# Analiza CSS okvira za web aplikacije

**Kerečeni, Luka**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Luka Kerečeni**

# ANALIZA CSS OKVIRA ZA WEB APLIKACIJE

**ZAVRŠNI RAD**

**Varaždin, 2021.**

**SVEUČILIŠTE U ZAGREBU**

**FAKULTET ORGANIZACIJE I INFORMATIKE**

**V A R A Ž D I N**

**Luka Kerečeni**

**Matični broj: 45091**

**Studij: Informacijski sustavi**

# ANALIZA CSS OKVIRA ZA WEB APLIKACIJE

**ZAVRŠNI RAD**

**Mentor:**

Prof. dr. sc. Dragutin Kermek

**Varaždin, rujan 2021**

*Luka Kerečeni*

**Izjava o izvornosti**

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

*Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi*

---

**Sažetak**

Svrha ovog rada je analiza i usporedba CSS okvira za web aplikacije. U teorijskom dijelu rada objašnjava se pojam Weba kao platforme, dizajn korisničkog iskustva i korisničkog sučelja te se obrađuju glavni dijelovi CSSa i CSS preprocesora. Također, kroz rad se analizira i uspoređuje nekoliko najpopularnijih i najkorištenijih CSS okvira te se pomoću odabranog CSS okvira izrađuje web aplikacija. Prilikom izrade web aplikacije koristi se razvojno okruženje Visual Studio Code. Cijela aplikacija programirana je pomoću React biblioteke i TypeScripta, dok su glavni koraci prilikom izrade web aplikacije detaljno opisani i prikazani pomoću slika. Nakon izrade, aplikacija je objavljena na servisima poput Netlifyja ili Vercela i postala javno dostupna za korištenje, a programsko rješenje u cijelosti je dostupno na GitHubu.

**Ključne riječi:** web, web aplikacija, korisničko sučelje, analiza, CSS, CSS okvir, CSS preprocesor

# Sadržaj

# 1. Uvod

Ubrzan razvoj interneta posljednjih godina doveo je do pojave mnogo novih servisa i mogućnosti. Mogućnosti koje nudi internet kao mreža ne mogu se usporediti ni sa kojim drugim rješenjem. Razvoj interneta ne samo da je olakšao i ubrzao protok informacija, već je stvoreno mnogo radnih mjesta i struka u području koje praktički nije ni postojalo prije nekog vremena. Kada govorimo o internet servisima prva pomisao pada na web aplikacije. No naravno, ovo je ogromno područje, mnogo kompleksnije nego što se zapravo smatra na prvu. Web programiranje nikad nije bilo kompleksnije sa većom potrebom uporabe raznih, a često i ne povezanih alata i tehnika, stoga se web programeri u današnje vrijeme moraju konstantno nadograđivati u znanju i biti u toku s vremenom i tehnologijama.

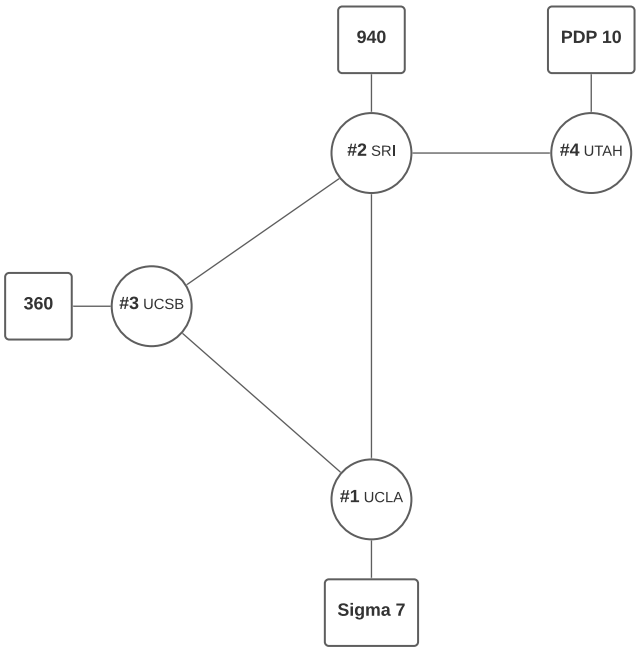
Cilj ovog rada je predočiti i približiti napredak web tehnologija i procesa od samih početaka do današnjice. U početku rada biti će objašnjen sam početak Interneta te njegov razvoj kroz nadolazeće godine, pa sve do pojave Web-a kao platforme. Nakon razjašnjenja osnovnih pojmova Interneta i Web-a na red dolazi tema kojoj se u današnje vrijeme pridaje velika pažnja prilikom izrade neke aplikacije. Budući da je čovjek, između ostalog i vizualno biće, izuzetno je važno kako će krajnji proizvod izgledati i biti prezentiran korisnicima. U samim počecima izrade web aplikacija dizajnu se nije pridavala velika pažnja već je bilo važnije da krajnji proizvod funkcionira kako je programer to zamislio. U današnje vrijeme proces dizajniranja aplikacije nikad nije bio kompleksniji i zahtjevniji. Velika pažnja posvećuje se detaljima i korisničkom iskustvu jer to razlikuje jednu dobru aplikaciju koju će korisnik htjeti koristiti ponovno od one koju će koristiti eventualno jednom i nikad više. Nakon dizajna na red dolazi poglavlje o glavnim blokovima svake web aplikacije – HTML-u i CSS-u. Napretkom tehnologije dolazi i do razvoja pomoćnih alata koji znatno olakšavaju proces programiranja, a neki od njih su svakako CSS preprocesori i CSS okviri. Kako bi lakše odabrali neki od mnogih pomoćnih alata, napravljena je analiza i usporedba nekoliko najkorištenijih. Teorija je sastavni i važan dio svakog područja, ali važno je teoriju pretočiti i u praksu. Budući da je tema ovog rada analiza CSS okvira za web aplikacije, u posljednjem poglavlju biti će prikazan i objašnjen proces izrade male web aplikacije pomoću odabranih tehnologija i CSS okvira Tailwind CSS.

# 2. Web kao platforma

Budući da se često pojmovi Interneta i Weba poistovjećuju, ovo poglavlje pojasnit će razliku između ova dva pojma, nastanak i arhitekturu Interneta, što je to World Wide Web i kako Web zapravo funkcionira.

## 2.1. Internet

Prvi korak k nastajanju Interneta kakvog danas poznajemo napravljen je krajem 60-tih godina prošlog stoljeća formiranjem mreže pod nazivom ARPANET. ARPANET je nastao vrlo spontano, bez velikih planova o razvitku velike globalne mreže kakva postoji danas. Zadaća tadašnje uspostavljene mreže bila je razmjena podataka i informacija među znanstvenicima diljem SAD-a. Početnu mrežu činila su dva spojena računala, računalo na Sveučilištu UCLA i SRI sa Stanforda, a kasnije su se mreži priključila još dva računala, računalo sa UCSB-a (Santa Barbara) te računalo iz Utaha. (Slika 1.)



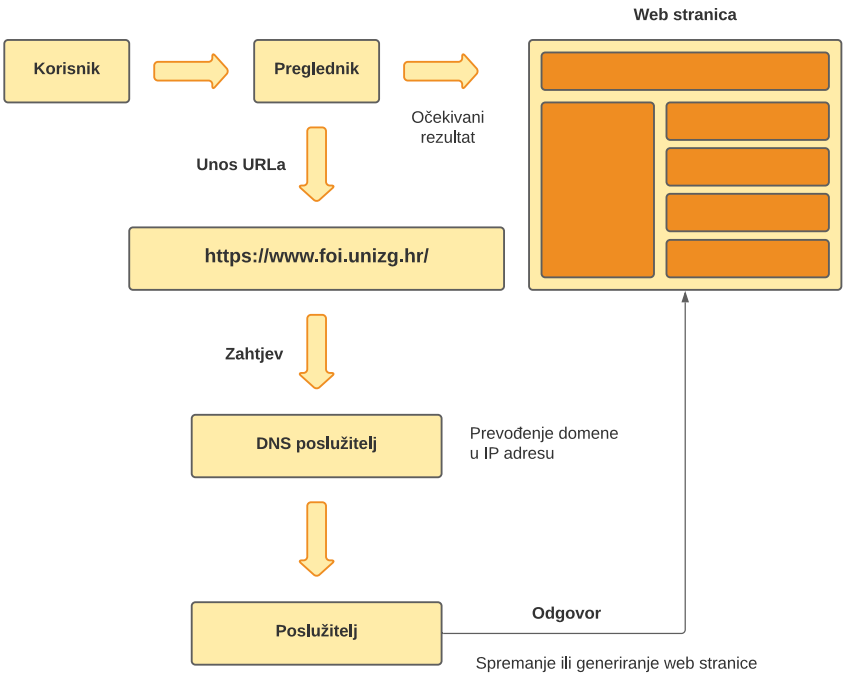

Slika 1: ARPANET, shematski prikaz mreže (Prema: Alex McKenzie, 2009)

Svakom idućom godinom broj umreženih računala nastavio je rasti pa je tako krajem 1972. godine bilo četrdesetak umreženih, što vojnih, što civilnih računala. Istovremeno dolazi

do pojave drugih računalnih mreža koje se vremenom ujedinjuju i u tom trenutku nastaje Internet. Internet je javno dostupna globalna mreža koja povezuje računala i računalne mreže diljem svijeta putem IP protokola. Važno je znati da svako računalo u mreži ima svoju jednoznačnu IP adresu, a svaka poruka koja se šalje cijepa se u pakete. Način na koji će se cijepati i ponovno sastavljati poruke određuje TCP protokol. Svaki paket ima svoju ishodišnu i odredišnu adresu i broj paketa. Nakon što paketi stignu na odredište ponovno se sastavljaju te čine cjelovitu poruku.

## 2.2. World Wide Web

Prijelomni trenutak koji je doveo do ubrzanog razvoja Interneta te uvođenje istog u svakodnevicu ljudi je uspostava mrežne usluge World Wide Web (WWW ili samo Web). Web je jedna u nizu usluga koje pruža Internet, a koja omogućuje pregled hipertekstualnih dokumenata. Hipertekstualni dokument može sadržavati slike, tekst ili neki drugi multimedijski sadržaj, a međusobno su povezani pomoću hiperpoveznica. Za prikaz i korištenje njihovog sadržaja koriste se web preglednici. Ideja međusobno povezanih dokumenata na računalu nastala je još u 1940-tim od strane Vannevara Busha, dok se prvi online hipertekst počinje razvijati u 1960-tim. Godine 1989. Tim Berners Lee sa European Particle Physics Laborathory u CERNu objavio je dokument pod nazivom „Information management: A proposal“ u kojem je opisao način prijenosa informacija putem Interneta pomoću hiperteksta. U početku su dizajnirani jezik za specificiranje sadržaja dokumenta, današnji HyperText Markup Language (HTML), i protokol za preuzimanje dokumenata i interpretiranje sadržaja, današnji HyperText Transfer Protocol (HTTP). U 1990. godini dolazi do implementacije prvog web preglednika, a 1991. godine Web postaje dostupan i izvan CERN-a. Budući da je Web postao dostupan i izvan CERN-a mnoge tvrtke iskoristile su priliku te su se počele predstavljati potencijalnim klijentima i nuditi im svoje proizvode i usluge. Danas gotovo da i nema tvrtke koja u svom poslovanju ne koristi Internet ili pak Web, pogotovo u ovo današnje doba kada je prijeko potreban zbog rada na daljinu i globalizacije poslovanja.




Slika 2: Web, shematski prikaz rada (Prema: Maximilian Schwarzmüller, 2019)

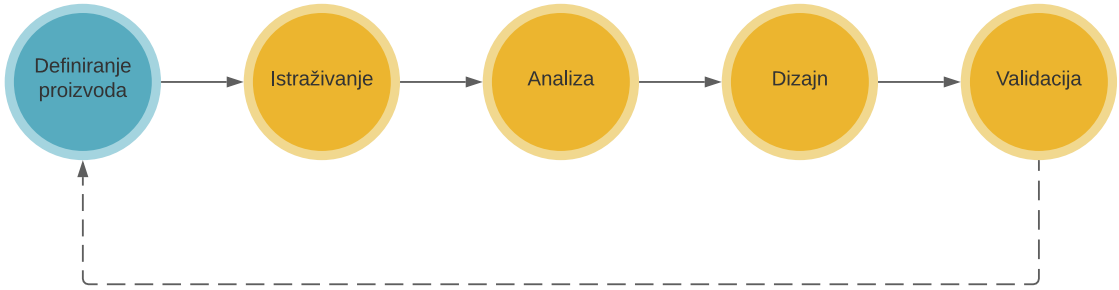
Na slici 2. shematski je prikazan način rada Weba. Otvaranjem web preglednika i unosom URL-a šalje se zahtjev DNS poslužitelju. DNS poslužitelj je računalo koje drži tablicu naziva korespondirajućih IP adresa, a ima zadatak da niz znakova URL-a pretvori u IP adresu. U svijetu postoji 13 korijenskih poslužitelja. Nakon što korisnik konačno bude spojen s traženim poslužiteljem, na ekranu će biti prikazan traženi sadržaj. Traženi sadržaj obično je web dokument kao dio sadržaja web stranice koja je spremljena kao datoteka na web poslužitelju ili se vraća kao rezultat zahtjeva na web aplikaciji. Kao protokol koji služi za komunikaciju između poslužitelja (servera) i klijenta (web preglednika) koristi se HTTP. U današnje vrijeme koristi se HTTPS verzija HTTP-a koja koristi SSL/TLS zaštitu i skriva podatke koji se razmjenjuju između poslužitelja i klijenta.

# 3. Dizajn modernog korisničkog sučelja

U današnje vrijeme velika pozornost stavlja se upravo na dizajniranje korisničkog sučelja. Pojam dizajna korisničkog iskustva i korisničkog sučelja (eng. UX/UI design) prvi je predstavio poznati dizajner i psiholog Donald Norman krajem 1990-ih. Istraživanja su pokazala da je korisnicima važnije kako aplikacija ili web stranica vizualno izgleda kao i korisničko iskustvo, pristupačnost i jednostavnost korištenja same aplikacije ili web stranice od mnoštva neorganiziranih ili čak i nepotrebnih funkcionalnosti implementiranih u samoj aplikaciji. Upravo iz tog razloga u proces stvaranja aplikacija uvode se dizajn korisničkog iskustva i korisničkog sučelja.

## 3.1. Dizajn korisničkog iskustva

Dizajn korisničkog iskustva (eng. User Experience Design – UX design) je proces koji timovi za dizajn koriste prilikom stvaranja proizvoda koji korisnicima pružaju smislena i relevantna iskustva. To uključuje dizajn cjelokupnog postupka stjecanja i integriranja proizvoda, aspekte brendiranja, dizajna, upotrebljivosti i funkcije. Postoji pet ključnih koraka u procesu dizajna korisničkog iskustva.




Slika 3: Koraci dizajna korisničkog iskustva (Prema: Nick Babich, 2020)

### 3.1.1. Definiranje proizvoda

Jedan od najvažnijih koraka u procesu dizajna korisničkog iskustva svakako je definiranje proizvoda. Cilj ovog koraka je da tim koji radi na proizvodu razumije njegov stvarni opseg i kontekst postojanja te kao takav postavlja temelj za preostale korake koji slijede u procesu. U ovom koraku članovi tima dobavljaju sve potrebne informacije o proizvodu na kojem

će raditi. Upravo iz tog razloga važno je održati sastanak sa ključnim sudionicima u procesu kako bi im prenijeli poslovne ciljeve, potrebe korisnika i zahtjeve. Izuzetno je važno dobiti odgovore na tri ključna pitanja; Što je proizvod na kojem se radi?, Tko će koristiti proizvod? i zašto bi netko koristio taj proizvod? Glavni rezultati ove fazu su stvaranje korisničkih osoba, korisničkih priča i dijagrama slučajeva korištenja. Korisnička osoba (eng. User persona) je alat kojim se dizajneri služe u procesu dizajna korisničkog iskustva. Svaka od korisničkih osoba opisuje tip korisnika konačnog proizvoda, a izrađuje se na temelju stvarnih intervjuiranih korisnika uz izmišljene osobne podatke.

### 3.1.2. Istraživanje

Ova faza uključuje istraživanje korisnika i istraživanje tržišta. Postoji mnogo načina na koje se provodi ova faza, a neki od njih su: intervjuiranje ciljane grupe ljudi usmenim putem ili u obliku anketa te istraživanje postoje li slični proizvodi, a samim time i konkurenti na tržištu. Ciljevi ove faze su dobivanje detaljnog uvida u ciljanu grupu korisnika, njihove potrebe, način korištenja proizvoda te analiza i usporedba proizvoda sa postojećim proizvodima na tržištu.

### 3.1.3. Analiza

Sve informacije prikupljene u fazi istraživanja koriste se kao ulazni podaci u fazi analize. Prema prikupljenim informacijama tim dizajnera radi korisničke persone, zamišljene korisnike koji predstavljaju različite tipove korisnika proizvoda, odlučuje koje funkcionalnosti su potrebne, a koje su bespotrebne po mišljenju potencijalnih korisnika, izrađuju korisničke priče te ih povezuju sa korisničkim osobama.

### 3.1.4. Dizajn

U ovoj fazi sva idejna rješenja, sve informacije i dijagrami objedinjuju se u grafičko sučelje. U početku je cilj izraditi jednostavne skice proizvoda (eng. sketch), često ih dizajneri rade pomoću olovke i običnog papira radi bržeg ispravljanja i vizualiziranja raznih ideja. Nakon uspješno završenog skiciranja proizvoda na red dolazi izrada skice dizajna proizvoda (eng. wireframe, mockup) u nekom od alata. Skica dizajna proizvoda daje jasan uvid u to kako će komponente funkcionirati zajedno i strukturu unutar korisničkog sučelja, a sve u svrhu lakšeg donošenja odluka prilikom izrade završnog vizualnog dizajna. Kako bi tim imao jasan uvid u funkcionalnosti prije same izrade proizvoda izrađuju se prototipovi (eng. prototypes). Prototipovi su simulacije nastale u nekom od alata u svrhu poboljšanja iskustva interakcije korisnika i proizvoda. Posljednji korak u fazi dizajna je pribavljanje, priprema i specificiranje

slika, ikona i tekstova (eng. assets) te ukoliko se radi o većem projektu, izrada sustava komponenti, uzoraka i stilova koji će biti elementi korisničkog sučelja.

### 3.1.5. Validacija

Validacija ili testiranje važan je korak u procesu dizajna korisničkog sučelja zbog toga što daje odličan uvid u to kako se korisnici snalaze u dizajnu proizvoda. Dobivanje povratnih informacija izuzetno je važno kako bi tim mogao bolje sagledati cjelokupnu sliku te popraviti eventualne nedostatke ili nadograditi proizvod. Idealno, proces validacije ili testiranja trebali bi proći i sudionici u izradi i određeni uzorak potencijalnih korisnika kako bi se ispunili zahtjevi svih strana. Dizajn korisničkog iskustva nije linearan, već je iterativni proces upravo zato kako bi se proizvod konstantno nadograđivao, a samim time postao sve bolji za buduće korisnike.

## 3.2. Dizajn korisničkog sučelja

Dizajn korisničkog sučelja (eng. User Interface Design – UI design) je proces kojim se služe dizajneri prilikom kreiranja korisničkih sučelja, a koji u fokus stavlja izgled i stil. Jedan je od koraka u procesu dizajna korisničkog iskustva, a kao ulaz koristi izlazne vrijednosti prethodnih koraka. Glavni cilj ovog procesa je izraditi korisničko sučelje koje će krajnjem korisniku biti lako, ugodno i intuitivno za korištenje. Postoje tri vrste korisničkih sučelja:

1. Grafičko korisničko sučelje (eng. Graphical User Interface - GUI) – korisnici komuniciraju s vizualnim prikazima na digitalnim kontrolnim pločama. Radna površina računala je grafičko korisničko sučelje.
2. Korisničko sučelje upravljano glasom (eng. Voice User Interface - VUI) – korisnici upravljaju pomoću vlastitog glasa. Neki od primjera korisničkih sučelja upravljanih glasom su pametni asistenti (eng. Smart assistants) kao Siri (Apple) ili Alexa (Amazon).
3. Korisničko sučelje bazirano na gestama (eng. Gesture – based interface) – korisnici upravljaju 3D modelima pomoću pokreta tijela. Obično se takva sučelja nalaze kod implementacije virtualne stvarnosti (eng. Virtual reality, VR).
4. Znakovno korisničko sučelje (eng. Character User Interface - CUI) – preteča GUI-u, koristi se tipkovnica za komunikaciju preko naredbi s računalom kao u MS DOS-u. Na zaslonu nema slika ili grafika te je riječ o primitivnom korisničkom sučelju.

Postoji niz preporuka koje bi se trebale slijediti pri izradi korisničkog sučelja.

1. Svi uobičajeni elementi trebaju raditi predvidljivo tako da ih ljudi mogu koristiti na uobičajen način. Iza prikaza treba slijediti odgovarajuća funkciju.
2. Održavati vidljivost. Jasno naznačiti ikone kao i primjerice sjene na gumbima.
3. Sučelja bi trebala biti jednostavna, samo sa elementima koji su od pomoći korisniku.
4. Usredotočenost na hijerarhiju i čitljivost.
    a. Korištenje pravilnog poravnanja.
    b. Korištenje pravilnog poravnanja.
    c. Skretanje pozornosti na ključne značajke pomoću odgovarajućih boja, svjetline i kontrasta. Također, važno je odabrati odgovarajuću veličinu, debljinu i razmak između slova tako da korisnik brzim pogledom može jasno prepoznati važne informacije.
5. Smanjiti broj akcija za određeni događaj i staviti fokus na glavnu funkciju određene stranice.
6. Intuitivno postaviti pokretače određenih akcija.
7. Informirati i udijeliti korisniku povratnu informaciju.
8. Koristiti prikladan uzorak kako bi se smanjilo opterećenje korisnika.
9. Zadržati konzistentnost.
10. Uvijek pružiti prirodan sljedeći korak za korisnika bez obzira na kontekst. [6]

# 4. CSS stilske upute

Budući da HTML nije bio prevelikih mogućnosti u smislu naprednog oblikovanja sadržaja, dolazi do potrebe proširenja HTML-a koje je realizirano u vidu CSS-a. CSS (eng. Cascading Style Sheet) jednostavan je mehanizam stilskih listova koji pruža autorima i čitateljima da prikvače stil (npr. font, boje i razmake) HTML dokumentima.

U današnje vrijeme postoje 4 specifikacije/razine CSS-a:

1. CSS1 (1996., revizija 2008.) – čitljiv te je moguće izraziti stil na uobičajen način terminologije stolnog izdavaštva. Kasnije zamijenjen s CSS2.
2. CSS2 (1998. – 2008.), zamijenjen s CSS2.1 (2004., revizija 2011.) – podrška za apsolutno pozicioniranje, brojanje, prekid stranice i sl. CSS2.1 uvodi nove selektore, apsolutno, relativno i fiksno pozicioniranje elemenata, slojevitost elemenata, koncept tipova medija. Ispravlja pogreške u CSS2 te izbacuje slabo podržane ili nepotpuno interoperabilne osobine.
3. CSS3 (1999., revizija 2011.) – podijeljen u više dokumenata koji se zovu moduli. Gradi se na CSS2 modul po modul koristeći CSS2.1 specifikaciju kao svoju jezgru. Svaki modul dodaje funkcionalnost i/ili zamjenjuje dio CSS2.1 specifikacije.
4. CSS4 (2011., revizija 2012. u razvoju)

Postoje tri tipa stilskih listova:

1. Preglednički – svaki element ima svoje predefinirane osobine. Postoji realne opasnost da jedan element nema iste osobine kod različitih preglednika i/ili njihovih verzija.
2. Autorski – osobine elemenata koje obično definira dizajner.
3. Korisnički – ideja nije potpuno saživjela za pristupačnost. Korisnik može definirati vlastite osobine za određene elemente (npr. veći font, kontrast i sl.). [1]

Važno je napomenuti redoslijed prioriteta pa tako autorski nadjačava preglednički, a korisnički nadjačava autorski.

# 4.1. Uvod u CSS

Najnovija verzija CSS-a sadrži mnoštvo raznih uputa koje pomažu kod pisanja stilova, a neke od njih bit će tema ovog poglavlja. Cilj ovog poglavlja je napraviti uvod u osnove te prikazati mogućnosti koje posjeduje CSS3.

## 4.1.1 Povezivanje HTML-a i CSS-a

Kao što je navedeno u prethodnom poglavlju, HTML nema mogućnost oblikovanja sadržaja već služi kao jezik oznaka koje predstavljaju strukturne, prezentacijske i semantičke informacije. Postoje 3 načina smještanja CSS uputa:

1. Jednom u dokumentu – koristi se style element u zaglavlju dokumenta.

```
<!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>Početna stranica</title>
    <style>
        h1 {
            font-size: 20px;
            font-weight: bold;
            color: blue;
        }
        p {
            font-size: 14px;
            color: red;
        }
    </style>
</head>
<body>
    <h1>Ovo je Početna stranica</h1>
    <p>Ovo je opis početne stranice</p>
</body>
</html>
```

Programski kod 1: Smještanje CSS uputa jednom u dokumentu

2. Po potrebi u dokumentu – pomoću atributa style koji se dodaje željenom elementu.

```
<!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>Početna stranica</title>
</head>
<body>
    <h1 style="font-size: 20px; font-weight: bold; color: blue;">Ovo je Početna
stranica</h1>
    <p style="font-size: 14px; color: red;">Ovo je opis početne stranice</p>
</body>
</html>
```

Programski kod 2: Smještanje CSS uputa po potrebi u dokumentu

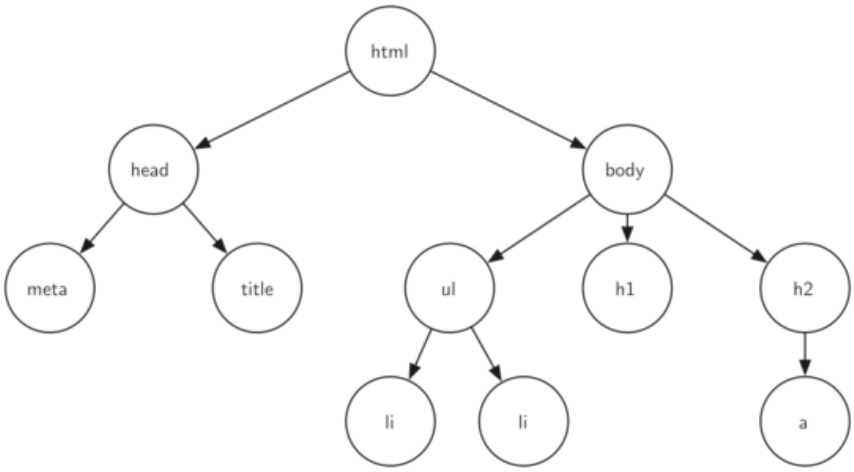
3. U vanjskoj datoteci – u zaglavlju dokumenta koristi se element link kojim se odredi putanja do datoteke koja sadrži CSS upute.

```
<!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>Početna stranica</title>
</head>
<body>
    <h1>Ovo je Početna stranica</h1>
    <p>Ovo je opis početne stranice</p>
</body>
</html>
```

Programski kod 3: Smještanje CSS uputa u vanjskoj datoteci

### 4.1.2. Selektori

Selektori su dio CSS-a pomoću kojih se označuju pojedini HTML elementi na koje se primjenjuje navedeni stil. Postoji čitav niz selektora, a u ovom poglavlju biti će objašnjeni samo oni najčešće korišteni. Za početak, važno je napomenuti kako zapravo izgleda HTML struktura dokumenta. Na slici 7. vidljivo je da elementi unutar stabla imaju svoje podređene i nadređene elemente. Primjerice, elementi li sadržani su u elementu ul, a element ul sadržan je u elementu body. Selektorom možemo identificirati i označiti bilo koji od elemenata stabla.




Slika 4: Hijerarhijsko stablo HTML dokumenta

Postoji mnogo načina korištenja CSS uputa:

1. Implicitno – najjednostavniji način od svih, selektor odgovara imenu HTML elementa i primjenjuje se na svaki istovrsni element u dokumentu. Prednost korištenja ovog selektora je u tome što ne zahtijeva dodatne intervencije unutar HTML-a. Nedostatak je u tome što se pomoću ovog selektora ne mogu postići različiti stilovi za paragrafe u dokumentu.

```
p {
  font-size: 18px;
  font-weight: bold;
  color: blue;
  margin: 6px 10px;
}
```

Programski kod 4: Implicitni način korištenja CSS uputa

2. Eksplicitno – uputa koja se koristi uz atribut class kojem se pridružuje selektor uz znak . (povezan na HTML element ili samostalno).

```
.paragraph {
  font-size: 18px;
  font-weight: bold;
  color: blue;
  margin: 6px 10px;
}
```

Programski kod 5: Eksplicitni način korištenja CSS uputa

3. Jednoznačno – uputa koja se koristi uz atribut id kojem se pridružuje selektor uz znak # (povezan na HTML element ili samostalno), garantira jedinstvenu vrijednost u cijelom dokumentu.

```
#paragraph {
  font-size: 18px;
  font-weight: bold;
  color: blue;
  margin: 6px 10px;
}
```

Programski kod 6: Jednoznačan način korištenja CSS uputa

4. Pseudo klase – koriste se kod elemenata koji imaju unaprijed pripremljene poddijelove kojima se mogu promijeniti osnovne osobine, a odvajaju se znakom : i kombiniraju se s implicitnim, eksplicitnim ili jednoznačnim načinom korištenja CSS uputa.

```
a:hover {
  text-decoration: underline;
  color: yellowgreen;
}
```

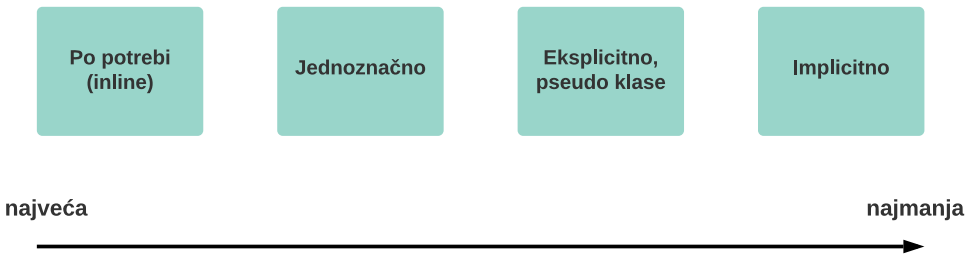
Programski kod 7: Korištenje pseudo klasa u CSS uputama

5. Po potrebi (eng. inline) – koristi se po potrebi tako da se HTML elementu dodaje atribut style te se unutar njega navode svojstva i vrijednosti.

```
<!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>Početna stranica</title>
</head>
<body>
    <h1 style="font-size: 20px; font-weight: bold; color: blue;">Ovo je Početna
stranica</h1>
    <p style="font-size: 14px; color: red;">Ovo je opis početne stranice</p>
</body>
</html>
```

Programski kod 8: Korištenje CSS uputa po potrebi

Budući da postoji cijeli niz kombinacija raznih selektora, na kraju se postavlja sasvim očekivano pitanje, a to je koji od selektora ima najveću važnost i kako CSS to interpretira. Upravo je ovo uobičajen razlog zbog kojeg se CSS upute ne primjenjuju na neke HTML elemente, a trebale bi. Važno je upamtiti redoslijed kojim CSS interpretira pojedine selektore. Prema slici 13. vidljivo je da style ima najveći prioritet, zatim slijede jednoznačni, eksplicitni te implicitni način korištenja CSS uputa.

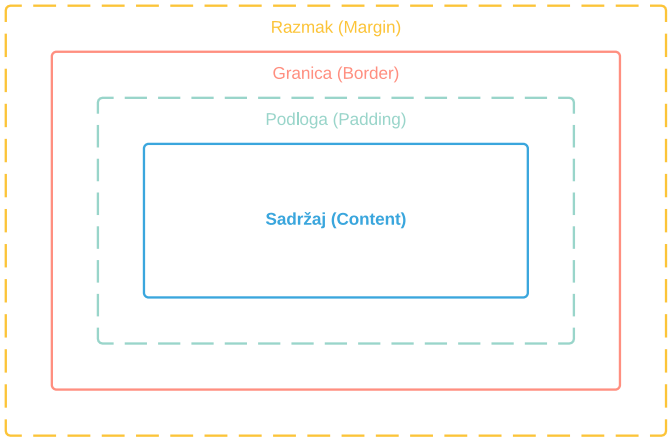



Slika 5: Prioriteti prilikom korištenja CSS uputa

### 4.1.3. Model kutije

Model kutije (eng. Box Model) predstavlja ključni pojam prilikom izrade dizajna i rasporeda elemenata. Na svaki od HTML elemenata može se gledati kao na kutiju ili pravokutnik kojem se mogu dodijeliti određene osobine. Prema slici 14. model kutije sastoji se od četiri dijela:

1. Sadržaj (eng. Content) – predstavlja sadržaj HTML elementa (tekst, slika, i sl.), može biti mijenjan pomoću svojstava width ili height
2. Podloga (eng. Padding) – predstavlja područje između razmaka i sadržaja, kontrolira se pomoću svojstva padding
3. Granica (eng. Border) – predstavlja crtu oko HTML elementa, odgovara svojstvu border
4. Razmak (eng. Margin) – predstavlja razmak od ostalih HTML elemenata, koristi svojstvo margin




Slika 6: Prikaz modela kutije

Svaki od dijelova modela kutije može se podešavati simetrično ili asimetrično, ovisno o potrebi. Unutar CSS-a postoje 3 tipa modela kutije, a to su redom:

1. Blok elementi (eng. Block) – element se prelama u novi red te ako autor nije odredio drugačije, zauzima sav dostupan prostor unutar njegovog nadređenog elementa. Primjer takvih elemenata su h1 i p elementi.
2. Linijski elementi (eng. Inline) – elementi koji dolaze u nizu, jedan za drugim. Primjer su a ili img elementi.
3. Linijski – blok elementi (eng. Inline-block) – kombinacija su linijskog i blok tipa elemenata.

### 4.1.4. Fleksibilnost

Fleksibilnost (eng. Flexbox) naziv je za jednodimenzionalnu metodu prema kojoj se elementi dizajna raspoređuju u redove ili stupce. Dugo vremena postojala su jedina dva načina za kreiranje rasporeda elemenata u obliku plutajućih (eng. floats) elemenata i pozicioniranja (eng. positioning). Osnova ove metode je postavljanje CSS osobine display kontejnera uz vrijednost flex. Elementi unutar kontejnera popunjavaju preostali slobodan prostor unutar kontejnera prema dodatno danim uputama. Vrsta prikaza određuje se osobinom flex-direction te može poprimiti sljedeće vrijednosti:

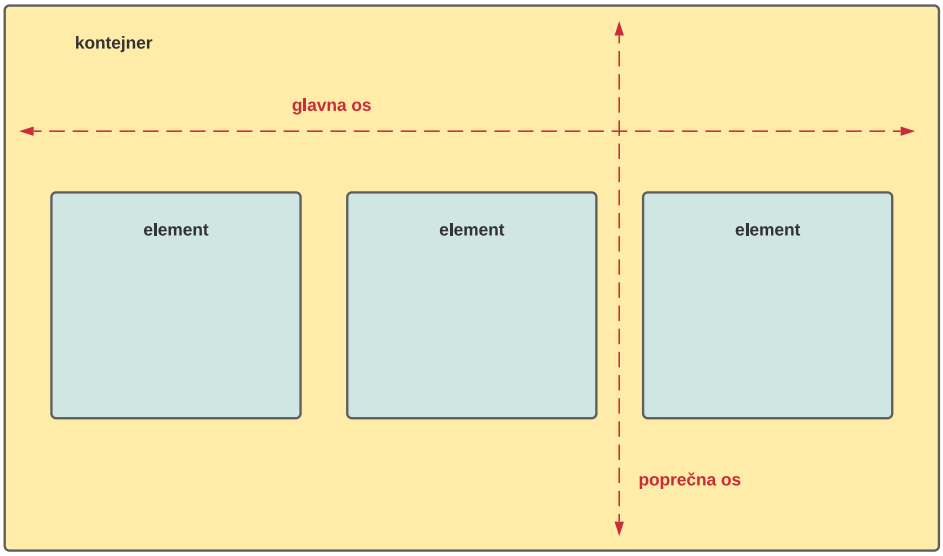
1. row – elementi su raspoređeni po retku, početna vrijednost nakon postavljanja osobine display na vrijednost flex
2. column – elementi su raspoređeni po stupcu
3. row-reverse – elementi su raspoređeni po retku obrnutim redoslijedom
4. column-reverse – elementi su raspoređeni po stupcu obrnutim redoslijedom

Osim kontrole rasporeda elemenata unutar kontejnera, postoje i druge mogućnosti pomoću sljedećih osobina:

1. flex-wrap – kontrola prijeloma, može poprimiti vrijednosti wrap, no-wrap i wrap-reverse
2. flex-flow – skraćena osobina koja objedinjuje osobine flex-direction i flex-wrap
3. justify-content – služi za poravnanje sadržaja, moguće vrijednosti su center, flex-start, flex-end, space-around i space-between
4. align-items – okomito poravnanje sadržaja, vrijednosti: center, flex-start, flex-end, stretch i baseline
5. align-content – služi za poravnanje flex linija, moguće vrijednosti su: space-around, space-between, center, stretch, flex-start i flex-end

Djeca fleksibilnih elemenata ili kontejnera automatski postaju fleksibilne stavke. Za njih se mogu koristiti sljedeće osobine:

1. order – može poprimiti brojčanu vrijednost
2. flex-grow – govori koliko će određeni element rasti u odnosu na ostale elemente unutar kontejnera
3. flex-shrink - govori koliko će se određeni element smanjiti u odnosu na ostale elemente unutar kontejnera
4. flex-basis – postavlja inicijalnu dužinu elementa
5. align-self – postavlja okomito poravnanje sadržaja i nadjačava kontejnerovu osobinu align-items, a može imati vrijednosti center, flex-start, flex-end, stretch ili baseline

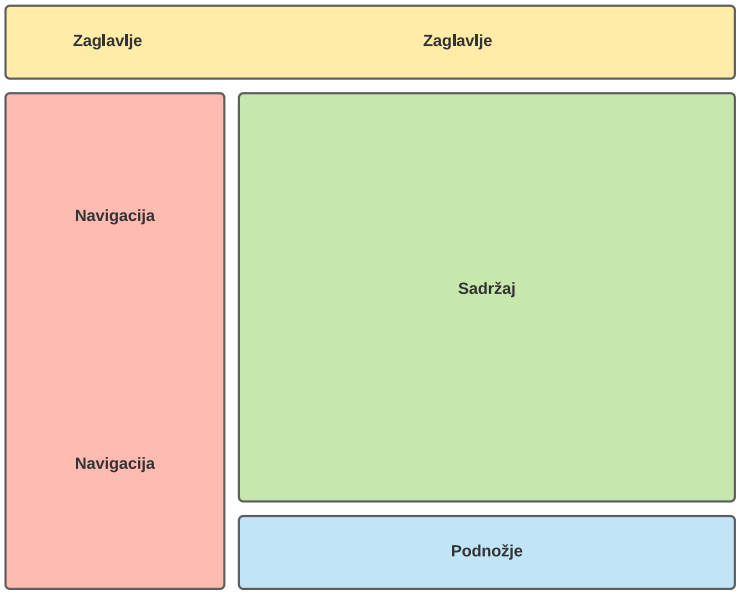



Slika 7: Prikaz metode fleksibilnosti

### 4.1.5. Rešetka

Rešetka (eng. Grid) je dvodimenzionalan sustav za izradu kompleksnih sadržaja. Kod ovog sustava elementi se raspoređuju u stupce i redove, a sam sustav posjeduje mnoštvo raznih specijalnih opcija koje znatno olakšavaju izradu kompleksnih vrsta dizajna. Cijeli proces kreće od koraka u kojem se glavnom kontejneru pridružuje osobina display uz vrijednost grid. Budući da ova osobina daje samo jednostupčanu rešetku, potrebno je osobinama grid-template-columns ili grid-template-rows odrediti broj stupaca ili redaka putem njihove širine. Dimenzije svakog stupca ili retka mogu se zadati u pikselima, fragmentima, postocima, kao i u nekim relativnim jedinicama kao što su rem, em, vw, vh ili ch. Pomoću osobine gap određuje se razmak između stupaca i redaka, a njega je također moguće iskazati u nekim od prethodno navedenih jedinica. Jedna od glavnih osobina koje su olakšale izradu kompleksnih sučelja je grid-template-areas. Funkcionira tako da se svakom elementu unutar rešetke pomoću osobine grid-area može zadati unikatno ime te se pomoću imena element raspoređuje unutar rešetke (Slika 8). Poravnanja elemenata unutar rešetke funkcioniraju podjednako kao i kod metode fleksibilnosti te se za njih koriste iste osobine.

```
grid-template-columns: 200px 1fr;

grid-template-areas: "zaglavlje zaglavlje"
                     "navigacija sadržaj"
                     "navigacija podnožje";
```




Slika 8: Prikaz korištenja osobine grid-template-areas kod rešetke

## 4.2. CSS preprocesori

CSS preprocesori alati su koji proširuju CSS dodajući mu širok spektar novih funkcionalnosti koje nisu tipične za CSS jezik. Često se te iste funkcionalnosti mogu vidjeti kod većine standardnih programskih jezika. CSS preprocesor uzima kod napisan u formatu i prema uputama predviđenim za određeni preprocesor i kompilira ga u tradicionalni CSS kako bi ga web preglednik razumio. Glavni razlozi zašto bi se neki od CSS preprocesora trebao koristiti na projektu su DRY (eng. Don't Repeat Yourself) princip te dostupnost korištenja kompleksnih logičkih operatora i uvjetnih iskaza.

### 4.2.1. Sintaksa

Kao što je spomenuto u uvodnom dijelu ovog poglavlja, postoji više CSS preprocesora, a prema tome svaki od njih koristi svoja pravila pisanja koda. Pravila kojih se je potrebno pridržavati prilikom pisanja koda jednom riječju zovemo sintaksom. Sass (eng. Syntactically Awesome StyleSheets) posjeduje dvije verzije sintakse:

1. Scss sintaksa – ova sintaksa ne razlikuje se od standardne CSS sintakse, koriste se zagrade i točka – zarez, dokument ima nastavak .scss
2. Sass sintaksa – kod ove sintakse ne koriste se zagrade i točka – zarez, a blokovi naredbi odvajaju se uvlačenjem, dokument ima nastavak .sass

```
.container {
  width: 100%;
  border: 1px solid black;
  p {
    font-size: 18px;
    color: red;
  }
}
```

Programski kod 9: Scss sintaksa

```
.container
  width: 100%
  border: 1px solid black
  p
    font-size: 18px
    color: red
```

Programski kod 10: Sass sintaksa

Less (eng. Leaner Style Sheets) posjeduje jednu standardnu verziju sintakse koja izgleda potpuno isto kao standardna CSS sintaksa. Koristi zagrade i točku-zarez. Dokument ima .less nastavak.

### 4.2.2. Varijable

Jedna od glavnih osobina koje CSS preprocesori posjeduju svakako su varijable. Uvođenje varijabli u pisanje CSS koda značajno je olakšalo sam proces pisanja i posebno kasnijeg održavanja ili dodatnih korekcija. Iako standardni CSS posjeduje varijable, one se uvelike razlikuju od preprocesorskih varijabli:

1. Preprocesorske varijable se kompiliraju u običan CSS tako da se mjesto varijable zamjenjuje vrijednosti varijable, dok kod CSS varijabli to nije slučaj.

```
:root {
  --bg-light: #ffffff;
}

.container {
  width: 100%;
  background-color: var(--bg-
light);
}
```

```
$bg-color-light: #ffffff;

.container {
  width: 100%;
  background-color: #ffffff;
}
```

Programski kod 11: Prikaz CSS dokumenta nakon kompiliranja

2. CSS varijable mogu imati različite vrijednosti za različite elemente, a preprocesorske varijable imaju samo jednu vrijednost u danom trenutku.

3. Kod izmjene preprocesorskih varijabli prethodna vrijednost korištene varijable ostaje ista, dok se kod CSS varijabli vrijednost varijable mijenja u svakom dijelu programskog koda.

```
$variable: value 1;

.rule-1 {
  value: $variable;
}

$variable: value 2;

.rule-2 {
  value: $variable;
}
```

```
.rule-1 {
  value: value 1;
}

.rule-2 {
  value: value 2;
}
```

Programski kod 12: Način korištenja preprocesorskih varijabli

Također, važno je spomenuti da kod preprocesora postoji djelokrug varijable. Prema tome, varijable mogu biti globalne ili lokalne. Globalne varijable dostupne su u svakom dijelu programskog koda, dok je djelokrug lokalnih varijabli samo unutar bloka u kojoj je lokalna varijabla deklarirana. Preprocesori posjeduju i već ugrađene varijable, a jedna od njih je konstanta PI koja dolazi iz matematičkog modula preprocesorskih varijabli. Trenutno Sass preprocesor sadrži sedam modula, a to su redom: sass:color, sass:list, sass:map, sass:math, sass:meta, sass:selector i sass:string.

### 4.2.3. At-pravila

Mnogo dodatnih funkcionalnosti dolazi u obliku at-pravila. U nastavku slijedi opis svakog pojedinog pravila:

1. @use – služi za učitavanje funkcija, mixina ili varijabli iz ostalih Sass stilskih uputa. Stilske upute učitane pomoću ovog pravila nazivaju se moduli. Ovo pravilo važno je navesti na početku stilskih uputa kako bi se učitani moduli mogli koristiti u daljnjem kodu.
2. @forward – učitava Sass stilske upute i priprema funkcije, mixine i varijable za korištenje pomoću @use pravila. Glavna zadaća ovog pravila je omogućiti lakšu organizaciju kroz više dokumenata dok je za učitavanje svih dokumenata potrebna samo jedna putanja.
3. @mixin i @include - @mixin pravilo omogućuje definiranje stilova koji se mogu koristiti više puta. Mixin može i ne mora primati parametre. Zadani parametar označava se dvotočkom i on će biti apliciran ukoliko se određeni parametar ne proslijedi u mixin. Pravilo @include koristi se za učitavanje određenog @mixin pravila unutar stilskih uputa. Desni dio koda prikazuje kako će lijevi dio koda izgledati nakon kompilacije u CSS stilske upute.

```
@mixin replace-text($image, $x:
50%, $y: 50%) {
  text-indent: -99999em;
  overflow: hidden;
  text-align: left;

  background: {
    image: $image;
    repeat: no-repeat;
    position: $x $y;
  }
}

.mail-icon {
  @include replace-
text(url("/images/mail.svg"),
0);
}
```

```
.mail-icon {
  text-indent: -99999em;
  overflow: hidden;
  text-align: left;
  background-image:
url("/images/mail.svg");
  background-repeat: no-repeat;
  background-position: 0 50%;
}
```

Programski kod 13: Prikaz @mixin i @include pravila

4. @function – ovo pravilo omogućuje definiranje kompleksnih operacija koje se također mogu koristiti više puta. Glavna razlika između @mixin i @function pravila je u tome što @mixin za izlaz daje Sass stilske upute, a @function daje jednu vrijednost koja će se koristiti kao vrijednost za neki parametar unutar stilskih uputa.

```
@function invert($color,
$amount: 100%) {
  $inverse: change-
color($color, $hue: hue($color)
+ 180);
  @return mix($inverse, $color,
$amount);
}

$primary-color: #036;
.header {
  background-color:
invert($primary-color, 80%);
}
```

```
.header {
  background-color: #523314;
}
```

Programski kod 14: Prikaz @function pravila

5. @extend – pravilo koje omogućuje nasljeđivanje drugih, već zadanih stilskih uputa. Pravilo je vrlo fleksibilno i moguće su razne kombinacije sa mixinima i funkcijama.

```
.error {
  border: 1px #f00;
  background-color: #fdd;

  &--serious {
    @extend .error;
    border-width: 3px;
  }
}
```

```
.error, .error--serious {
  border: 1px #f00;
  background-color: #fdd;
}
.error--serious {
  border-width: 3px;
}
```

Programski kod 15: Prikaz @extend pravila

6. @error i @warn – pravila koje posebno dolaze do izražaja kod kreiranja funkcija i mixina unutar Sass stilskih uputa. Koriste se kako bi korisnik bio obaviješten o pogrešci unutar funkcije ili mixina. Pravilo @error zaustavlja izvođenje funkcije ili mixina dok @warn samo daje obavijest bez zaustavljanja.

7. @debug – pravilo posebno korisno kod pisanja stilskih uputa. Omogućuje ispis vrijednosti neke varijable ili izraza zajedno sa imenom dokumenta i brojem linije unutar koda.

```
@mixin inset-divider-
offset($offset, $padding) {
  $divider-offset: (2 *
$padding) + $offset;
  @debug "divider offset:
#{$divider-offset}";

  margin-left: $divider-offset;
  width: calc(100% -
#{$divider-offset});
}
```

```
test.scss:3 Debug: divider
offset: 132px
```

Programski kod 16: Prikaz @debug pravila

8. @at-root – pravilo pomoću kojeg će sve stilske upute navedene unutar ovog pravila biti prikazane u korijenu dokumenta.

```
@media print {
  .page {
    width: 8in;

    @at-root (without: media) {
      color: #111;
    }

    @at-root (with: rule) {
      font-size: 1.2em;
    }
  }
}
```

```
@media print {
  .page {
    width: 8in;
  }
}
.page {
  color: #111;
}
.page {
  font-size: 1.2em;
}
```

Programski kod 17: Prikaz @at-root pravila

9. @if, @else, @each, @for, @while – pravila koja funkcioniraju potpuno isto kao u svim ostalim programskih jezicima, a služe za kontrolu toka programskog koda. Imaju važnu primjenu kod @mixin i @function pravila.

### 4.2.4. Operatori

Sass preprocesor sadrži uobičajene matematičke i logičke operatore. Neki od operatora su sljedeći:

1. ==, != - koriste se za provjeru jesu li dvije vrijednosti iste
2. +, -, *, /, % - standardni matematički operatori, posebno korisni kod funkcija
3. <, <=, >, >= - koriste se za provjeru jesu li dva broja veća, jednaka ili manja jedan od drugog
4. and, or, not – logički operatori, koriste se kod vrijednosti tipa boolean
5. +, -, / - mogu se koristiti kod vrijednosti tipa string

Važno je napomenuti da postoje prioriteti prilikom izvođenja operacija. Prema tome, redoslijed izvođenja je sljedeći:

1. Unarni operatori not, +, -, /
2. *, /, % operatori
3. +, - operatori
4. >, >=, <, <= operatori
5. ==, != operatori
6. and operator
7. or operator
8. = operator, kad je dostupan

### 4.2.5. Usporedba Sass i Less preprocesora

Budući da je Sass najkorišteniji CSS preprocesor današnjice, on je i detaljnije obrađen u prethodnih nekoliko poglavlja. Naravno, postoji još podosta CSS preprocesora, a jedan koji svakako može konkurirati Sass-u je Less i upravo zato će se u ovom poglavlju napraviti njihova usporedba (Tablica 1). Prema svemu navedenom u tablici lako je zaključiti da je Sass i zasluženo popularniji i korišteniji CSS preprocesor. Sass jednostavno daje više slobode, elegantniji je svojom sintaksom, posjeduje više korisnih stvari poput at-pravila i korisničkih funkcija, a dokumentacija mu se svodi na realne primjere iz prakse.

| | Sass | Less |
|---|---|---|
| **Instalacija** | kompilator u C++, za ostale jezike potrebna druga verzija kompilatora | napisan u čistom JavaScriptu i lagan za instalaciju |
| **Varijable** | varijabla se definira pomoću znaka $ | varijabla se definira pomoću znaka @ |
| **Gniježđenje (eng. Nesting)** | posjeduje, osim gniježđenja selektora, mogućnost gniježđenja CSS osobina | posjeduje mogućnost gniježđenja selektora |
| **Mixins** | definira se pomoću pravila @mixin | definira se pomoću eksplicitnog selektora |
| **Nasljeđivanje** | definira se pomoću pravila @extend | definira se pomoću pseudo klase :extend |
| **Funkcije** | definira se pomoću pravila @function | ne posjeduje specijalno pravilo za definiranje korisničkih funkcija |
| **Operatori** | posjeduje razne matematičke i logičke operatore | posjeduje razne matematičke i logičke operatore |
| **Upravljanje pogreškama** | posjeduje pravilo @error za upravljanje pogreškama | ne posjeduje specijalna pravila za upravljanje pogreškama |
| **Dokumentacija** | posjeduje vrlo dobro organiziranu dokumentaciju | posjeduje vrlo dobro organiziranu dokumentaciju |

Tablica 1: Usporedba Sass i Less preprocesora

# 5. CSS okviri

Jedan od glavnih problema prilikom izrade nekog projekta je konstantno ponavljanje istih koraka i kretanje svakog projekta od početka, što dodatno usporava cijeli proces programiranja. Kako bi si programeri olakšali posao i znatno ubrzali početak rada izmišljeni su CSS okviri. CSS okviri (eng. CSS frameworks) alati su kojima se programeri služe kako bi lakše, brže i jednostavnije krenuli u izradu nekog projekta. Sadrže skup CSS stilskih uputa koje su unaprijed pripremljene od strane osoba koje su izradile CSS okvir i spremne za korištenje. Umjesto da svaki put troše vrijeme na cijelo postavljanje temelja za neki projekt, koristeći CSS okvir programer može u vrlo kratkom roku izraditi pokazno korisničko sučelje. CSS okviri posebno su korisni kod vrlo velikih timova te kada komponente moraju imati standardan izgled kroz cijelo korisničko sučelje. Mana takvih CSS okvira je monoton, jednoličan i standardiziran izgled komponenti korisničkih sučelja. Korištenje CSS okvira od strane neiskusnih programera, koji nisu upućeni u CSS, može biti vrlo zbunjujuće budući da CSS okviri sakrivaju podosta čistog CSS koda unutar svoje implementacije. Da bi izbjegli neželjene posljedice važno je dobro preispitati što je zapravo cilj i treba li se uopće koristiti CSS okvir u danom projektu.

## 5.1. Vrste CSS okvira

CSS okvire trenutno dijelimo na dvije kategorije – CSS okviri temeljeni na gotovim komponentama i CSS okviri temeljeni na klasama. CSS okviri temeljeni na gotovim komponentama (eng. Component-based CSS frameworks) alati su kod kojih postoje već definirane, stilizirane komponente spremne za implementaciju u korisničko sučelje. Korištenjem takve vrste CSS okvira programer u vrlo kratkom vremenskom roku može izraditi funkcionalno korisničko sučelje. No, postoji naravno i određen broj mana za takvu vrstu CSS okvira. Glavna mana je u tome što korisnička sučelja izrađena uporabom CSS okvira temeljenim na gotovim komponentama izgledaju jednolično, monotono i vrlo prepoznatljivo svakom tko je imao doticaj sa takvom vrstom CSS okvira. Neki od trenutno najkorištenijih CSS okvira temeljenim na gotovim komponentama svakako su: Bootstrap, Material UI, Foundation, Bulma, Semantic UI i Materialize. S druge strane, CSS okviri temeljeni na klasama (eng. Utility-based CSS frameworks) alati su kod kojih se određene stilske upute primjenjuju preko klasa koje su definirane unutar određenog CSS okvira. Ova vrsta CSS okvira programeru daje vise slobode, fleksibilnosti i lakše upravljanje stilskim uputama prilikom izrade vlastitih komponenti budući da ne postoje već stilizirane komponente. CSS okviri temeljeni na klasama dobar su izbor prilikom izrade prilagođenih projekata gdje je potrebno određene elemente korisničkog

sučelja izraditi po želji klijenta. Budući da će se u praktičnom dijelu ovog rada koristiti CSS okvir Tailwind CSS, on će biti analiziran u nastavku ovog poglavlja.

## 5.2. CSS okvir Tailwind CSS

Tailwind CSS jedan je od najkorištenijih CSS okvira današnjice. Temeljen je na klasama što programeru omogućuje veliku fleksibilnost, slobodu i dodatna proširenja prilikom izrade korisničkih sučelja. Svaka uobičajena CSS stilska uputa posjeduje klasu unutar Tailwind CSS okvira koja je spremna za dodavanje u HTML dokument. Instalacija Tailwind CSS-a u projekt moguća je u dvije varijante. Prva varijanta uključuje instalaciju Tailwind CSS-a putem CDN-a (eng. Content Delivery Network) dodavanjem putanje do Tailwind CSS datoteke unutar HTML dokumenta. Druga opcija, koja je trenutno i korištenija, odnosi se na instalaciju Tailwind CSS paketa putem npm-a ili yarn-a. Neke od važnijih CSS uputa i njihova primjena putem Tailwind CSS-a bit će objašnjene u sljedećim poglavljima kroz prilagođene primjere iz prakse.

### 5.2.1. Fontovi i boje

Kao što je vidljivo Tailwind CSS primjenjuje razne stilske upute preko definiranih klasa. Sve što je moguće napraviti putem CSS-a, moguće je i putem Tailwind CSS-a. Na konkretnom primjeru programskog koda primijenjene su sljedeće stilske upute:

1. w-full – postavlja vrijednost širine sekcije na 100%
2. h-64 – postavlja vrijednost visine sekcije na 16rem
3. bg-red-500 – postavlja vrijednost boje pozadine na točno definiranu nijansu crvene boje, nijansa određene boje određuje se pomoću broja na kraju klase
4. font-sans – postavlja vrijednost font-family upute na Sans
5. text-xl – postavlja vrijednost veličine fonta na 1.25rem i visine linije na 1.75rem
6. font-bold – postavlja vrijednost debljine fonta na bold
7. tracking-wide – postavlja vrijednost razmaka između riječi na 0.025rem
8. text-center – postavlja pozicioniranje naslova na sredinu
9. text-indigo-500 – postavlja vrijednost boje teksta na točno definiranu nijansu ljubičaste boje, nijansa boje također se određuje pomoću broja na kraju klase
10. underline – postavlja podcrtavanje teksta

```
<section class="w-full h-64 bg-red-500">
	<h1 class="font-sans text-xl font-bold tracking-wide text-center text-indigo-500 underline">Ovo je naslov</h1>
	<p class="font-serif text-lg font-light tracking-tight text-left text-blue-500">Ovo je odlomak teksta</p>
</section>
```

Programski kod 18: Prikaz korištenja stilskih uputa kod fontova i boja

## 5.2.2. Podloga, granica i razmak

Tailwind CSS posjeduje cijeli niz klasa pomoću kojih se mogu primijeniti podloge, granice ili razmaci:

1. Podloga – klase koje Tailwind CSS koristi za primjenu podloge kreću sa slovom p (eng. padding). Klasa p-broj znači da se podloga primjenjuje na sve strane prema modelu kutije. Postoji mogućnost primjene podloge po osima, stoga razlikujemo klase px-broj i py-broj. Klasa px-broj primjenjuje podlogu po glavnoj osi, dok klasa py-broj primjenjuje podlogu po poprečnoj osi. Ukoliko se podlogu želi primijeniti na točno određenu stranu modela kutije potrebno je uz slovo p navesti i prvo slovo strane na koju se podloga želi primijeniti. Prema tome klasa je ovog formata p[strana modela kutije]-broj.
2. Granica – klase za primjenu granice kreću sa riječi border-broj. Broj označuje vrijednost debljine granice, ali kada iza ključne riječi border stoji ime boje tada govorimo o boji granice.
3. Razmak – klase za primjenu razmaka kreću sa slovom m (eng. margin). Klasa razmaka funkcionira apsolutno na isti način kao i klasa podloge.

```
<section class="w-full h-64 my-8 mx-6">
	<div class="p-5 mt-3 mb-2 border-2 border-gray-300">
		<h2>Lorem ipsum dolor sit amet.</h2>
		<p>Lorem ipsum dolor sit amet consectetur adipisicing elit.<p>
	</div>
</section>
```

Programski kod 19: Prikaz korištenja stilskih uputa kod podloge, granice i razmaka

### 5.2.3. Fleksibilnost i rešetka

Fleksibilnost ili rešetka se glavnom kontejneru zadaje primjenom klasa flex, odnosno grid. Svaki od elemenata unutar fleksibilnog kontejnera ili rešetke može imati sljedeće klase:

1. flex-direction – smjer prostiranja elemenata unutar kontejnera definira se klasama flex-row, flex-row-reverse, flex-col ili flex-col-reverse
2. flex-wrap – prijelom elemenata definira se klasama flex-wrap, flex-wrap-reverse i flex-nowrap
3. flex-grow i flex-shrink – definiraju se klasama flex-grow-0 i flex-grow ili flex-shrink-0 i flex-shrink
4. gap – razmak između stupaca ili redaka određuje se klasama gap-broj ili gap-x-broj i gap-y-broj
5. order – razmještaj elemenata određuje se klasama order-broj
6. grid-template-columns – broj stupaca rešetke određuje se klasama grid-cols-broj
7. grid-template-rows – broj redaka rešetke određuje se klasama grid-rows-broj
8. justify-content – poravnanje elemenata po glavnoj osi, klase koje se koriste mogu biti: justify-start, justify-end, justify-center, justify-between, justify-around ili justify-evenly
9. align-items – poravnanje elemenata po poprečnoj osi, klase koje se koriste mogu biti: items-start, items-end, items-center, items-baseline ili items-stretch

```
<section class="w-full h-64 flex flex-col justify-center items-center">
	<div class="p-5 mt-3 mb-2">
		<h2>Lorem ipsum dolor sit amet.</h2>
	<p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>
	</div>
	<div class="p-5 mt-3 mb-2">
		<h2>Lorem ipsum dolor sit amet.</h2>
	<p>Lorem ipsum dolor sit amet consectetur adipisicing elit.</p>
	</div>
</section>
```

Programski kod 20: Prikaz korištenja stilskih uputa za fleksibilnost

### 5.2.4. Responzivnost

Tailwind CSS omogućuje i primjenu responzivnosti. Responzivnost se određuje pomoću pomoćnih klasa. Unutar okvira postoji pet različitih predviđenih veličina ekrana, a one su definirane na sljedeći način:

1. sm – veličina od 640px i uputa `@media (min-width: 640px)`
2. md – veličina od 768px i uputa `@media (min-width: 768px)`
3. lg – veličina od 1024px i uputa `@media (min-width: 1024px)`
4. xl – veličina od 1280px i uputa `@media (min-width: 1280px)`
5. 2xl – veličina od 1536px i uputa `@media (min-width: 1536px)`

Svaka od ovih pomoćnih klasa može se primijeniti na određenu drugu klasu. Ukoliko pomoćna klasa nije iskorištena nad određenom klasom, smatra se kao početno stanje za sve manje veličine ekrana od iduće pomoćne klase. Dakle, prema programskom kodu 21 element div kao početno stanje imat će boju pozadine prema klasi bg-teal-500, na ekranima veličine veće ili jednake 768px imat će boju pozadine prema klasi bg-red-500, a na ekranima veličine veće ili jednake 1024px imat će boju pozadine prema klasi bg-teal-500.

```
<div class="bg-teal-500 md:bg-red-500 lg:bg-teal-500">
  <!-- ... -->
</div>
```

Programski kod 21: Prikaz korištenja pomoćnih klasa responzivnosti

Naravno, ovo je samo jedan mali dio primjene pomoćnih klasa kod okvira Tailwind CSS. Stvari postaju mnogo kompleksnije kada se pomoćne klase kombiniraju sa fleksibilnosti ili rešetkom što omogućuje izradu i prilagođavanje kompleksnih dizajna različitim veličinama ekrana. Glavna prednost Tailwind CSS-ovih pomoćnih klasa je u tome što stavljaju izradu dizajna za mobilne uređaje (eng. Mobile-first) na prvo mjesto, a zatim se prelazi na veće ekrane. Time se mnogo štedi na programskom kodu, a kasnije izmjene su znatno olakšane budući da je izrada dizajna za mobilne uređaje kompleksniji proces.

# 6. Razvoj web aplikacije pomoću CSS okvira

U ovom poglavlju detaljno će biti opisan proces razvoja web aplikacije pomoću CSS okvira Tailwind CSS. Aplikacija je razvijana pomoću Tailwind CSS-a zbog toga što, kao što je navedeno u prethodnom poglavlju, Tailwind CSS omogućuje programeru da sam kreira komponente koje su mu potrebne na projektu umjesto da koristi već definirane komponente koje možda neće ispunjavati sve potrebne zahtjeve projekta. Tehnologije koje su korištene prilikom izrade su: HTML, CSS, JavaScript/React, Firebase i neki dodatni paketi koji će biti pojašnjeni u nastavku poglavlja. Također, za dohvaćanje podataka o filmovima i serijama koristi se TMDB API servis.

## 6.1. Opis funkcionalnosti web aplikacije

Web aplikacija izrađena u ovom poglavlju nosi naziv MovieApp. Cilj aplikacije je korisnicima omogućiti praćenje odgledanih filmova ili serija, njihovo pretraživanje te prikaz važnijih informacija o istima. Web aplikacija MovieApp posjeduje sljedeće funkcionalnosti:

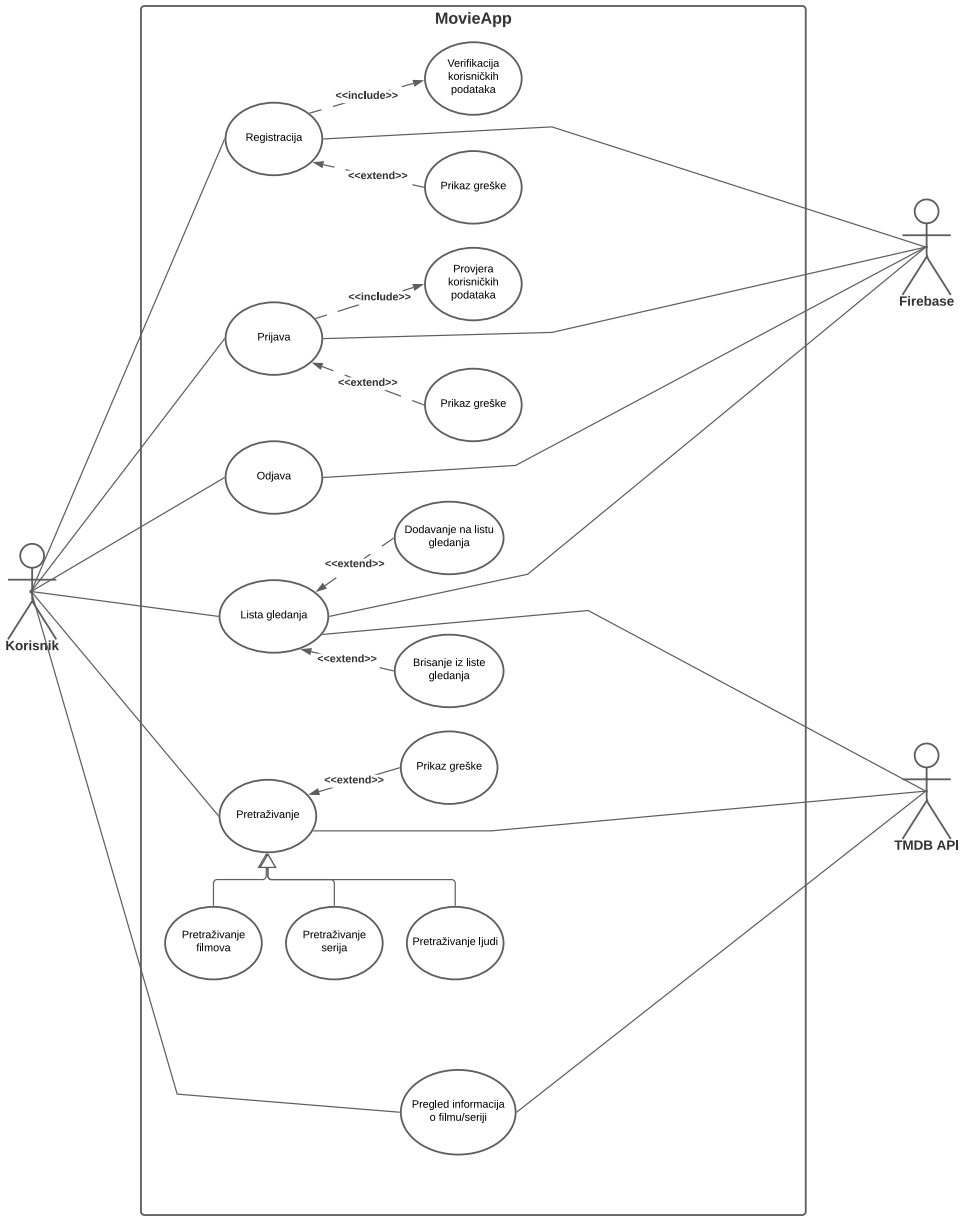
1. Registracija – budući da aplikacija koristi Firebase Auth servis, cijeli proces registracije novog korisnika odvija se unutar tog servisa. Upisom traženih korisničkih podataka i pritiskom na gumb Sign Up, korisnički podaci se verificiraju i šalju Firebase Auth servisu koji kreira novog korisnika sa prethodno poslanim korisničkim podacima. Nakon uspješnog kreiranja novog korisnika servis nazad šalje obavijest o uspješnosti.
2. Prijava – procesom prijave također uvelike upravlja Firebase Auth servis. Upisom traženih korisničkih podataka i pritiskom na gumb Log In, korisnički podaci šalju se Firebase Auth servisu koji provjerava postoji li korisnik unutar servisa. Ukoliko korisnik ne postoji na ekranu se prikazuje obavijest da korisnički podaci ne postoje, a ukoliko korisnik postoji u servisu, Firebase Auth servis ga prijavljuje te se korisniku daje mogućnost dodavanja ili brisanja filmova ili serija sa liste gledanja.
3. Prikaz informacija o filmu ili seriji – pozicioniranjem pokazivača iznad slike filma ili serije, prikazuje se njezin naziv. Klikom na naslov, otvara se posebna informativna stranica o filmu ili seriji. Informativna stranica sadrži neke važnije

informacije kao što su: godina izlaska, naslov, status filma ili serije, opis, prosječna ocjena te broj sezona ili epizoda ako je riječ o seriji.

4. Pretraživanje – pretraživanje se odvija u dva koraka. Korisnik upisuje željeni pojam za pretraživanje te odabire jednu od tri ponuđene kategorije prema kojima može pretraživati uneseni pojam. Za sve podatke o filmovima i serijama brine se TMDB API servis preko kojeg se odvija i proces pretraživanja. Uneseni pojam i kategorija šalju se kao parametri u zahtjevu upućenom TMDB API-u. Kao rezultat pretraživanja, TMDB API servis šalje nazad podatke, ukoliko oni postoje, ili grešku ukoliko traženi pojam ne postoji.
5. Dodavanje ili brisanje sa liste gledanja – nakon uspješne registracije i prijave u aplikaciju, korisniku je dozvoljeno dodavanje ili brisanje filmova ili serija sa njegove liste gledanja. Prazna lista gledanja kreira se odmah, prilikom uspješne registracije novog korisnika, unutar Firebase Firestore servisa. Korisnik pritiskom gumba smještenog u desnom gornjem uglu kartice filma ili serije dodaje isti na svoju listu gledanja te se izgled gumba mijenja. Ovisno o izgledu gumba korisnik će znati nalaze li se već određeni film ili serija na njegovoj listi gledanja ili ne. Ukoliko se film ili serija već nalaze na listi gledanja, pritiskom na gumb korisnik će izbrisati određeni film ili seriju sa njegove liste gledanja.

## 6.2. Dijagram slučajeva korištenja

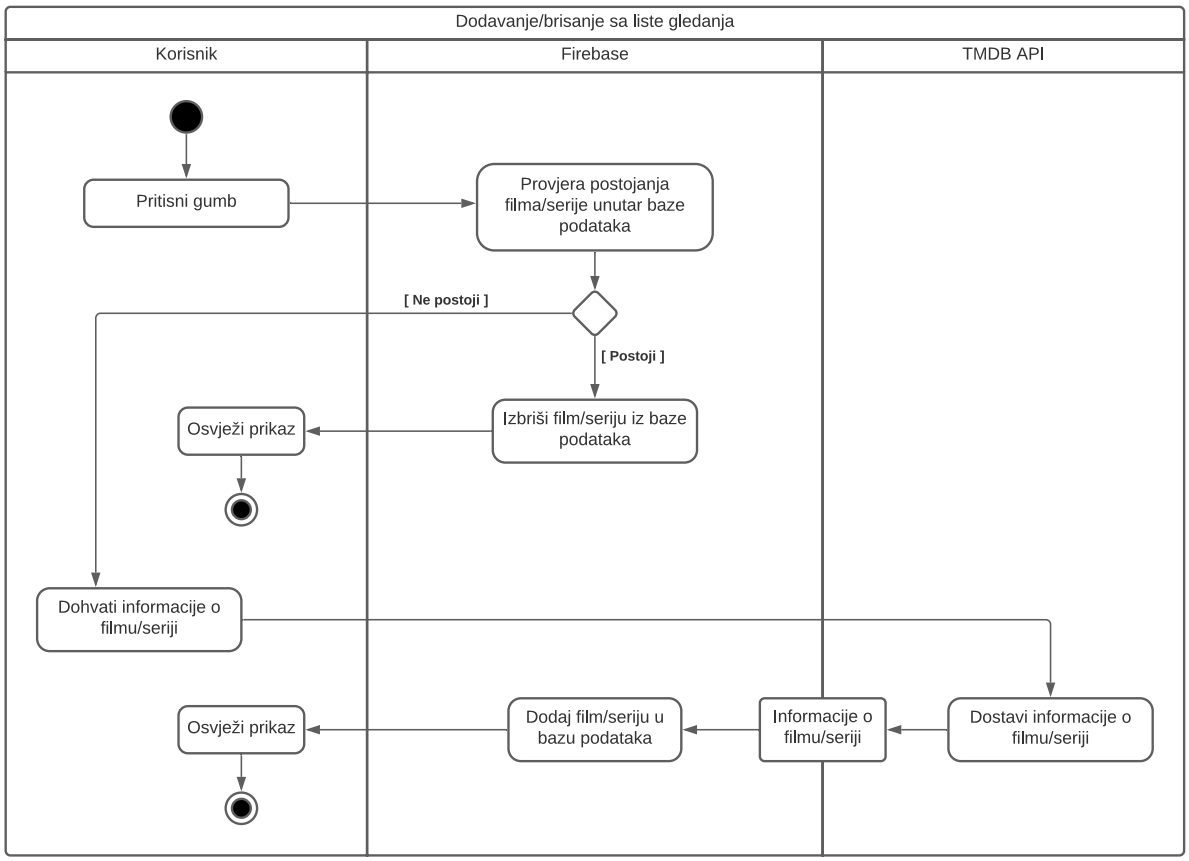
Dijagram slučajeva korištenja (eng. Use Case Diagram) opisuje što sustav radi s motrišta vanjskog promatrača. Budući da trenutna web aplikacija nije toliko opširna, u ovom poglavlju prikazan je dijagram slučajeva korištenja cijele aplikacije, a koja je opisana u prethodnom poglavlju. Kod ove web aplikacije imamo tri učesnika (eng. Actor), jedan primarni i dva sekundarna. Korisnik predstavlja primarnog učesnika, dok Firebase i TMDB API servisi predstavljaju sekundarne učesnike unutar dijagrama slučajeva korištenja.




Slika 9: Prikaz dijagrama slučajeva korištenja

## 6.3. Dijagrami slijeda aktivnosti

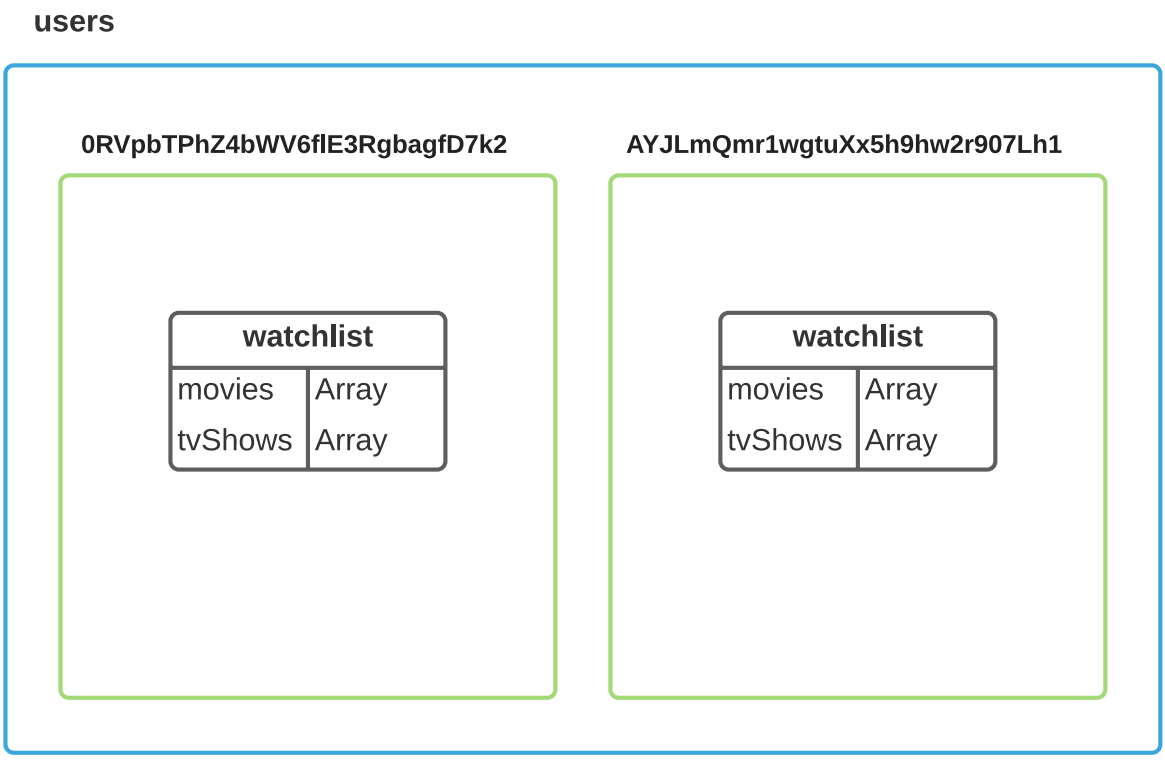
Dijagram slijeda aktivnosti (eng. Activity Diagram) vrlo je koristan za analiziranje slučajeva korištenja opisujući njegovu unutarnju logiku. Svaki dijagram slijeda aktivnosti sadrži tri osnovne stavke: početak korištenja, opis što slučaj korištenja radi, odnosno tijek posla od početne do završne točke i završetak slučaja korištenja. Kroz dijagrama slijeda aktivnosti biti će prikazan proces dodavanja i brisanja filma ili serije sa liste gledanja.




Slika 10: Prikaz dijagrama slijeda aktivnosti

## 6.4. Shema baze podataka

Baza podataka nalazi se u sklopu Firebase servisa i kao takva je u NoSQL formatu. Potrebe za bazom podataka ove aplikacije jako su male pa se tako u bazu podataka spremaju samo filmovi ili serije u obliku liste gledanja. Budući da se radi o NoSQL tipu baze podataka, ona se sastoji samo od kolekcije korisnika. Kolekcija korisnika sadrži dokumente unikatnih korisničkih ID-eva kako bi aplikacija znala da se podaci unutra odnose na točno određenog korisnika. Svaki dokument sadrži stavku watchlist tipa objekt (eng. Object) dok se unutar objekta nalaze još dvije stavke movies i tvShows, oboje tipa polje (eng. Array).




Slika 11: Prikaz sheme baze podataka

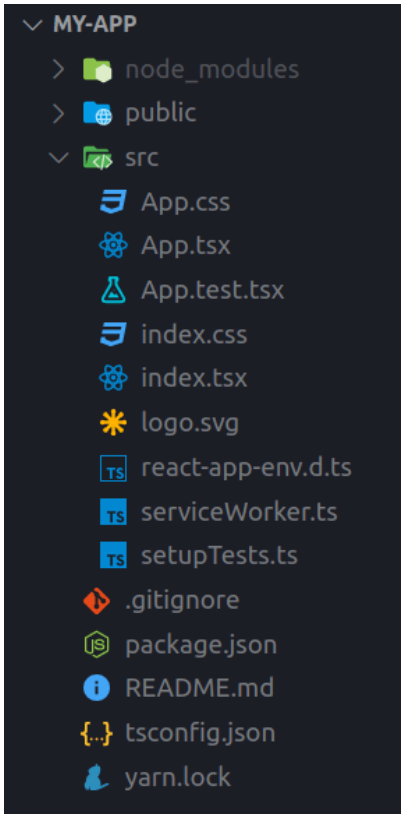
## 6.5. Postavljanje projekta

Budući da se prilikom izrade projekta koristi npm (eng. Node Package Manager) za generiranje početnih datoteka. Node Package Manager standardni je upravitelj paketa za JavaScript.

```
npx create-react-app my-app –template typescript
```

Programski kod 22: Generiranje početnih datoteka projekta

Ova naredba generirat će početnu strukturu React projekta i omogućiti korištenje TypeScripta unutar istog (Programski kod 22). U početku je važno napraviti čišćenje nepotrebnih standardnih datoteka te napraviti dobru strukturu projekta kako bi se projekt mogao što lakše skalirati u kasnijem razvoju (Slika 12).




Slika 12: Prikaz početne strukture projekta

Nakon postavljanja početnih datoteka vrijeme je za instalaciju Tailwind CSS-a. Tailwind CSS instalira se u projekt putem npm-a (Programski kod 23). Cijeli proces opisan je unutar Tailwind CSS dokumentacije pod kategorijom Installation.

```
npm install -D tailwindcss@npm:@tailwindcss/postcss7-compat postcss@^7 autoprefixer@^9
```

Programski kod 23: Prikaz naredbe za instalaciju Tailwind CSS-a

Budući da Create React App ne podržava nativnu konfiguraciju PostCSS preprocesora potrebno je instalirati Craco (Programski kod 24).

```
npm install @craco/craco
```

Programski kod 24: Prikaz naredbe za instalaciju Craco-a

Nakon instalacije potrebno je izmijeniti datoteku package.json na sljedeći način. Sve standardne skripte potrebno je promijeniti u nove skripte pokretane od strane Craco-a (Programski kod 25).

```
{
    // ...
    "scripts": {
     "start": "react-scripts start",
     "build": "react-scripts build",
     "test": "react-scripts test",
     "start": "craco start",
     "build": "craco build",
     "test": "craco test",
      "eject": "react-scripts eject"
    },
  }
```

Programski kod 25: Prikaz package.json datoteke

Kako bi Craco znao koje dodatke treba koristiti potrebno je kreirati konfiguracijsku datoteku za Craco (Programski kod 26).

```
// craco.config.js
module.exports = {
  style: {
    postcss: {
      plugins: [
        require('tailwindcss'),
        require('autoprefixer'),
      ],
    },
  },
}
```

Programski kod 26: Prikaz Craco konfiguracijske datoteke

Konačno, na kraju ovog dugotrajnog procesa potrebno je kreirati konfiguracijsku datoteku za Tailwind CSS, a ona se generira pomoću npm naredbe (Programski kod 27).

```
npx tailwindcss-cli@latest init
```

Programski kod 27: Prikaz naredbe za generiranje konfiguracijske datoteke Tailwind CSS-a

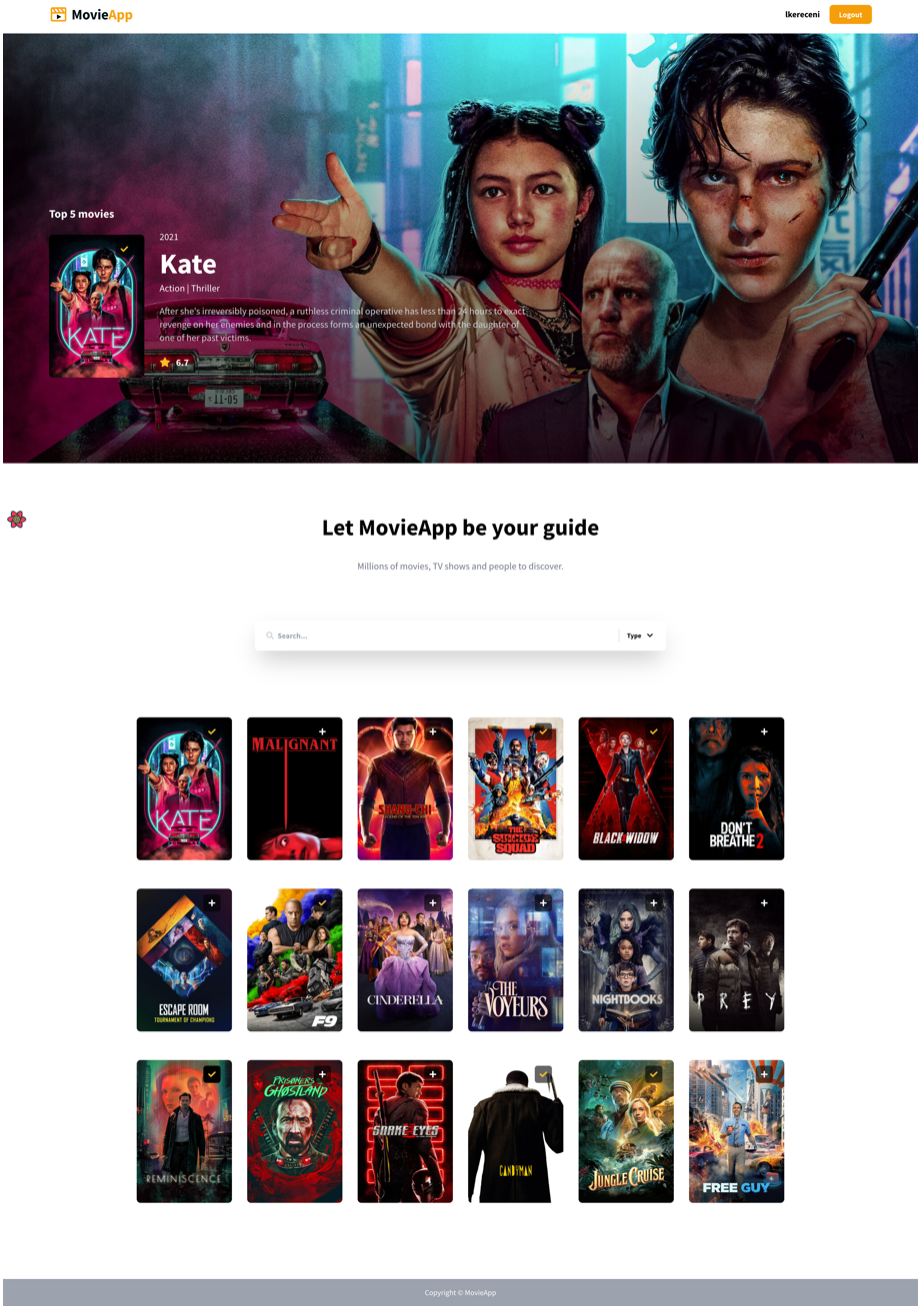
Konfiguracijska datoteka omogućuje promjenu ili dodavanje Tailwind CSS klasa te pridruživanje ili prilagođavanje vrijednosti. Nakon generiranja konfiguracijska datoteka izgleda ovako (Programski kod 28).

```
// tailwind.config.js

module.exports = {
  purge: [],
  darkMode: false, // or 'media' or 'class'
  theme: {
    extend: {},
  },
  variants: {
    extend: {},
  },
  plugins: [],
}
```

Programski kod 28: Prikaz Tailwind CSS konfiguracijske datoteke

## 6.6. Prikaz dijelova aplikacije

U ovom poglavlju prikazani su najvažniji prikazi ekrana aplikacije. Na slici 13 prikazan je prikaz ekrana početne stranice aplikacije.




Slika 13: Prikaz ekrana početne stranice

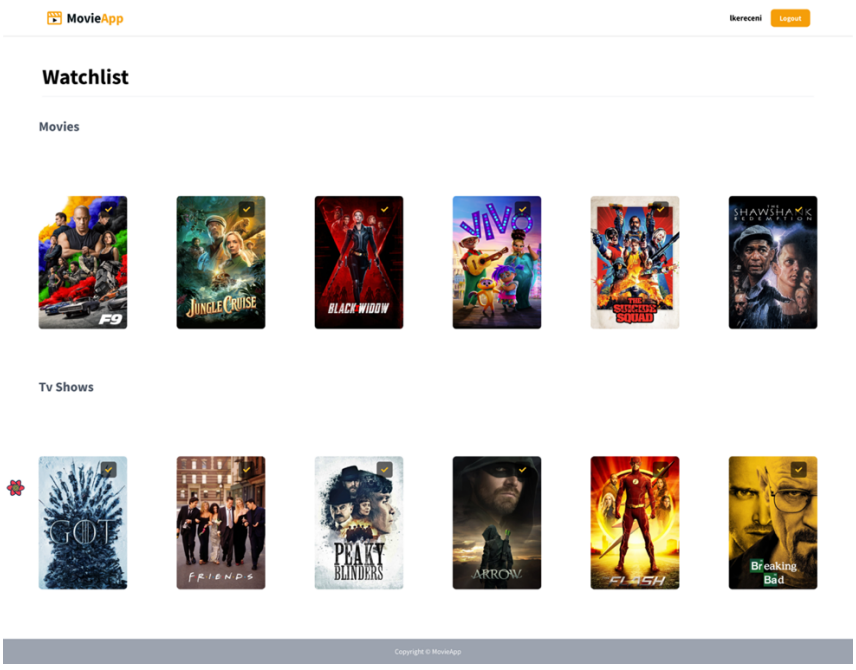
Budući da početna stranica sadrži mnogo komponenti, biti će prikazan programski kod funkcionalnosti pretraživanja (Programski kod 29). Na početku programskog koda uvoze se sve potrebne komponente, servisi, React funkcije i TypeScript tipovi koji služe za pravilan rad komponente pretraživanja. Funkcija useState omogućuje pohranu trenutnog stanja komponente. Do trenutnog stanja dolazi se pomoću prvog parametra, a postavljanje trenutnog stanja vrši se pomoću drugog parametra koji je ujedno i funkcija za postavljanje. Asinkrone funkcije fetchSearchedData i fetchPopularMovies služe za dohvaćanje traženih podataka i trenutno popularnih filmova, a iste se predaju unutar useQuery vanjskih funkcija koje upravljaju cijelim procesom dohvaćanja. Kada se podaci dohvate, spremni su za prikaz u željenom obliku. Cijela Search komponenta je zapravo funkcija koja u sebi sadrži još određenih komponenti kako bi prikaz liste pretraživanja bio upravo takav kakav je na prikazu ekrana. Na početku su prikazani samo trenutno popularni filmovi, ali nakon što korisnik odabere tip i upiše određeni pojam za pretraživanje, odmah se umjesto trenutno popularnih filmova prikazuje rezultat pretraživanja.

```
import { FC, useState } from 'react';
import { useQuery } from 'react-query';
import SearchInput from '@core/components/shared/SearchInput';
import GridView from '@core/components/shared/GridView';
import Poster from '../../poster/components/Poster';
import { IType } from '../interfaces/search.interface';
import { IResponse } from '@core/interfaces/response.interface';
import api from '@core/services/api';

const Search: FC = () => {
    const [term, setTerm] = useState<string | null>(null);
    const [type, setType] = useState<IType | null>(null);
    const fetchSearchedData = async (): Promise<IResponse | undefined> => {
        const config = {
            params: {
                query: term,
            },
        };
        const { data } = await api.get(`/search/${type?.value}`, config);
        return { ...data, media_type: type?.value };
    };
    const fetchPopularMovies = async (): Promise<IResponse | undefined> => {
        const { data } = await api.get('/trending/movie/week');
        return data;
    };
    const { data, isLoading } = useQuery(
        ['searchedData', term, type],
        fetchSearchedData,
        { enabled: !!(term && type) }
    );
    const { data: popularMovies, isLoading: isPopularMoviesLoading } = useQuery(
        'popularMovies',
        fetchPopularMovies
    );

    return (
        <section className="flex flex-col items-center px-6 lg:px-24 my-12">
            <SearchInput
                icon="fas fa-search"
                placeholder="Search..."
                term={term}
                setTerm={setTerm}
                type={type}
```

```
                setType={setType}
            />
            <GridView>
                {!isPopularMoviesLoading && !term ? (
                    <>
                        {popularMovies?.results
                            .slice(0, 18)
                            .map((movie: any) => (
                                <Poster
                                    key={movie.id}
                                    data={movie}
                                    mediaType={movie.media_type}
                                />
                            ))}
                    </>
                ) : (
                    <>
                        {!isLoading &&
                            data?.results.map((result: any) => (
                                <Poster
                                    key={result.id}
                                    data={result}
                                    mediaType={data.media_type}
                                />
                            ))}
                    </>
                )}
            </GridView>
        </section>
    );
};
export default Search;
```

Programski kod 29: Prikaz programskog koda Search komponente

Sljedeća slika prikazuje prikaz ekrana liste spremljenih filmova i serija (Slika 14). Filmovi i serije prikazani su u redove i stupce te sortirani prema tipu.




Slika 14: Prikaz ekrana liste spremljenih filmova i serija

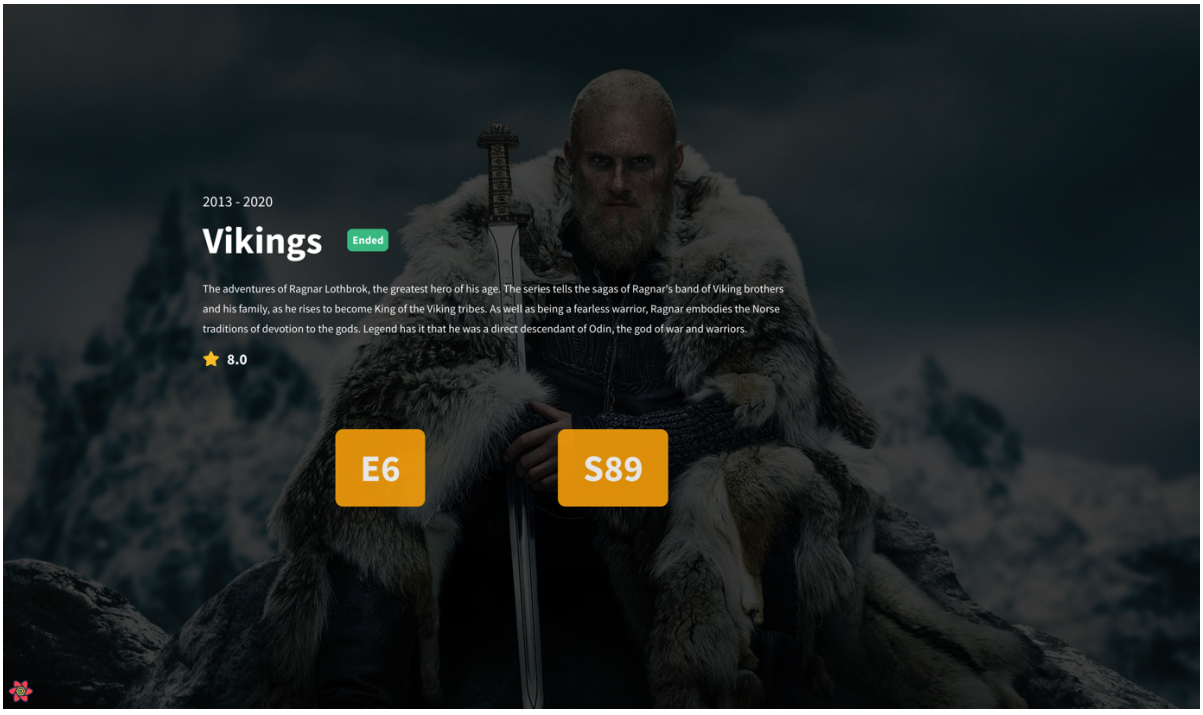
Programski kod 30 prikazuje programski kod Watchlist komponente unutar projekta. Također, na početku programskog koda uvoze se React funkcije, autentifikacijski kontekst, React komponente, te funkcije iz Firebase servisa. React funkcija useContext služi za pristupanje kontekstu, u ovom slučaju autentifikacijskom kontekstu. Autentifikacijski kontekst služi za pohranu informacija o trenutno prijavljenom korisniku. Ukoliko korisnik nije prijavljen u aplikaciju, kontekst će biti prazan te korisnik neće moći pristupiti listi spremljenih filmova i serija. Uvezene Firebase funkcije omogućuju dohvaćanje liste spremljenih filmova i serija, a funkcija za dohvaćanje se pokreće pomoću React funkcije useEffect. React funkcija useEffect omogućuje pokretanje određenih akcija odmah pri postavljanju komponente ili ovisno o nekom određenom vanjskom uvjetu. Dohvaćena lista sprema se u trenutno stanje Watchlist komponente. Spremanje podataka u trenutno stanje pokreće ponovno postavljanje komponente i prikaz liste na ekranu.

```
import { FC, useEffect, useState, useContext } from 'react';
import { AuthContext } from '../../auth/context/AuthContext';
import GridView from '@core/components/shared/GridView';
import { firebase, firebaseFirestore } from '@core/services/firebase';
import Poster from '../../poster/components/Poster';

const Watchlist: FC = () => {
    const user = useContext(AuthContext);
    const [watchlist, setWatchlist] = useState<
        firebase.firestore.DocumentData | undefined
    >(undefined);
    useEffect(() => {
        firebaseFirestore
            .collection('users')
            .doc(user?.uid)
            .onSnapshot(doc => {
                const watchlist = doc.data();
                setWatchlist(watchlist?.watchlist);
            });
    }, []);

    return (
        <section className="mx-10">
        <h1 className="text-3xl md:text-5xl font-bold text-center md:text-left mt-8 mb-4
        md:mt-14 md:mx-8 lg:mx-12 lg:my-12 py-5 border-b-2 border-gray-100">
                Watchlist
            </h1>
            <h2 className="text-2xl md:text-3xl font-bold text-gray-600 md:mx-4 lg:mx-10">
                Movies
            </h2>
            <GridView>
                {watchlist?.movies.map((movie: any) => (
                    <Poster
                        key={movie.id}
                        mediaType="movie"
                        isOnWatchlist
                        data={movie}
                    />
                ))}
            </GridView>
```

```
            <h2 className="text-2xl md:text-3xl font-bold text-gray-600 md:mx-4 lg:mx-10">
                Tv Shows
            </h2>
            <GridView>
                {watchlist?.tvShows.map((tvShow: any) => (
                    <Poster key={tvShow.id} data={tvShow} />
                ))}
            </GridView>
        </section>
    );
};

export default Watchlist;
```

Programski kod 30: Prikaz programskog koda Watchlist komponente

Posljednji prikaz ekrana prikazuje informativnu stranicu o odabranom filmu ili seriji (Slika 15). Programski kod 31 prikazuje programski kod cijele informativne stranice o filmu ili seriji.




Slika 15: Prikaz detalja o filmu ili seriji

Klikom korisnika na određeni film ili seriju, korisnika se preusmjerava na informativnu stranicu o filmu ili seriji. Pomoću funkcije useLocation dohvaća se id odabranog filma ili serije koji služi za daljnje dohvaćanje informacija o filmu ili seriji. Određeni id prosljeđuje se asinkronoj funkciji fetchDetails u kojoj se određuje radi li se o filmu ili seriji i prema tome se formira zahtjev prema TMDB API-u. Cijeli proces dohvaćanja

podataka kontrolira useQuery funkcija. Nakon što su podaci dohvaćeni i kontrolna varijabla isLoading poprime vrijednost istine, dohvaćeni podaci spremni su za prikaz na ekran. Prikaz informacija također se formira ovisno o tipu, odnosno da li se dohvaćene informacije odnose na film ili seriju. Neke od informacija koje se prikazuju na ekranu su godina izlaska, naslov filma ili serije, trenutni status u kojem se film ili serija nalaze, kratki opis, prosječna ocjena te ukoliko se radi o seriji prikazuje se broj sezona i epizoda.

```
import { FC } from 'react';
import { useLocation } from 'react-router-dom';
import { useQuery } from 'react-query';
import api from '@core/services/api';

const DetailsPage: FC = () => {
    const { state } = useLocation<any>();
    const fetchDetails = async (): Promise<any | undefined> => {
        if (!state?.release_date) {
            const { data } = await api.get(`/tv/${state?.id}`);
            return data;
        } else {
            const { data } = await api.get(`/movie/${state?.id}`);
            return data;
        }
    };
    const {
        data: details,
        isLoading,
        isFetchedAfterMount,
    } = useQuery('mediaDetails', fetchDetails, { refetchOnMount: 'always' });

    return (
        <div className="w-screen h-screen relative overflow-hidden">
            {!isFetchedAfterMount ||
                (!isLoading && (
                    <>
                        <div
                            className={`absolute w-full left-80 ${
                                details?.release_date ? 'inset-y-1/2' : 'top-72'
                            } z-10`}>
                            {details?.release_date ? (
                                <p className="text-white text-2xl">
                                    {details?.release_date?.substring(0, 4)}
                                </p>
                            ) : (
                                <p className="text-white text-2xl">
                                    {details?.first_air_date?.substring(0, 4)} -{' '}
                                    {details?.last_air_date &&
                                        details?.last_air_date.substring(0, 4)}
                                </p>
                            )}
                            <div className="flex items-center mt-4 text-white">
                                <a
                                    href={
                                        details?.imdb_id &&
                                        `https://www.imdb.com/title/${details?.imdb_id}`
                                    }
                                    target="_blank"
```

```
                                    rel="noreferrer"
                                     className="text-white text-6xl font-bold no-underline
                                     hover:underline cursor-pointer"
                                    style={{
                                        textDecorationColor: '#f59e09',
                                        color: 'inherit',
                                    }}>
                                    {details?.title || details?.name}
                                </a>
                                <span
                                    className={`flex h-9 items-center ${
                                        details?.status !== 'Released'
                                            ? 'bg-green-500'
                                            : 'bg-yellow-500'
                                    } text-white text-lg font-bold p-2 ml-10 rounded-lg`}>
                                    {details?.status}
                                </span>
                            </div>
                            <p className="w-1/2 text-white text-lg mt-8 leading-8">
                                {details?.overview}
                            </p>
                            <div className="mt-4">
                                <i className="fas fa-star text-yellow-400 text-2xl"></i>
                                <span className="text-white text-2xl font-bold ml-3">
                                    {details?.vote_average.toFixed(1)}
                                </span>
                            </div>
                            {!details?.release_date && (
                                    <div className="flex justify-evenly w-1/2 text-white font-
                                    bold text-6xl mt-24">
                                    <div className="bg-yellow-500 rounded-xl px-10 py-8
                                    opacity-90">
                                        <span>
                                            E{details?.number_of_seasons}
                                        </span>
                                    </div>
                                     <div className="bg-yellow-500 rounded-xl px-10 py-8
                                     opacity-90">
                                        <span>
                                            S{details?.number_of_episodes}
                                        </span>
                                    </div>
                                </div>
                            )}
                        </div>
                        <div className="w-full h-full absolute bg-black opacity-70"></div>
                        <img
                            className="w-full h-full object-cover"

                              src={`${process.env.REACT_APP_TMDB_IMAGES_URL}${details?.backdro
                              p_path}`}
                            alt="details-bg" />
                    </>
                ))}
            </div>
    );
};

export default DetailsPage;
```

Programski kod 31: Prikaz programskog koda DetailsPage komponente

# 7. Zaključak

Ubrzanim razvojem Interneta kao mreže nastao je Web. Kako se Web kao platforma izuzetno brzo razvija i širi, dolazi do potrebe za razvojem novih tehnologija koje bi omogućile nekakav napredak. Razvojem tehnologije i raznih tehnika izrade korisničkih sučelja proces dizajna korisničkog sučelja zadnjih godina dobio je dodatno na značaju. Sve više se uz pojam korisničkog sučelja veže i pojam korisničkog iskustva. Dizajn korisničkog iskustva je proces koji timovi za dizajn koriste prilikom stvaranja proizvoda koji korisnicima pružaju smislena i relevantna iskustva. U današnjem svijetu postoji mnogo programskih jezika, okvira, biblioteka i alata kojima se programeri mogu koristiti prilikom izrade sadržaja na Webu, ali mnogo je teži zadatak odabrati onaj pravi koji će ispuniti sve zahtjeve. Analizom CSS okvira pojašnjene su kategorije i razlike između CSS okvira današnjice, kao i prednosti, ali i nedostaci jednog naprema drugom.

CSS okvir Tailwind CSS, uz Bootstrap, jedan je od najkorištenijih CSS okvira u današnje vrijeme koji omogućuje izuzetno brzo razvijanje korisničkih sučelja aplikacije. Njegova sintaksa bazirana na klasama omogućuje programeru da kreira korisničko sučelje sa minimalnim znanjem CSS-a. Mogućnost razvoja aplikacija korištenjem okvira dodatno je olakšao i ovako dugotrajan proces. Proces pripreme koji uključuje početne postavke projekta i nije baš najjednostavniji za potpune početnike i tu se vidi mogućnost napretka u nadolazećim vremenima.

Praktičan dio razvijan je pomoću JavaScript biblioteke React i CSS okvira TailwindCSS. Iz priloženih prikaza ekrana u prethodnom poglavlju može se vrlo brzo uočiti da se određene komponente u dizajnu same aplikacije ponavljaju, odnosno mogu se ponovno iskoristiti na raznim mjestima, ali u druge svrhe. Glavna prednost izrade aplikacija pomoću React-a je mogućnost izrade komponenti koje zajedničkim spajanjem i interakcijom čine jednu cjelinu koja se naziva web aplikacija. Korištenjem takvog pristupa i kombinacijom React-a i CSS okvira TailwindCSS postoji mogućnost dodatne optimizacije krajnje aplikacije. Budući da se sve ponavljajuće komponente nalaze samo na jednom mjestu u projektu te se cijeli stil nalazi također u dokumentu pojedine komponente dodatno se štedi i na stilskim dokumentima koji bi se potencijalno pojavljivali za svaku izrađenu React komponentu. Glavna karakteristika TailwindCSS-a je u tome da se prilikom kompilacije izbacuju sve neiskorištene klase te se time dodatno štedi na veličini same aplikacije. Objava programskog koda aplikacije na servisu kao što je GitHub olakšava izradu aplikacija u timovima kao i pregledavanje koda od treće strane ukoliko je to dozvoljeno, a objavom web aplikacije na nekom od servisa kao što su Netlify ili Vercel ona postaje dostupna korisnicima na korištenje.

# Popis literature


[1] D. Kermek, „Internet i Web. Povijest Interneta i Weba. Razvoj Interneta i Weba“, nastavni materijali na predmetu Web dizajn i programiranje [Moodle], Sveučilište u Zagrebu, Fakultet organizacije i informatike, Varaždin

[2] M. Schwarzmüller, „How The Web Works“, 2019. [Na internetu]. Dostupno: https://academind.com/tutorials/how-the-web-works/ [pristupano: 16.02.2021.]

[3] Norman, D., (2013), The design of everyday things, Basic Books, New York

[4] N. Babich, „The UX Design Process: Everything You Need To Know“, [Blog post]. 2020. [Na internetu]. Dostupno: https://xd.adobe.com/ideas/guides/ux-design-process-steps/ [pristupano: 18.05.2021.]

[5] J. Prša (bez dat.) The UX Strategy [Na internetu]. Dostupno: https://q.agency/blog/the-ux-strategy [pristupano: 10.07.2021.]

[6] Interaction Design Foundation (bez dat.) User Interface Design [Na internetu]. Dostupno: https://www.interaction-design.org/literature/topics/ui-design [pristupano: 10.07.2021.]

[7] J. Duckett, HTML & CSS: design and build websites, 1. izd. Indianapolis, IN: John Wiley & Sons, Inc., 2011.

[8] T. Firdaus, „CSS Preprocessors Compared: Sass vs LESS“, [Blog post]. 2019. [Na internetu]. Dostupno: https://www.hongkiat.com/blog/sass-vs-less/ [pristupano: 13.08.2021.]

[9] Tailwind CSS (bez dat.) Tailwind CSS Docs [Na internetu]. Dostupno: https://tailwindcss.com/docs [pristupano: 14.08.2021.]

[10] J. O. Meiert, The Little Book of HTML/CSS Frameworks, 1.izd. Sebastopol, CA: O'Reilly Media, Inc., 2015.

[11] N. Rappin, Modern CSS with Tailwind: Flexible Styling Without the Fuss, 1.izd. Raleigh, NC: The Pragmatic Programmers, LLC., 2021.

# Popis slika

# Popis tablica

# Popis programskih kodova

# Prilozi

[1] Izvorni kod praktičnog dijela završnog rada dostupan je na GitHub servisu:

https://github.com/kereceniluka/movie-app-ts

[2] Objavljen praktičan dio završnog rada na Netlify servisu:

https://movie-app-lkereceni.netlify.app/