume u konzoli preglednika
console.log(suma);
}while(broj != 0)
// ispis završne sume
document.write(`Ukupno potrošeni iznos je ${suma}.`);
```

Slika 4.20. Primjer upotrebe programske petlje *do while*

- Programske petlje while i do while koristimo kada ne znamo unaprijed koliko će se puta petlja izvršiti.

Programska petlja for

```
for(inicijalizacija; uvjet; korak){
        /*naredbe koje će
        se izvršiti ukoliko
        je uvjet zadovoljen*/
}
//naredbe iza
```

Kad je **točno poznat broj ponavljanja** nekog postupka ili su poznati početni i krajnji uvjet, koristi se petlja for.

Slika 4.21. Opći oblik programske petlje *for*

Programsku petlju for pokazat ćemo na sljedećem primjeru:
Marko četiri puta dnevno svojim električnim turističkim brodom obilazi park prirode. Radi se o kružnoj vožnji bez ulazaka i izlazaka. Svi putnici ulaze i izlaze na istom mjestu. Ulaznice prodaje isključivo na pristaništu prije polaska. Na brodić stane maksimalno 50 putnika u jednoj vožnji. Na kraju radnog dana provjerava stanje blagajne i odnosi utržak u banku. Svaki dan započinje s 0 kn u blagajni. Cijena svih ulaznica je 5 kn.

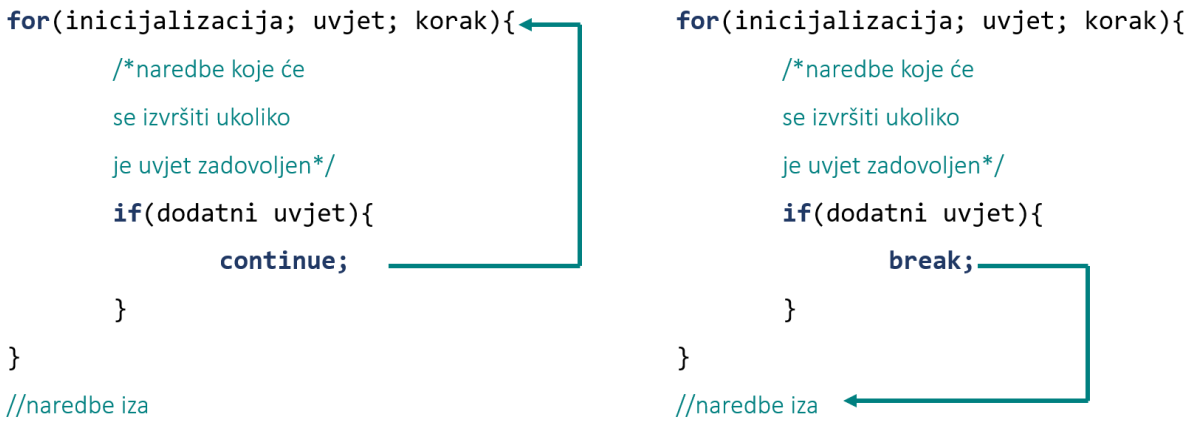
Možemo uočiti provjeru je li uneseni broj ulaznica veći od maksimalnog broja putnika te ponavljanje unosa ako to jest slučaj.

```
for (let i=1;i<=4;i++){
   brojProdanihUlaznica=parseInt(prompt(`Unesite broj
   prodanih ulaznica za ${i}. vožnju: `));
   if(brojProdanihUlaznica>50){
       brojProdanihUlaznica=parseInt(prompt(`Unijeli ste
       prevelik broj, ponovite unos broja prodanih ulaznica
       za ${i}. vožnju: `));
   }
   ukupniBrojProdanihUlaznica+=brojProdanihUlaznica;
}
stanjeBlagajne+=ukupniBrojProdanihUlaznica*5;
console.log(ukupniBrojProdanihUlaznica);
console.log(stanjeBlagajne);
```

Slika 4.22. Primjer upotrebe programske petlje *for*

Continue i break

U blokove programskih petlji često ugrađujemo dodatno testiranje određenih uvjeta, pa ovisno o njihovoj zadovoljenosti koristimo naredbe **continue** i **break**. Naredba **continue** preskače obradu trenutne vrijednosti iteracijske varijable i nastavlja sa sljedećom vrijednošću, dok **break** prekida izvršavanje petlje te se program nastavlja izvršavati naredbama iza petlje. Ove naredbe možemo uključiti u bilo koju od petlji. Uočite primjenu u programskoj petlji for:

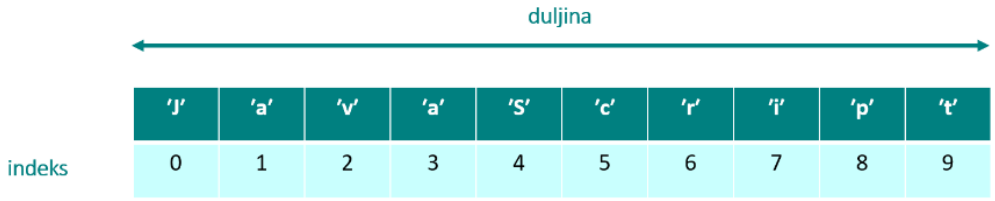


Slika 4.23. Rad s naredbama continue i break u programskoj petlji for

- Programske petlje možemo ugraditi jednu u drugu, što nazivamo **petljom u petlji ili ugniježđenom petljom.**
- Ukoliko je uvjet petlje uvijek zadovoljen, petlja se izvodi **beskonačno**. Stoga je jako važno dobro definirati uvjet izvršavanja petlje i testirati njeno izvođenje te pažljivo primjenjivati ključne riječi **break** i **continue.**

4.2.4. Nizovi i metode za manipulaciju nizovima u skriptnom jeziku JavaScript

Niz je posebna vrsta varijable, kolekcija u koju možemo pohraniti više od jedne vrijednosti. Pojedine vrijednosti u nizu nazivaju se **elementi niza**. Pojedini elementi u nizu dohvaćaju se brojkom, tzv. **indeksom elementa**. Indeksi elemenata počinju se brojati od nule, odnosno prvi element ima indeks 0, a ostali elementi imaju indeks koji označava koliko su mjesta udaljeni od prvog elementa niza.



Slika 4.24. Indeksiranje elemenata niza

- Indeksiranje elemenata nizova počinje s 0!

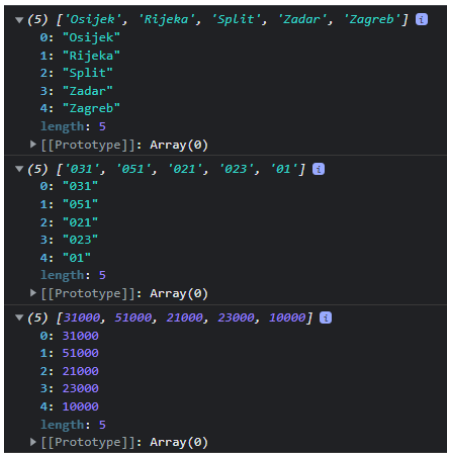
Nizovi se mogu proširivati i smanjivati po volji. Podaci unutar niza mogu biti **bilo kojeg tipa**, a sam je niz tipa **objekt**.

Nizove možemo kreirati **upotrebom literala** ili upotrebom ključne riječi ***new***.

- Preporuka je niz deklarirati primjenom tipa **const**.
- Preporuka je koristiti **literale** za kreiranje nizova.

```
//kreiranje niza pomoću literala
const gradovi=['Osijek','Rijeka','Split','Zadar','Zagreb'];
//kreiranje niza pomoću ključne riječi new
const pozivniBrojevi=new  Array('031','051','021','023','01');
const postanskiBrojevi =new Array(31000,51000,21000,23000,10000);
console.log(gradovi);
console.log(pozivniBrojevi);
console.log(postanskiBrojevi);
```

Slika 4.25. Kreiranje nizova pomoću literala i ključne riječi *new*



U konzoli preglednika možemo pogledati elemente nizova s pripadajućim indeksima i duljinu niza.

Slika 4.26. Ispis nizova u konzoli preglednika

Metode/Svojstva	Opis
.length	svojstvo koje vraća broj znakova u stringu
push()	dodaje element na kraj niza
pop()	briše zadnji element niza
sort()	reda (sortira) elemente niza (*defaultno* od a do z, nije primjenjivo za rad s brojčanim vrijednostima)
reverse()	obrne redoslijed elemenata u nizu
forEach()	iterira kroz niz
map()	prosljeđuje svaki element niza funkciji i vraća niz koji sadrži vrijednosti vraćene tom funkcijom
unshift()	dodaje element na početak niza

Tablica 4.9. Dio metoda za rad s nizovima u JavaScriptu

```
const gradovi=['Osijek','Rijeka','Split','Zadar','Zagreb'];
console.log(gradovi);
//dodaje Pula na kraj niza
gradovi.push('Pula');
//Mijenja vrijedost elementa s indeksom 1 iz Rijeka u Sisak
gradovi[1]="Sisak";
//dodaje element na početak niza (Vinkovci)
gradovi.unshift('Vinkovci');
//sortira niz po abecedi od a-z
gradovi.sort();
console.log(gradovi);
//briše zadnji element niza (Zagreb)
gradovi.pop();
console.log(gradovi);
```

```
▶ (5) ['Osijek', 'Rijeka', 'Split', 'Zadar', 'Zagreb']
▶ (7) ['Osijek', 'Pula', 'Slavonski Brod', 'Split', 'Vinkovci', 'Zadar', 'Zagreb']
▶ (6) ['Osijek', 'Pula', 'Slavonski Brod', 'Split', 'Vinkovci', 'Zadar']
```

Slika 4.27. Primjeri primjene metoda za rad s nizovima i pripadajući ispis u konzoli preglednika

- Elemente nizova ne možemo brisati upotrebom metode **delete** jer primjena ove metode ne briše elemente niza nego njihovu vrijednost, pa na navedenom mjestu ostaje element čija je vrijednost **undefined**.

Mnoštvo je metoda za rad s nizovima, stoga ćemo u ovom dijelu pokazati samo neke, dok će se neke pojaviti u kasnijim primjerima rada s funkcijama i objektima.

Na sljedećem primjeru vidimo da **metoda sort() nije prikladna** za sortiranje brojčanih nizova jer brojeve tretira kao stringove. Zbog toga za sortiranje brojčanih nizova trebamo primjenjivati **funkciju za usporedbu**. Funkcija uspoređuje dva po dva člana tako što ih oduzima, ako vrati negativan rezultat metoda sort pozicionira prvi element na niži indeks nego drugi.

```
const masa = [40, 100, 1, 5, 25, 10];
console.log(masa);
masa.sort();
console.log(masa);
masa.sort(function(a, b){return a-b});
console.log(masa);
```

```
▶ (6) [40, 100, 1, 5, 25, 10]
▶ (6) [1, 10, 100, 25, 40, 5]
▶ (6) [1, 5, 10, 25, 40, 100]
```

Slika 4.28. Primjeri problema sortiranja brojčanih nizova i priklanog rješenja te pripadajući ispis u konzoli preglednika

Višedimenzionalni nizovi

Višedimenzionalni niz jest niz koji sadrži **niz jednodimenzionalnih nizova.**

Dvodimenzionalni niz možemo si vizualizirati kao tablicu s određenim brojem redaka i stupaca. Za pristup vrijednostima elemenata potrebna su nam **dva cjelobrojna brojčana indeksa** od kojih prvi predstavlja **oznaku retka**, a drugi **oznaku stupca**.

```
const podaciDjelatnika=[['Ivan','Ivić',385991234567],
                        ['Ana','Anić',385992345671],
                        ['Petar','Perić',385993456712],
                        ['Marija','Marić',385994567124]];
console.log(podaciDjelatnika);
```

```
▼Array(4) ℹ
  ▶0: (3) ['Ivan', 'Ivić', 385991234567]
  ▶1: (3) ['Ana', 'Anić', 385992345671]
  ▶2: (3) ['Petar', 'Perić', 385993456712]
  ▶3: (3) ['Marija', 'Marić', 385994567124]
    length: 4
  ▶[[Prototype]]: Array(0)
```

Slika 4.29. Deklaracija i inicijalizacija dvodimenzionalnog niza podaciDjelatnika i ispis u konzoli preglednika

	[0]	[1]	[2]
[0]	'Ivan'	'Ivić'	385991234567
[1]	'Ana'	'Anić'	385992345671
[2]	'Petar'	'Perić'	385993456712
[3]	'Marija'	'Marić'	385994567123

Tablica 4.10. Vizualizacija dvodimenzionalnog niza podaciDjelatnika

Želimo li, primjerice, dohvatiti vrijednost mobilnog broja Ane Anić, potrebno je u kodu pozvati:

podaciDjelatnika[1][2].

Za dodavanje nizova u dvodimenzionalni niz možemo koristiti metodu **push()** ili metodu **splice()**, a za brisanje zadnjeg niza **pop()**. Isto tako možemo dodavati elemente pojedinim nizovima ili ih brisati iz njih, što zahtijeva pažljivo indeksiranje.

```
//kreiranje niza pomoću literala
const podaciDjelatnika=[['Ivan','Ivić',385991234567],
                        ['Ana','Anić',385992345671],
                        ['Petar','Perić',385993456712],
                        ['Marija','Marić',385994567123]];
//brisanje zadnjeg niza iz dvodimenzionalnog polja - sad imamo 3 retka
podaciDjelatnika.pop();
//dodavanje novog niza na kraj dvodimenzionalnog niza - sad imamo 4 retka
podaciDjelatnika.push(['Marko','Marković',385995671234]);
//umetanje niza između Ivan Ivić i Ana Anić - sad imamo 5 redaka
podaciDjelatnika.splice(1,0,['Jozefina','Jozić',385996712345]);
console.log(podaciDjelatnika);
```

```
▼ (5) [Array(3), Array(3), Array(3), Array(3), Array(3)]
  ▶ 0: (3) ['Ivan', 'Ivić', 385991234567]
  ▶ 1: (3) ['Jozefina', 'Jozić', 385996712345]
  ▶ 2: (3) ['Ana', 'Anić', 385992345671]
  ▶ 3: (3) ['Petar', 'Perić', 385993456712]
  ▶ 4: (3) ['Marko', 'Marković', 385995671234]
    length: 5
  ▶ [[Prototype]]: Array(0)
```

Slika 4.30. Primjena metoda za rad s dvodimenzionalnim nizovima podaciDjelatnika i ispis u konzoli preglednika

Iteracije nad nizovima

Iteracije nad nizovima možemo provesti **primjenom programskih petlji**, primjerice petljom **for**.

```
const gradovi=['Osijek','Rijeka','Split','Zadar','Zagreb'];
for(let i=0;i<gradovi.length;i++)
{
    document.write(`${gradovi[i]} `);
}
document.write(`<br>`);
for (let i of gradovi) {
        document.write(`${i} `);
    }
```

```
Osijek Rijeka Split Zadar Zagreb
Osijek Rijeka Split Zadar Zagreb
```

Slika 4.31. Iteracija kroz jednodimenzionalan niz gradovi primjenom programske petlje for; ispis u konzoli preglednika

Isto postižemo primjenom metode **forEach()**, koja zahtijeva poznavanje koncepta funkcija.

```
gradovi.forEach (function(podatak){
    document.write(`${podatak}<br>`);
});
```

Slika 4.32. Iteracija kroz jednodimenzionalan niz gradovi primjenom programske metode forEach()

Kroz **dvodimenzionalne nizove** također možemo **iterirati** primjenom petlje for i metode **forEach()**.

```
const podaciDjelatnika=[['Ivan','Ivić',385991234567],
                        ['Ana','Anić',385992345671],
                        ['Petar','Perić',385993456712],
                        ['Marija','Marić',385994567123]];
for (let i of podaciDjelatnika) {
    for (let j of i) {
        document.write(`${j} `);
    }
    document.write(`<br>`);
}
document.write(`<br>`);
podaciDjelatnika.forEach((djelatnik) => {
    djelatnik.forEach((podatak) => {
        document.write(`${podatak} `);
    });
    document.write(`<br>`);
});
```

```
Ivan Ivić 385991234567
Ana Anić 385992345671
Petar Perić 385993456712
Marija Marić 385994567123

Ivan Ivić 385991234567
Ana Anić 385992345671
Petar Perić 385993456712
Marija Marić 385994567123
```

Slika 4.33. Iteracija kroz dvodimenzionalni niz primjenom programske petlje for i metode forEach()

4.3. Funkcije u skriptnom jeziku JavaScript

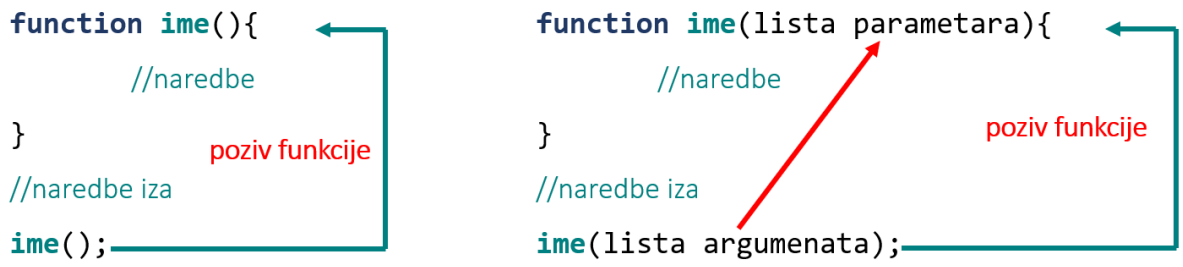
Funkcija je **blok koda** koji izvršava određeni **zadatak**.

Složenije probleme prije pisanja programskog koda raščlanjujemo na manje pojedinačne zadatke kako bismo ih lakše razumjeli i riješili. U programskom kodu za te pojedinačne zadatke kreiramo funkcije.

Deklaracija funkcije u JavaScriptu započinje ključnom riječi **function**, zatim slijedi **ime funkcije** (najbolje je da opisuje ulogu funkcije), pa **()**, koja može biti prazna ili se u njoj može nalaziti popis parametara koje funkcija prima. Nakon toga slijedi **blok naredbi unutar {…}**.

Deklaracijom funkcije kreiramo funkciju, ali ona se **ne izvodi dok god nije pozvana**. Ako funkcija prima parametre u pozivu joj ih je potrebno proslijediti ili je moguće postaviti **unaprijed zadane vrijednosti parametara** pri deklaraciji funkcije. Ove vrijednosti koriste se ukoliko pri pozivu funkcije nisu proslijeđeni potrebni argumenti.

- Vrijednosti koje funkcija prima, s kojima radi, nazivamo **parametrima**, a vrijednosti koje funkciji prosljeđujemo prilikom poziva nazivamo **argumentima**.



Slika 4.34. Poziv funkcije bez parametara i s njima

Često se jednom definirane funkcije višestruko puta koriste s različitim argumentima za dobivanje različitih rezultata.

JavaScript omogućuje deklaraciju funkcije pomoću konstruktora **function()** i ključne riječi **new**, iako je moguće raditi bez njih.

Ukoliko vrijednost izračunata u funkciji nije potrebna za daljnje izvođenje programa, ona se iz funkcije može, primjerice, ispisati i nema potrebe za vraćanjem vrijednosti na mjesto poziva funkcije. Izvedba funkcije ipak rezultira
vraćenom vrijednošću koja je **nedefinirana** (engl. *undefined*).

```
// deklaracija funkcije
function pozdrav(){
    console.log('Pozrdav iz JavaScript funkcije');
}
//poziv funkcije
pozdrav();
```

Slika 4.35. Primjer funkcije bez parametara i bez upotrebe ključne riječi return

Ukoliko je vrijednost izračunata u funkciji potrebna za daljnji tijek programa, unutar bloka funkcije koristimo ključnu **riječ return**.

> Ključnu riječ **return** postavljamo na sam kraj tijela (bloka) funkci jer se u suprotnom sve naredbe napisane iza ovog izraza nikad neće izvesti.

```
// deklaracija funkcije
function zbrajalica(a, b) {
    console.log(a + b);
}
//višestruki poziv funkcije
zbrajalica(1,2);
zbrajalica(3,4);

//deklaracija funkcije
function kub(b) {
    return b**3;
}
//unos korisnika
let radijus = parseFloat(prompt("Unesite radijus sfere u mm: "));
//poziv funkcije
let rna3=kub(radijus);
let obujamSfere=4/3*Math.PI*rna3;
obujamSfere=obujamSfere.toPrecision(2);
//ispis rezultata
document.write(`Obujam sfere radijusa ${radijus} mm je ${obujamSfere}`);
```

Slika 4.36. Primjeri funkcija s parametrima bez vraćanja vrijednosti i s njim

Osim **korisnički kreiranih funkcija** JavaScript ima i mnoštvo ugrađenih funkcija, primjerice **Math.sqrt()**, **Math.round()** i sl.

Prednosti upotrebe funkcija:

- jednom definiranu funkciju moguće je višestruko koristiti
- program je jednostavniji jer su pojedini zadaci razdvojeni
- olakšana je čitljivost programskog koda i održavanje istog.

Funkcije se u JavaScriptu mogu koristiti kao bilo koja druga varijabla, odnosno **funkcija kao vrijednost** ili **izraz** (engl. *expression*).

```
ar proizvodnaCijenaProizvoda=parseFloat(prompt("Unesite proizvodnu cijenu proizvoda u kn: "))
ar PDV=function(iznos){return iznos*0.25;};
ar cijenaProizvoda=proizvodnaCijenaProizvoda+PDV(proizvodnaCijenaProizvoda);
ocument.write(`Prodajna cijena proizvoda je ${cijenaProizvoda} kn`);
```

Slika 4.37. Funkcija kao varijabla

- **Hoisting** je defaultno ponašanje JavaScripta u kojem se deklaracije postavljaju na vrh njihova dosega. To nam omogućuje **pozivanje funkcije prije deklaracije**.

Samopozivajuće funkcije izvode se automatski bez pozivanja. Za njih je karakteristično smještanje u () nakon čega opet slijedi ();

```
//samopozivajuća funkcija
(function () {
    console.log("Mene nitko nije pozvao, a izvršila sam se!");
})();
```

```
Mene nitko nije pozvao, a izvršila sam se!
```

Slika 4.38. Primjer samopozivajuće funkcije

U gornjem primjeru vidimo da samopozivajuća funkcija **nema ime**, nego samo ključnu riječ za deklaraciju, () i blok s naredbama. Takve funkcije nazivamo **anonimnim funkcijama**. Anonimne funkcije koriste se kao **parametri drugih funkcija**.

```
let zbrajalica = function(x, y) {
    return x + y;
  }
//arrow funkcija
const zbrajalica = (x, y) => x + y;
```

Arrow funkcije omogućuju nam skraćivanje zapisa. Argumente zapisujemo unutar okruglih zagrada, a umjesto ključne riječi **return**, koristimo =>.

Slika 4.39. Primjer zapisa funkcije kao vrijednosti bez upotrebe arrow funkcije i s njom

4.4. Objekti u skriptnom jeziku JavaScript

Objekt je **skup svojstava** od kojih svako ima **naziv** i **vrijednost**.

4.4.1. Definiranje objekata u skriptnom jeziku JavaScript

Objekt se prepoznaje po vitičastim zagradama. U objekt možemo pohraniti vrijednosti različitih tipova.

Objekte možemo kreirati:

- primjenom literala – definiranje i kreiranje objekata u jednom iskazu
- pomoću ključne riječi **new**
- pomoću konstruktora
- pomoću **Object.create()**.

Slika 4.40. Primjeri kreiranja objekata pomoću literala

4.4.2. Svojstva objekata u skriptnom jeziku JavaScript

Pojedinim svojstvima objekta možemo **pristupiti** navođenjem imena objekta i naziva koji sadrži vrijednost, primjerice matični broj objekta polaznik1 možemo dohvatiti na dva načina:

polaznik1.maticniBroj

polaznik1['maticniBroj'].

Primjerice, za ispis prva 3 svojstva objekta polaznik1 možemo koristiti:

```
document.write(polaznik1.ime+' '+polaznik1.prezime+' '+polaznik1.maticniBroj+'<br>');
```

Ivan Ivić 1010

Slika 4.41. Dohvaćanje svojstava objekta i ispis na stranici

4.4.3. Metode objekata kao funkcionalni dijelovi objekata

Unutar objekta moguće je ugnijezditi druge objekte.

```
const polaznik1 = {
    ime: 'Ivan',
    prezime:'Ivić',
    maticniBroj:1010,
    pozdrav:function(){document.write(`Pozdrav!`)}
};
```

Slika 4.42. Ugnježđivanje metode u objekt

U gornjem primjeru možemo uočiti ugnježđivanje anonimne funkcije u objekt.
Želimo li unutar metode objekta koristiti vrijednosti objekta, koristimo ključnu riječ **this.**

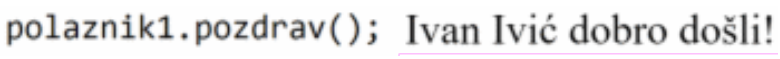
```
const polaznik1 = {
    ime: 'Ivan',
    prezime:'Ivić',
    maticniBroj:1010,
    pozdrav:function(){document.write(this.ime+' '+this.prezime+' dobro došli!')}
};
```

Slika 4.43. Primjer upotrebe vrijednosti objekta unutar metode objekta

4.4.4. Pristupanje metodama objekta

Za dohvaćanje metode objekta navodimo:

imeObjekta.nazivSvojstva();



Slika 4.44. Dohvaćanje metode objekta i ispis na stranici

4.5. Document Object Model (DOM) za upravljanje događajima

Document Object Model (DOM) jest **API (*Application Programming Interface*)** za upravljanje HTML dokumentima. DOM sadrži funkcije koje vam omogućuju učinkovito dodavanje, uklanjanje i izmjenu dijelova dokumenta.

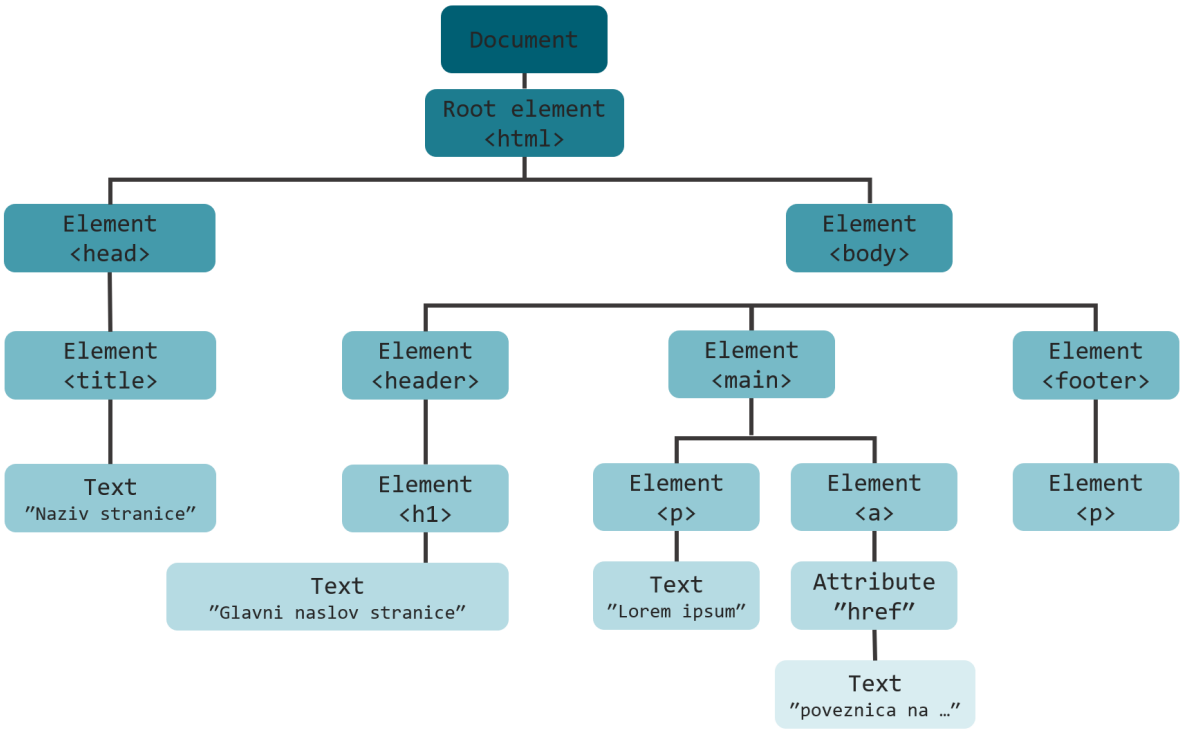
DOM definira:

- HTML elemente kao objekte
- svojstva svih HTML elemenata
- metode pristupa svim HTML elementima
- događaje za sve HTML elemente.

Želimo li pristupiti bilo kojem elementu HTML stranice, uvijek započinjemo s pristupom objektu dokumenta. DOM predstavlja HTML dokument kao **hijerarhiju čvorova.**

Za selektiranje DOM elementa možemo koristiti:

- **id** elementa
- **klasu** elementa
- **oznaku** elementa (tag)
- **CSS selektor**
- **selektor atributa** (title, href…)



Slika 4.45. DOM stablo HTML dokumenta

Metode DOM-a dodjeljuju se objektu **document**, primjerice:

```
document.getElementById('poruka').innerHTML = "Hello World!";
```

Osim DOM-a postoji i **BOM (Browser Object Model)**, koji omogućuje JavaScriptu upravljanje ponašanjem preglednika preko objekta **window**. Metode BOM-a dodjeljuju se objektu window, iako se, s obzirom na to da objekt window podržavaju svi preglednici, zapis često preskače.

Čak je i objekt dokumenta svojstvo objekta prozora, ali zapis objekta window preskače se:

```
window.document.getElementById('poruka');
```

```
document.getElementById('poruka');
```

Za rad s objektom window često koristimo metode **window.open(); i window.close();.**

4.5.1. Događaji na elementima HTML dokumenta

Kada se u HTML dokumentu koristi JavaScript, onda skripta može **reagirati** na **događaje**.

- Događaj je svaka interakcija između korisnika i aplikacije gdje svaka korisnička akacija ima unaprijed definiranu reakciju.

Događaji mogu biti:

- **generirani ponašanjem preglednika** (npr. stranica se učitala)
- **korisnički uvjetovani** (npr. korisnik klikne na neki element stranice, prijeđe mišem preko njega, stisne neku tipku i sl.).

HTML elementima dodajemo **atribute (*id, class*)** ili ih pomoću imena oznake (tag) povezujemo sa skriptom. **Reakcija** na te događaje izvođenje je koda ili dijela koda iz skripte te uzročno-posljedično **promjena na mrežnoj stranici.**

Događaj	Opis
onchange	na promjenu HTML elementa
onclick	na korisnikov klik na HTML element (obično na gumb ili sliku)
onmouseover	kada korisnik prelazi mišem preko HTML elementa
onmouseout	kada korisnik mišem izlazi izvan HTML elementa
onkeydown	kada korisnik stisne tipku na tipkovnici
onload	kada je preglednik završio učitavanje stranice

Tablica 4.11. Događaji JavaScripta

4.5.2. Metode objekta document za pristup elementima dokumenta

Metode za dohvaćanje HTML elemenata omogućuju nam djelovanje skripte na točno određeni dio dokumenta.

Metoda	Opis
document.getElementById('id')	dohvaća element na temelju imena id-a
document.getElementsByTagName('ime')	dohvaća element na temelju imena oznake
document.getElementsByClassName('ime	dohvaća element na temelju imena klase

Tablica 4.12. Metode za dohvaćanje HTML elemenata

U kombinaciji s različitim događajima elemente možemo mijenjati, dodavati ili uklanjati.

Metoda	Opis
element.innerHTML = *new html content*	mijenja sadržaj HTML elementa
element.attribute = *new value*	mijenja vrijednost atributa HTML elementa
element.style.property=*new style*	mijenja stilski izgled HTML elementa
element.setAttribute(attribute, value)	mijenja vrijednost atributa HTML elementa

Tablica 4.13. Metode za izmjenu HTML elemenata

Metoda	Opis
document.createElement(*element*)	kreira novi element u HTML dokumentu
document.removeChild(*element*)	uklanja HTML element
document.appendChild(*element*)	dodaje HTML element
document.replaceChild(*new, old*)	zamjenjuje HTML element drugim
document.write(*text*)	ispisuje u HTML dokument

Tablica 4.14. Metode za dodavanje i uklanjanje HTML elemenata

4.5.3. Izrada animacija JavaScriptom

Za izradu animacija koristimo metode:

- **setInterval(imeFunkcije, odgoda[ms])** – poziva funkciju koja joj je dodijeljena s definiranim rokom odgode između poziva
- **clearInterval()** – uklanja metodu setInterval()

- **setTimeout(imeFunkcije, odgoda[ms])** – poziva funkciju koja joj je dodijeljena nakon isteka definiranog roka odgode
- **clearTimeout()** – uklanja metodu setTimeout()

Za animaciju kreiramo ili ugradimo element koji ćemo animirati te ga stiliziramo CSS-om ako to želimo. Preko nekog od atributa dohvatimo ga u skripti.

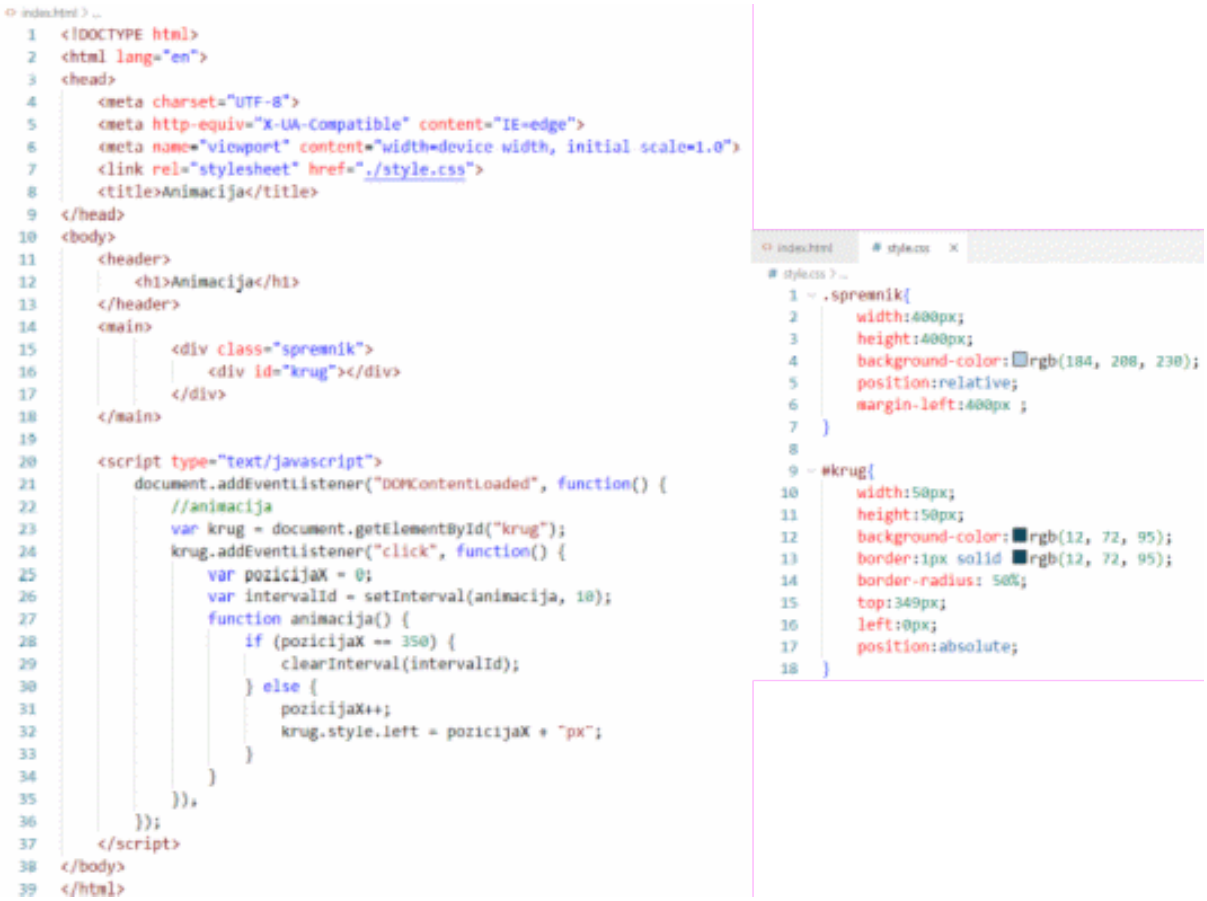
Animaciju možemo postaviti da se izvršava **automatski** ili na **temelju nekog događaja** (engl. *event*).

- Animaciju ćemo lakše definirati ako element koji želimo animirati postavimo u spremnik, a zatim tom spremniku postavimo u listi stilova svojstvo **position:relative;** dok elementu koji ćemo animirati dodijelimo svojstvo **position:absolute;.**

Animaciju ostvarujemo **promjenom vrijednosti liste stilova.**

Pogledajmo sljedeći primjer. Postavili smo spremnik u kojemu će se element krug kretati s lijeva na desno. Oba elementa imaju id kako bismo ih lakše dohvatili u skripti. U CSS-u smo postavili osnovna oblikovanja. Budući da će se krug početi kretati na klik postavili smo eventListener metodu koja će onemogućiti bilo kakve evente dok se ne iscrta cijeli sadržaj stranice. Element krug postavili smo u varijablu krug kako bismo mogli upravljati svojstvom left. Ono je u početku 0 u odnosu na element roditelj (spremnik), a pokretanjem anonimne funkcije koju aktivira klik događaj svojstvo left poprima svakih 10 ms novu vrijednost + 1 px. Za tu vrijednost mijenjamo mu svojstva style.left. Kad dosegne vrijednost 350 px aktivira se metoda clearInterval() i krug se prestane kretati.

Ukoliko bismo željeli da se krug giba vertikalno prema gore mijenjali bismo mu svojstvo top tako što bismo oduzimali vrijednosti varijabli koju bismo definirali za to svojstvo. Ako bismo željeli da se giba istovremeno i gore i desno mijenjali bismo i svojstvo top i svojstvo left.



```html
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=edge">
    <meta name="viewport" content="width=device width, initial scale=1.0">
    <link rel="stylesheet" href="./style.css">
    <title>Animacija</title>
</head>
<body>
    <header>
        <h1>Animacija</h1>
    </header>
    <main>
            <div class="spremnik">
                <div id="krug"></div>
            </div>
    </main>

    <script type="text/javascript">
        document.addEventListener("DOMContentLoaded", function() {
            //animacija
            var krug = document.getElementById("krug");
            krug.addEventListener("click", function() {
                var pozicijaX = 0;
                var intervalId = setInterval(animacija, 10);
                function animacija() {
                    if (pozicijaX == 350) {
                        clearInterval(intervalId);
                    } else {
                        pozicijaX++;
                        krug.style.left = pozicijaX + "px";
                    }
                }
            }),
        });
    </script>
</body>
</html>
```

```css
.spremnik{
    width:400px;
    height:400px;
    background-color:rgb(184, 208, 230);
    position:relative;
    margin-left:400px ;
}

#krug{
    width:50px;
    height:50px;
    background-color:rgb(12, 72, 95);
    border:1px solid rgb(12, 72, 95);
    border-radius: 50%;
    top:349px;
    left:0px;
    position:absolute;
}
```

Slika 4.46. Primjer izvršavanja animacije na klik primjenom setInterval() i clearInterval()

- Važno je uočiti dodavanje vrijednosti mjerne jedinice kako bi preglednik znao na kojoj poziciji iscrtati element, a za to mu je potrebna i **brojčana vrijednost** i **mjerna jedinica**. Mjerna jedinica nije potrebna jedino za vrijednost 0.

Možemo napraviti da se animacija odvija kad, primjerice, kliknemo na neki gumb.

```html
<!DOCTYPE html>
<html>
<head>
    <title>Rad s timerom</title>
    <meta charset="utf-8">
    <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>
    <img src="slika.png" id="slika">
    <script src="javascript.js"></script>
    <center>
    <form>
        <input type="button" value="Pokreni" id="pokreni">
        <input type="button" value="Vrati na početak" id="vrati">
    </form>
    </center>
</body>
</html>
```

```css
body {
    background-image: url("pozadina.jpg");
    background-size: cover;
}
img{
    width: 100px;
    height: 100px;;

}
input {
    font-size: 25px;
    border-radius: 50px;
    border-width: 0px;
    color: #b81c1c;
    background-color: #c8d0e2;
}
```

```javascript
document.addEventListener("DOMContentLoaded", function() {
    var animacija;
    var pokreni=document.getElementById('pokreni');
    var vrati=document.getElementById('vrati');
//inicijalno postavljanje pozicije slike
    slika = document.getElementById('slika');
    slika.style.position = 'relative';
    slika.style.left = '0px';
    slika.style.top = '0px';
    pokreni.addEventListener("click",function(){
        //svakih 50 ms dodaje se 20 px pomaka po x osi i 10 px pomaka po y osi
        slika.style.left = parseInt(slika.style.left) + 20 + 'px';
        slika.style.top = parseInt(slika.style.top) + 10 + 'px';
        animacija = setTimeout(pokreni,50);
    });
    vrati.addEventListener('click',function(){
    //poništavanje vremena i postavljanje na početnu poziciju
    clearTimeout(animacija);
        slika.style.left = '0px';
        slika.style.top = '0px';
    });
});
```

Slika 4.47. Primjer izvršavanja i poništavanja animacije klikom na gumb

4.6. Optimizacija za algoritme pretraživanja sadržaja dokumenta SEO (engl. *Search Engine Optimization*)

Optimizacija mrežne stranice (engl. *search engine optimization* – *SEO*) proces je prilagodbe mrežnih stranica i sadržaja na njima kako bi ih korisnici lako pronašli putem tražilica, odnosno kako bi ih usmjerili na našu stranicu.

Pretraga se izvodi na temelju **ključnih riječi** i fraza.

Tražilice pomoću **programa za indeksiranje**, tzv. ***crawlera*** pronalaze, skeniraju (pretražuju) i indeksiraju stranice na temelju:

- **kvalitete i relevantnosti informacija**
- povezanosti **ključnih riječi i sadržaja** stranice
- povezanosti s **drugim mrežnim stranicama**
- **analize ponašanja korisnika** stranice:
  - zadržavaju li se na stranici neko vrijeme
  - brzo se vraćaju na stranicu pretrage
  - sa stranice prate druge veze
  - zanemaruju vašu stranicu u rezultatima pretrage (ne posjećuju je)
- **brzine učitavanja** stranice
- analize je li **prilagođena prikazu na mobilnim uređajima**
- provjere koliko je sadržaja na stranici **originalno** (a koliko kopija sadržaja koji se već nalaze na drugim stranicama).

SEO je kompleksan proces i treba ga raditi kontinuirano.

SEO optimizaciju možemo podijeliti:

- **optimizaciju na stranici** (engl. *on-site* ili *on-page*) – obuhvaća sve radnje poduzete na poboljšanju sadržaja web-stranice kako bi bila prepoznatljiva *web*-tražilicama
- **optimizaciju izvan stranice** (engl. *off-site*) – radnje poduzete izvan mrežne stranice, koje utječu na njezino rangiranje na *web*-tražilicama.

4.6.1. Tehnike optimizacije za algoritme pretraživanja sadržaja *web*-stranice

Kako bi stranice što bile što bolje rangirane na tražilicama, potrebno ih je optimizirati.

Prvi korak optimizacije **odrediti** je koje **pojmove** korisnici pretražuju, odnosno koje **ključne riječi** želite da vašu mrežnu stranicu rangiraju na tražilicama.

Za odabir ključnih riječi važni su:

- **opseg pretraživanja** – koliko korisnika pretražuje po određenoj ključnoj riječi – što više korisnika pretražuje pomoću nje, veća je vjerojatnost da ćete privući korisnike na svoju stranicu
- **relevantnost** – pojam može biti često korišten na tražilicama, ali to ne znači da je taj pojam važan za vašu stranicu, odnosno da je povezan s vašim sadržajima (što je ključno za rangiranje)
- **konkurentnost** – ako ključna riječ koju koristite ima puno pretraga, to znači da se teže visoko pozicionirati zbog velike konkurencije.

Zato treba razmisliti tko je ciljana publika stranice, što su zainteresirani pronaći, kakve probleme imaju, kakvim jezikom komuniciraju te tko nam je konkurencija.

Nakon toga moguće je kreirati inicijalan popis ključnih riječi i sjedišta koji mogu pomoći pri pronalasku dodatnih informacija i ključnih riječi.

Postoje besplatne **online aplikacije** koje nam mogu pomoći pronaći koliko se često neki pojam pojavljuje u pretragama te nam nude preuzimanje popisa ključnih riječi povezanih s tom pretragom.

Ako već imamo mrežnu stranicu, možemo analizirati pretragom po konkretnim ključnim riječima kako će je tražilice rangirati.

Na temelju ovih istraživanja možemo odrediti koje ključne riječi postižu najbolje rangiranje i nude nabolje prilike za dobro pozicioniranje te koliko su mrežne stranice konkurencije koje ih sadrže kvalitetne, pouzdane i relevantne.

4.6.2. Tehnike optimizacije za algoritme pretraživanja koje se primjenjuju na samoj *web*-stranici (engl. *on-page* ili *on-site*)

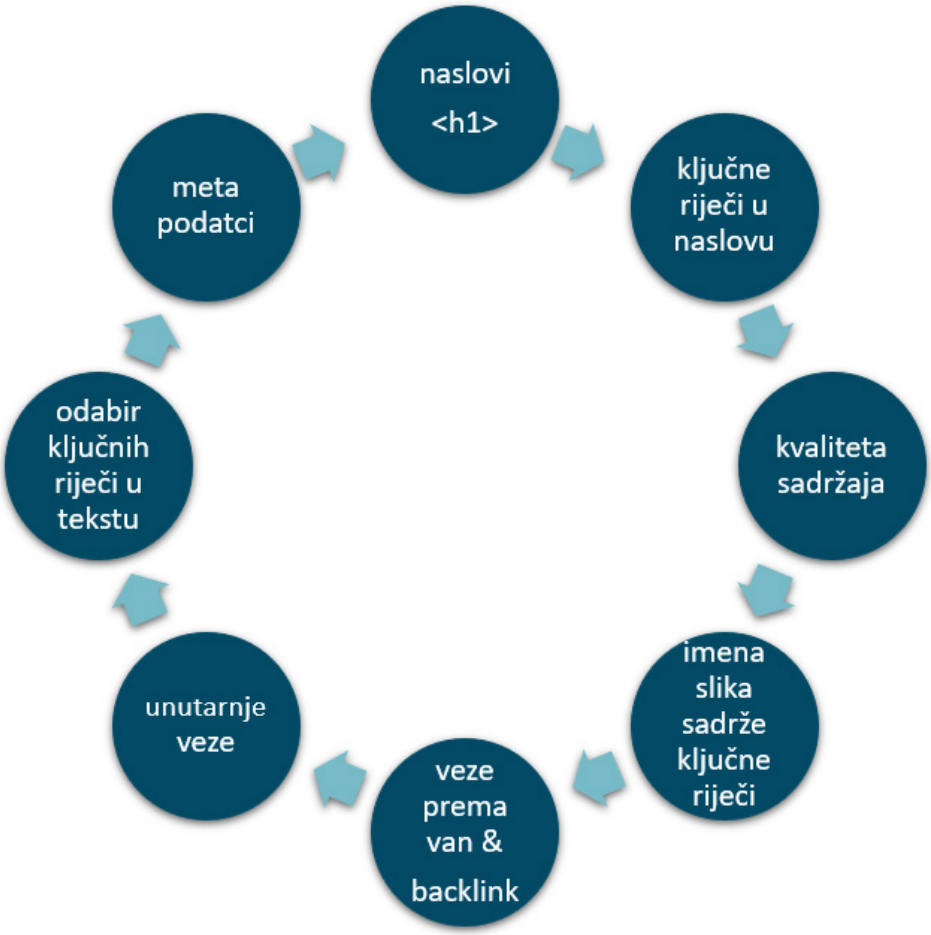
Cilj *on-page* optimizacije osigurati je da tražilice razumiju tematiku stranice i ključne riječi te da ih mogu povezati s relevantnim pretragama.

Osnovni čimbenici za „privlačenje" tražilica:

- **Optimizacija ključnih riječi** – potrebno je krenuti od popisa ključnih riječi koje treba implementirati u sadržaje stranice. Svaku stranicu treba povezati s ključnim pojmom i s njim povezanim pojmovima.
- Važno je u **naslov** stranice pametno sročiti i u njega ugraditi ključnu riječ. Optimalno je koristiti 50 do 60 znakova za naslov, a ključnu riječ treba postaviti na početak ili što bliže njemu.
- **Naslov** postaviti unutar **<h1>** oznake.
- Ključna riječ treba se nalaziti u **prvih 100 riječi prvog paragrafa** na stranici.
- Oblikovati naslov metaoznake **<title>** tako da sadrži ključnu riječ i da je povezan sa sadržajima na stranici.
- U metaoznaku **<description>** postaviti kratki opis.
- **Slikama** na stranici postaviti imena koja ih vežu uz sadržaj i sadrže ključnu riječ te isto dodati u opisni atribut alt, optimizirati veličinu slikovnih sadržaja.
- **Povezati stranicu** s drugim mrežnim sjedištima poveznicama (linkovima), posebice s društvenim mrežama i sličnim stranicama.
- Kreirati **unutarnje veze** na stanici, odnosno povezati sadržaje r azličitih stranica sjedišta. Poveznice koji vode na naslovnicu , poveznice za kretanje po kategorijama i potkategorijama trebaju biti dobro povezane, jednostavne i jednoznačno definirane jer se tako omogućuje pretraživačima da prepoznaju više sadržaja unutar jednog web-sjedišta.
- Veća količina sadržaja na stranici postiže bolje pozicioniranje na tražilicama.
- Ako stranica sadrži više multimedijskih sadržaja (video, dijagrami, zvuk), dulje će zadržati pažnju korisnika.

- Osigurati kvalitetan, jedinstven, relevantan i aktualan sadržaj.
- Dobro postavili izgled (engl. *layout*) stranice i naglasiti bitne sadržaje.
- **Backlink** podrazumijeva da postoje poveznice s drugih stranica na našu, što je teško postići ukoliko stranica nije visoko pozicionirana na tražilicama ili nema kvalitetan sadržaj.

Kako bi postigli bolje rezultate na tražilicama, neki posežu za različitim manipulacijama, od kojih je jedna **Black hat SEO**, koji se temelji na manipulaciji algoritmima koje koriste tražilice, pretjerivanju s ključnim riječima, skrivanju ključnih riječi u kodu, što korisnici stranice ne vide, ali tražilice vide, kupovini poveznica i sl.



Slika 4.48. Tehnike optimizacije na stranici

4.6.3. Tehnike optimizacije za algoritme pretraživanja koje se primjenjuju izvan *web*-stranice (engl. *off-site* ili *off-page*)

Odnosi se na aktivnosti izvan same stranice kako bi se postigli bolje rezultate na tražilicama, prvenstveno osiguravanjem poveznica na našu stranicu s drugih stranica i društvenih mreža.

Što je više poveznica s pouzdanih, visokorangiranih stranica prema vašoj stranici i sadržaju na njoj, to bolje. Treba biti oprezan pri biranju mrežnih stranica koje mogu podijeliti naše mrežne stranice, odnosno stranicu koju optimiziramo, jer njihova kvaliteta utječe na odluku tražilice o tome kako će pozicionirati povezane mrežne stranice.

4.6.4. Optimizacija tehničkih standarda *web*-stranice za algoritme pretraživanja

U tehnički SEO spada rad na tehničkim uvjetima koji utječu na brzinu učitavanja mrežne stranice, kao što su neispravne poveznice, pogreške 404, 403 i 500, ispravak grešaka servera, prilagodba na responzivni način rada, provjera HTTPS-a, otkrivanje i popravak grešaka indeksiranja, otkrivanje umnoženih metapodataka, umnoženog sadržaja, optimizacija slika i slično.

Slika 4.49. Koraci u optimizaciji mrežnog sjedišta

Pitanja za ponavljanje

1. Objasnite koje mogućnosti upotreba JavaScripta donosi na mrežnim stranicama.
2. Na koje načine možemo ugraditi skripte JavaScripta u HTML?
3. Gdje se izvode operacije JavaScripta?
4. Koji standard definira JavaScript?
5. Može li se JavaScript primijeniti i na strani poslužitelja?
6. Na koje sve načine JavaScript može prikazati podatke?
7. Čemu primarno služi metoda console.log()?
8. Objasnite što su varijable i na koji ih način imenujemo u JavaScriptu.
9. Usporedite primjenu ključnih riječi var, let i const u odnosu na doseg (scope) i promjenu vrijednosti.
10. Nabrojite vrste tipova podataka u JavaScriptu.
11. Objasnite što podrazumijevaju vrijednost null, undefined i NaN.
12. Nabrojite vrste pop up prozora.
13. Kako se u JavaScriptu mogu izvršavati pretvorbe tipova podataka?
14. Nabrojite nekoliko metoda za pretvorbu.
15. Objasnite razliku između unarnih i binarnih operatora.

16. Nabrojite nekoliko metoda za rad s tekstom.
17. Objasnite način rada uvjetnog grananja if else if else na općenitom primjeru.
18. Što nam omogućuje primjena `` (tzv. backtick)?
19. Kad primjenjujemo programsku petlju for, a kad while i do while?
20. Koja je funkcija naredbi continue i break?
21. U kojem će se slučaju pojaviti beskonačna petlja?
22. Objasnite zašto metoda sort() nije prikladna za sortiranje brojčanih nizova?
23. Koje sve tipove podataka možemo pohraniti u niz u JavaScriptu?
24. Nabrojite metode iteracija nad nizovima.
25. Objasnite ulogu funkcija u JavaScriptu.
26. Koja je razlika između parametara i argumenata funkcije?
27. Objasnite ulogu ključne riječi return.
28. Objasnite pojam samopozivajuće funkcije.
29. Što je u JavaScriptu objekt?
30. Objasnite na koji način možemo pristupiti pojedinim svojstvima objekta.
31. Objasnite ulogu ključne riječi this kod objekata.

32. Usporedite pojmove DOM i BOM.

33. Nabrojite načine za selektiranje elemenata iz DOM stabla?

34. Nabrojite nekoliko događa nad HTML elementima.

35. Nabrojite nekoliko metoda za dohvaćanje HTML elemenata u JavaScript.

36. Nabrojite nekoliko metoda za izmjenu, dodavanje i uklanjanje HTML elemenata.

37. Objasnite primjenu metoda setInterval() i clearInterval().

38. Kako ostvarujemo animaciju u JavaScriptu?

39. Što podrazumijeva pojam optimizacije mrežne stranice?

40. Nabrojite nekoliko tehnika za optimizaciju mrežne stranice.

Literatura:

1. **Learn JavaScript Programming**
   Preuzeto 1.5.2022. s https://www.programiz.com/javascript
2. **W3 schools JavaScript tutorial**
   Preuzeto 3.5.2022. s https://www.w3schools.com/js/default.asp
3. **MDN Javascript tutorial.**
   Preuzeto 5.5.2022. s https://developer.mozilla.org/en-US/docs/Web/JavaScript
4. **GeeksforGeeks JavaScript tutorial.**
   Preuzeto 15.5.2022. s https://www.geeksforgeeks.org/javascript/?ref=ghm
5. **SEO Optimization – Learn to Optimize for SEO.**
   Preuzeto 20.5.2022. s https://www.wordstream.com/seo
6. **What is SEO? Your Complete Step-By-Step Guide**
   Preuzeto 1.6.2022. s https://neilpatel.com/what-is-seo/

5. Stilski jezik za vizualno oblikovanje *web*-stranice – SASS

U OVOM POGLAVLJU NAUČIT ĆETE:

- Što je SASS?
- Koje su mu komparativne prednosti u odnosu na CSS?
- Kako koristiti SASS za lakše pisanje liste stilova?

5.1. Uvod u stilski jezik za vizualno oblikovanje SASS

5.1.1. Svojstva stilskog jezika za vizualno oblikovanje web-stranice SASS koji se prevodi u CSS

SASS (engl. *Syntactically Awesome Style Sheet*) besplatno je proširenje CSS-a koje su osmislili 2006. Hampton Catlin i Natalie Wizenbaum kako bi se pojednostavilo pisanje CSS-a. Ima vlastitu sintaksu, a kompajlira se u CSS listu stilova. **Kompatibilan** je sa svim verzijama CSS-a, a omogućuje nam **lakše i brže pisanje**

Slika 5.1.
Logotip SASS-a

liste stilova. CSS liste stilova mogu biti jako dugačke, složene i teške za održavanje jer se niz CSS pravila zbog načina selektiranja sadržaja često ponavlja.

Dodatna svojstva i mogućnosti SASS-a jesu:

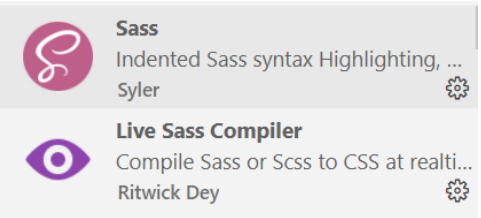
- rad s varijablama
- rad s logičkim operacijama
- ugnježđivanje pravila,
- nasljeđivanje pravila
- primjena funkcija i sl.

Prednosti upotrebe SASS-a:

- kreiramo čišći i jednostavniji kod
- automatiziramo zadatke koji se ponavljaju
- smanjujemo broj pogrešaka
- kreiramo dijelove koda koji se mogu višekratno upotrijebiti i
- osiguravamo kompatibilnost sa starijim verzijama preglednika.

5.1.2. Instalacija SASS-a

Da bismo mogli zapisivati kod u SASS-u i kompajlirati ga u CSS, trebamo ga instalirati na računalo ili ugraditi proširenje za SASS u VS Code. U VS Codeu obavezno dodajemo **Live Sass Compiler**, a možemo dodati i druga SASS proširenja.



Slika 5.2. Proširenja za rad sa SASS-om u VS Codeu

- Instalaciju za SASS možete preuzeti s: https://sass-lang.com/install.

Nakon dodavanja proširenja u VS Code u mapu projekta možemo dodati dokument ekstenzije .scss. Najbolje ga je nazvati ***style.scss*** jer će se tada kompajlirati u ***style.css***. Kad stvorimo datoteku, kliknemo na **Watch Sass**, koji nam pri svakoj promjeni i pohrani kompajlira **.scss** u **.css**. Stranicu možemo pokrenuti pomoću Live Servera klikom na **Go Live**.

Slika 5.3. Pokretanje automatskog generiranja css datoteka i otvaranje HTML stranice

Osim ***style.css*** generira se i ***style.map.css***, koja pregledniku omogućuje prepoznavanje izvorne datoteke.

Za upotrebu kompajlirane style.css liste stilova, ako se datoteka style.css nalazi u mapi css, potrebno je u HTML-u dodati poveznicu:

```
<link rel="stylesheet" href=../css/style.css ">
```

- Važno je napomenuti da **nikada ne mijenjamo kod u .css** datoteci nego isključivo u .scss jer se svakim kompajliranjem .scssa ponovno „prepiše“ css dokument.

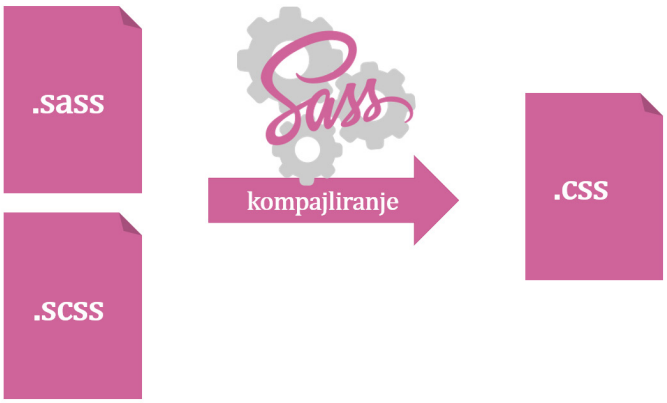
5.2. Osnove stilskog jezika SASS

5.2.1. Sintaksa stilskog jezika SASS

Razlikujemo dvije sintakse jezika SASS:

- **.sass** – starija verzija u kojoj se ne koriste **{ }** niti **;** za završetak pravila – parova svojstvo: vrijednost
- **.scss** – sličniji standardom CSS-u, koristi selektore za koje se piše niz pravila svojstvo vrijednost unutar **{ }**

Kodovi zapisani u SASS-u kompajliraju se u standardni CSS.



Slika 5.4. Povezanost SASS datoteka s CSS-om

5.2.2. Varijable stilskog jezika SASS

Varijable nam omogućuju pohranu različitih vrijednosti (pravila) za kasnije korištenje, čime izbjegavamo ponavljanje istog koda na više mjesta. U varijable možemo pohraniti primjerice boje, fontove i drugo.

Za **kreiranje varijable** koristimo oznaku **$** pa **ime**: i onda **vrijednost**, primjerice:

```
$primary-color: #CF649A;
```

Varijable se u CSS ne kompajliraju kao varijable, nego kao **vrijednosti** u pridruženo pravilo.

style.scss

```
$primary-color: #CF649A;
$primary_font:sans-serif,
'cursive';
body {
    background-color: $primary-color;
    font-family: $primary_font;
}
```

kompajliranje

style.css

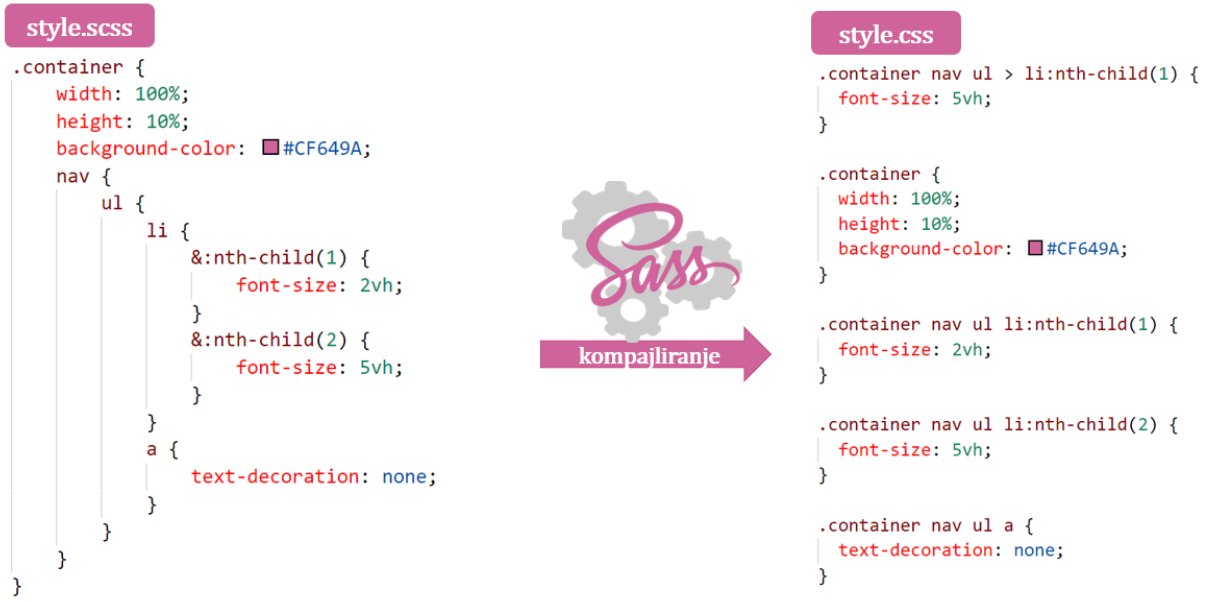
```
body {
  background-color: #CF649A;
  font-family: sans-serif, "cursive";
}
```

Slika 5.5. Usporedni prikaz koda u .scssu i koda kompajliranog u css

5.2.3. Ugnježđivanje

U HTML-u su elementi imali jasnu hijerarhijsku strukturu u kojoj se točno znalo koji se element postavlja unutar drugog elementa te je ta hijerarhija bila i jasno vizualno uočljiva, dok u CSS takve hijerarhije nije bilo.

SASS nam omogućuje ugnježđivanje elemenata kako smo ih ugnježđivali i u HTML-u i jasno vizualno prikazuje hijerarhiju. Međutim kompajliranjem u .css vizualna hijerarhija gubi se, ali logika ugnježđivanja elemenata i dalje je jasno vidljiva.



Slika 5.6. Usporedni prikaz koda u .scssu primjenom svojstva ugnježđivanja elemenata i koda kompajliranog u css

Možemo uočiti da se ugniježđeni elementi u CSS kompajliraju u obliku roditelj – potomci (***selector descendant***), a možemo koristiti i selektor **&** koji **zamjenjuje prethodno ugniježđeni selektor** (roditelja).

5.3. Uključivanje vanjskih datoteka

Pojedine dijelove koda možemo izdvojiti u zasebne datoteke pa ih po potrebi pozvati u druge datoteke, što nam omogućuje **modularno** održavanje koda. Takav dokument

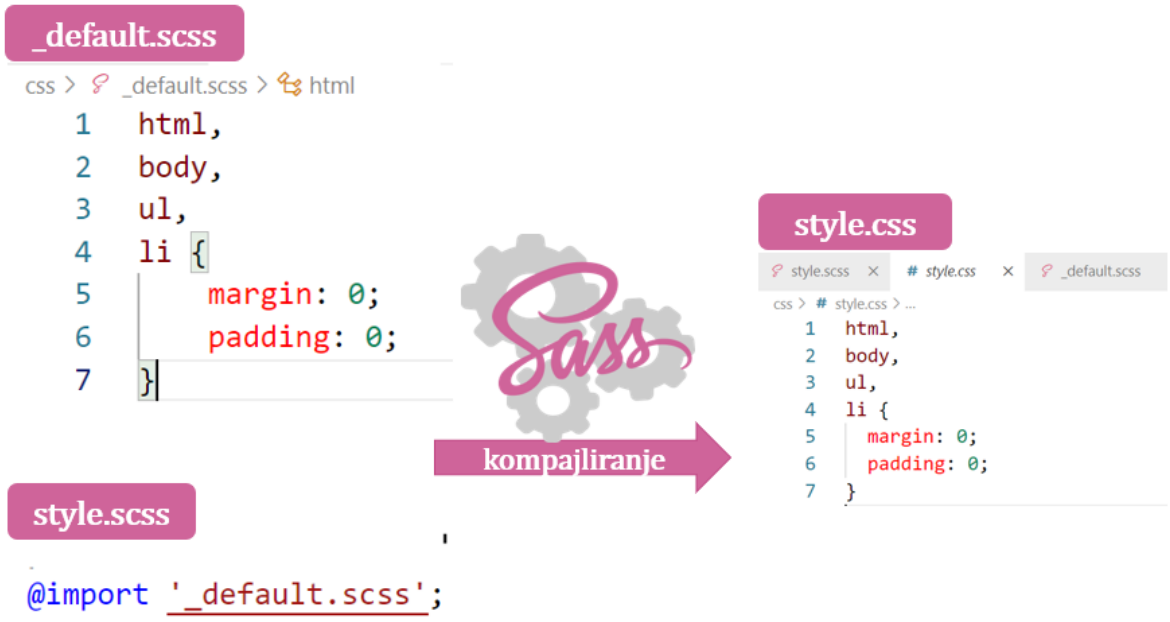
nazivamo **djelomični scss** ili **partial**.

Pri imenovanju takvih datoteka koristimo ispred imena **prefiks** _ (primjerice _default.scss).

Za uvoz u drugu datoteku koristimo ključnu riječ **@import.**

```
@import '_default.scss';
```

Nije nužno dodati _ ni ekstenziju pri pozivu imena.



Slika 5.7. Primjer upotrebe djelomičnog SASS-a

- Prisjetimo se da smo ključnu riječ **@import** već koristili, primjerice, kod uvoza fontova u CSS.

5.4. Pravila i ključne riječi stilskog jezika SASS

5.4.1. Uloga i primjena mixina

Upotreba **mixina** omogućuje nam kreiranje skupova CSS deklaracija koje kasnije uključujemo u pravila unutar određenih selektora. Mixine definiramo u obliku:

```
@mixin ime {
svojstvo: vrijednost;
}
```

style.scss

```
@mixin flexSredina() {
    display: flex;
    justify-content: center;
    align-items: center;
}

p {
    @include flexSredina();
}
```

kompajliranje

style.css

```
p {
    display: -webkit-box;
    display: flex;
    -webkit-box-pack: center;
    -ms-flex-pack: center;
    justify-content: center;
    -webkit-box-align: center;
    -ms-flex-align: center;
    align-items: center;
}
```

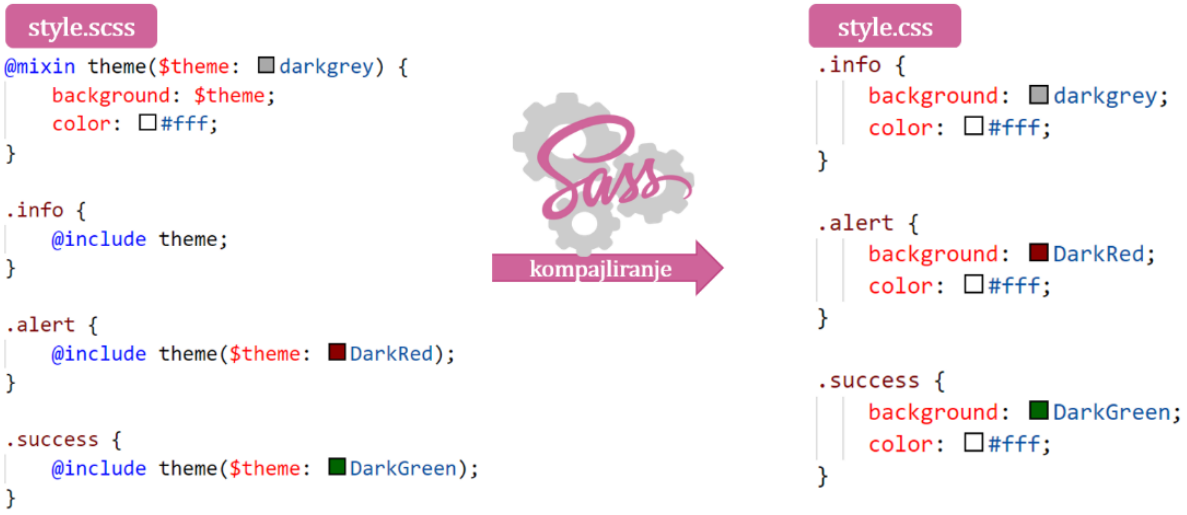
Slika 5.8. Primjer upotrebe mixina

U mixine možemo prosljeđivati **argumente** ili ranije definirane **varijable**, koji se tada pišu unutar zagrada iza imena.

```
@mixin ime(lista argumenata ili
varijabli) { svojstvo: vrijednost;
}
```

5.4.2. Uloga i primjena ključne riječi @include

Da bismo svojstva definirana u mixinu dodijelili elementima stranice, dodajemo ih sektorima primjenom ključne riječi **@include**.



Slika 5.9. Primjer prosljeđivanja argumenata mixinu

- Mixine možemo koristiti za odabir tamne ili svijetle teme sučelja ili za promjenu rasporeda na stranici u ovisnosti o veličini zaslona uređaja.

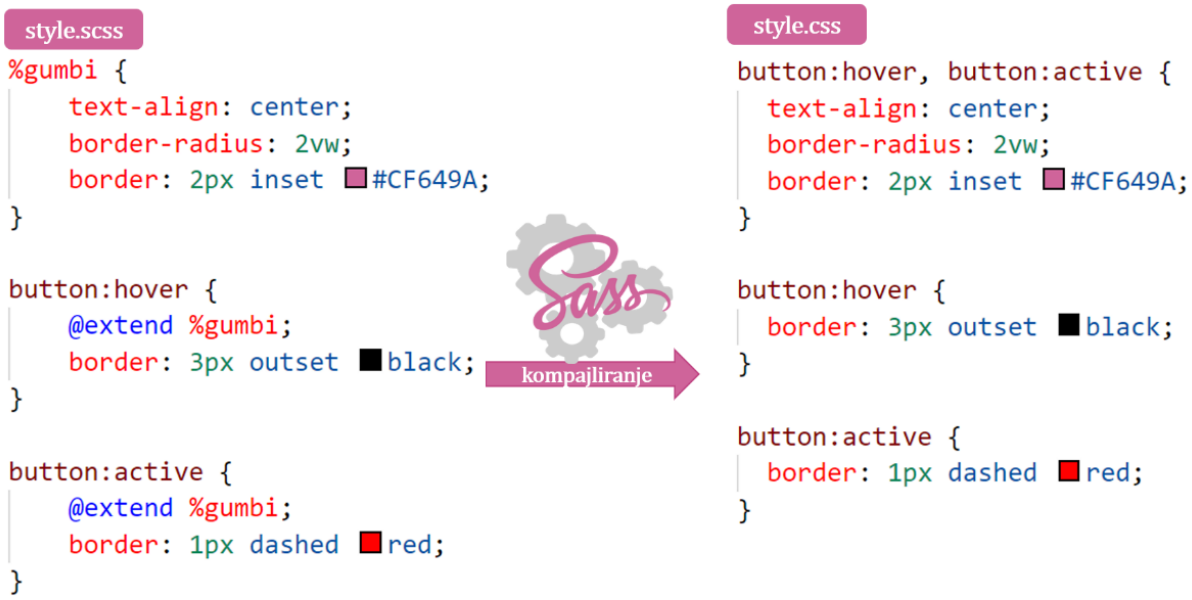
5.4.3. Proširivanje pravila – uloga i primjena ključne riječi @extend

Element možemo proširiti tako da se na njega primjenjuju stilovi elementa na koji smo ga proširili. Radi se o svojstvu sličnom mixinu, no ovdje imamo slično stilizirane elemente koji se razlikuju u detaljima.

Element preuzima svojstva elementa na koji smo ga proširili, a možemo mu dodati i vlastita svojstva. U tu svrhu možemo zapisati skup pravila koja će biti zajednička u više elemenata. Selektoru dodajemo % ispred:

```
%ime{
svojstvo:
vrijednost;
}
```

Zatim dodajemo taj skup pravila željenom selektoru pomoću ključne riječi **@extend**.



Slika 5.10. Primjer proširivanja pravila

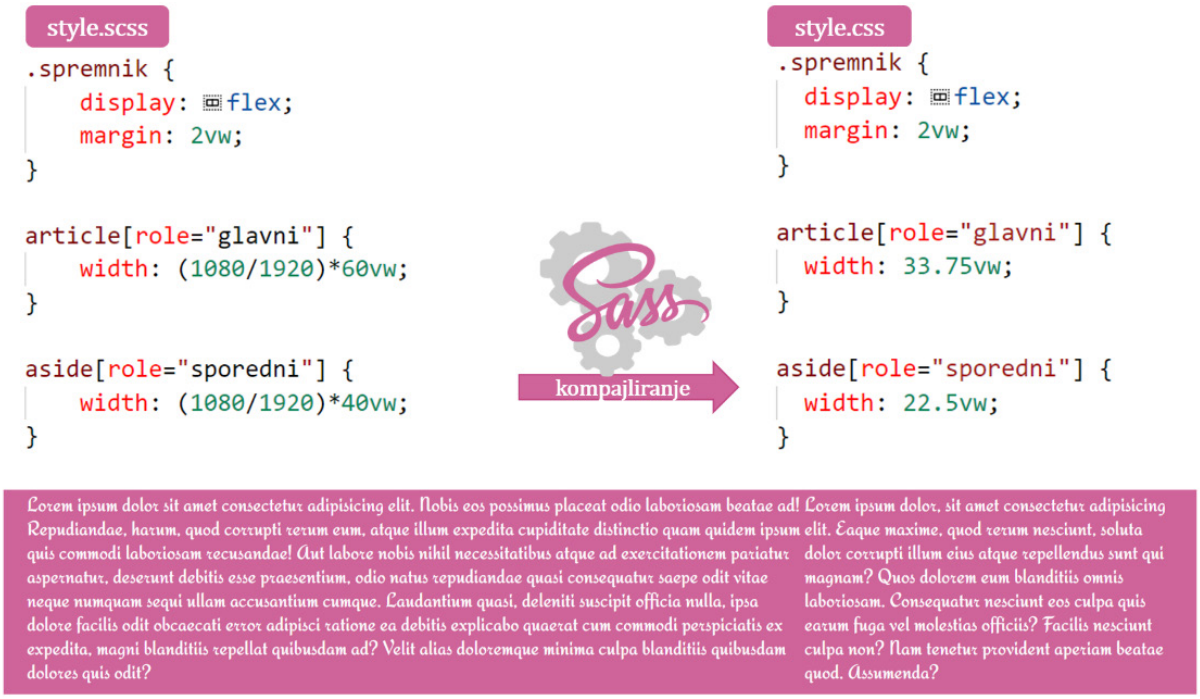
5.5. Matematičke i logičke operacije i kontrolne strukture

5.5.1. Matematičke operacije u SASS-u

SASS nam omogućuje primjenu matematičkih operacija na vrijednostima. Možemo koristiti primjerice: **+**, **-**, *****, **/** i **%**.

To nam omogućuje **izračun vrijednosti** pojedinih **svojstava** elemenata na stranici.

- Važno je napomenuti da prilikom izračuna **ne smijemo miješati** različite mjerne jedinice!



lika 5.11. Primjer upotrebe matematičkih operatora za izračun širine elemenata stranice

5.5.3. Numeričke funkcije u SASS-u

Numeričke funkcije koristimo da bismo manipulirali brojčanim vrijednostima.

Funkcija	Opis funkcije
abs(n)	Vraća apsolutnu vrijednost broja.
ceil(n)	Zaokružuje broj na prvu veću cjelobrojnu vrijednost.
floor(n)	Zaokružuje broj na prvu manju cjelobrojnu vrijednost.
max(n1,n2,...)	Vraća najveću vrijednost unesenog niza brojeva.
min(n1, n2,...)	Vraća najmanju vrijednost unesenog niza brojeva.
random()	Vraća nasumičan broj između 0 i 1.
random(n)	Vraća nasumičan cijeli broj između 1 i n.
round(n)	Zaokružuje broj na najbliži cijeli broj.

Tablica 5.1. Primjeri numeričkih funkcija u SASS-u

5.5.4. Kontrolne strukture i logičke operacije u SASS-u

Kako bismo mogli primijeniti određene stilove ili grupe stilova na pojedine elemente ovisno o njihovu trenutnom stanju, pomoći nam mogu **@if**, **@else** i **logičke operacije**.

@if omogućuje nam provjeru točnosti nekog izraza. Ako je testirani uvjet zadovoljen, pišemo skup pravila koja će se primijeniti u tom slučaju, a ako nije, možemo pisati **@else** u kojem slijedi skup pravila koja se izvršavaju u tom slučaju ili možemo zadržati trenutna svojstva bez primjene @else.

```
@if <provjera uvjeta>
{
svojstvo:
vrijednost;
}@else{
svojstvo:
vrijednost;
}
```

style.scss

```scss
$light-background: #CF649A;
$light-text: white;
$dark-background: #640635;
$dark-text: #edd9f048;
@mixin theme-colors($light-theme: true) {
    @if $light-theme {
        background-color: $dark-background;
        color: $dark-text;
    }
    @else {
        background-color: $light-background;
        color: $light-text;
    }
}

.zaglavlje {
    @include theme-colors($light-theme: true);
    body.dark & {
        @include theme-colors($light-theme: false);
    }
}
```

style.css

```css
.zaglavlje {
  background-color: #640635;
  color: #edd9f048;
}

body.dark .zaglavlje {
  background-color: #CF649A;
  color: white;
}
```

Slika 5.12. Primjer primjene kontrolne strukture pomoću @if i @else

Strukturu ponavljanja koristimo kad pravila želimo **primijeniti na više elemenata** nekog skupa elemenata, primjerice **<li>** unutar **<ul>** ili **<ol>**. Razlikujemo ključne riječi **@for** i **@while.**

```
@for <varijabla> from <početna vrijednost> to/through <krajnja vrijednost> {
svojstvo: vrijednost;
}
```

Ukoliko primjenjujemo ključnu riječ **to**, nije uključena krajnja vrijednost, dok je kod ključne riječi **through** ona uključena.

style.scss

```scss
$base-color: #010f14;
@for $i from 1 through 3 {
    li:nth-child(n + #{$i}) {
        background-color: lighten($base-color, $i * 20%);
    }
}
```

kompajliranje

style.css

```css
li:nth-child(n + 1) {
  background-color: #065875;
}

li:nth-child(n + 2) {
  background-color: #0ba1d6;
}

li:nth-child(n + 3) {
  background-color: #51cbf6;
}
```

Slika 5.13. Primjer primjene strukture ponavljanja

style.scss

```scss
$sizes: 40px,
50px,
80px;
@each $size in $sizes {
    .icon-#{$size} {
        font-size: $size;
        height: $size;
        width: $size;
    }
}

$icons: ("eye": "\f112", "start": "\f12e", "stop": "\f12f");
@each $name,
$glyph in $icons {
    .icon-#{$name}:before {
        display: inline-block;
        font-family: "Icon Font";
    }
}
```

kompajliranje

style.css

```css
.icon-40px {
  font-size: 40px;
  height: 40px;
  width: 40px;
}

.icon-50px {
  font-size: 50px;
  height: 50px;
  width: 50px;
}

.icon-80px {
  font-size: 80px;
  height: 80px;
  width: 80px;
}

.icon-eye:before {
  display: inline-block;
  font-family: "Icon Font";
}

.icon-start:before {
  display: inline-block;
  font-family: "Icon Font";
}

.icon-stop:before {
  display: inline-block;
  font-family: "Icon Font";
}
```

Slika 5.14. Primjer primjene @each na listi i mapi

Pitanja za ponavljanje

1. Koje dvije sintakse jezika SASS poznajemo? Koju više koristimo i zašto?
2. Koje su komparativne prednosti korištenja sintakse SASS za izradu liste stilova u odnosu na CSS?
3. Kako možemo kreirati varijablu u SASS-u?
4. Objasni pojam djelomični SASS (*partial*) i na koji ga način uključujemo u drugu .scss datoteku.
5. Što nam omogućuje upotreba mixina i na koji ih način uključujemo unutar odabranih selektora?
6. Kako možemo primijeniti već napisan skup stilova proširivanjem pravila? Pojasni ulogu *@extend.*
7. Kako nam matematičke operacije pomažu u SASS-u?
8. Nabroji i objasni nekoliko numeričkih funkcija.
9. Nabroji i objasni nekoliko funkcija za rad sa znakovnim nizovima.
10. Koju ključnu riječ koristimo za provjeru točnosti uvjeta?
11. Po čemu se razlikuju to i through u strukturi petlja *@for*?
12. Koja je razlika između *@for* i *@each*?

Literatura:

1. **Službena stranica Sass**
   Preuzeto 12.2.2022. s https://sass-lang.com/guide
2. **Sass Tutorial for Beginners - CSS With Superpowers, freeCodeCamp.org.**
   Preuzeto 12.2.2022. s: https://www.youtube.com/watch?v=_a5j7KoflTs&t=4458s
3. **Službena stranica W3 Schools: Tutorijal Sass.**
   Preuzeto 13.2.2022. s: https://www.w3schools.com/sass/default.php

6. Razvojni okvir Bootstrap

U OVOM POGLAVLJU NAUČIT ĆETE:

- Što je Bootstrap?
- Kako ugraditi i primijeniti pojedine elemente Bootstrapa u vlastite mrežne stranice?
- Kako izraditi raspored elemenata i oblikovati sadržaje primjenom Bootstrapa?

6.1. Što je Bootstrap i kako ga primijeniti

6.1.1. Primjena Bootstrapa u izradi prilagodljivih *web*-sjedišta

Slika 6.1. Logotip Bootstrapa

Bootstrap je popularan frontend razvojni okvir, koji su razvili Mark Otto i Jacob Thornton za potrebe Twittera te javno objavili kao open source 2011. na GitHubu. Bazira se na HTML-u, CSS-u i JavaScriptu.

Osnovna svojstva:

- potpuno **besplatan** za preuzimanje i upotrebu (engl. *open source*)
- podržavaju ga svi suvremeni preglednici
- omogućuje brzu i jednostavnu izradu *web*-sjedišta – **štedi vrijeme**
- sadrži mnoštvo **dosljedno vizualno stiliziranih** komponenti za ugradnju u *web*-sjedišta
- potpuno prilagođen za različite veličine zaslona (**responzivan**)
- baziran je na pristupu izrade – prvo za male zaslone (engl. **mobile first**)
- komponente je lako prilagoditi vlastitim potrebama
- moguće ga je kombinirati s drugim bibliotekama.

Izvor: Vecteezy.com

Slika 6.2. Bootstrap je potpuno prilagođen za različite veličine zaslona (responzivan)

- Prije početka rada možemo sa službenog **Bootstrap** *web*-sjedišta preuzeti datoteke s kompajliranim CSS-om i JS-om https://getbootstrap.com/docs/5.1/getting-started/download/ kako bi nam pri pisanju koda VS Code nudio opcije završetka sintakse klasa i sl.

Potrebno je raspakirati dokument koji ste preuzeli te postaviti mape **css** i **js** u mapu projekta.

Druga je opcija koristiti Bootstrap preko CDN-a (engl. *Content Delivery Network*) bez preuzimanja datoteka i postavljanja u mape projekta. Na taj način smanjuje se vrijeme učitavanja stranice jer se CDN serveri nalaze na različitim lokacijama širom svijeta i šalju s najbližeg servera.

- Preuzimamo početni predložak (engl. starter template) sa stranice: https://getbootstrap.com/docs/5.1/getting-started/introduction/

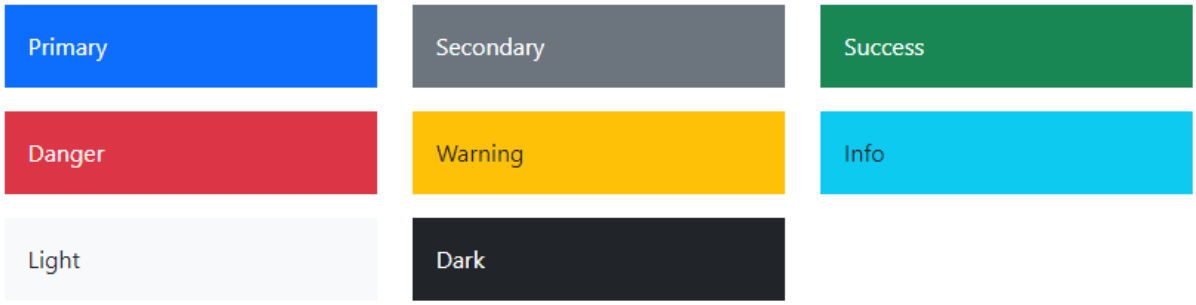
```
Starter template > index.html > html
<!doctype html>
<html lang="en">

<head>
    <!-- Required meta tags -->
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">

    <!-- Bootstrap CSS -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/css/bootstrap.min.css" rel="stylesheet"
    integrity="sha384-1BmE4kWBq78iYhFldvKuhfTAU6auU8tT94WrHftjDbrCEXSU1oBoqyl2QvZ6jIW3" crossorigin="anonymous">

    <title>Hello, world!</title>
</head>

<body>
    <h1>Hello, world!</h1>

    <!-- Optional JavaScript; choose one of the two! -->

    <!-- Option 1: Bootstrap Bundle with Popper -->
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.bundle.min.js" integrity="sha384-ka7Sk0Gln4gmtz2MlQnikT1wXgYsOg+OMhuP+IlRH9sENBO0LRn5q+8nbTov4+1p'
    crossorigin="anonymous"></script>

    <!-- Option 2: Separate Popper and Bootstrap JS -->
    <!--
    <script src="https://cdn.jsdelivr.net/npm/@popperjs/core@2.10.2/dist/umd/popper.min.js" integrity="sha384-7+zCNj/IqJ95wo16oMtfsKbZ9ccEh31eOz1HGyDuCQ6wgnyJNSYdrPa03rtR1zdB"
    crossorigin="anonymous"></script>
    <script src="https://cdn.jsdelivr.net/npm/bootstrap@5.1.3/dist/js/bootstrap.min.js" integrity="sha384-QJHtvGhmr9XOIpI6YVutG+2QOK9T+ZnN4kzFN1RtK3zEFEIsxhlmWl5/YESvpZ13"
    crossorigin="anonymous"></script>
    -->
</body>

</html>
```

lika 6.3. Izgled početnog predloška

Na slici možemo uočiti povezivanje s datotekama i skriptama Bootstrapa i Poppera.

- Ako je korisnik već preuzimao datoteke Bootstrapa s istog CDN-a, one će se koristiti iz memorije preglednika, što **ubrzava otvaranje stranice.**
- Pri izradi HTML dokumenta uvijek postavite viewport metaoznaku kako biste omogućili povećanje sadržaja zaslona osjetljivih na dodir i osigurali i spravan prikaz na mobilnim uređajima.

Osnovne tematske boje Bootstrapa jesu:



Slika 6.4. Osnovna paleta boja u Bootstrapu

Dodatne boje dostupne su u vidu SASS varijabli primjerice:

`$purple-500`

6.1.2. Načini primjene i upotreba klasa i spremnika

- Bootstrap komponente strukturirane su kao niz **spremnika s klasama (class)** koje su oblikovane CSS-om ili JavaScriptom.

Spremnici su temeljni element komponenti Bootstrapa, a koriste se za osiguravanje prostora oko sadržaja i za njegovo lakše pozicioniranje.

Razlikujemo tri različite klase spremnika:

- `.container`, koji primjenjuje maksimalnu širinu **max-width** na svim točkama prekida
- `.container-fluid`, koji primjenjuje širinu **width: 100%** na svim točkama prekida
- `.container-{breakpoint}`, koji primjenjuje širinu **width: 100%** do navedene točke prekida.

Ako pri izradi responzivnog spremnika fiksne širine koristimo klasu `.container`, širina spremnika mijenjat će se na različitim točkama prekida ili veličinama zaslona. Ako koristimo klasu `.container-fluid`, širina spremnika uvijek će biti 100 % bez obzira na točke prekida ili veličinu zaslona.

U početno zadanim postavkama spremnik nema boju pozadine ili okvir, no možemo ih jednostavno oblikovati dodavanjem klasa za boju teksta, pozadine, okvira i sl.

- Više o Bootstrap helper klasama možete pronaći na: https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/bootstrap-helper-classes.php

6.2. Ugradnja i oblikovanje gumba i navigacijske trake

6.2.1. Ugradnja i oblikovanje gumba na stranici

Gumbi su važan dio *web*-stranice. Koriste se, primjerice, za slanje ili poništavanje podataka iz obrazaca, prikazivanje ili skrivanje sadržaja, preusmjeravanje korisnika na drugu stranicu i sl.

Bootstrap omogućuje brz i jednostavan način za stvaranje i prilagodbu gumba. Želimo li na stranicu ugraditi gumbe, možemo kreirati gumbe različitih oblikovanja:

- osnovni gumb
- gumb s određenom bojom pozadine
- uokvireni gumb kojemu se boja pozadine prikazuje kad mišem prijeđemo preko njega i sl.

```
<button type="button" class="btn">Obični gumb</button>
<button type="button" class="btn btn-btn btn-success">Gumb s obojenom pozadinom </button>
<button type="button" class="btn btn-outline-info">Gumb s okvirom</button>
<button type="button" class="btn btn-secondary" disabled>Onemogućeno klikanje</button>
<button type="button" class="btn btn-warning btn-lg">Veliki gumb</button>
<button type="button" class="btn btn-light btn-sm">Mali gumb</button>
<button type="button" class="btn-close" aria-label="Close"></button>
```

Slika 6.5. Ugradnja gumba

Unutar oznake **<button>** dodajemo klase **btn**, **btn-**, **btn-outline**- i sl. Možemo izraditi gumbe na koje nije moguće kliknuti **disabled**, velike **btn-lg** ili male gumbe **btn-sm** te gumbe za zatvaranje **btn-close** i sl.

Više gumba možemo grupirati upotrebom klase **class="btn-group"**

```
<div class="m-4">
    <div class="btn-group">
        <button type="button" class="btn btn-success">
        <i class="bi-eye"></i> Pogledaj
    </button>
        <button type="button" class="btn btn-warning">
        <i class="bi-pencil"></i> Uredi
    </button>
        <button type="button" class="btn btn-danger">
        <i class="bi-trash"></i> Obriši
    </button>
    </div>
</div>
```

Slika 6.6. Ugradnja grupe gumba

6.2.2. Ugradnja i oblikovanje navigacijske trake

Za kreiranje **navigacije** možemo koristiti klase `.nav` ili `.navbar` unutar oznake **<ul>**, a zatim dodati pojedine poveznice unutar oznaka **<li>**.

- **.nav** nam omogućuje stvaranje osnovne navigacije, a **.navbar** izradu responzivnog navigacijskog zaglavlja.

Pojedine komponente navigacije oblikovane su pomoću svojstva **flexbox**.

U navigaciju možemo dodati primjerice:

- aktivnu stranicu
- padajuće izbornike navigacije
- logotip tvrtke
- polje za unos traženog sadržaja i sl.

Poveznice navigacije mogu se rasporediti: **horizontalno** (jedna uz drugu) ili **vertikalno** (jedna ispod druge).

Na malim zaslonima navigacija se skalira tako da se prikazuje u vidu jednog gumba (engl. *burger menu*) primjenom klasa `navbar-toggler-icon` i `.navbar-toggler`.

Klik na gumb otvara vertikalno raspoređen izbornik. Postoje i komponente oblikovane JavaScriptom koje uključuju različite efekte na pojedine poveznice navigacije.

```
<nav class="navbar navbar-expand-lg navbar-dark bg-dark">
    <div class=" container-fluid ">
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent"
        aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
        <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0">
                <li class="nav-item"><a class="nav-link active" aria-current="page" href="# ">Home</a></li>
                <li class="nav-item"><a class="nav-link" href="#">Link</a></li>
                <li class="nav-item dropdown"><a class="nav-link dropdown-toggle" href="#" id="navbarDropdown" role="button"
                data-bs-toggle="dropdown" aria-expanded="false">Dropdown</a>
                    <ul class="dropdown-menu" aria-labelledby="navbarDropdown">
                        <li><a class="dropdown-item" href="#">Action</a></li>
                        <li><a class="dropdown-item" href="#">Another action</a></li>
                        <li>
                            <hr class="dropdown-divider">
                        </li>
                        <li><a class="dropdown-item" href="#">Something else here</a></li>
                    </ul>
                </li>
                <li class="nav-item"><a class="nav-link disabled">Disabled</a></li>
            </ul>
            <form class="d-flex ">
                <input class="form-control me-2" type="search" placeholder="Search" aria-label="Search">
                <button class="btn btn-outline-light" type="submit">Search</button>
            </form>
        </div>
    </div>
</nav>
```

lika 6.7. Ugradnja navigacije i skaliranje navigacije

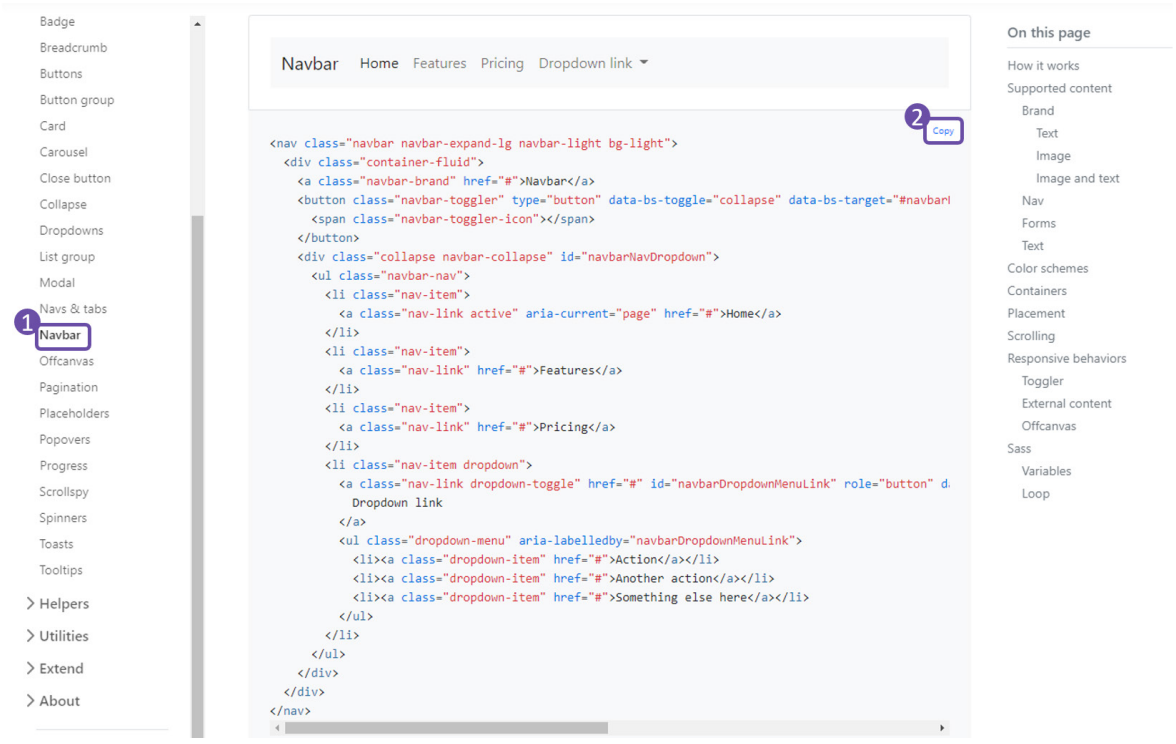
6.3. Izrada rasporeda elemenata i oblikovanje sadržaja primjenom Bootstrapa

- Na mrežnom sjedištu https://getbootstrap.com/ nalazi se niz primjera – od upotrebe pojedinih komponenti ovog razvojnog okvira do gotovih predložaka cijelog sjedišta.

6.3.1. Raspoređivanje elemenata na stranici

Pojedine Bootstrap komponente možemo lako ugraditi u svoje web-sjedište:

- S popisa komponenti na mrežnoj stranici (slika 6.5. lijevo) odaberemo željeni element.
- U središnjem dijelu stranice odaberemo željeno oblikovanje komponente, kopiramo HTML kod i zalijepimo u svoj HTML dokument u dio stranice u kojem želimo da se odabrani sadržaj nalazi. Ponuđene su nam različite opcije boja, veličina, oblika i sl., ovisno o vrsti komponente koju želimo ugraditi.

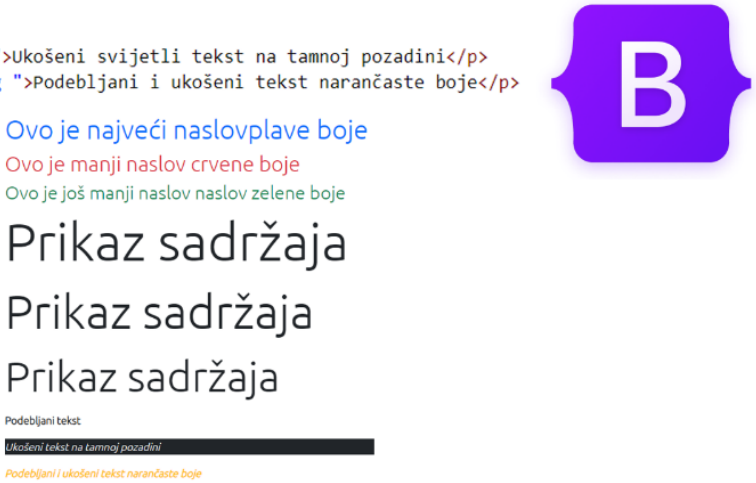


Slika 6.8. Primjer odabira komponente za ugradnju u web-stranicu

6.3.2. Prilagođavanje tekstualnih sadržaja

Ugrađujemo li na stranicu tekstualne sadržaje, umjesto uobičajenih oznaka elemenata (<h1>-<h6>, <p> i sl.) i pridružene liste stilova možemo dodati atribute različitih klasa, uz koje se automatski vežu liste stilova Bootstrapa.

```
<!--klase naslova (heading)-->
<p class="h1 text-primary ">Ovo je najveći naslov plave boje</p>
<p class="h2 text-danger ">Ovo je manji naslov crvene boje</p>
<p class="h3 text-success ">Ovo je još manji naslov naslov zelene boje</p>
<!--klase prikaza (display)-->
<p class="display-1 ">Prikaz sadržaja</p>
<p class="display-2 ">Prikaz sadržaja</p>
<p class="display-3 ">Prikaz sadržaja</p>
<!--oblikovanje stilova teksta-->
<p class="fw-bold ">Podebljani tekst</p>
<p class="fst-italic text-light bg-dark ">Ukošeni svijetli tekst na tamnoj pozadini</p>
<p class="fw-bold fst-italic text-warning ">Podebljani i ukošeni tekst narančaste boje</p>
```



Slika 6.9. Primjer oblikovanja tekstualnih sadržaja primjenom atributa klasa

- Bootstrap koristi određene fontove ovisno o operacijskom sustavu. Fontove je lako promijeniti i prilagoditi, primjerice ugradnjom Google fontova: https://fonts.google.com/

```
link href="https://fonts.googleapis.com/css2?family=Ubuntu:wght@300&display=swap" rel="stylesheet
style>
   body {
       font-family: 'Ubuntu', sans-serif;
   }
/style>
```

lika 6.10. Primjer ugradnje obitelji fontova

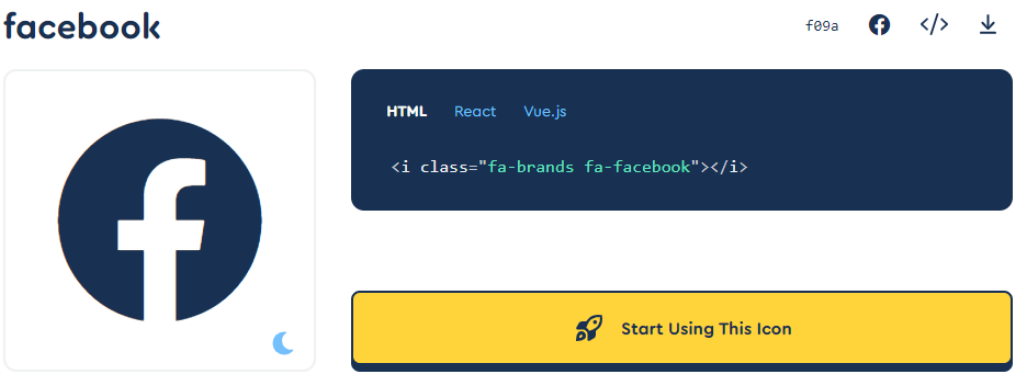
6.3.3. Poruke i upozorenja

Na naše stranice možemo po potrebi ugraditi poruke i/ili upozorenja. Korisnik ih nakon čitanja može zatvoriti tako da im dodamo gumb za zatvaranje. Vizualno ih možemo dodatno urediti dodavanjem **ikona**.

- Bootstrap ima i svoj repozitorij ikona koje možemo koristiti na svojim stranicama: https://icons.getbootstrap.com/
- Možemo koristiti i druge repozitorije ikona. Primjerice, možemo preuzeti kit sa stranice Font Awesome https://fontawesome.com/start.

Vaš jedinstveni kod ugradite u **<head>** stranice i možete ugrađivati različite ikone, prateći jednostavnu sintaksu:

```
src="https://kit.fontawesome.com/_____.js" crossorigin=
                "anonymous"></script>
```



Slika 6.11. Primjer koda za ugradnju Font Awesome ikone

Ikonama možemo mijenjati veličinu i boju dodajući im atribute iz Bootstrapa.

```
<i class="fa-brands fa-facebook
   fa-2x text-primary"></i>
```

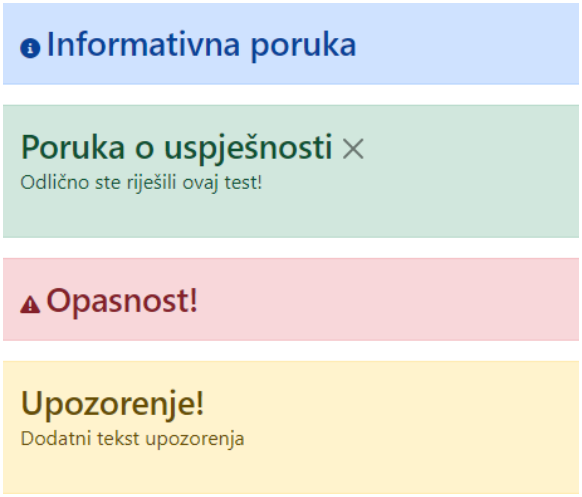
uvećano 2 puta

plava boja

```html
<div class="alert alert-primary">
    <i class=" fa-solid fa-circle-info"></i>
    <span class="h3" fw-bold>Informativna poruka</span>
</div>
<div class="alert alert-success alert-dissmisable">
    <span class="h3" fw-bold>Poruka o uspješnosti</span>
    <button class="btn-close" data-bs-dismiss="alert"></button>
    <p>Odlično ste riješili ovaj test!</p>
</div>
<div class="alert alert-danger">
    <i class="fa fa-warning"></i> <span class="h3" fw-bold>Opasnost!</span>
</div>
<div class="alert alert-warning">
    <span class="h3" fw-bold>Upozorenje!</span>
    <p>Dodatni tekst upozorenja</p>
</div>
```

Dodavanje Font Awesome ikone

Omogućavanje zatvaranja poruke



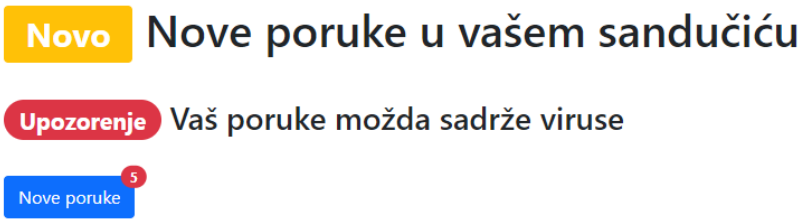
lika 6.12. Primjer ugradnje poruka i upozorenja te oblikovanja primjenom atributa klasa

6.3.4. Značke

Kako bismo **naglasili** ili **nadopunili sadržaje**, možemo ugraditi **značke (engl. *badge*)**, bilo kao zasebni element bilo kao dio gumba.

Njihova se veličina prilagođava veličini neposredno nadređenog elementa.

```
<h1><span class="badge bg-warning">Novo</span> Nove poruke u vašem sandučiću</h1><br>
<h3><span class="badge bg-danger rounded-pill">Upozorenje</span> Vaš poruke možda sadrže
viruse</h3><br>
<button type="button" class="btn btn-primary position-relative">
    Nove poruke
    <span class="position-absolute top-0 start-100 translate-middle badge rounded-pill
    bg-danger">
      5
    </span>
</button>
```



lika 6.13. Dodavanje znački

6.4. Primjena strukture rešetke u izradi sadržaja *web*-dokumenata

Struktura rešetke (engl. *grid layout*) u Bootstrapu oblikovana je kao *flexbox* s najviše 12 stupaca na stranici. Stupce možemo grupirati stvarajući šire stupce.

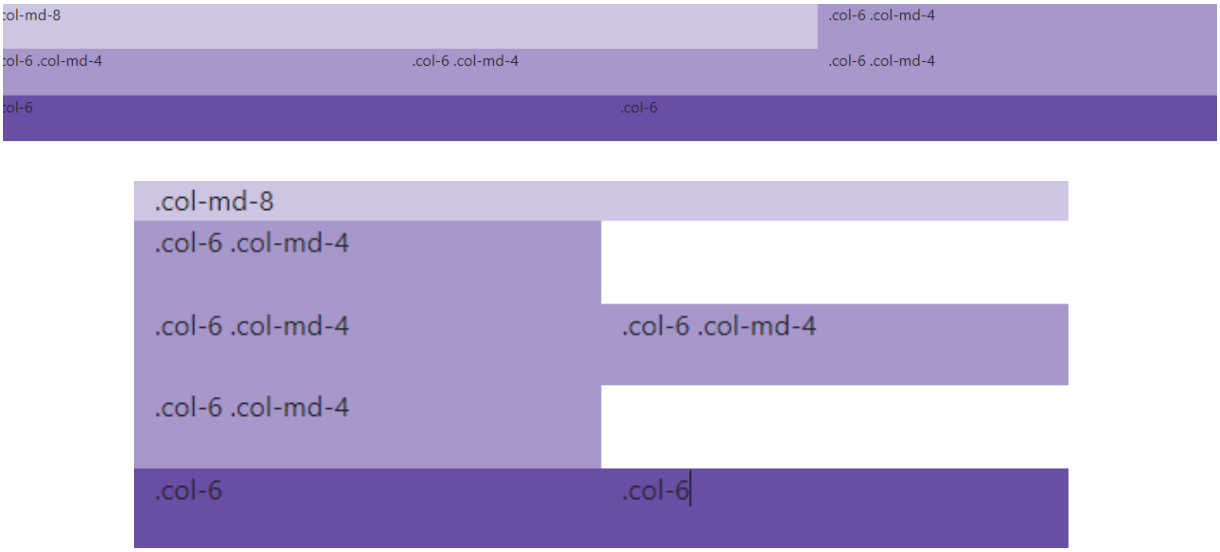
Bootstrap ima ugrađene oblikovane klase za različite veličine zaslona koje se prilagođavaju ovisno o šest točaka loma, od kojih svaka ima svoju oznaku klase (engl. *class prefix*).

	xs <576px	sm ≥576px	md ≥768px	lg ≥992px	xl ≥1200px	xxl ≥1400px
Container `max-width`	None (auto)	540px	720px	960px	1140px	1320px
Class prefix	`.col-`	`.col-sm-`	`.col-md-`	`.col-lg-`	`.col-xl-`	`.col-xxl-`
# of columns	12					
Gutter width	1.5rem (.75rem on left and right)					
Custom gutters	Yes					
Nestable	Yes					
Column ordering	Yes					

Slika 6.14. Točke loma za različite veličine zaslona uređaja

Sadržaje na stranici možemo oblikovati u stupce jednakih ili različitih širina upotrebom različitih **.col- klasa**.

```
<div class="container">
    <div class="row">
        <div class="col-md-8">.col-md-8</div>
        <div class="col-6 col-md-4">.col-6 .col-md-4</div>
    </div>
    <div class="row">
        <div class="col-6 col-md-4">.col-6 .col-md-4</div>
        <div class="col-6 col-md-4">.col-6 .col-md-4</div>
        <div class="col-6 col-md-4">.col-6 .col-md-4</div>
    </div>
    <div class="row">
        <div class="col-6">.col-6</div>
        <div class="col-6">.col-6</div>
    </div>
```



Slika 6.15. Oblikovanje sadržaja u stupce pomoću strukture rešetke i prilagodba prikaza sadržaja ovisno o maksimalnoj širini zaslona

6.5. Ugradnja i oblikovanje tablica

Tablice u Bootstrapu izrađujemo nizom elemenata **<table>, <thead>, <tr>, <th> i <td>.** Unutar oznaka možemo dodati klase **`.table–`** sa standardnim bojama, primjenjujući na taj način prilagođena oblikovanja Bootstrap liste stilova.

Primjerice, možemo kreirati:

- tzv. prugaste tablice `.table-striped`
- tablice s efektom promjene stilova prilikom prelaska miša `.table-hover`
- s okvirom oko ćelija `.table-bordered` ili
- bez okvira `.table-borderless`.

```
<table class="table table-dark table-hover">
    <thead>
        <tr>
            <th>Ime</th>
            <th>Prezime</th>
            <th>Dob</th>
        </tr>
    </thead>
    <tbody>
        <tr>
            <td>Ana</td>
            <td>Anić</td>
            <td>20</td>
        </tr>
        <tr>
            <td>Ivan</td>
            <td>Ivić</td>
            <td>25</td>
        </tr>
        <tr>
            <td>Marko</td>
            <td>Marković</td>
            <td>22</td>
        </tr>
    </tbody>
</table>
```

Ime	Prezime	Dob
Ana	Anić	20
Ivan	Ivić	25
Marko	Marković	22

lika 6.16. Oblikovanje tablica Bootstrap klasam

Ukoliko je tablica preširoka, možemo joj dodati horizontalni scrollbar klasom `.table-responsive`. Ovo svojstvo možemo podesiti za različite točke loma, primjerice `.table-responsive-sm` ili .`table-responsive-md`.

6.6. Ugradnja i oblikovanje multimedijskih sadržaja

6.6.1. Primjena Bootstrapa pri oblikovanju ugrađenih slika i videosadržaja

Bootstrap klase omogućuju nam da slikovnim sadržajima podesimo **prilagodljivost** (responzivnost) tako **da nikad ne postanu širi od pripadajućeg roditeljskog elementa** te da ih lakše **pozicioniramo** na stranici.

Slike na stranicu ugrađujemo primjenom oznake **`<img>`**, dok responzivnost postavljamo primjenom klase **`.img-fluid`**. Tada slika poprima svojstva **`max-width: 100%`**; i **`height: auto`**; u odnosu na element roditelja.

Slike možemo pozicionirati upotrebom **helper float klasa**:

- **`.float-start`**
- **`.float-end`**
- **`.float-none`**

```
<img src="img/avatar-muski.png" class="rounded float-start" alt="Muški avatar">
<img src="img/avatar-zenski.png" class="rounded float-end" alt="Ženski avatar">
```

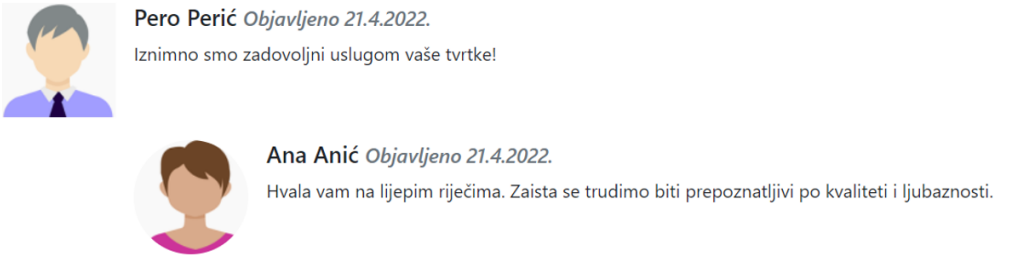
Slika 6.17. Oblikovanje slike primjenom klasa .float- i rounded

- Koristimo li oznaku **`<picture>`** da bismo postavili više različitih slika koje će se prikazivati ovisno o širini zaslona uređaja, važno je klasu **`.img-`** (primjerice .img-fluid, .img-thumbnail) dodijeliti u oznaci **`<img>`**, a ne u oznaci **`<picture>`**.

Slikama možemo promijeniti oblik primjenom klasa `.rounded` ili `.rounded-circle` te podesiti način prilagodbe veličine primjenom klasa `.flex-shrink` i `.flex-grow.`

```
<!--Multimedija-->
<!--Slike-->
<div class="d-flex">
    <div class="flex-shrink-0">
        <img src="img/avatar-muski.png" alt="Muški avatar">
    </div>
    <div class="flex-grow-1 ms-3">
        <h5>Pero Perić <small class="text-muted"><i> Objavljeno 21.4.2022. </i></small></h5>
        <p>Iznimno smo zadovoljni uslugom vaše tvrtke! </p>
        <br>
        <!-- Ugniježđeni media object -->
        <div class="d-flex mt-4">
            <div class="flex-shrink-0">
                <img src="img/avatar-zenski.png" class="rounded-circle" alt="Ženski avatar">
            </div>
            <div class="flex-grow-1 ms-3">
                <h5>Ana Anić <small class="text-muted"><i> Objavljeno 21.4.2022. </i></small></h5>
                <p>Hvala vam na lijepim riječima. Zaista se trudimo biti prepoznatljivi po kvaliteti i ljubaznosti.</p
            </div>
        </div>
    </div>
</div>
```

Postavljanje kružnog oblika slike



Slika 6.18. Oblikovanje slike klasom rounded-circle

Za **oblikovanje omjera širine i visine** ugrađenih videosadržaja pomoću elementa `<iframe>` možemo koristiti skup Bootstrap klasa `.ratio`:

- `.ratio-1x1`
- `.ratio-4x3`
- `.ratio-16x9`
- `.ratio-21x9`

```
<!--Video-->
<div class="ratio ratio-16x9">
    <iframe width="560" height="315" src="https://www.youtube.com/embed/
    -qfEOE4vtxE" title="YouTube video player" frameborder="0"></iframe>
</div>
```

Slika 6.19. Oblikovanje videa klasama .ratio

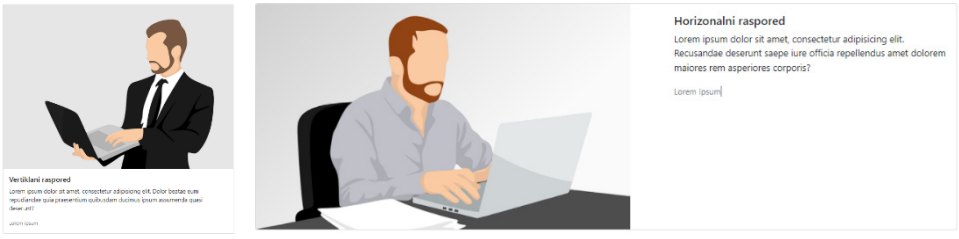
6.6.2. Oblikovanje sadržaja na stranici primjenom Bootstrap elementa *cards*

Kartice su uokvireni spremnici koji nam omogućuju ugradnju različitih multimedijskih sadržaja: slike, teksta, poveznica i sl. Uređene su flexboxom i prema zadanim postavkama **imaju padding**, **a nemaju marginu**.

Možemo ih ugraditi u grupama, bez primjene strukture rešetke ili s njom. Upotreba strukture rešetke omogućuje nam lakšu prilagodbu sadržaja različitim veličinama zaslona. Pojedine elemente kartica možemo po potrebi dodavati ili uklanjati (sliku, naslov, tekst, podnožje ili poveznice).

Raspored elemenata u kartici možemo oblikovati vertikalno ili horizontalno.

```
<div class="card" style="max-width: 640px;">
    <img src="img/slika1.png" class="card-img-top" alt="...">
    <div class="card-body">
        <h5 class="card-title">Vertiklani raspored</h5>
        <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit. Dolor beatae
        eum repudiandae quia praesentium quibusdam ducimus ipsum assumenda quasi deserunt?</p>
        <p class="card-text"><small class="text-muted">Lorem ipsum</small></p>
    </div>
</div>
<div class="card mb-3" style="max-width: 1200px;">
    <div class="row g-0">
        <div class="col-md-7">
            <img src="img/slika2.png" class="img-fluid rounded-start" alt="...">
        </div>
        <div class="col-md-5">
            <div class="card-body">
                <h5 class="card-title">Horizonalni raspored</h5>
                <p class="card-text">Lorem ipsum dolor sit amet, consectetur adipisicing elit.
                Recusandae deserunt saepe iure officia repellendus amet dolorem maiores rem
                asperiores corporis?</p>
                <p class="card-text"><small class="text-muted">Lorem ipsum</small></p>
            </div>
        </div>
    </div>
</div>
```



Slika 6.20. Horizontalni i vertikalni raspored sadržaja u kartici

6.7. Izmjenjivanje multimedijskih sadržaja

6.7.1. Ugradnja funkcionalnosti za izmjenu multimedijskih sadržaja (engl. *carousel*)

Komponenta engl. *carousel* omogućuje nam kružno izmjenjivanje slika ili slajdova. Izrađena je pomoću CSS 3D transformacija i JavaScripta.

Carousel nema automatizirano dimenzioniranje slajdova, nego ih je potrebno unaprijed pripremiti tako da budu jednakih dimenzija ili podesiti njihove dimenzije u listi stilova.

Pri ugradnji ove komponente možemo, a i ne moramo koristiti prikaz koji uključuje **kontrolne elemente i indikatore.**

Nužno je jednom od slajdova dodijeliti klasu .active.

```
<div id="carouselExampleCaptions" class="carousel slide" data-bs-ride="carousel">
    <div class="carousel-indicators">
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="0" class="active" aria-current="true" aria-
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="1" aria-label="Slide 2"></button>
        <button type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide-to="2" aria-label="Slide 3"></button>
    </div>
    <div class="carousel-inner">
        <div class="carousel-item active">
            <img src="img/slider3.jpg" class="d-block w-100" alt="...">
            <div class="carousel-caption d-none d-md-block">
                <h5>Monitori</h5>
                <p>Podnaslov prvog slajda</p>
            </div>
        </div>
        <div class="carousel-item">
            <img src="img/slider1.jpg" class="d-block w-100" alt="...">
            <div class="carousel-caption d-none d-md-block">
                <h5>Digitalni svijet</h5>
                <p>Podnaslov drugog slajda</p>
            </div>
        </div>
        <div class="carousel-item">
            <img src="img/slider2.jpg" class="d-block w-100" alt="...">
            <div class="carousel-caption d-none d-md-block">
                <h5>Globe</h5>
                <p>Podnaslov trećeg slajda</p>
            </div>
        </div>
    </div>
    <button class="carousel-control-prev" type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide="prev">
      <span class="carousel-control-prev-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Previous</span>
    </button>
    <button class="carousel-control-next" type="button" data-bs-target="#carouselExampleCaptions" data-bs-slide="next">
      <span class="carousel-control-next-icon" aria-hidden="true"></span>
      <span class="visually-hidden">Next</span>
    </button>
</div>
```

Slika 6.21. Primjer ugradnje *carousela*

Izvor: Pixabay.com

Slika 6.22. Izgled pojedinih slajdova

Po potrebi možemo dodati više slika, ukloniti naslove i podnaslove ili kontrolne elemente i indikatore.

Pitanja za ponavljanje

1. Nabroji nekoliko svojstava Bootstrapa.
2. Objasni na koje je načine u našim projektima moguće koristiti Bootstrap?
3. Kako su strukturirane komponente Bootstrapa?
4. Koje komponente Bootstrapa možemo koristiti za ugradnju i oblikovanje navigacije?
5. Koje sve oznake (class prefix) možemo koristiti kod oblikovanja sadržaja strukturom rešetke pomoću Bootstrapa?
6. Nabroji nekoliko klasa za oblikovanje tablica pomoću Bootstrapa.
7. Opiši načine oblikovanja multimedijskih sadržaja pomoću Bootstrapa.

Literatura:

1. **Službena stranica Bootstrap.**
   Preuzeto 1.3.2022. s https://getbootstrap.com/
2. **Bootstrap Tutorial.**
   Preuzeto 19.4.2022. s https://www.tutorialrepublic.com/twitter-bootstrap-tutorial/
3. **Bootstrap 5 Crash Course 2022, Naveen Saggam.**
   Preuzeto 5.3.2022. s https://www.udemy.com/course/bootstrap-5-crash-course-2022/

7. JavaScript razvojni okvir - React

U OVOM POGLAVLJU NAUČIT ĆETE:

- Što je React?
- Kako primijeniti React komponente za izgradnju korisničkog sučelja?
- Kako u Reactu rukujemo događajima?
- Kao izraditi formu i validirati podatke koje korisnik unesi u formu?
- Zašto testirati funkcionalnosti komponenti?
- Kako komunicirati sa sustavima putem REST API-a?

Slika 7.1. React logo

React je *open source* JavaScript biblioteka **za izradu korisničkih sučelja** (engl. *user interface UI*), odnosno komponenti za korisnička sučelja (gumbi, slike, komentari i sl.). Izradio ju je **Jordan Walke** za potrebe Facebooka 2011. Jednostavan je i lako se uči.

Možemo ga koristiti za izradu **aplikacija na jednoj stranici** (engl. *single page application SPA*). Većina resursa učitava se samo jednom prilikom inicijalnog iscrtavanja

stranice. Ovisno o zahtjevu korisnika prema poslužitelju prikazat će se određeni podaci, a svakim novim zahtjevom poslužitelj će dostavljati nove podatke, koji će se mijenjati na stranici. Izmjenjuju se samo oni podaci za koje je korisnik zatražio da se izmjene. Primjenom SPA-a omogućeno nam je stvaranje velikih *web*-aplikacija koje u interakciji s korisnikom **osvježavaju pojedine sadržaje** na stranici, odnosno **dijelove stranice**, a bez ponovnog učitavanja cijele stranice. Ovakav pristup izbjegava prekid korisničkog iskustva učitavanjem uzastopnih stranica, čineći ponašanje aplikacije nalik onome desktop aplikacija.

- React se koristi za **prikaz MVC aplikacija** (engl. *Model View Controler*).
- React pojednostavljuje način na koji se podaci pohranjuju i obrađuju, koristeći **state** i **props** objekte.
- Ne koristi se za rad s poslužiteljima.
- Pomoću malih izoliranih dijelova koda – **komponenti**, stvaramo kompleksno, efikasno i fleksibilno korisničko sučelje (UI).
- HTML DOM stabla suvremenih web-aplikacija velika su i dinamička.
  To znači da se DOM često ažurira, što usporava rad aplikacija i angažira procesor. React istovremeno koristi dvije instance virtualnog DOM-a: **ažurno stanje i prethodno stanje** za usporedbu primjenom ***The Diffing Algorithm*** te ažurira samo one elemente koji su se promijenili.

7.1. Postavljanje radne okoline za rad s Reactom

Da bismo radili s Reactom potrebno je instalirati **Node.js**, a potom i **Node Packet Manager (NPM).**

- Node.js preuzimamo s: https://nodejs.org/en/download/

Raspakiramo komprimiranu datoteku i pokrećemo instalaciju pomoću

```
node.exe
```

Po završetku instalacije možemo u **naredbenom retku (CMD command prompt)** provjeriti koju smo verziju instalirali unosom:

```
node –v
```

NPM se sastoji se od mnoštva paketa koji nam omogućuju obavljanje različitih zadataka.

Da bismo globalno instalirali **NPM**, u naredbenom retku (cmd) pišemo naredbu

```
npm install npm@latest –g
```

- Mogli bismo globalno instalirati i React, ali to ne želimo zato što će svaki projekt imati drugu verziju Reacta, mnoštvo različitih paketa i njihovih ovisnosti (engl. dependencies) te ako bismo naknadno radili update različitih verzija, možda nam neke aplikacije ne bi radile.

Kreirat ćemo **novu mapu** za svaki pojedini novi projekt, smjestiti se pomoću naredbenog retka u tu mapu ili u ugrađenom terminalu aplikacije Visual Studio Code te ćemo napraviti `npm init` i dobiti mapu

```
package.json
```

te pokrenuti npm install react i dobiti mapu

```
node–modules
```

u kojoj se nalaze **paketi i ovisnosti** koje React treba za rad.

Isto možemo postići s

```
npm i ––save
```

no tada nam se pohranjuju i sve ovisnosti za pojedinu verziju Reacta u `package.json.`

Nakon instalacije potrebnih preduvjeta možemo izraditi React aplikaciju u željenoj mapi primjenom:

```
npx create–react–app my–app.
```

Prebacimo se u mapu projekta:

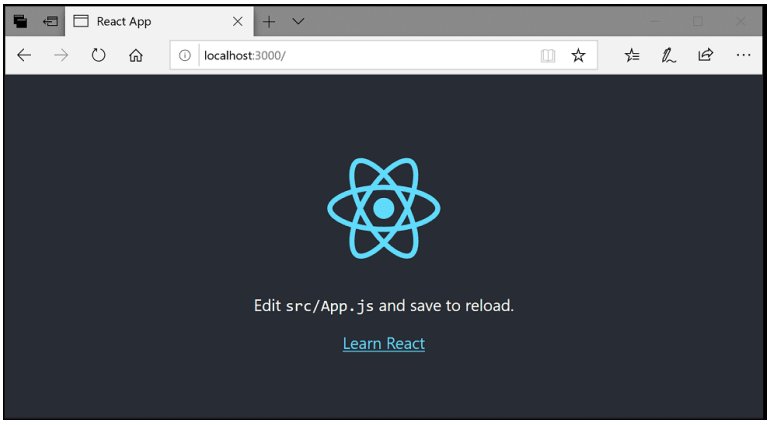
```
cd my-app
```

i pokrenemo React

```
npm start.
```

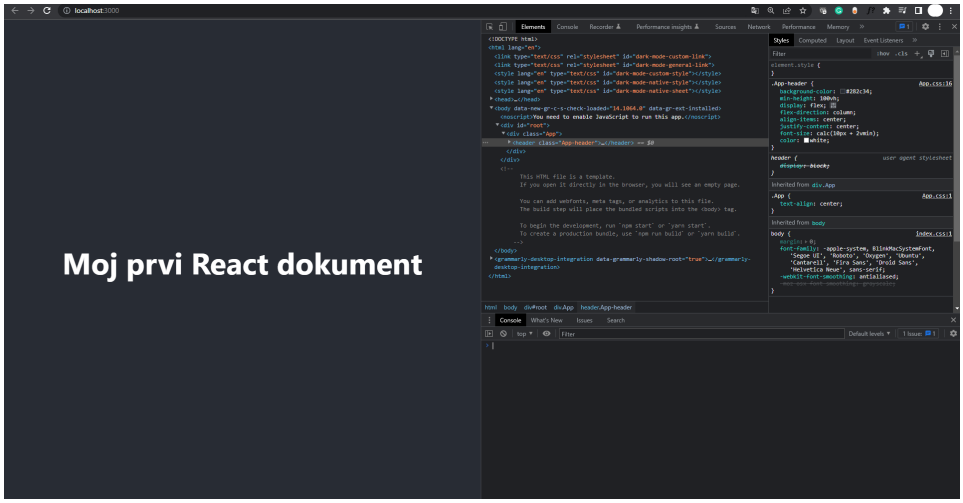
- Pokrenemo li izradu aplikacije i nakon toga ne stavimo ime my-app nego samo . sve će se datoteke nalaziti unutar mape projekta koji smo kreirali bez stvaranja mape my-app.

Tada se **index.html** iz mape **public** novog projekta automatski otvara u pregledniku pomoću lokanog servera. Cijela aplikacija kontrolira se iz dokumenta **App.js** koji se nalazi u mapi **src.**



Slika 7.2. Početna stranica pri radu s Reactom

- Za lakši rad s Reactom korisno je preuzeti **React Developer Tools** proširenje u preglednik.
  https://reactjs.org/link/react-devtools



Proširenje nam pomaže da pratimo ponašanje koda. Otvaramo ga desnim klikom u prozoru preglednika i biramo **Provjeri** (engl. *inspect*). Ukoliko postoje problemi s izvršavanjem, ovdje nam se ispisuju pogreške i problemi.

Za zaustavljanje rada servera potrebno je u konzoli pritisnuti:

CTRL+C, pa odabrati **Y**

a za ponovno pokretanje

`npm start.`

Prilikom pohrane izmjena promjene se automatski izvršavaju na stranici i nije potrebno osvježavati stranicu.

- React projekt možemo kreirati i tako što u VS Codeu otvorimo novu mapu pa u terminalu aplikacije upišemo naredbe:

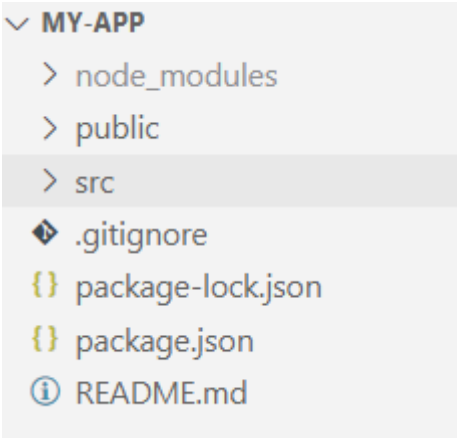
npx create-react-app my-app

kad je postavljen projekt:

cd my-app

pa otvorimo projekt u pregledniku:

npm start.



7.3. Mape i datoteke React projekta

U mapi React projekta nalazi se niz drugih mapa i datoteka.

Mapa **node_modules** sadrži različite pakete koje smo instalirali s npm-om. Budemo li željeli dodati dodatne pakete, upravo ćemo ih tu trebati instalirati. U mapi **public** nalazi se **index.html**, ikona Reacta i sl. Datoteka **gitignore** zanemaruje dio datoteka ukoliko odlučimo projekt podići na Github.

Datoteka **package.json** sadrži popis svih paketa koji pripadaju aplikaciji i skripti koje nam omogućuju optimizaciju projekta prije objavljivanja. Za dodavanje paketa koristimo:

```
npminstall ——save <ime-paketa>.
```

Datoteka **package-lock.json** sadrži popis verzija svih instaliranih paketa.

Mapa **src** jest mapa u kojoj ćemo raditi i u koju ćemo dodavati naše komponente te na taj način graditi aplikaciju. Iz nje možemo ukloniti sve datoteke osim **App.js** i **index.js**, no možemo ih i ostaviti jer nam ne smetaju.

JavaScript **modul** samostalan je i odvojeni dio koda koji obavlja neku funkciji. Modula može biti više, a možemo ih dodavati, mijenjati ili uklanjati prema potrebi bez narušavanja rada sustava. Kod dijelimo u više manjih datoteka koje se spajaju u željenu funkcionalnost.
Primjenom modula lakše je održavanje koda i ponovna upotreba.

- Da bi se programski kod napisan u jednoj datoteci mogao koristiti u drugoj datoteci, isti je potrebno **izvesti** (engl. ***export***) iz datoteke u kojoj je napisan i **uvesti** (engl. ***import***) u datoteku u kojoj ga želimo koristiti. Razlikujemo ***default*** i ***named*** *import* i *export*. U pravilu ćemo koristit *default* jer je jednostavniji.

Istovremeno možemo uvesti više modula.

Prvi dokument u kojemu ćemo raditi bit će **App.js.**

U **App.js** datoteci za početak je potrebno uvesti React:

```
import React from ″react″;
```

zatim kreiramo komponentu primjenom funkcije **App().**

App.js M ×

src > App.js > ...

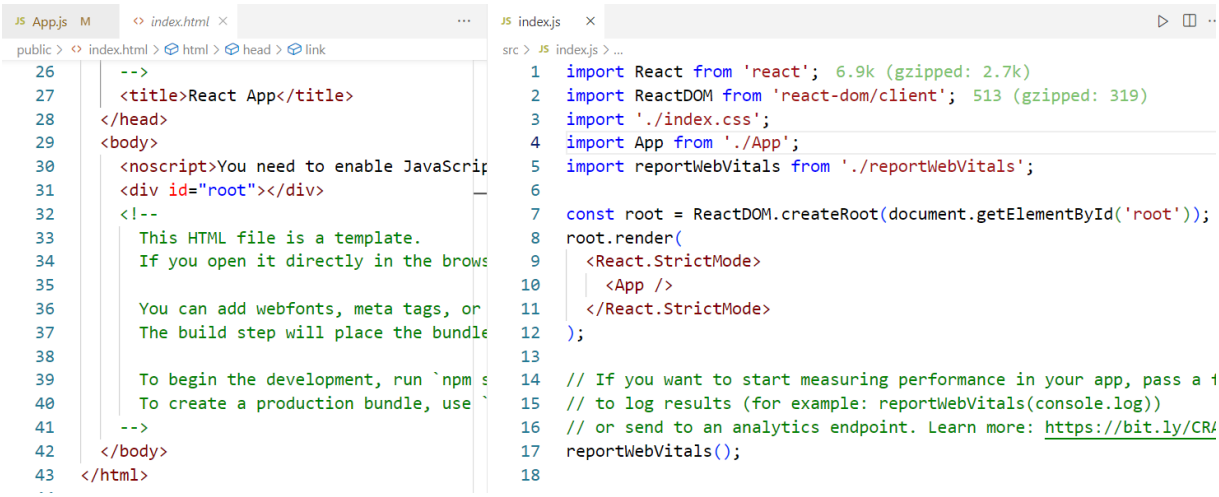
```
1  import React from "react";  6.9k (gzipped: 2.7k)
2
3  function App() {
4    return (
5      <div className="App">
6        <h1>Hello from React component!</h1>
7      </div>
8  )}
9
10 export default App;
```

Slika 7.4. Izrada osnovne komponente

Funkciju App izvozimo upotrebom

```
export default App;.
```

Komponentu koju smo kreirali prikazat će index.html iz mape public. Datoteka index.html sadrži jedan jedini element div, čiji je `id=″root″`. Sve izvodi index.js koji komponentu renderira upravo u taj root.



Slika 7.5. Izrada osnovne komponent

7.2. JavaScript sintaksno proširenje (JSX)

U prethodnom primjeru vidjeli smo da smo unutar funkcije pisali kod koji izgleda kao HTML. Radi se o **JSX-u (JavaScript XML)** – dodatku (engl. *extension*) sintaksi JavaScripta. Tijekom prevođenja aplikacije sav kod napisan korištenjem JSX-a pretvorit će se u odgovarajući JavaScript kod. Prevođenje se izvodi automatski pomoću **Babela**, JavaScript prevoditelja koji smo instalirali pokretanjem naredbe `create-react-app`.

Sintaksa mu je **jednostavna** za upotrebu, oznake (engl. *tags*) sliče HTML-u, a mogu osim naziva sadržavati i atribute. Moguće je ostvariti i **ugnježđivanje oznaka** (engl. *children*).

React je zasnovan na **dizajniranju pojedinačnih komponenti**, čime nam je omogućena ponovna upotreba koda. Pomoću komponenti gradimo strukturu korisničkog sučelja.

- Svaka komponenta vraća neki JSX element, a želimo li da ih vrati više, oni moraju biti ugniježđeni u jednom vršnom elementu (najčešće), što React tada smatra jednim elementom.

7.2.1. Izrada grafičkog sučelja primjenom sintakse JavaScript XML – JSX

Kako smo već naveli, React nam omogućuje **pisanje koda sličnog HTML-u** te **kombiniranje** s **naredbama JavaScripta**.

JavaScript dodajemo u **{ }**. U { } možemo dodati **konstante**, **varijable** i **funkcije** JavaScripta unutar HTML oznaka ili **bilo koji drugi validan JavaScript izraz**.

App.js M ×

my-app > src > App.js > ...

```
function App() {
  const ime='JSX';
  const lista = (
  <ul>
    <li>HTML</li>
    <li>CSS</li>
    <li>JavaScript</li>
  </ul>
  );
  const spremnik = (
    <div>
      <h2>Naslov</h2>
      <p>Lorem ipsum dolor sit amet consectetur adipisicing elit. Sunt iusto nihil sint temporibus necessitatibus quam
      eius nostrum? Ipsum veritatis ea obcaecati tempora et, quisquam quod? Libero est obcaecati voluptatibus optio?</p>
    </div>);
  return (
    <div className="App">
      <header className="App-header">
        {/*naslovi*/}
          <h1>Ovo je JSX</h1>
          <h1>Pozrdav, {ime}!</h1>
        {/*umetanje izračuna*/}
          <p>React je {10*10} puta bolji uz JSX!</p>
        {/*umetanje liste*/}
        <div>{lista}</div>
        {/*umetanje spremika*/}
        {spremnik}
      </header>
    </div>
  );
}
```

Slika 7.6. Upotreba JSX-a u Reactu

Pokrenemo `npm start` za našu aplikaciju u terminalu VS Codea, koji nam kompajlira aplikaciju i otvori je u zadanom pregledniku.

```
Windows PowerShell
Compiled successfully!

You can now view my-app in the browser.

  Local:            http://localhost:3000
  On Your Network:  http://172.26.144.1:3000

Note that the development build is not optimized.
To create a production build, use npm run build.

webpack compiled successfully
```

Slika 7.7. Kompajliranje aplikacije u terminalu VS Codea

Ovo je JSX

Pozrdav, JSX!

React je 100 puta bolji uz JSX!

- HTML
- CSS
- JavaScript

Naslov

Lorem ipsum dolor sit amet consectetur adipisicing elit. Sunt iusto nihil sint temporibus necessitatibus quam eius nostrum? Ipsum veritatis ea obcaecati tempora et, quisquam quod? Libero est obcaecati voluptatibus optio?

Slika 7.8. Prikaz JSX sadržaja u pregledniku

7.2.2. Napredni koncepti korištena sintakse JavaScript XML – JSX

App.js M ×

src > App.js > ...

```
1  import React from "react";  6.9k (gzipped: 2.7k)
2
3  function App() {
4    const tekst =()=>{
5      document.write("Tekst iz funkcije");
6    }
7    return (
8      <div className="App">
9        <h1>Hello from React component!</h1>
10       <button onClick={tekst}>Klikni za ispis</button>
11     </div>
12 )}
13
14 export default App;
```

Slika 7.9. Upotreba JSX-a te funkcije i događaja onClick u Reactu

Unutar App.js možemo kreirati i funkcije čije povratne vrijednosti možemo ugraditi u HTML upotrebom JSX sintakse kako bismo renderirali dinamičke podatke u našim komponentama.

- Ukoliko nam VS Code za React ne dovršava automatski HTML oznake, potrebno je kliknuti na alatnu traku u donjem dijelu prozora i u *Select Language Mode* izborniku dodati React.

7.3. Komponente i svojstva komponenti u Reactu

Komponenta je **temeljni dio Reacta**, osnovni gradivni blok, odnosno djelić korisničkog sučelja. Kreirat ćemo ih unutar nove mape **components** u mapi **src**.

Jedna je komponenta JavaScript **klasa** ili **funkcija** koja prima proizvoljan broj parametara koji se nazivaju svojstva (eng. *property*) i **vraća React element** koji opisuje kako će izgledati određeni dio korisničkog sučelja (engl. *User Interface, UI*).

Komponente se mogu ugnijezditi jedna unutar druge.

Pojedinačne komponente stvaramo kao **JavaScript dokumente**.

Razlikujemo **složene** (pametne, engl. *statefull*) i **jednostavne** (prezentacijske, engl. *stateless*) komponente.

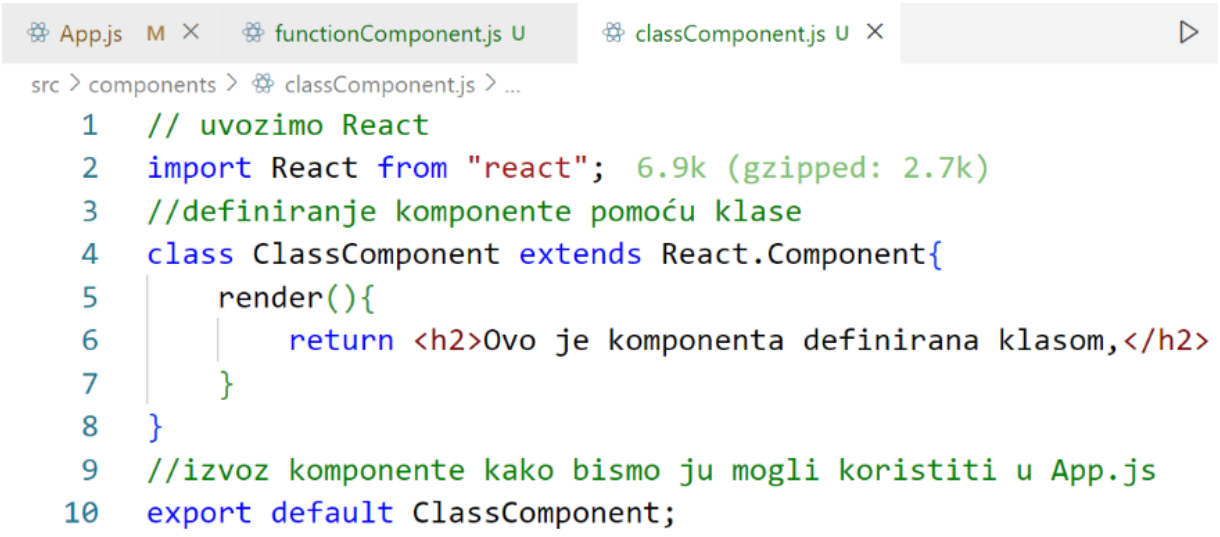
U Reactu komponente se mogu definirati upotrebom:

1. **funkcije** (metode)

Slika 7.10. Primjer definiranja komponente upotrebom funkcije

- Instaliramo li u VS Code dodatak **ES7 React/Redux/React-Native/JS snippets**, u novoj komponenti upišemo **rfc** (engl. *React Functional Component*) i stisnemo **Enter** – to nam kreira predložak komponente definiran pomoću funkcije.

2. **klase** (koristimo standard ECMAScript 6) – kreiranje komponente pomoću klase ostvaruje se tako da se naslijedi klasa `React.Component` pomoću ključne riječi **extends**.



Slika 7.11. Primjer definiranja komponente upotrebom klase

- Komponenta kreirana pomoću klase mora sadržavati metodu **render()** jer klasa ne može direktno vratiti JSX sadržaj.

Oba načina deklariranja komponenti mogu vratiti **samo jedan objekt**. Odnosno React komponenta vraća samo jedan React element odnosno JavaScript objekt koji sadrži tip i svojstva među kojima može biti objekt *children* (bit će objašnjen kasnije). Nije moguće kreirati komponentu koja vraća više komponenti jednu ispod druge. Ako to želimo postići moramo komponente „zamotati" u neki drugi HTML element, kako smo već naveli ranije.

Komponente kreiramo u mapi **src** našeg projekta. Ime komponente uvijek započinje **velikim slovom** i ima ekstenziju **.js**.

Komponente kreirane kao zasebne datoteke izvozimo upotrebom naredbe `export default`, a zatim uvozimo i koristimo u dokumentu **App.js** koji upravlja izvođenjem projekta. Dokument je **automatski stiliziran** pomoću **App.css** koji se kreira automatski pri izradi React aplikacije, ali potrebno je uvesti dokument u App.js.

U primjeru (slika 7.12.) uočavamo uvoz i upotrebu ClassComponent i FunctionComponent.

App.js M × functionComponent.js U × classComponent.js U

src > App.js > default

```
1  import React from "react";  6.9k (gzipped: 2.7k)
2  import './App.css';
3  //uvoz vanjskih komponenti
4  import ClassComponent from "./components/classComponent";
5  import FunctionComponent from "./components/functionComponent";
6  function App() {
7    return (
8      <div className="App">
9        <h1>Hello from React component!</h1>
10       <div className="App-header">
11       {/*Primjena vanjskih komponenti pomoću samozatvarajućih oznaka*/}
12         <ClassComponent/>
13         <FunctionComponent/>
14       </div>
15     </div>
16 );}
17
18 export default App;
```

Slika 7.12. Primjer uvoza i upotrebe komponenti u App.js

Imena React **komponenti** počinju velikim slovom.

- U Reactu ne možemo koristiti atribut class jer je **class** ključna riječ koja se koristi za kreiranje klase, nego koristimo atribut **className**, čiju vrijednost koristimo kao selektor u CSS-u.

Rezultat je odmah po pohrani vidljiv na stranici i nije potrebno osvježavanje.

Slika 7.13. Primjena komponenti na stranic

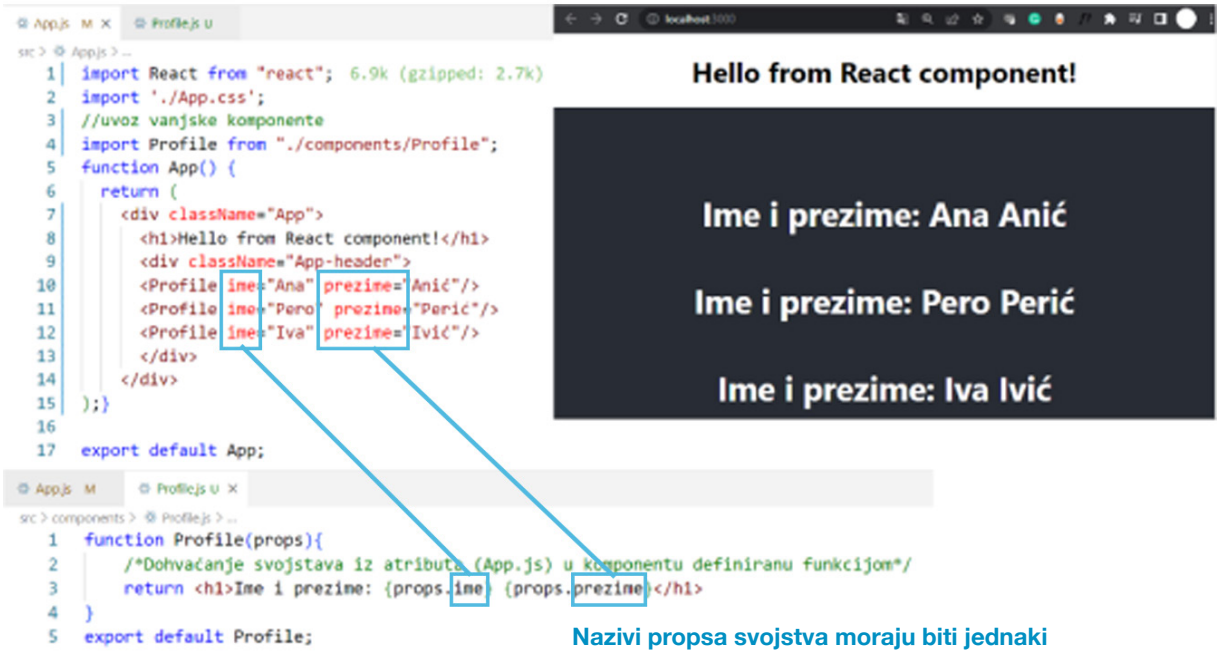
Datoteku **App.css** možemo po želji uređivati.

U primjeru kreiranja komponente funkcijom funkcija nije primala argumente, no najčešće ćemo joj željeti proslijediti neke ulazne podatke, manipulirati njima i tek onda vratiti rezultat.

Props

Za kreiranje dinamičkih komponenti stranice koristit ćemo **svojstva** (engl. *properties*) **props**, koja ćemo prosljeđivati kao **argumente funkcijama** preko **HTML atributa**. Radi se o objektu koji sadrži informacije kojima **upravljamo ponašanjem komponente**. Komponenta ih dobiva izvana i **ne može ih mijenjati**.

- Primjenom propsa komponente međusobno **komuniciraju**.
- Pomoću atributa možemo onoliko vrijednosti koliko želimo/trebamo koristiti (primjerice *ime i prezime*) proslijediti komponenti kao props i onda ih dohvatiti u funkciji.



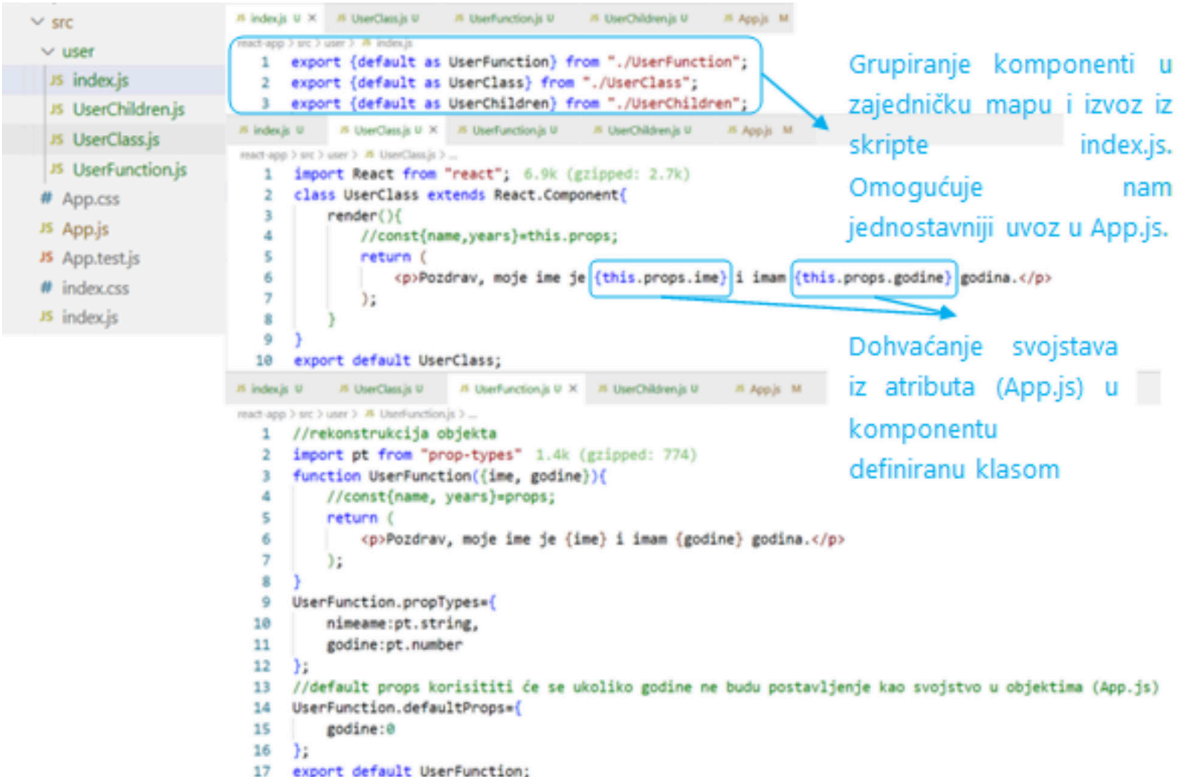
Slika 7.14. Primjena propsa u komponenti na stranici

Unutar oznaka pojedinih komponenti moguće je ugnijezditi druge elemente – djecu. U komponenti ih onda pozivamo sintaksom `{props.children}`.

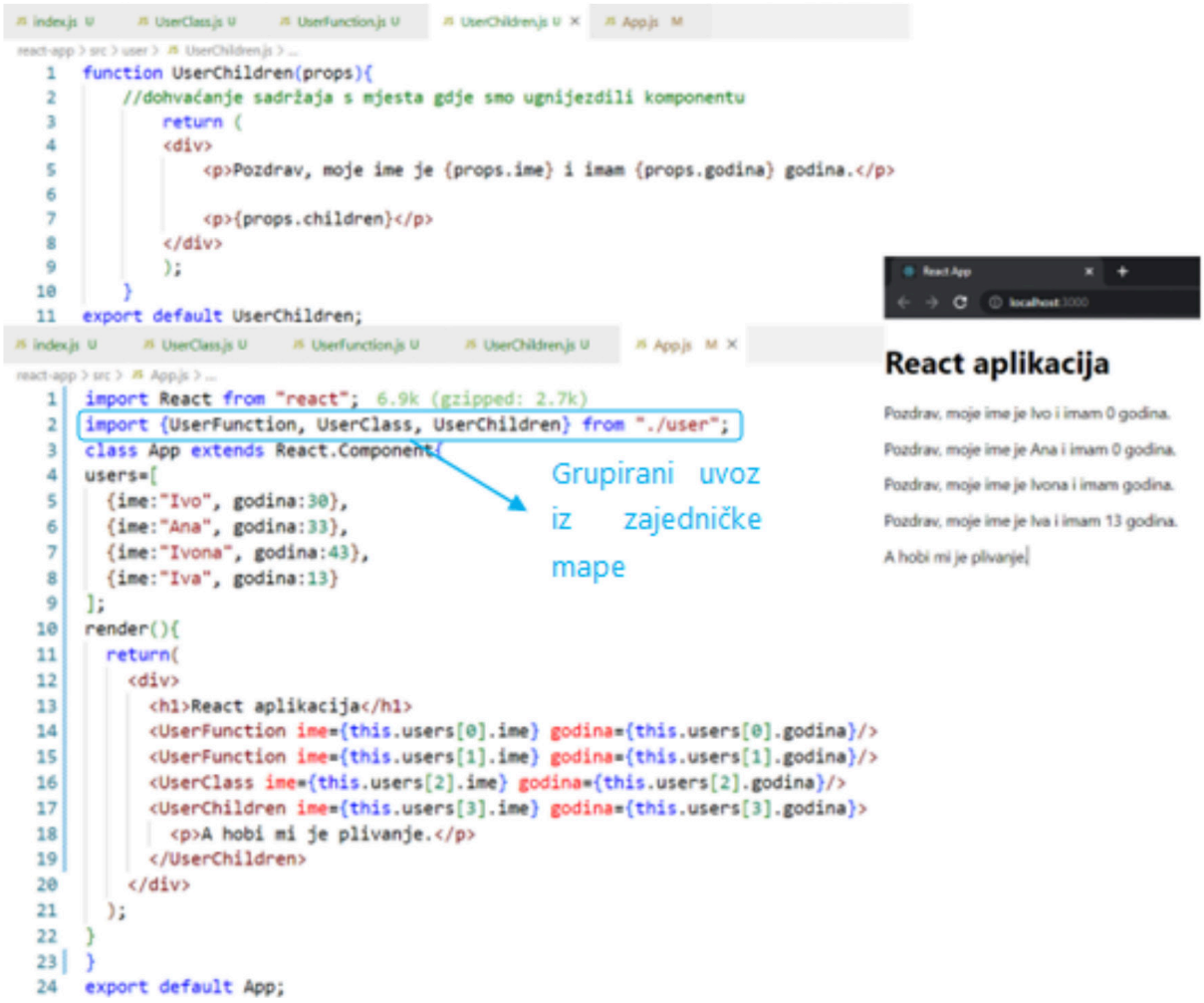
- React aplikacija stablo je ugniježđenih React komponenti.

Na sljedećim isječcima koda možemo uočiti kreiranje komponenti i pomoću klase i pomoću funkcije te primjenu child propsa. Kod klasa nužna je upotreba ključne riječi **this**.

`this.props.name.`



Slika 7.15. Kreiranje komponenti pomoću klase i pomoću funkcije



Slika 7.16. Primjeri upotrebe propsa i child propsa iz različito kreiranih komponenti

Svojstva komponenti utječu na izgled korisničkog sučelja, no pri primjeni vrijednosti podataka propsa ta se svojstva **ne mogu dinamički mijenjati** unutar komponente (engl. immutable). Podaci iz propsa u komponenti su ***read-only***, tj. možemo ih samo upotrijebiti, ali ih ne možemo mijenjati. Želimo li promijeniti izgled dijela korisničkog sučelja (UI-a) primjenom propsa, komponentu je potrebno **uništiti** i **stvoriti novu**, kojoj ćemo predati nove vrijednosti propsa. Ovo je ozbiljno ograničenje za upotrebu propsa. Da bismo to izbjegli, u Reactu je osmišljen koncept **unutarnjeg stanja komponenata** (engl. *state*).

Stanja

Stanje je **promjenjivi skup podataka** (engl. *mutable*) unutar **state** objekta komponente. Svaka komponenta ima vlastiti, privatan skup stanja, koja je **moguće mijen-**

jati samo unutar komponente koja ih sadrži (lokalno).

- Komponenta ima kontrolu nad svojim stanjem i može ga mijenjati pomoću metode **setState**.

Stanje je **varijabla** koju pomoću JSX-a možemo dodati u **metodu render()** i na taj način utjecati na promjene na korisničkom sučelju.

- O stanju možemo razmišljati kao o svim podacima koji bi trebali biti spremljeni i izmijenjeni bez nužnog dodavanja u bazu podataka – na primjer dodavanje i uklanjanje artikala iz košarice prije potvrde kupnje.

Stanje predstavlja ugrađeni objekt koji se može inicijalizirati:

1. izravno unutar **klase**
2. unutar metode **constructor().**

- Props i state slični su, ali za razliku od propsa, kod promjene podataka unutar objekta **state** u komponenti dolazi do **automatskog ponovnog renderiranja** te komponente bez njezina uništavanja.

Pojedine komponente mogu imati mnoštvo stanja. Podatak pohranjen kao stanje pod nekim imenom koristi se u obliku:

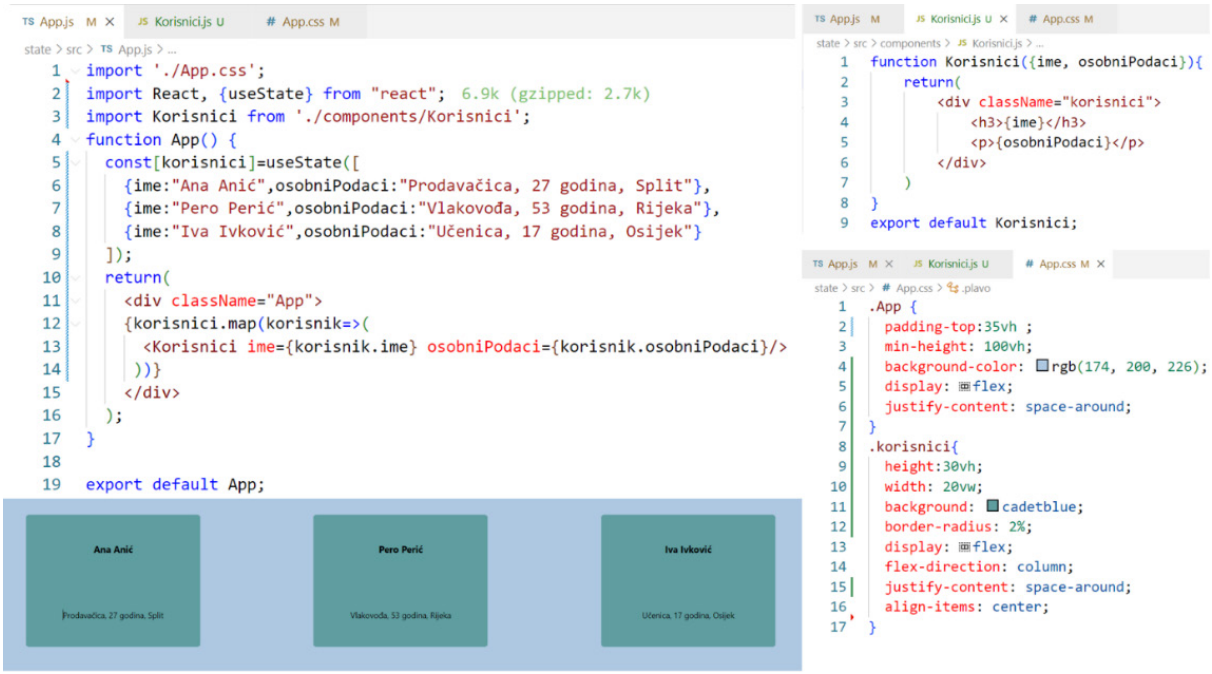
```
this.state.ime
```

Props	State
Možemo ih smatrati parametrima koji se prosljeđuju komponenti.	Varijable deklarirane u tijelu funkcije.
Ne mogu se mijenjati u komponenti.	Mogu se mijenjati.
Dohvaćamo ih pomoću: this.props.	Dohvaćamo ih pomoću: this.state.

Tablica 7.1. Usporedba svojstava props i state objekata

Da bismo koristili **state** u komponentama definiranim funkcijom, moramo uvesti **{useState} hook.**

Primjer upotrebe objekta state.



Slika 7.17. Primjena objekta state i komponente definirane funkcijom

- State se smije mijenjati samo pomoću metode setState, nakon čega se komponenta ponovno iscrtava (renderira) kako bi prikazala promjene u stanju. Ponovnim renderiranjem komponente renderiraju se i sve njezine *child* komponente.

7.4. Stanje i životni vijek React komponenti

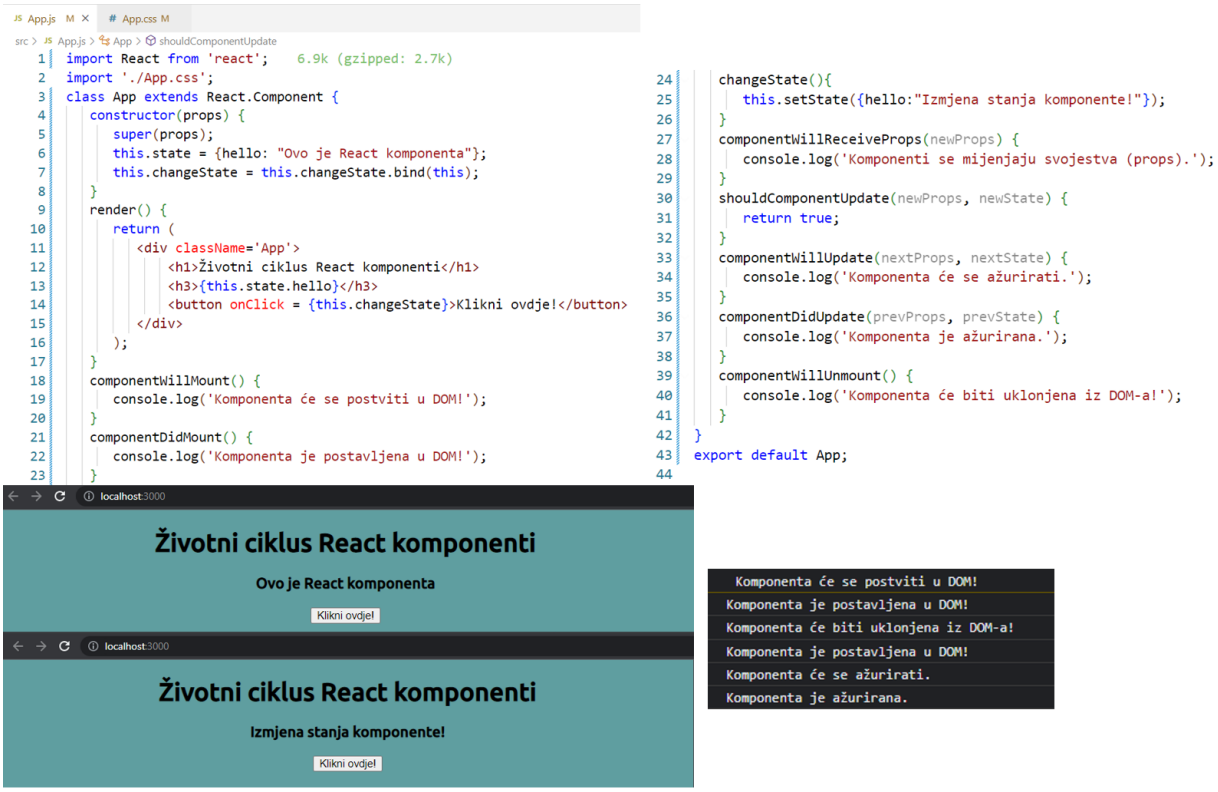
Životni vijek komponente čini niz metoda koje se pozivaju u različitim fazama postojanja komponente. Nazivamo ih ***lifecycle method***s:

1. početna faza – jednokratno kreiranje komponente u kojoj komponenta sadrži početna svojstva (props) i stanja (state), što definiramo u konstruktoru komponente. Koristimo metode **getDefaultProps()** i **getInitalState()**

2. kreiranje instance **komponente** i **postavljanje sadržaja** koji komponenta vraća **u DOM** (engl. *mounting*) – u ovoj fazi možemo koristiti metode **componentWillMount()**, **componentDidMount()** i **render()**. Jedina metoda koja je obvezna jest render()

3. ažuriranje komponente (engl. updating) – u ovoj fazi na temelju interakcije s korisnikom imamo promjenu svojstava (props) i mijenjamo stanja (state) komponente. Cilj nam je prikazati najnoviju verziju komponente. Radi se o fazi koja se ponavlja po potrebi. Možemo koristiti metode **componentWillReceiveProps()**, **shouldComponentUpdate()**, **componentWillUpdate()**, **componentDidUpdate()**, **render()**. Jedina metoda koja je obvezna jest render()

4. uklanjanje komponente iz DOM-a (engl. *unmounting*) – zadnja faza u životu komponente. Primjenom metode **componentWillUnmount()** prije uklanjanja komponente provodimo čišćenje i oslobađanje resursa.



Slika 7.18. Životni ciklus React komponenti

7.5. Obrada korisnički kreiranih događaja

Kao i JavaScript DOM, i u Reactu možemo kreirati akcije na događaje koje čini korisnik.

- **Događaj** je svaka interakcija između korisnika i aplikacije pri čemu svaka korisnička akcija ima unaprijed definiranu reakciju. Aktiviranjem događaja izvršava se funkcija dodijeljena tom događaju, što obično rezultira promjenom stanja neke komponente ili cijele aplikacije.

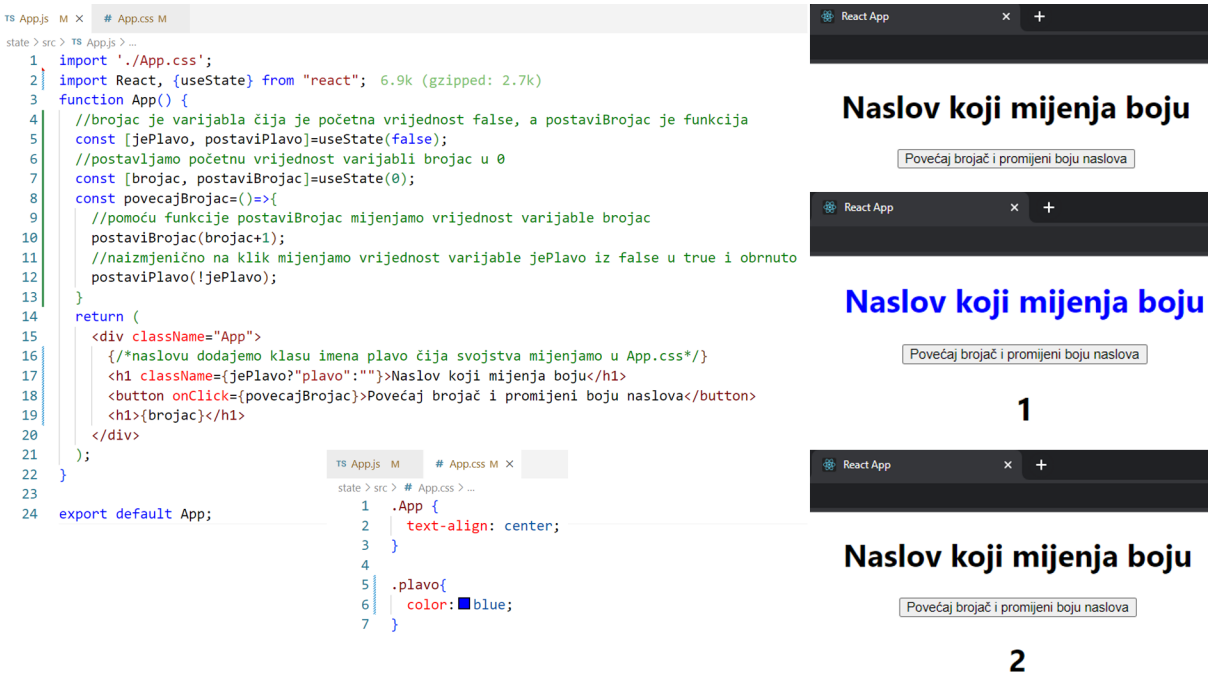
Imamo jednake događaje koje smo koristili u JavaScript DOM-u, ali koristimo **camelCase** pri njihovu zapisu:

HTML DOM	React	Opis
onchange	onChange	promjena unutar elementa aktivira događaj
onclick	onClick	klik na element aktivira događaj
onmouseover	onMouseOver	prelazak pokazivačem miša preko elementa aktivira događaj
onmouseout	onMouseOut	izlazak pokazivača miša izvan elementa aktivira događaj
onload	onLoad	učitavanje sadržaja (primjerice stranice) aktivira događaj

Funkcije koje reagiraju na događaje zapisujemo unutar **{}**.

```
<button onClick=
 {showMessage}>
      Hello!
</button>
```

Na sljedećem primjeru kreirat ćemo konstante određenih vrijednosti i njima pridružene funkcije koje će te vrijednosti mijenjati onClick upotrebom objekta state.



Slika 7.19. Primjeri upotrebe objekta state dinamičkim osvježavanjem vrijednosti

7.6. Rad s obrascima (formama)

Obrasci su dio svake moderne web-aplikacije. Omogućuju **interakciju korisnika i aplikacije**. Obično ih koristimo za autentifikaciju korisnika, dodavanje korisnika, pretraživanje, filtriranje, rezervacije, kontakt i sl.

U Reactu razlikujemo dva tipa unosa (engl. input) korisnika: **kontrolirane i nekontrolirane komponente**. U nastavku promatrat ćemo kontrolirani unos.
U Reactu kreiramo komponente koje rukuju podacima iz obrasca.

Dobar primjer upotrebe **state** objekta jest lista zadataka u kojoj se broj zadataka mijenja, a sa svakom promjenom ponovno se renderira promijenjena komponenta.

Imamo element obrasca za unos teksta te dva gumba. Kad unesemo zadatak u polje za unos i potvrdimo ga klikom na dodaj zadatak, zadaci se ispisuju na stranici uz klikabilan checkbox element. Kad označimo *checkbox*, možemo brisati označene elemente klikom na *Izbriši izvršeni zadatak.*

Slika 7.20. Rad s elementima forme primjenom objekta state

- Za generiranje jedinstvenih id-a preuzmemo i instaliramo **uuid** u naš projekt preko naredbenog retka ili ugrađenog terminala pomoću:

```
npm i uuid
```

a zatim ga uvezemo u App.js:

```
import uuidv4 from 'uuid/v4'
```

Promjenama podataka u obrascu rukujemo pomoću događaja, primjerice elementu obrasca dodjeljujemo atribut kojemu dodjeljujemo vrijednost svojstva stanja komponente.

Kako bi unutar funkcije koja rukuje događajem dohvatili vrijednost elementa obrasca i pohranili ju u odgovarajuće svojstvo, pišemo naziv svojstva u [].

```
/*dodajemo funkcije (hooks) useState- omogućuje nam dohvaćanje stanja,
useRef - omogućuje nam referencijranje elemenata u našem slučaju selemeta za unos teksta
useEffect */
import React, {useState, useRef, useEffect} from "react";  6.9k (gzipped: 2.7k)
import TodoList from "./TodoList";
import './App.css';
/*uvoz biblioteke za kreiranje jedinstvenih id-a*/
import uuidv4 from "/node_modules/uuid/dist/v4";
const LOCAL_STORAGE_KEY='todoApp.todos';

function App() {
  /*U početku nam je varijabla todos (lista zadataka) prazna, pa stvaramo prazno polje,
  drugi parametar je funkcija setTodos koju ćemo pozivati kako bismo ažurirali podatke
  iz liste zadataka*/
  const[todos, postaviZadatak]= useState([ ]);
  const todoIme=useRef();

  /*pozivanje pohranjenih zadataka prilikom ponovnog otvaranja stranice*/
  useEffect(()=>{
    const pohranjeniZadaci=JSON.parse(localStorage.getItem(LOCAL_STORAGE_KEY));
    if (pohranjeniZadaci) postaviZadatak(pohranjeniZadaci);
  },[]);

  /*pohrana u lokalni spremnik (storage)*/
  useEffect(() => {
    localStorage.setItem(LOCAL_STORAGE_KEY, JSON.stringify(todos));
  }, [todos]);

  /*dodavanje zadataka u listu*/
  function  dodajZadatak(){
    const ime=todoIme.current.value;
    if (ime==='') return
    postaviZadatak(ranijiZadaci=>{
      return [...ranijiZadaci,{ id:uuidv4(),ime:ime,izvrsenost:false}];
    })
    todoIme.current.value=null;
  }

  /*označavanje izvršenih zaataka*/
  function postavljanjeIzvrsenosti(id){
    /*stvaramo kopiju liste */
    const newTodos=[...todos];
    const todo=newTodos.find(todo=>todo.id===id);
    todo.izvrsenost=!todo.izvrsenost;
    postaviZadatak(newTodos);
  }

  /*brisanje izvršenih zadataka*/
  function izbrisiZadatak(){
    const newTodos=todos.filter(todo=>!todo.izvrsenost);
    postaviZadatak(newTodos);
    }

  return (
    <div className="App">
      <h1>Lista zadataka</h1>
      <div className="AppBody">
      {/*ugradnja komponente TodoList.js kojoj prosljeđujemo props {todos}*/}
      <TodoList todos={todos} postavljanjeIzvrsenosti={postavljanjeIzvrsenosti}/>
      <input ref={todoIme} type="text" />
      <button onClick={dodajZadatak}>Dodaj zadatak</button>
      <button onClick={izbrisiZadatak}>Izbriši izvršeni zadatak</button>
      <div>Imate {todos.filter(todo=>!todo.izvrsenost).length} zadataka za izvršiti</div>
      </div>
    </div>
  );
}

export default App;
```

Slika 7.21. App.js s funkcijama aplikacije i elementima forme

App.js M | TodoList.js U | Todo.js U | App.css M

todo > src > TodoList.js > ...

```
1  import React from 'react';  6.9k (gzipped: 2.7k)
2  import Todo from './Todo';
3  /*funkciji prosljeđujemo props {todos} */
4  export default function TodoList({todos, postavljanjeIzvrsenosti}) {
5
6    return (
7        todos.map(todo=>{
8          /*key nam omogućuje da se ponovno renderiraju samo elementi liste koji
9          su se promijenili umjesto da se ponovno renderiraju svi elementi*/
10         return <Todo key={todo.ime} postavljanjeIzvrsenosti={postavljanjeIzvrsenosti} todo={todo}/>
11       })
12   );
13 }
```

App.js M | TodoList.js U | Todo.js U | App.css M

todo > src > Todo.js > ...

```
1  import React from 'react';  6.9k (gzipped: 2.7k)
2
3  export default function Todo({todo, postavljanjeIzvrsenosti}) {
4     function klikanjeIzvrsenosti(){
5        postavljanjeIzvrsenosti(todo.id);
6     }
7    return (
8      <div>
9      <label>
10        <input type="checkbox" checked={todo.izvrsenost} onChange={klikanjeIzvrsenosti}/>
11        {todo.ime}
12     </label>
13
14     </div>
15   );
16 }
```

Slika 7.22. Komponente liste aplikacije

- ***useEffect hook*** koristimo kada želimo uzrokovati nuspojave (engl. *side effects*) prije ili poslije iscrtavanja (renderiranja) komponente definirane funkcijom. Može nam zamijeniti glavne lifecycle metode: componentDidMount(), componentDidUpdate() te componentWillUnmount().

App.js M | TodoList.js U | Todo.js U | # App.css M ×

todo > src > # App.css > ...

```css
@import url('https://fonts.googleapis.com/css2?family=Ubuntu:wght@300;500&display=swap');
*{
    margin: 0;
    padding: 0;
}
body{
    background-color: cadetblue;
    font-family: 'Ubuntu', sans-serif;
}
.App{
    font-weight: 500;
    margin: 5vw;
}
.AppBody{
    font-weight: 300;
    padding: 2vw;
}
.AppBody button{
    background-color: rgb(184, 203, 204);
    border: 1px outset black;
    padding: 1vw;
    border-radius: 5px;
    font-family: 'Ubuntu', sans-serif;
    font-weight: 100;
    margin: 1vw;
}
.AppBody input{
    background-color: rgb(236, 241, 241);
    border: 1px outset black;
    padding: 1vw;
    border-radius: 5px;
    font-family: 'Ubuntu', sans-serif;
    font-weight: 100;
    margin: 1vw;
}
```

Slika 7.23. Lista stilova aplikacije

7.7. Testiranje prikaza i funkcionalnosti komponenti aplikacije

Testiranje aplikacije jedna je od najznačajnijih i najvažnijih faza u procesu razvoja životnog ciklusa aplikacije. Testiranje je metoda koja se koristi za **procjenu funkcionalnosti aplikacije**. Provjeravamo na koji se način renderira (iscrtava) određena komponenta u ovisnosti o interakciji s korisnikom.

Izrada automatiziranih testova odnosi se na pisanje programskog koda koji provjerava reagiraju li komponente onako kako se to od njih očekuje, odnosno kako je specificirano prilikom izrade aplikacije.

Najčešće metodologije testiranja web-aplikacija jesu:

- **jedinični testovi** (engl. *unit testing*) – testiramo različite pojedinačne metode, svojstva i interaktivne korisničke elemente. Čine bazu testnog paketa aplikacije. Obično su male i brojne. Postoji pristup u kojemu devel operi prvo izrađuju testove, a tek onda pišu programski kod aplikacije – engl. *Test Driven Development (TDD)*
- **integracijski testovi** (engl. *integration test)* – testiramo kako različiti moduli funkcioniraju kao cjelina, primjerice komunikacija s bazom podataka i sl.
- **funkcionalni testovi** (engl. *functional test*) – promatramo aplikaciju s korisničke pozicije i provjeravamo radi li aplikacija kako se od nje očekuje.

Slika 7.24. Hijerarhija provedbe automatiziranih testova

U Reactu za izradu automatiziranih testova možemo, primjerice, koristiti biblioteke **Jest – test.js** (*JavaScript Testing Framework*), koja je već instalirana izradom React aplikacije, **Selenium** i mnoge druge.

Testiranje koda > {} package.json > ...

```
1  {
2    "devDependencies": {
3      "jest": "^27.5.1"
4    },
       ▷ Debug
5    "scripts": {
6      "test": "jest"
7    }
8  }
```

Jest je testni okvir koji je razvio Facebook. Potrebno ga je instalirati:

```
npm install --save-dev jest
```

Zatim dodajemo test naredbu u package.json.

Slika 7.25. Dodavanje testa u package.json

Izrađujemo skriptu koju ćemo testirati (script.js) i testnu skriptu (script.test.js).

Testiranje koda > js > JS script.js > ...

```
1  function zbroj(a, b){
2      return a + b;
3  }
4  module.exports={zbroj};
```

Testiranje koda > js > __tests__ > JS script.test.js > ...

```
1  const mojeFunkcije = require("../script");
2  /* identična vrijednost (vrijednost i tip) */
   Run | Debug
3  test('1 + 2 jednako 3', () => {
4      expect(mojeFunkcije.zbroj(1, 2)).toEqual(3);
5  });
```

Slika 7.26. Funkcija koju testiramo i test za funkciju

Test pokrećemo tako što u terminal upišemo npm test ili kliknemo na Run u testnoj skripti.

```
PASS  js/__tests__/script.test.js
  √ 1 + 2 jednako 3 (2 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        3.808 s
```

Slika 7.27. Rezultat testiranja

Selenium je radni okvir za testiranje koji možemo dodati u preglednike Mozilla Firefox i Google Chrome.

- Preuzmemo instalacijske datoteke s https://github.com/mozilla/geckodriver/releases/

  i postavimo u mapu SeleniumWebrivers, primjerice na disku C.

U dijelu sustava Uređivanje varijabli okruženja sustava biramo -> Svojstva sustava -> Varijable okruženja -> Path -> Uređivanje -> Novo -> C:\SeleniumWebdrivers

U node.js command promptu prebacimo se u mapu C:\SeleniumWebdrivers i pokrenemo pomoću

```
start geckodriver.
```

U željenoj mapi kreiramo i pokrenemo

```
npm init
```

te pomoću Enter potvrdimo sve ponuđeno. U našoj mapi sad je package.json, a mi ćemo kreirati index.js.

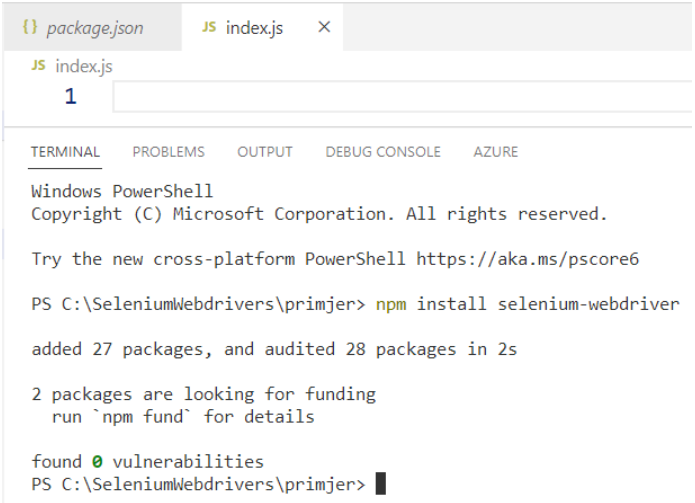
package.json > ...

```
{
  "name": "primjer",
  "version": "1.0.0",
  "description": "\tSelenium Example Project",
  "main": "index.js",
  ▷ Debug
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC"
}
```

Slika 7.28. Primjer package.json

Potrebno je instalirati Selenium webdriver, što možemo učiniti u terminalu VS Coda.

```
npm install selenium-webdriver
```



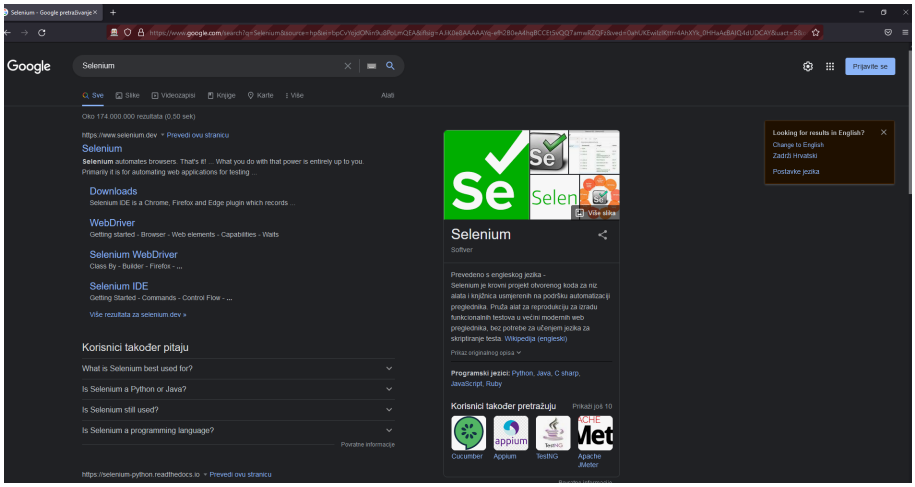
Slika 7.29. Instalacija Selenium webdrivera

```
index.js  ×  {} package.json
index.js > ...
1  const {Builder, By, Key, until} = require('selenium-webdriver');  273.2k (gzipped: 81.2k)
2  async function primjerSelenium() {
3    let driver = await new Builder().forBrowser('firefox').build();
4      await driver.get('http://www.google.com');
5      await driver.findElement(By.name('q')).sendKeys('Selenium', Key.RETURN);
6  }
7  primjerSelenium();
```

Slika 7.30. Automatizirani zahtjev za Selenium

Sad su nam instalirani moduli i ovisnosti za Selenium i pišemo kod u index.js.

U terminalu upišemo node index i dobijemo rezultat u pregledniku.

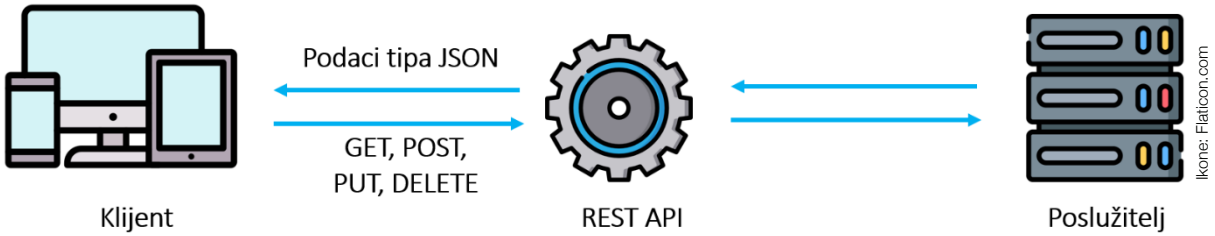


Slika 7.31. Rezultat izvođenja automatiziranog zahtjeva za Selenium

7.8. Komunikacija sa sustavima putem REST API arhitekture

API (engl. *Application Programming Interface*) skup je pravila koja određuju kako aplikacije ili uređaji međusobno komuniciraju i povezuju se. Primjerice, komunikacija React *web*-aplikacije s poslužiteljem na kojemu su pohranjeni podaci.

REST (engl. *REpresenational State Transfer*) konvencija je za organizaciju arhitekture mrežnih API-ja. **REST API arhitekturu** čine **klijent**, **poslužitelj** i **resursi**, a zahtjevima se upravlja putem **HTTP**-a. REST API **sučelje** je (posrednik) između klijenta i poslužitelja, postavlja zahtjeve za resurse i isporučuje odgovore.



Slika 7.32. REST API model (arhitektura)

Da bi klijent i poslužitelj mogli učinkovito komunicirali, a istovremeno postojali neovisno, potreban nam je standardiziran komunikacijski protokol, obično HTTP, čime ostvarujemo **interoperabilnost informacijskih sustava**, koji mogu biti pisani različitim programskim jezicima, na različitim platformama i sl.

Zahtjev se šalje od klijenta do poslužitelja u obliku web URL-a kao **HTTP** (GET, POST, PUT ili DELETE) **zahtjev**. Kada klijent uputi zahtjev koristeći RESTful API, poslužitelj prenosi standardizirani prikaz stanja resursa klijentu. Ne šalje se stvarni resurs već prikaz njegova stanja – tj. stanje resursa u određenom trenutku. Glavne metode za rad s resursima od strane klijenta jesu:

- POST – stvaranje novog resursa
- GET – dohvaćanje resursa
- PUT – uređivanje ili ažuriranje postojećeg resursa
- DELETE – brisanje resursa.

JSON API prima zahtjeve i vraća odgovore u formatu JSON (engl. JavaScript Object Notation).

- JSON je popularan **„lagani" format** za pohranu i prijenos informacija na internetu. Nije ovisan o programskom jeziku, a čitljiv je i ljudima i strojevima. Sastoji se od parova **ime: vrijednost:**

```
{"djelatnici":[
    {"prezime":"Ana", "ime":"Anić"},
    {"prezime":"Pero", "ime":"Perić"},
    {"prezime":"Iva", "ime":"Anić"},
    ]
}
```

Odvajanjem problematike korisničkog sučelja od problematike pohrane podataka unapređuje se fleksibilnost sučelja na svim platformama i poboljšava se skalabilnost pojednostavljivanjem komponenata poslužitelja. Uz to, razdvajanje omogućuje svakoj komponenti da se samostalno razvija. Nema bojazni da će komponente nenamjerno mijenjati jedna drugu tijekom neovisnih promjena, pojednostavljujući procese optimizacije i skaliranja.

Svaka komunikacija između klijenta i poslužitelja neovisna je, odnosno svaki je novi zahtjev klijenta prema poslužitelju nov, neovisan o prethodnom. Sa svakim zahtjevom šalju se sve potrebe informacije koje trebaju poslužitelju za slanje odgovora. Poslužitelj ne pohranjuje nikakve informacije o klijentu, čime se ostvaruje brža i pouzdanija komunikacija.

Glavne karakteristike REST servisa:

- nepostojanje stanja (engl. stateless)
- mogućnost keširanja (engl. cacheable)
- jedinstveno sučelje (engl. uniform interface)
- izričito korištenje HTTP metoda te
- korištenje XML i/ili JSON formata poruke za komuniciranje.

Pitanja za ponavljanje

1. Opišite što je React i za što se koristi.
2. Objasnite na koji način postavljamo radnu okolinu za rad s Reactom.
3. Što nam omogućuje JavaScript sintaksno proširenje?
4. Na koje načine možemo kreirati komponente u Reactu?
5. Objasnite razliku između svojstava (props) i stanja (state).
6. Nabrojite životne faze React komponenti.
7. Navedite primjer jednog korisnički kreiranog događaja i kako biste izradili funkciju koja odgovara na taj događaj.
8. Što nam omogućuje upotreba obrazaca?
9. Objasnite postupak testiranja aplikacija.
10. Opišite REST API arhitekturu i nabrojite prednosti upotrebe.

Literatura:

1. **Službena stranica React.**
   Preuzeto 16.6.2022. s https://reactjs.org/tutorial/tutorial.html
2. **React tutorijal.**
   Preuzeto 1.6.2022. s https://www.javatpoint.com/reactjs-tutorial
3. **Započnimo s Reactom**
   Preuzeto 1.6.2022. s https://developer.mozilla.org/en-US/docs/Learn/Tools_and_testing/Client-side_JavaScript_frameworks/React_getting_started
4. **W3 schools React tutorijal.**
   Preuzeto 1.6.2022. s https://www.w3schools.com/react/default.asp
5. **React tutorijal za početnike.**
   Preuzeto 1.6.2022. s https://ibaslogic.com/react-tutorial-for-beginners/
6. **React tutorijal.**
   Preuzeto 1.6.2022. s https://code.visualstudio.com/docs/nodejs/reactjs-tutorial
7. **React tutorijal.**
   Preuzeto 1.6.2022. s https://www.geeksforgeeks.org/reactjs-tutorials/?ref=lbp
8. **10 glavnih koncepata Reacta.**
   Preuzeto 15.6.2022. s https://payalpaul2436.medium.com/10-main-core-concept-you-need-to-know-about-react-303e986e1763

8. Zaštita na radu

U OVOM POGLAVLJU NAUČIT ĆETE:

- razlikovati mjere zaštite na radu
- razlikovati različite izvore opasnosti pri radu s računalom
- navesti mjere zaštite od električnog udara na radnom mjestu
- objasniti upotrebu računala na siguran način primjenom mjera zaštite od ozljeda
- opisati i demonstrirati vježbe opuštanja i razgibavanja radi sprečavanja ozljeda na radnom mjestu

8.1. Osnove zaštite na radu

Zaštita na radu u Republici Hrvatskoj regulirana je Zakonom o zaštiti na radu („Narodne novine", br. 071/2014, 118/2014, 094/2018, 096/2018) te podzakonskim propisima donesenima na temelju Zakona o zaštiti na radu. Svi poslodavci trebaju je organizirati i provoditi.

Zaštita na radu sadrži sustav pravila, mjera i aktivnosti čijom se primjenom osiguravaju sigurni uvjeti na radu te zaštita zdravlja na radu. Cilj zaštite na radu prevencija je, odnosno sprečavanje ozljeda na radu te profesionalnih bolesti koje se mogu javiti prilikom obavljanja nekog posla.

Poslodavac može na različite načine organizirati zaštitu na radu, a to ovisi o djelatnosti kojom se bavi i o broju djelatnika. Razina opasnosti utvrđuje se postupkom procjene rizika, što poslodavac provodi za sve poslove, odnosno sva postojeća radna mjesta unutar ustanove.

Osnovni pojmovi vezani uz zaštitu na radu jesu:

- **Poslodavac** je fizička ili pravna osoba za koju radnik obavlja posao.
- **Radnik** je fizička osoba koja obavlja poslove za poslodavca.
- **Mjesto rada** svako je mjesto na kojemu se obavljaju poslovi iz djelokruga poslodavca.
- **Izdvojeno mjesto rada** mjesto je na kojemu radnik obavlja posao, a nije prostor poslodavca, može biti kod kuće ili u drugom prostoru.
- **Povjerenik** za zaštitu na radu osoba je koja je izabrana da zastupa prava radnika iz područja zaštite na radu.
- **Stručnjak zaštite na radu** osoba je koju je poslodavac odredio za obavljanje poslova zaštite na radu.
- **Ozljeda na radu** ozljeda je koju je radnik zadobio tijekom rada u prostoru rada, bilo kod poslodavca ili drugom prostoru gdje se obavlja rad.
- **Profesionalna bolest** jest bolest nastala kao posljedica djelovanja štetnosti u tijeku rada i/ili radnoj okolini.
- **Stres** predstavljaju zdravstvene i psihičke promjene koje su nastale kao posljedica kontinuiranog utjecaja izvora stresa na radu kroz duži vremenski period.

Slika 8.1. Ilustracija zaštite na radu

Nezgodom na radu smatra se svaki neželjeni, nepredviđeni događaj koji za posljedicu ima ozljedu na radu koja dovodi do bolovanja, materijalnih gubitaka radnika i poslodavca zbog izgubljenih sati rada ili materijalni gubitak zbog oštećenja opreme ili zastoja u proizvodnom procesu. Nezgode na radu nastaju najčešće kao posljedica ljudske pogreške jer radnik ne zna sigurno raditi, ne može raditi na siguran način ili ne želi raditi na siguran način.

Može biti da radnik nije osposobljen za rad na siguran način i da ne zna do kojih posljedica njegov način rada može dovesti.

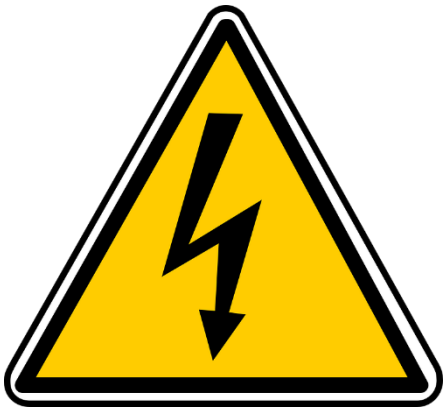
Najčešće zbog zdravstvenih razloga radnik ne može raditi na siguran način. U tom slučaju poslodavac je dužan rasporediti ga na poslove koje je u mogućnosti obavljati.

Kad radnik ne želi raditi na siguran način iako poznaje mjere zaštite na radu, on ugrožava sebe i druge djelatnike i poslodavac mora reagirati na adekvatan način, upozorenjem, razgovorom ili ga udaljiti s tog radnog mjesta.

8.2. Opasnosti i način zaštite na radnom mjestu

Izvori opasnosti pri radu s računalom mogu biti:

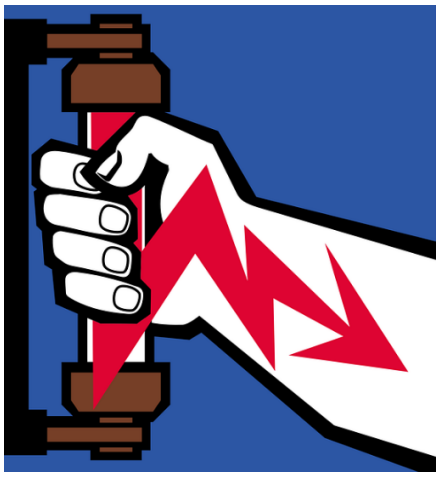
- opasnost od električkog udara
- opasnosti od buke
- opasnosti od štetnih zračenja
- opasnosti od požara i eksplozija, sredstva za zaštitu od požara.



Opasnosti od električkog udara izloženi su svi zaposlenici i posjetitelji ustanove kada električne instalacije i informatička oprema i drugi uređaji nisu ispravni ili pravilno održavani.

Električna struja ako prolazi kroz ljudski organizam može izazvati teške ozljede ili smrtne posljedice.

Slika 8.2. Opasnost od strujnog udara



Slika 8.3. Ilustracija strujnog udara

Nezgode od strujnog udara događaju se zbog izravnog dodira s neispravnom opremom ili oštećenom izolacijom na kablovima, sklopkama, utičnicama i slično. Elektroničku opremu i električne instalacije potrebno je periodički redovito pregledavati i održavati i na taj način osigurati da ne postoji opasnost od izravnog slučajnog dodira.

Kod uočenih smetnji ili kvarova potrebno je odmah isključiti napon ili izvući utikač uređaja iz utičnice. Isto tako ne smiju se dirati oštećeni prekidači ili utikači jer mogu biti pod naponom. Prilikom zamjene žarulje potrebno je isključiti napon. U slučaju pregrijane ili na dodir vruće utičnice ili uređaja potrebo ih je isključiti kako ne bi izazvali požar, a nakon toga pozvati električara da otkloni kvar.

U slučaju ozljede električnim udarom i ako se ozlijeđeni nalazi u strujnom krugu, potrebno ga je osloboditi od strujnog kruga poštujući mjere sigurnosti za sebe i ozlijeđenog. Najsigurnija opcija isključiti je strujni krug prekidačem ili uklanjanjem vodiča iz blizine ozlijeđenog. U tome vam može pomoći komad suhog drveta.

Osiguranje od buke jedan je segment koji također utječe na sigurnost zaposlenika. Bukom se smatra svaki neželjeni zvuk koji je zaposleniku neugodan ili mu smeta. Djelovanje buke na čovjeka ovisi o njezinoj jačini i frekvenciji. Buka može izazvati gubitak sluha, napetost, smanjenje koncentracije, probleme s razumijevanjem govora, umor i razdražljivost. Posebno je opasno oštećenje sluha jer ima trajnu posljedicu. Sluh izgubljen zbog buke ne može se liječiti. Buka se razlikuje po visini frekvencije, a buka visokih frekvencija štetnija je od buke dubokih tonova.

Buku izražavamo u mjernim jedinicama:

- decibelima (dB) – intenzitet buke
- Hertzima (Hz) – frekvencija.

Prag čujnosti je intenzitet buke 0 dB kod frekvencije 1000 Hz. Kod opremanja ureda treba paziti da uređaji koji se koriste ne stvaraju buku veću od 60 dB, koliko je propisano pravilnikom za takve prostore. Ukoliko je za rad potrebna oprema koja stvara buku veću od propisane, potrebno ju je smjestiti u zasebnu prostoriju. Stalna izloženost buci manje je opasna za zdravlje zaposlenika od povremene zato što se organizam čovjeka prilagođava okolini u kojoj stalno boravi. Osobna zaštitna sredstva koja se mogu koristiti slušalice su i čepići.

Čovjek je svakodnevno izložen nekim vrsta štetnog zračenja i one mogu utjecati na njegovo zdravstveno stanje. Dijele se na ionizirajuća zračenja, čiji su izvor radioaktivni elementi i susreću se u zdravstvenoj djelatnosti, te neionizirajuća zračenja, koja proizvode određeni uređaji. U neionizirajuća zračenja spadaju elektromegnetsko polje i elektromegnetski valovi.

Izvori neionizirajućih elektromagnetskih polja kojima su zaposlenici u uredu svakodnevno izloženi jesu antene baznih postaja, radarski sustava, televizijskih i radijskih postaja, mikrovalne pećnice, mobilni telefoni, bežični internet, bežični telefoni, daljinski upravljači, indukcijske ploče za kuhanje te protuprovalni sustavi. Udaljenost od takvih uređaja smanjuje razinu zračenja.

Slika 8.4. Opasnost od elektromagnetskog polja

Preporuke za prevenciju odnose se na izbjegavanje prekomjerne upotrebe mobilnih uređaja, izbjegavanje boravka pored uređaja koji su izvor elektromagnetskih polja, izbjegavanje kontinuiranog nošenja mobilnih uređaja uz tijelo, korištenje slušalica tijekom razgovora ili pisanje poruka umjesto telefonskih razgovora.

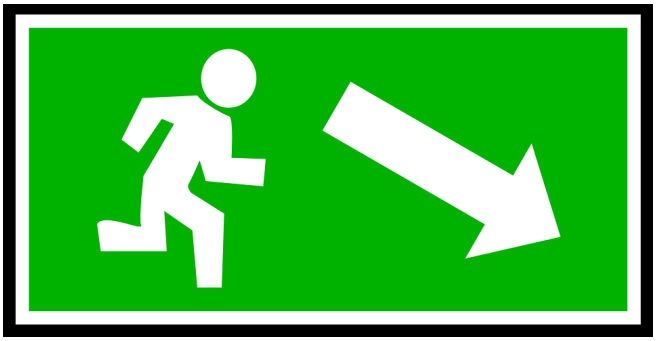
Uzrok požara može biti neodgovarajuća upotreba radne opreme ili kvar na elektroinstalacijama, kao i nepažnja zaposlenika ili posjetitelja. U slučaju curenja plina može doći do eksplozije jer čak i iskra prekidača može izazvati eksploziju. Mjere prevencije redovito su održavanje plinskih uređaja i instalacija, kao i postavljanje detektora. Na-

Slika 8.4. Opasnost od požara

jvažnije je da zaposlenici budu osposobljeni za zaštitu na radu iz područja zaštite od požara kako bi, u slučaju nastanka požara, mogli intervenirati brzo i učinkovito. Uredi moraju imati vatrogasne aparate za slučaj požara, a zaposlenici trebaju znati kako njima rukovati. Vatrogasni aparati moraju se servisirati jednom godišnje, pristup do njih uvijek mora biti slobodan. Ustanova mora imati jasno istaknut plan evakuacije, a putevi predviđeni za evakuaciju u svakom trenutku trebaju biti prohodni. Ista pravila vrijede i za hidrante, kojima uvijek mora biti dostupan pristup i u blizini smješten ormarić s opremom za gašenje požara.

Mjere prevencije osposobljenost su zaposlenika za brzu reakciju i početno gašenje požara, svakodnevni pregled radnog mjesta, isključivanje opreme i rasvjete i slično.

U slučaju opeklina opečeni dio staviti pod tekuću čistu vodu ili u posudu s hladnom vodom. Potrebno je skinuti nakit i odjeću s opečenog dijela tijela. Ako postoji dio odjeće koji je zalijepljen za opeklinu, ne diramo ga. Iznimno dio odjeće uklanjamo s opekline kada je riječ o kemikalijama koje mogu izazvati dodatna oštećenja kože. Nastale plikove ne dirati i ne bušiti. Opeklinu zaštititi sterilnom gazom i previti zavojem ili maramom. Ukoliko se radi o većim opeklinama, ozlijeđenom treba dati da pije što više tekućine.



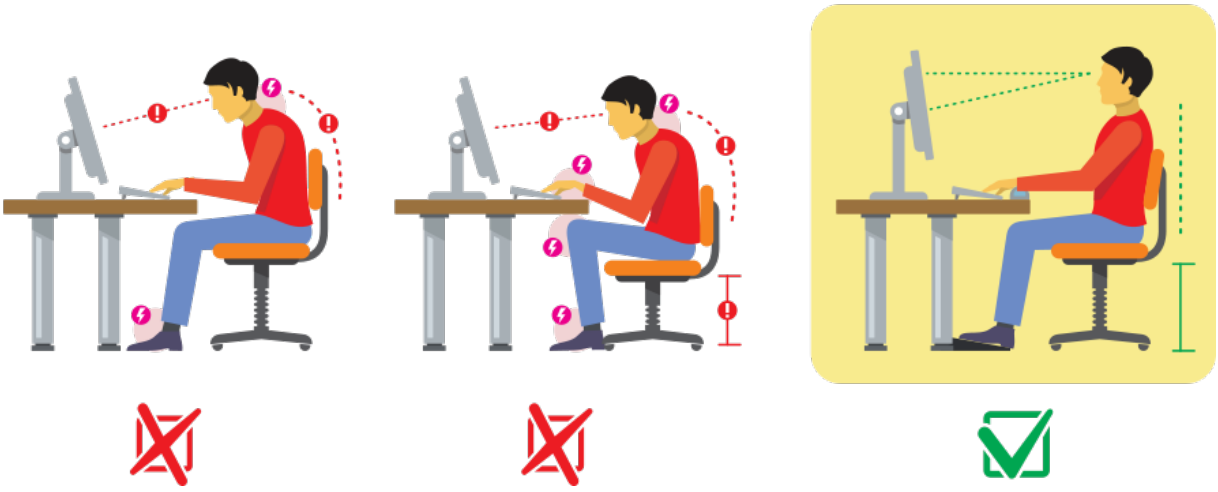
Slika 8.5. Oznaka puta za evakuaciju

Obveza je poslodavca osigurati sredstva za rad i osobnu zaštitnu oprema zaposlenika kako bi bili sigurni pri radu, kao i održavati istu. Oštećenu opremu ili sredstva osobne zaštite mora ukloniti i isključiti iz upotrebe ako na njoj postoje oštećenja i kao takva predstavljaju rizik za djelatnike.

8.3. Ergonomija i mjere prevencije pri radu s računalom

U radu s računalom treba osigurati da radnik može često mijenjati svoj položaj te da oprema ne bude izvor opasnosti od ozljede ili oštećenja zdravlja djelatnika. Poteškoće uzrokuju dugotrajno sjedenje u istom položaju, dugotrajno gledanje u zaslon, ponavljajući pokreti izazivaju bolove u mišićima i zglobovima. Bolesti povezane uz rad s računalom jesu bolesti vida, mišićnog sustava, koštano-zglobnog i kardiovaskularnog sustava. Nažalost, posljedice se osjete tek u kroničnoj fazi bolesti.

Ergonomija treba rad učiniti što ugodnijim i sigurnijim te smanjiti broj profesionalnih oboljenja. Ergonomija se koristi znanstvenim informacijama prilikom dizajniranja radne opreme i radne okoline.



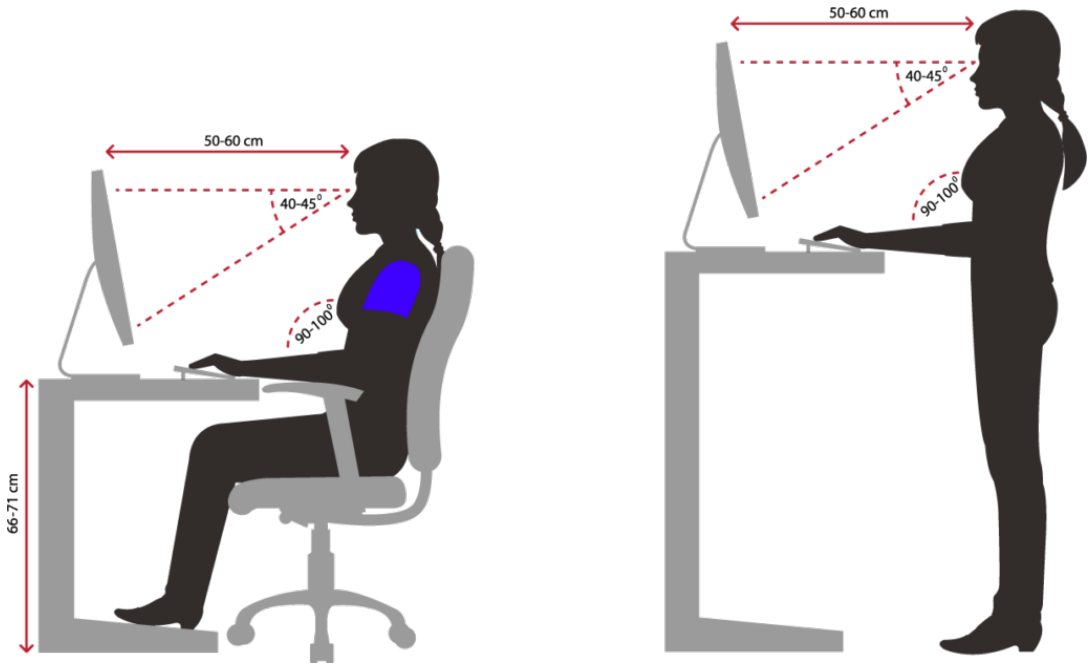
Slika 8.6. Ispravan i neispravan položaj tijela pri radu s računalom

Radno mjesto potrebno je prilagoditi svakom djelatniku. Pravilnim držanjem tijekom rada s računalom moguće je izbjeći zdravstvene probleme.

Preporučeni položaji za:

- **ruke i nadlaktice** – vodoravan položaj, paralelan s površinom poda
- **glava** – ravan položaj ili lagano naginjanje prema naprijed
- **ramena** – opuštena s nadlakticama koje vise uz tijelo
- **laktovi** – položeni uz tijelo pod kutom između 90° i 120°
- **stopala** – potpuno spuštena na pod ili oslonac za noge
- **leđa** – u potpunosti naslonjena na naslon stolice
- **koljena** – u ravnini s bokovima, savijena pod 90°.

Osnovni element uredske opreme je radni stol jer određuje raspored ostalih uređaja, a i položaj tijela tijekom rada. Radni stol mora imati dovoljno mjesta da se na njega smjeste svi potrebni uređaji za rad i da omogući djelatniku komotan rad s monitorom na njemu. Odnosno, dubina stola mora omogućiti da na stol stanu ruke i podlaktica dok se koristi tipkovnica. Poželjno je da radni stol bude podesiv po visini i da ima stalak za miš i tipkovnicu. Preporučena visina sjedećeg stola je između 66 i 71 cm. Minimum širine stola je 120 cm. Ispod stola treba biti dovoljno prostora za udobno sjedenje.



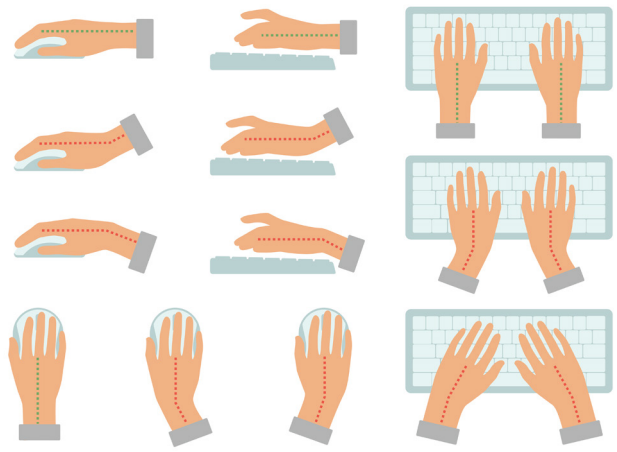
Slika 8.7. Ispravan položaj tijela kod podesivih stolova

Kod stola za rad u stajaćem položaju visina radnog stola mora biti između 95 i 118 cm, ovisno o visini zaposlenika. Kod ovakve vrste stolova potrebno je paziti da se osigura potrebna dubina za stopala, 15 cm u dubinu i 12 cm u visinu.

Zaslon monitora od očiju treba biti udaljen najmanje 50 cm. Gornji rub zaslona treba biti u visini očiju. Monitor treba biti pomičan kako bi svaki djelatnik, ovisno o položaju za vrijeme rada, mogao prilagoditi njegov smjer i nagib.

Uredska stolica treba biti stabilna, prilagodljive visine, tako da svakom djelatniku može omogućiti udoban položaj i prilagodbu. Naslon mora biti potpora cijelom dužinom leđa i imati mogućnost podešavanja nagiba i visine. Stolica mora biti pokretna na 5 kotačića.

Zadovoljavajuću razinu osvijetljenosti prostorije treba osigurati općom i/ili lokalnom rasvjetom, kao i primjeren kontrast između zaslona i pozadine na računalu, poštujući specifične zahtjeve djelatnika i njegova radnog mjesta. Minimalna propisna razina osvijetljenosti za radna mjesta za računalom jest 300 luxa. Zaslon mora biti namješten i nagnut tako da se izbjegne zrcaljenje rasvjete na zaslonu. Problem refleksije i bliještanja moguće je spriječiti pravilnim smještajem radnog mjesta i uređaja sa zaslonom u odnosu na izvori svjetlosti, prozore, druge otvore ili svijetle površine. Kako bi se spriječio ulazak Sunčeve svjetlosti, na prozorima trebaju biti odgovarajući zastori ili kapci. Zaslon treba biti postavljen pod kutom od 90° prema prozoru ili nekom drugom izvoru svjetla.



Slika 8.8. Pravilan i nepravilan položaj ruku pri upotrebi miša i tipkovnice

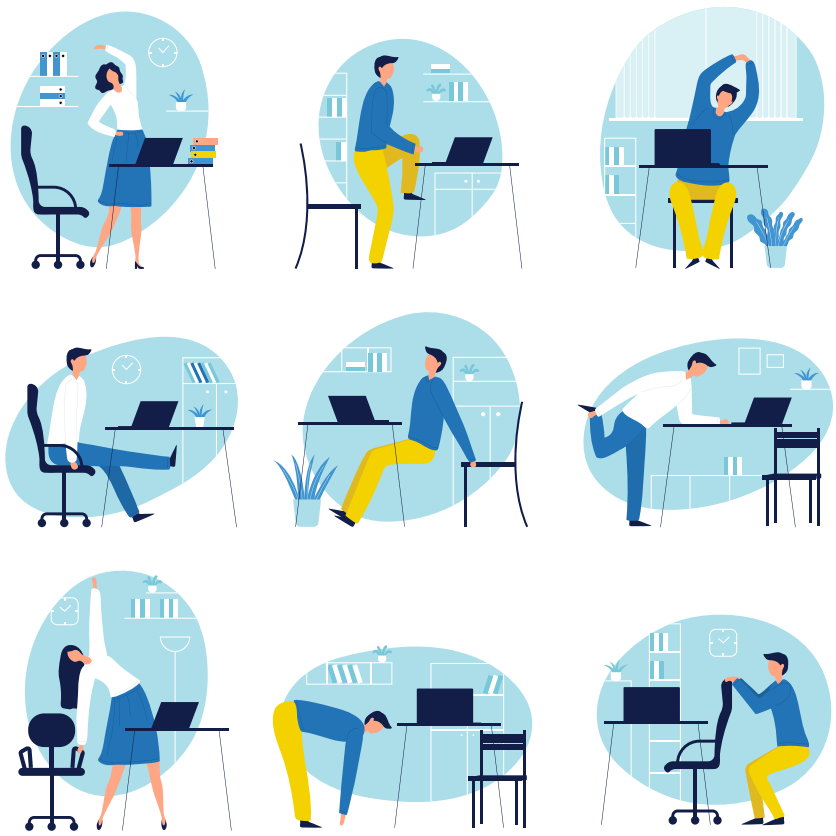
Tijekom upotrebe miša i tipkovnice treba paziti na pravilan položaj ruku kako ne bi došlo do ozljede zbog opterećenja zglobova, ruke i šake prilikom ponavljajućih pokreta (engl. *repetiton strain injury*).

Za prirodno držanje ruku poželjno je da tipkovnica bude pokretna na radnoj površini stola. Poželjno

je koristiti ergonomske tipkovnice koje imaju raspored tipki prilagođen prirodnom položaju ruku. Nagib tipkovnice na podlogu trebao bi iznositi 5 – 10°, visina tipkovnice 3 cm, a visina tipki 12 – 15 mm s razmakom između tipki 18 – 20 mm.

Preporuke za prevenciju ozljeda ruku jesu:

- šaka i prsti moraju biti u ravnini s podlakticom
- prsti trebaju biti ravni u odnosu na šaku, bez savijanja zgloba
- razgibavati zglobove i prste više puta tijekom dana
- tipkovnice moraju biti na ravnoj i tvrdoj podlozi i udaljene tako da ispred njih na stol stane cijela podlaktica
- miš i tipkovnica moraju biti dostupni iz istog položaja
- pomicati cijelu ruku pri upotrebi miša, a ne samo zglobove
- ramena trebaju biti opuštena
- stolica treba imati naslon za ruke.



Slika 8.10. Primjer vježbi rasterećenja

Za očuvanje zdravlja kod rada u uredu pravilno sjedenje nije dovoljno, već je od velike važnosti promijeniti položaj tijela ili aktivnosti tijekom rada. Kod organizacije posla treba omogućiti radniku promjenu aktivnosti ili stanku od 5 minuta nakon svakog sata rada. U toj pauzi mogu se organizirati vježbe rasterećenja. Preporuka je više puta tijekom dana uraditi vježbe rasterećenja, koje će pomoći u smanjenju napora tijekom rada.

Vijeće ministara europske zajednice 1989. i 1990. godine donijelo je odluku o unošenju Direktive o zdravlju i zaštiti na radu u povelju o socijalnim pravima (engl. Social Charter). Direktiva 89/391/EEC ima namjeru poticanja poboljšanja zdravstvenih i sigurnosnih uvjeta na radu. Odnosi se na radni prostor, radnu opremu, osobnu zaštitnu opremu, ručno rukovanje teškim predmetima, rad s monitorima, rad s kancerogenim tvarima te rad s biološkim agentima.

Posebnu važnost ima direktiva 90/270/EEC o minimalnim zahtjevima u pogledu sigurnosti i zaštite zdravlja pri radu sa zaslonima, ali isključuje prijenosna računala.

Ova direktiva definira zahtjeve za:

- **monitore** – oblik slova, veličina i definicija, stabilnost i treperenje slike, kon trola svjetline i kontrasta, podešavanje kuta zaslona, sjaj i refleksija
- **tipkovnice** – podešavanje nagiba i odvajanje od zaslona, podrška za šaku i ruku, sjaj i refleksija, raspored tipki i čitljivost
- **radne površine** – sjaj i refleksija, prostor za opremu, držači dokumenata
- **stolice** – podešavanje i oslonac za noge
- **radnu okolinu** – rasvjeta, sjaj i refleksija, buka, toplina, zračenje i vlažnost zraka
- **korisnička sučelja** – prikladnost softvera, lakoća korištenja prilagođena vještini korisnika, vrijeme odziva i svojstva, izgled.

Pitanja za ponavljanje

1. Kojim su zakonskim aktima propisane mjere zaštite na radu?
2. Navedite i opišite osnovne pojmove vezane uz zaštitu na radu.
3. Koje su obveze poslodavca?
4. Koji su izvori opasnosti pri radu s računalom?
5. Nabrojite mjere prevencije strujnog udara.
6. Kako tretirati opekline u slučaju požara?
7. Nabrojite i opišite mjere prevencije rada s računalom.
8. Što propisuje direktiva 90/270/EEC?

„Sadržaj publikacije/emitiranog materijala isključiva je odgovornost RCK ELPROS“

Projekt je sufinancirala Europska unija iz Europskog socijalnog fonda.