Razvoj web aplikacija pomoću okvira React

Kajić, Mladen

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mladen Kajić

RAZVOJ WEB APLIKACIJA POMOĆU OKVIRA REACT

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Mladen Kajić

Matični broj: 0016147999

Studij: Informacijski i poslovni sustavi

Razvoj web aplikacija pomoću okvira React

ZAVRŠNI RAD

Mentor:

Prof. dr. sc. Dragutin Kermek

Varaždin, rujan 2023.

Mladen Kajić

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/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Završni rad bavi se procesom razvoja web aplikacije korištenjem Reacta kao programskog okvira. U njemu će se razmatrati jezici i specifikacije koje se koriste u web tehnologijama, a koje su neophodne za razumijevanje osnova razvoja web aplikacija.

Također će biti opisane kategorije programskih jezika koji se koriste za razvoj na strani preglednika, kao i komponente koje se koriste za razvoj web aplikacija. Uspoređivat će se različiti okviri koji se koriste na strani preglednika, s naglaskom na React.

Jedan od ključnih aspekata rada je sigurnost web aplikacije, a ukratko će biti opisane neke od najvažnijih sigurnosnih značajki koje su neophodne za razvoj sigurnih web aplikacija.

U sklopu rada izradit će se web aplikacija uz pomoć Reacta. Bit će opisani neki od ključnih koncepata i tehnologija potrebnih za razvoj web aplikacija pomoću Reacta. Razvijena web aplikacija bit će praktičan primjer primjene teorijskih koncepata opisanih u radu, te će pokazati kako se pomoću Reacta mogu izraditi moderne, brže i skalabilne web aplikacije.

Ključne riječi: web aplikacija; HTML; CSS; JavaScript; razvojni okvir; React; sigurnost

Sadržaj

1. Uvod

U današnjem dobu, web aplikacije igraju ključnu ulogu u našim životima. Web aplikacije omogućavaju korisnicima pristup raznim uslugama, sadržajima i funkcionalnostima putem web preglednika na različitim platformama i uređajima. One su postale neophodan alat za komunikaciju, e-trgovinu, obrazovanje, zabavu, bankarstvo, upravljanje projektima i još mnoge druge svakodnevne aktivnosti.

Razvoj web aplikacija zahtijeva napredne tehnologije, strukturirane pristupe i efikasne okvire kako bi se osigurala intuitivna korisnička interakcija, brza izvedba, visoka sigurnost i skalabilnost. Jedan od najpopularnijih i najmoćnijih programskih okvira koji se koristi za razvoj web aplikacija je React.

React je razvojni okvir otvorenog koda koji se koristi za izgradnju korisničkog sučelja web aplikacija. On pruža efikasan način za organiziranje i upravljanje kompleksnim komponentama, omogućava ponovnu upotrebu koda i olakšava razvoj skalabilnih aplikacija. React pruža bržu i učinkovitiju reaktivnost korisničkog sučelja u odnosu na tradicionalne pristupe.

Razvoj web aplikacija pomoću Reacta zahtijeva poznavanje osnova HTML-a, CSS-a i JavaScripta, ali također pruža mogućnost korištenja modernih praksi kao što su komponentna arhitektura, deklarativno programiranje i upotreba JSX sintakse. React se također integrira s raznim alatima i bibliotekama koje olakšavaju razvoj, kao što su Redux za upravljanje stanjem aplikacije, React Router za upravljanje rutama i Axios za upravljanje HTTP zahtjevima.

Ovaj završni rad će istražiti ključne koncepte i tehnologije potrebne za razvoj web aplikacija pomoću Reacta. Kroz praktični primjer izrade web aplikacije, demonstrirat će se snaga i fleksibilnost Reacta u stvaranju modernih, interaktivnih i web aplikacija visokih performansi. Također će se istražiti sigurnosne značajke i najbolje prakse koje treba uzeti u obzir prilikom razvoja sigurnih web aplikacija.

2. Web aplikacije

Web aplikacije su posebna vrsta softvera koja se izvodi preko web preglednika. Nezavisno od operacijskog sustava uređaja, one omogućuju korisnicima da pristupe aplikaciji u bilo kojem trenutku i na bilo kojem mjestu gdje postoji pristup internetu. Ova vrsta aplikacija izuzetno je praktična jer ne zahtijeva instalaciju softvera na uređaju.

Kako bi se web aplikacije mogle pravilno izgraditi i funkcionirati, koriste se razni programski jezici i tehnologije. Web aplikacija obično se sastoji od dva dijela: korisničke (eng. front-end) i poslužiteljske (eng. back-end) strane.

Korisnička strana je onaj dio aplikacije s kojim korisnik ima izravnu interakciju. To je dio web aplikacije koji korisnik vidi, sastavljen od vizualnih elemenata i korisničkog sučelja. Za izgradnju korisničke strane koristi se HTML za strukturu stranice, CSS za oblikovanje i stiliziranje, te JavaScript za interaktivnost. Ponekad se koriste i razni okviri i biblioteke poput Reacta, Angulara ili Vue.js-a kako bi se olakšao i ubrzao razvoj.

Poslužiteljska strana, s druge strane, jest onaj dio aplikacije koji korisnik ne vidi izravno. Ona uključuje server, aplikacijski softver i bazu podataka. Ova strana je zadužena za obradu zahtjeva koje šalje korisnička strana, upravljanje bazom podataka, provođenje poslovne logike, autentifikaciju, autorizaciju, i slično. Za izgradnju poslužiteljske strane koriste se programski jezici poput Pythona, Rubyja, PHP-a, Jave i JavaScript-a.

Komunikacija između korisničke i poslužiteljske strane odvija se putem HTTP protokola. Korisnik šalje HTTP zahtjev serveru, koji zatim odgovara HTTP odgovorom. Iz sigurnosnih razloga, često se koristi HTTPS, sigurnosna verzija HTTP-a koja štiti podatke tijekom prijenosa.

Pored tradicionalnih web aplikacija, sve je popularnije korištenje tzv. jednostraničnih aplikacija (Single Page Applications, SPA). Ove aplikacije funkcioniraju tako da kada korisnik pristupa web stranici, stranica se učitava samo jednom, a dodatni sadržaj i stranice se dinamički učitavaju i ažuriraju bez ponovnog učitavanja cijele stranice. Ovo omogućava bržu korisničku interakciju sličnu onoj na stolnim aplikacijama. Razlika između tradicionalnih web aplikacija i jednostraničnih aplikacija leži prvenstveno u načinu na koji se sadržaj prikazuje korisniku. U tradicionalnim web aplikacijama, svaka nova stranica ili interakcija korisnika zahtijeva od servera da pošalje novi HTML dokument. S druge strane, jednostranične aplikacije učitavaju cijeli sadržaj samo jednom, a naknadne interakcije s web stranicom rezultiraju samo ažuriranjem dijelova sadržaja na stranici. Kako bi se to postiglo, jednostranične aplikacije koriste određene tehnike programiranja, kao što su asinkroni pozivi i manipulacija DOM-a (eng. Document Object Model), koje omogućuju dinamičku promjenu

sadržaja stranice bez potrebe za učitavanjem čitave nove stranice. Korištenje jednostraničnih aplikacija rezultira smanjenjem mrežnog prometa i bržim učitavanjem stranice, ali je složenije za implementaciju, te je stoga uobičajeno koristiti programske okvire za njihovu implementaciju.[1]

2.1. HTML

HTML (eng. Hypertext Markup Language) je standardni jezik koji se koristi za izgradnju web stranica. On definira strukturu i sadržaj web stranica pomoću oznaka ili tagova. Glavna svrha HTML-a je organizacija informacija na web stranicama. To se postiže pomoću različitih elemenata i oznaka koji određuju vrstu sadržaja, kao što su naslovi, odlomci, slike, tablice, liste itd.

HTML elementi su osnovni građevni blokovi HTML-a. Oni predstavljaju različite dijelove web stranice i koriste se za definiranje strukture i organizacije sadržaja. HTML elementi imaju svoju otvarajuću oznaku ("<oznaka>") i zatvarajuću oznaku ("</oznaka>"), gdje oznaka predstavlja naziv elementa. Zatvarajuća oznaka označava kraj određenog elementa i koristi se za označavanje granica tog elementa. Na primjer, otvarajuća oznaka označava početak odlomka, a zatvarajuća oznaka označava kraj odlomka. Neki HTML elementi su samozatvarajući, što znači da nemaju zatvarajuću oznaku. Oni se zatvaraju koristeći oznaku za samozatvaranje. Ova oznaka ima oblik "<oznaka />" i koristi se kada element ne zahtijeva unutarnji sadržaj ili nije potrebno navesti dodatne atribute. [2]

Neki primjeri oznaka koje se koriste su sljedeći: oznaka <h1> se koristi za naslov prve razine, oznaka se koristi za odlomke, oznaka <a> se koristi za poveznice, oznaka <img> se koristi za prikaz slika, itd. Svaki od ovih elemenata ima svoju ulogu i značenje u strukturi web stranice.

HTML atributi se koriste unutar otvarajućih oznaka kako bi se pružile dodatne informacije ili upute o određenom elementu. Atributi pružaju dodatna svojstva ili konfiguracije za elemente. Svaki element može imati različite atribute koji se koriste za kontroliranje izgleda, ponašanja ili funkcionalnosti elementa.

Na primjer, oznaka <img> ima atribut src koji označava izvor slike koja će se prikazati, a oznaka <a> ima atribut href koji definira URL odredišta poveznice. Atributi se definiraju unutar otvarajuće oznake i obično imaju vrijednost koja se postavlja pomoću jednakosti, kao što je <img src="slika.jpg">.

Veza između HTML elemenata i atributa je takva da se atributi primjenjuju na određeni element kako bi pružili dodatne informacije ili funkcionalnosti za taj element. Na primjer, atributi se koriste za specificiranje izvora slike, odredišta poveznice, stilova, veličine i drugih svojstava elemenata.

HTML elementi i atributi zajedno omogućuju programerima da strukturiraju sadržaj web stranica i pruže dodatne informacije ili upute za prikaz ili funkcioniranje elemenata na stranici. Kombinacija elemenata i atributa omogućuje stvaranje bogatog i interaktivnog sadržaja na webu.

Osnovna struktura HTML stranice je sljedeća:

```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>Dokument</title>
</head>
<body>

</body>
</html>
```

Linija „<!DOCTYPE html>“ naziva se preambula te ona označava verziju HTML koja se koristi, kako bi preglednik ispravno interpretirao oznake koje se koriste.

Oznake <html> i </html> označavaju početak i kraj HTML dokumenta, te se unutar njih nalazi sav sadržaj HTML dokumenta.

Oznake <head> i </head> označavaju početak i kraj zaglavlja dokumenta. Zaglavlje sadrži meta informacije o stranici i nije prikazano izravno na stranici. U primjeru se u zaglavlju dokumenta nalaze meta podaci stranice te naslov stranice.

Linija „<meta charset="UTF-8" />“ specificira karakteristični skup znakova koji se koristi u dokumentu. UTF-8 je često korišten skup znakova koji podržava razne jezike.

Linija „<meta name="viewport" content="width=device-width, initial-scale=1.0" />“ specificira prikaznu površinu i inicijalno skaliranje za prilagodbu stranice na različite uređaje i veličine ekrana. „width=device-width“ znači da će širina prikazne površine biti jednaka širini uređaja, a „initial-scale=1.0“ postavlja inicijalno skaliranje na 1, što znači da će se stranica inicijalno prikazivati bez ikakvog skaliranja ili promjene veličine.

Oznake <title> i </title> postavljaju naslov stranice koji se prikazuje na kartici preglednika ili u rezultatima pretraživanja. U navedenom primjeru naslov stranice postavljen je na „Dokument“ te se može prilagoditi po potrebi.

Oznake <body> i </body> označavaju početak i kraj tijela dokumenta. Unutar tijela dodaje se stvarni sadržaj stranice koji će se prikazati korisniku.

Uobičajena struktura tijela web stranice sadrži nekoliko ključnih dijelova: zaglavlje stranice (eng. header), navigacija (eng. navigation), glavni sadržaj (eng. main content) i podnožje (eng. footer). Ovi dijelovi pomažu u organizaciji i boljoj navigaciji kroz sadržaj web stranice.

Zaglavlje je gornji dio web stranice koji često sadrži logotip, naslov stranice, navigacijske elemente ili druge važne informacije koje bi trebale biti vidljive na početku stranice. Zaglavlje pruža korisnicima brzi pregled o čemu se stranica radi i često sadrži elemente koji su dostupni na svim stranicama unutar web mjesta.

Navigacija je područje koje sadrži skup veza ili izbornika koje korisnici mogu koristiti za navigaciju kroz web stranicu. Navigacija omogućuje korisnicima jednostavno kretanje na različite dijelove web stranice ili na druge stranice unutar web mjesta.

Glavni sadržaj je središnji dio web stranice koji sadrži glavni informacijski sadržaj koji se prikazuje korisnicima. Ovdje se nalaze članci, slike, videozapisi ili bilo koji drugi sadržaj koji je specifičan za tu stranicu. Glavni sadržaj je obično smješten nakon zaglavlja i navigacije.

Podnožje je donji dio web stranice koji obično sadrži informacije o autorskim pravima, poveznice na uvjete korištenja, politiku privatnosti, kontaktne informacije ili druge važne informacije koje bi korisnicima mogle biti zanimljive. Podnožje je obično prisutno na svim stranicama unutar web mjesta.

Ova struktura tijela web stranice pomaže organizirati sadržaj na način koji olakšava korisnicima pronalaženje informacija i kretanje kroz stranicu. Svaki dio ima svoju svrhu i doprinosi boljoj korisničkoj navigaciji i iskustvu na web stranici. Naravno, ovisno o potrebama i dizajnu web stranice, ovi dijelovi mogu se prilagoditi ili nadopuniti drugim elementima kako bi se postigla željena funkcionalnost i izgled.[2]

2.2. CSS

CSS (eng. Cascading Style Sheets) je jezik koji se koristi za stilizaciju HTML elemenata i definiranje izgleda web stranica. Ovaj jezik omogućava odvajanje dizajna od sadržaja, što znači da se HTML koristi za strukturu i sadržaj stranice, dok se CSS koristi za oblikovanje i stilizaciju elemenata. CSS koristi se za definiranje različitih stilova elemenata kao što su boja teksta, pozadina, fontovi, veličina i raspored elemenata, te mnoge druge vizualne osobine.

CSS može se pisati unutar atributa HTML elementa na koji se želi primijeniti, može se pisati u zaglavlju HTML dokumenta, te se može pisati u zasebnoj datoteci. Kada se piše u zaglavlju HTML dokumenta, koriste se oznake <style> i </style>, te se unutar njih piše CSS kod. Kada se CSS piše u zasebnoj datoteci, ta datoteka ima ekstenziju .css, te se datoteka treba uključiti u HTML dokument, za što se koristi <link> oznaka.

Jednostavan primjer CSS-a:

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

Navedeni primjer mijenja boju pozadine tijela HTML stranice u plavu boju. U navedenom primjeru, „body“ je selektor, „background-color“ je svojstvo te je „blue“ vrijednost koja je dodijeljena tom svojstvu.

Neki od ključnih koncepata bitnih za razumijevanje CSS-a su sljedeći:

Selektori, pseudoklase, model kutije, responzivni dizajn.

2.2.1. Selektori

CSS selektori koriste se kako bi se definiralo na koje HTML elemente se treba primijeniti definirani stil. Selektori mogu biti jednostavni i kompleksni. Razlika je u tome što kompleksni selektor kombinira više jednostavnih selektora korištenjem tzv. kombinatora.

2.2.1.1. Jednostavni selektori

Jednostavni selektori dijele se na više vrsta: univerzalni, jednoznačni, implicitni, eksplicitni, selektor atributa.

Univerzalni selektor definira da se navedeni stil odnosi na sve elemente. Univerzalni selektor definira se znakom *.

Jednoznačni selektor definira da se definirani stil primjenjuje na onaj HTML element na stranici koji ima definiran jednak jedinstveni identifikator. HTML elementu se jedinstveni identifikator dodjeljuje atributom „id“. U pravilu se u HTML-u ne bi smio definirati isti jedinstveni

identifikator na više elemenata, tako da se stilovi definirani za jedinstveni identifikator primjene samo na jedan element na stranici. U CSS-u se jednoznačni selektor definira znakom #.

Implicitni selektor definira da se definirani stil primjenjuje na sve instance HTML elementa koji ima istu oznaku. Npr. ako je implicitni selektor p, definirani stil primijenit će se na sve odlomke odnosno <p> elemente na stranici.

Eksplicitni selektor definira da se definirani stil primjenjuje na sve HTML elemente na stranici koji imaju jednaku klasu. HTML elementu se klasa dodjeljuje atributom „class“. U CSS-u se eksplicitni selektor definira točkom.

Selektor atributa definira da se navedeni stil primjenjuje na sve elemente HTML elemente koji imaju navedeni atribut i vrijednost atributa. Na primjer, ako definiramo CSS stil selektorom atributa [class=“klasa“], definirani stil primijenit će se na sve HTML elemente koji imaju atribut „class“ čija je vrijednost „klasa“.

Selektori se također mogu kombinirati, npr. može se kombinirati implicitni selektor p i eksplicitni selektor klasa. Kombinacija tih selektora bila bi „p.klasa“, te bi se definirani stil primijenio samo na HTML elemente p, odnosno odlomke, koji imaju atribut „class“ čija je vrijednost „klasa“.

2.2.1.2. Kompleksni selektori

Kao što je ranije navedeno, kompleksni selektori koriste kombinatore kako bi povezali više jednostavnih selektora. Postoje 4 vrste kombinatora.

Kombinator pretka omogućava odabir elemenata koji su unutar drugih elemenata. Za definiranje kombinatora pretka koristi se razmak između 2 kombinatora. Na primjer, ako je definiran stil za „div a“, navedeni stil primijenit će se na sve <a> elemente koji se nalaze unutar elementa, odnosno kojima je element predak.

Kombinator djeteta omogućuje odabir samo onih elemenata koji su neposredno unutar drugih elemenata. Za definiranje kombinatora djeteta koristi se znak >. Na primjer, ako je definira stil za „div > a“, navedeni stil primijenit će se samo na one <a> elemente koji su smješteni izravno unutar elementa

Kombinator susjeda omogućuje odabir elemenata koji slijede drugi element te se nalaze na istoj razini. Za definiranje kombinatora susjeda koristi se znak tilda (~). Na primjer, ako je definiran stil za „div ~ a“, navedeni stil primijenit će se na sve elemente <a> koji se nalaze nakon elementa na istoj razini.

Kombinatori sljedećeg susjeda omogućuje odabir samo onih elemenata koji neposredno slijede drugi element na istoj razini. Za definiranje kombinatora sljedećeg susjeda

koristi se znak +. Na primjer, ako je definiran stil za „div + a“, navedeni stil primijenit će se samo na one elemente <a> koji se nalaze neposredno nakon elementa na istoj razini, odnosno između njih nema drugih elemenata na istoj razini.

2.2.2. Pseudoklase

Pseudoklase su posebne oznake koje se koriste za odabir elemenata u određenim stanjima ili određenog tipa. Pseudoklase dodaju interaktivnost i omogućavaju prilagođavanje stilova na temelju korisničkih radnji ili stanja elemenata. Nekoliko primjera često korištenih pseudoklasa:

- Pseudoklasa :hover – primjenjuje se na element kada se mišem pređe preko njega na stranici
- Pseudoklasa :active – primjenjuje se na element kada je aktivan, odnosno kada ga se pritisne mišem
- Pseudoklasa :focus – primjenjuje se na element kada je u fokusu, npr. polje za unos teksta je u fokusu kada korisnik klikne u njega kako bi upisao tekst

2.2.3. Model kutije

Model kutije (eng. box model) je osnovni koncept u CSS-u koji opisuje kako se HTML elementi prikazuju i zauzimaju prostor na web stranici. Prema modelu kutije, svaki HTML element se sastoji od četiri dijela: margine (eng. margin), granice (eng. border), punjenje (eng. padding) i sadržaj.

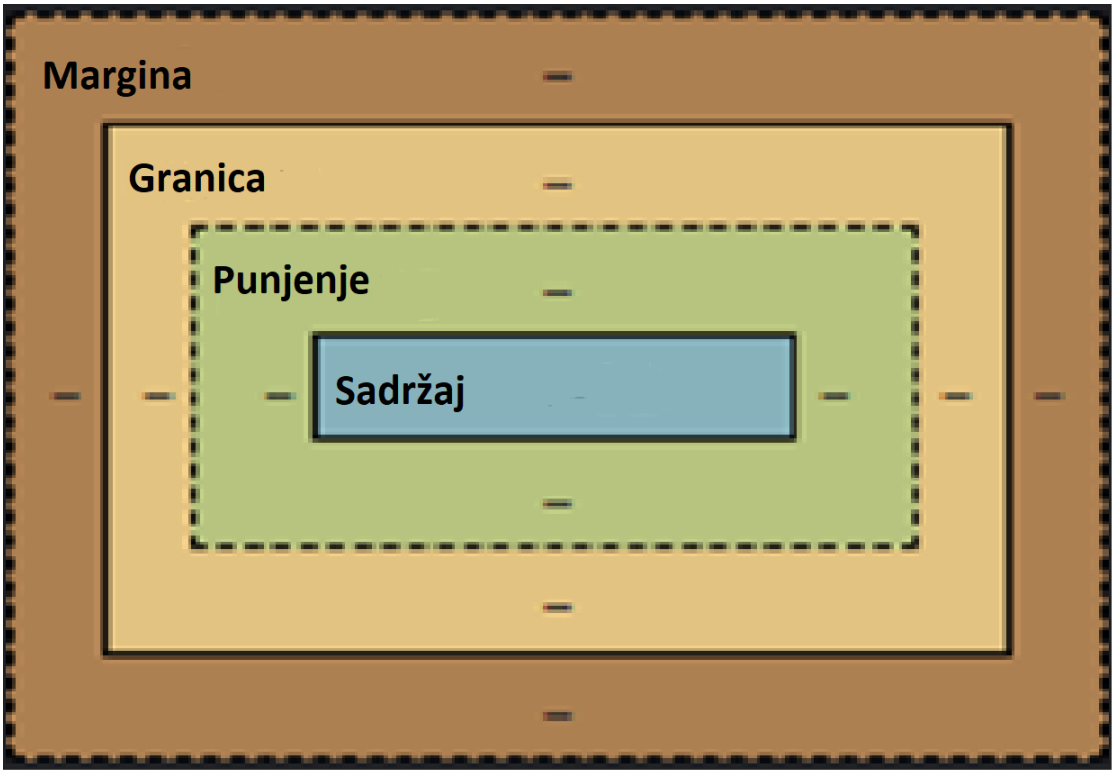
Margine su prazan prostor između elementa i susjednih elemenata. Margine se koriste za stvaranje razmaka između elemenata.

Granice definiraju liniju koja okružuje sadržaj elementa. Granice mogu biti različite debljine, stila i boje. One dijele prostor između margina i punjenja.

Punjenje je prostor između granica elementa i samog sadržaja. Ono pruža dodatni prostor unutar granica elementa. Punjenje se koristi za stvaranje prostora između granica i sadržaja, bez utjecaja na samu veličinu granica.

Sadržaj elementa je tekst, slike ili drugi sadržaj koji je unutar granica i punjenja. Sadržaj elementa zauzima prostor unutar granica i punjenja.

CSS svojstva poput margin, border, padding i width/height koriste se za kontrolu i prilagođavanje dimenzija i izgleda svakog dijela modela kutije. Na primjer, može se postaviti veličina granica, debljinu punjenja ili udaljenost margina koristeći ova svojstva kako bi se oblikovali elemente prema željenoj vizualnoj postavci.



Slika 1. Prikaz modela kutije (vlastita izrada)

2.2.4. Responzivni dizajn

Responzivni dizajn je pristup dizajnu i razvoju web stranica koji ima za cilj pružiti optimalno iskustvo korisnicima na različitim uređajima i zaslonima. Ideja je da se web stranica prilagodi i pravilno prikaže na širokom rasponu uređaja, uključujući računalne monitore, prijenosna računala, tablete i mobilne telefone.

Ključna karakteristika responzivnog dizajna je fleksibilnost. Umjesto da se koriste fiksne dimenzije i rasporedi, responzivni dizajn koristi fleksibilne jedinice kao što su postoci i "em" kako bi se elementi prilagodili veličini zaslona. Ovo omogućava da se elementi različito ponašaju na različitim uređajima.

Prednosti responzivnog dizajna su brojne. Korisnici dobivaju dosljedno i optimalno iskustvo bez obzira na uređaj koji koriste. Responzivni dizajn također olakšava održavanje web stranice jer se promjene mogu primijeniti na jednom mjestu, a ne na više verzija.

2.3. JavaScript

JavaScript je programski jezik koji se koristi na webu. Gotovo sve moderne web stranice koriste JavaScript za poboljšanje interaktivnosti i dinamičnosti korisničkog iskustva, a svi moderni web preglednici na stolnim računalima, tabletima i telefonima uključuju JavaScript interpretere, što ga čini najraširenijim programskim jezikom u povijesti. Tijekom posljednjeg desetljeća, Node.js je omogućio programiranje u JavaScriptu izvan web preglednika, a ogroman uspjeh Node.js-a znači da je JavaScript sada najviše korišteni programski jezik među programerima.[3]

JavaScript je jezik koji je dinamički tipiziran, što znači da nije potrebno unaprijed deklarirati tipove varijabli, jedna varijabla se može koristiti kao više tipova podataka. On podržava više paradigmi programiranja, uključujući proceduralno, objektno orijentirano i funkcionalno programiranje.

U web pregledniku, JavaScript može pristupiti i manipulirati HTML-om i CSS-om, omogućujući interakciju s korisnicima, stvarati dinamičke stranice, te omogućava još mnoge druge značajke koje se ne mogu postići samo s HTML-om i CSS-om.

JavaScript se u web stranici može pisati izravno u HTML dokument, te se u tom slučaju piše unutar oznaka <script> i </script>, ali se također može pisati u zasebnoj datoteci s ekstenzijom .js, te se u tom slučaju može pozvati u HTML dokumentu.

2.3.1. Osnove JavaScript-a

Neki od osnovnih koncepata koji su bitni za razumijevanje i korištenje JavaScript-a su varijable, grananja i petlje.

2.3.1.1. Varijable

Varijable su imenovani segmenti memorije u koje se može pohraniti određena vrijednost. One omogućuju spremanje podataka kao što su brojevi, tekst, nizovi, objekti ili bilo koja druga vrsta podataka. U JavaScriptu, postoje 3 ključne riječi koje se mogu koristiti za deklariranje varijabli: „let“, „var“ i „const“. Varijable se deklariraju te im se dodjeljuje inicijalna vrijednost na sljedeći način:

```
var x = 5;
let y = 6;
const z = 7;
```

Varijable deklarirane ključnom riječi „var“ dostupne su unutar cijele funkcije unutar koje su deklarirane, dok su varijable koje su deklarirane ključnom riječi „let“ dostupne samo unutar

programskog bloka unutar kojeg se nalaze. Također, varijable deklarirane ključnom riječi „var“ se „podižu“, što znači da se mogu koristiti u kodu i prije nego što su deklarirane, te se neće dogoditi greška u programu, dok se varijable deklarirane ključnim riječima „let“ i „const“ ne podižu, te ako se pokušaju koristiti prije deklariranja dogodit će se greška u programu.

Zato što su varijable deklarirane ključnom riječi „var“ dostupne unutar cijele funkcije, prilikom izvođenja sljedećeg koda će se u konzoli ispisati vrijednost varijable, odnosno 10:

```
if (true) {
  var x = 10;
}
console.log(x);
```

Prilikom izvođenja sljedećeg koda, dogodit će se greška, iako je jedina razlika u kodovima način na koji je deklarirana varijabla, zato što je korištena ključna riječ „let“:

```
if (true) {
  let x = 10;
}
console.log(x);
```

Ključna riječ „const“ koristi se kako bi se deklarirale varijable koje imaju konstantu vrijednost, odnosno vrijednost koja im je dodijeljena dok su deklarirane je konstantna te se ne može mijenjati.

2.3.1.2. Grananja

Grananja se u JavaScriptu koriste za kontrolu toka programa na temelju zadovoljenja određenih uvjeta. Omogućuju nam da izvršavamo određeni blok koda samo ako je određeni uvjet ispunjen. U JavaScriptu se najčešće koristi „if“ izraz za grananje, ali također se može koristiti izraz „switch“.

Kada se koristi „if“ izraz za grananje, definira se uvjet koji se promatra. Uvjet je izraz koji se evaluira kao logička vrijednost istina (eng. true) ili laž (eng. false). Ako je uvjet ispunjen (vrijednost je „istina“), izvršit će se blok koda unutar prvog dijela „if“ izraza. Ako uvjet nije ispunjen (vrijednost je „laž“), izvršit će se blok koda unutar „else“ dijela.

Primjer grananja korištenjem izrada „if“:

```
let broj = 10;
if (broj > 5) {
  console.log("Broj je veći od 5.");
} else {
  console.log("Broj je manji ili jednak 5.");
}
```

U navedenom primjeru provjerava se vrijednost varijable broj, te se na temelju vrijednosti te varijable određuje što će se dogoditi. Ako je vrijednost varijable veća od 5, u konzoli će se ispisati „Broj je veći od 5.“, a u suprotnom će se ispisati „Broj je manji ili jednak 5.“.

Druga vrsta grananja je grananje koristeći izraz „switch“. To grananje omogućuje provjeru više mogućih vrijednosti jedne varijable. „Switch“ izraz se koristi kada želimo izvršiti različite blokove koda ovisno o vrijednosti varijable. Prvo se provjerava vrijednost varijable, a zatim se izvršava odgovarajući blok koda unutar „case“ izraza koji odgovara toj vrijednosti. Ako niti jedan „case“ ne odgovara vrijednosti, izvršava se blok koda unutar „default“ izraza. Uz „switch“ grananje također se može koristiti izraz „break“. Izraz „break“ koristi se u grananju kako bi se prekinulo izvršavanje nakon odgovarajućeg bloka koda unutar „case“ izraza.

Primjer grananja korištenjem izraza „switch“:

```
let dan = 2;
switch(dan) {
  case 1:
    console.log("Ponedjeljak");
    break;
  case 2:
    console.log("Utorak");
    break;
  case 3:
    console.log("Srijeda");
    break;
  case 4:
    console.log("Četvrtak");
    break;
  case 5:
    console.log("Petak");
    break;
  case 6:
    console.log("Subota");
```

```
    break;
  case 7:
    console.log("Nedjelja");
    break;
  default:
    console.log("Nepoznat dan");
}
```

U navedenom primjeru provjerava se vrijednost varijable broj te se na temelju vrijednosti u konzolu ispisuje dan u tjednu koji odgovara tom broju.

2.3.1.3. Petlje

Petlje se koriste u JavaScriptu kako bi se ponavljali određeni blokovi koda dok je zadovoljen određeni uvjet. To omogućuje izvršavanje istog koda više puta bez ponavljanja istog koda za svaku pojedinu iteraciju. Postoje nekoliko vrsta petlji u JavaScriptu, uključujući „for“ petlju, „while“ petlju i „do-while“ petlju.

„For“ petlja se koristi kada unaprijed znamo broj ponavljanja. Sastoji se od inicijalizacijskog izraza, uvjetnog izraza i izraza ažuriranja. Primjer te petlje je sljedeći:

```
for (let i = 0; i < 5; i++) {
  console.log(i);
}
```

U tom primjeru, „let i = 0“ je inicijalizacijski izraz, njim se definira koja varijabla će se koristiti te koja joj je početna vrijednost. Zatim, „i < 5“ je uvjetni izraz, u njemu se definira uvjet te se petlja ponavlja dokle god je taj uvjet ispunjen. Napokon, „i++“ je izraz ažuriranja. Njim je definirano kako će se mijenjati vrijednost varijable svakom iteracijom petlje. Ovoj petlji je definirano da je početna vrijednost varijable 0, petlja će se izvršavati dokle god je vrijednost varijable manja od 5, te će se svakom iteracijom varijabla uvećati za 1, što znači da će se petlja izvršiti 5 puta.

„While“ petlja se koristi kada broj ponavljanja nije unaprijed poznat i ovisi o određenom uvjetu. While petlji potrebno je definirati uvjet, te se petlja izvršava dokle god je taj uvjet ispunjen. Primjer te petlje je sljedeći:

```
let i = 0;
while (i < 5) {
  console.log(i);
  i++;
}
```

„Do-while“ petlja je slična „while“ petlji. Jedina razlika je to što „do-while“ petlja provjerava uvjet na kraju petlje, što znači da će se uvjet provjeriti barem jednom čak i ako uvjet nije ispunjen. Primjer te petlje je sljedeći:

```
let i = 0;
do {
  console.log(i);
  i++;
} while (i < 5);
```

2.3.2. Polje

Polje, odnosno niz, je uređena lista elemenata. Svaki element u polju ima svoj indeks koji ga identificira. Indeksi počinju od 0 za prvi element, a zatim se povećavaju redom. Polja se u JavaScript-u stvaraju pomoću uglatih zagrada, te se elementi polja odvajaju zarezima.

Primjer polja u JavaScript-u:

```
let brojevi = [1, 2, 3, 4, 5];
console.log(brojevi[3]);
```

U ovom primjeru kreirano je polje od 5 elemenata te je u konzolu ispisan element polja koji ima indeks 3, što znači da će se u ovom slučaju u konzolu ispisati broj 4, zato što indeksi počinju od 0.

U kodu se također može iskoristiti ključna riječ „length“ nad poljem kako bi se dohvatio broj elemenata koji se nalaze u polju. Sljedeći primjer demonstrira kako se može iskoristiti „for“ petlja kako bi se redom dohvatili svi elementi polja, od prvog do zadnjeg, te ispisali u konzolu:

```
let brojevi = [1, 2, 3, 4, 5];
for(let i = 0; i < brojevi.length; i++){
  console.log(brojevi[i]);
}
```

2.3.3. Objekt

Objekt je složena struktura podataka koja se sastoji od svojstava koja imaju ključeve i vrijednosti. Ključevi su obično nizovi znakova ili identifikatori, a vrijednosti mogu biti bilo koja vrsta podataka, uključujući brojeve, tekst, nizove i objekte.

Objekti se stvaraju pomoću vitičastih zagrada. Svako svojstvo objekta sastoji se od ključa, dvotočke i odgovarajuće vrijednosti.

Primjer objekta:

```
var osoba = {
  ime: "Mladen",
  prezime: "Kajić",
  dob: 22,
};
```

2.3.4. Funkcije

Funkcije su blokovi koda koji se mogu pozivati i izvršavati u JavaScriptu. One omogućuju grupiranje logički povezanih dijelova koda i olakšavaju njihovu ponovnu upotrebu. Funkcije u JavaScriptu mogu primati argumente (ulazne vrijednosti), izvršavati određeni blok koda i vratiti rezultat (izlaznu vrijednost).

Primjer funkcije u JavaScript-u:

```
function zbroj(a, b) {
  return a + b;
}

let rezultat = zbroj(3, 5);
console.log(rezultat);
```

U navedenom primjeru definirana je funkcija naziva „zbroj“, koja prima dvije ulazne vrijednosti, te vraća rezultat zbrajanja te dvije ulazne vrijednosti (pretpostavlja se da su ulazne vrijednosti brojevi). Također je demonstriran poziv te funkcije, gdje se rezultat zbrajanja sprema u varijablu „rezultat“, te se vrijednost te varijable ispiše u konzolu, u ovom slučaju rezultat će biti 8, zato što su proslijeđene vrijednosti 3 i 5.

2.3.5. Klase

Klase su konstrukti u JavaScriptu koji se koriste za stvaranje objekata s određenim svojstvima (varijablama) i metodama (funkcijama) koje opisuju ponašanje tog objekta. Klase predstavljaju koncept objektno orijentiranog programiranja (OOP) u JavaScriptu.

Primjer klase u JavaScript-u:

```
class Osoba {
  constructor(ime, prezime) {
    this.ime = ime;
    this.prezime = prezime;
  }

  pozdrav() {
    console.log("Pozdrav, ja sam " + this.ime + " " + this.prezime);
  }
}

var kreiranaOsoba = new Osoba("Mladen", "Kajić");
kreiranaOsoba.pozdrav();
```

U ovom primjeru, definiramo klasu „Osoba“ s konstruktorom koji prima dvije vrijednosti - ime i prezime. Konstruktorom se postavljaju svojstva objekta "ime" i "prezime" na temelju predanih argumenata. Također, definiramo metodu „pozdrav“ koja ispisuje poruku pozdrava s imenom i prezimenom osobe.

Kada stvorimo objekt iz klase "Osoba" koristeći "new" ključnu riječ, konstruktor se poziva i inicijalizira svojstva objekta. Zatim, možemo pozvati metodu "pozdrav" nad stvorenim objektom i dobiti ispis na konzoli. U navedenom primjeru bi se u konzolu ispisalo „Pozdrav, ja sam Mladen Kajić“.

2.3.6. Node.js

Node.js je izvršno okruženje temeljeno na JavaScriptu koje omogućuje izvršavanje JavaScript koda izvan web preglednika. Umjesto da se koristi isključivo za razvoj korisničke strane (eng. front-end) web aplikacija, Node.js proširuje mogućnosti JavaScripta i omogućuje izvođenje JavaScript koda na serverskoj strani (eng. back-end).

Node.js ima široku primjenu - može se koristiti za izradu jednostavnih alatki za naredbeni redak, ali i kao serverski okvir za web aplikacije. Upotreba tehnologije ovisi o problemu koji treba riješiti, osobnim preferencijama i razini znanja razvojnih programera. Node.js se temelji na Googleovom V8 JavaScript engine-u, što znači da konstantno koristi najnovije jezične značajke. Također, Node.js je neblokirajuće okruženje, što znači da operacije koje se ne izvršavaju izravno u Node.js-u ne blokiraju izvršenje aplikacije, omogućujući joj obradu dodatnih zahtjeva i paralelnih zadataka pomoću asinkronih operacija. [4]

2.3.6.1. NPM

NPM (eng. Node Package Manager), odnosno upravitelj paketa za Node.js, je alat koji dolazi uz Node.js i koristi se za upravljanje JavaScript paketima i njihovim ovisnostima. Ovaj alat je ključan za razvoj Node.js aplikacija jer omogućuje lako dodavanje, upravljanje i ažuriranje biblioteka i modula koji se koriste u projektima.

NPM omogućuje inicijalizaciju Node.js projekta, prilikom čega se kreira datoteka „package.json“. Kako bi se inicijalizirao projekt, u naredbenom retku potrebno je upisati sljedeću naredbu:

```
npm init
```

Datoteka „package.json“ ključna je u svakom Node.js projektu. Ona služi za definiranje i upravljanje metapodacima projekta te praćenje ovisnosti između različitih paketa koje aplikacija koristi.

Primjer datoteke „package.json“:

```
{
  "name": "server",
  "version": "1.0.0",
  "description": "Server za zavrsni rad",
  "main": "server.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1",
    "start": "node server.js"
  },
  "author": "Mladen Kajić",
  "license": "ISC",
  "dependencies": {
    "bcrypt": "^5.1.1",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.1",
    "sqlite3": "^5.1.6"
  }
}
```

Osnovni metapodaci koji su vidljivi u datoteci su naziv projekta koji se nalazi pod atributom „name“, trenutna verzija projekta koja se nalazi pod atributom „version“, opis projekta koji se nalazi pod atributom „description“, autor projekta koji se nalazi pod atributom „author“, te licenca koja se primjenjuje na projekt koja se nalazi pod atributom „license“.

Osim metapodataka, u datoteci je pod atributom „dependencies“ naveden popis paketa koji su potrebno za ispravno izvršavanje aplikacije u produkcijskom okruženju. Navedeni paketi automatski će se instalirati kada se izvrši naredba „npm install“ u naredbenom retku.

Kako bi se dodala nova ovisnost, odnosno instalirao novi paket u projekt, koristi se naredba „npm install“ sa imenom paketa koji se instalira. Npr. paket „express“ može se instalirati u projekt sljedećom naredbom:

```
npm install express
```

Također, u datoteci se nalaze skripte pod atributom „scripts“. Tamo su navedene različite NPM skripte koje se koriste za automatizaciju raznih zadataka u projektu. Npr. skripta „start“ koristi se kako bi se pokrenuo glavni dio aplikacije. Skripta start se u naredbenom retku može pokrenuti sljedećom naredbom:

```
npm start
```

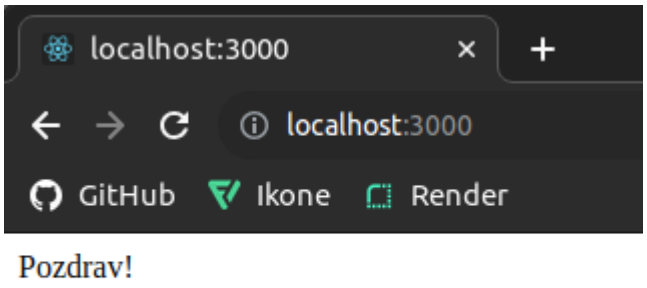
2.3.6.2. Express

Express je popularni razvojni okvir za Node.js koji olakšava razvoj web aplikacija. On pruža niz alata i sredstava za upravljanje putanjama, zahtjevima, odgovorima te omogućuje brzu i jednostavnu izgradnju web aplikacija.

Jednostavan primjer express aplikacije:

```
const express = require('express');
const server = express();

server.get('/', (zahtjev, odgovor) => {
  odgovor.send('Pozdrav!');
});

const port = 3000;
server.listen(port, () => {
  console.log(`Server pokrenut na portu ${port}`);
});
```

U primjeru je kreirana jednostavna Express aplikacija koja sadrži samo osnovnu putanju („/“), te prilikom slanja zahtjeva s GET metodom pošalje odgovor u kojem piše „Pozdrav!“. Također je definirano da je aplikacija pokrenuta na priključku (eng. port) 3000. Nakon pokretanja aplikacije lokalno na računalu, u pregledniku se može pristupiti aplikaciji preko putanje „localhost:3000“, prilikom čega se vidi sljedeći prikaz:



Slika 2. Prikaz jednostavne Express aplikacije (vlastita izrada)

Postupak kojim radi Express aplikacija uvijek slijedi isti obrazac. Express aplikacija prima zahtjev od klijenta. Na temelju odabrane HTTP metode i URL putanje, odabire se odgovarajuća putanja, i izvršavaju se jedna ili više funkcija povratnog poziva (eng. callback functions). Unutar povratne funkcije može se pristupiti objektima zahtjeva (eng. request) i odgovora (eng. response). Objekt zahtjeva je prvi argument povratne funkcije te sadrži korisnički zahtjev. Objekt odgovora je drugi argument predstavlja odgovor koji se šalje korisniku. [4]

Svojstvo	**Opis**
method	Sadrži HTTP metodu korištenu za slanje zahtjeva serveru.
originalUrl	Sadrži originalni URL zahtjeva. Ovo omogućava modifikaciju svojstva "url" koje sadrži iste informacije u svrhe aplikacije.
params	Sadrži vrijednost koja se sastoji od varijabilnih dijelova URL-a.
path	Omogućava pristup URL putanji
protocol	Sadrži protokol zahtjeva, poput HTTP-a ili HTTPS-a.
query	Omogućava pristup upitu (query string) koji je dio URL-a.

Tablica 1. Svojstva objekta zahtjeva [4]

Metoda	Opis
get(field)	Čita određeno zaglavlje (eng. header) odgovora.
set(field[, value])	Postavlja vrijednost određenog zaglavlja odgovora.
cookie(name, value[, options])	Postavlja vrijednost kolačića (eng. cookie).
redirect([status,]path)	Prosljeđuje zahtjev na drugu putanju.
status(code)	Postavlja statusni kod odgovora.
send([body])	Šalje HTTP odgovor.
json([body])	Šalje HTTP odgovor i pretvara predani objekt u JSON format.
end([data][, encoding])	Šalje HTTP odgovor. Ova metoda se uobičajeno koristi ako se ne šalju podaci poput HTML strukture, u suprotnom se koristi metoda send.

Tablica 2. Metode objekta odgovora [4]

3. Sigurnost web aplikacija

Web aplikacije su središnje točke za pristup informacijama i uslugama na internetu. S obzirom na njihovu ulogu u današnje vrijeme, važno je osigurati njihovu sigurnost kako bi se zaštitila povjerljivost, integritet i dostupnost podataka, kao i korisnici od potencijalnih prijetnji. Sigurnost web aplikacija obuhvaća niz mjera, uključujući kodiranje, autentifikaciju i kontrolu pristupa, kako bi se zaštitili sustavi, podaci, informacije i resursi od različitih prijetnji.

S obzirom na visoku vrijednost podataka u današnjem dobu, zaštita od gubitka podataka je od izuzetnog značaja. Različite prijetnje, uključujući prirodne katastrofe, tehničke kvarove, zlonamjerni softver ili neovlašteni pristup, mogu dovesti do gubitka ili oštećenja podataka. Stoga su odgovarajuće sigurnosne mjere neophodne za očuvanje integriteta i dostupnosti tih vrijednih resursa. Sigurnost web aplikacija ima ključnu ulogu u očuvanju privatnosti i povjerljivosti. U vrijeme kada su osobni podaci sve češća meta napada, mjere zaštite pomažu u sprječavanju neovlaštenog pristupa i osiguravanju da samo ovlaštene osobe mogu pristupiti osjetljivim informacijama.

Implementacija sigurnosnih značajki na web aplikacijama obično je brža u odnosu na stolne aplikacije, zato što stolne aplikacije moraju izdati zakrpe koje korisnici zatim moraju preuzeti, dok se kod web aplikacija svaka promjena automatski primijeni svim korisnicima. U nekim slučajevima to vodi do toga da se tokom razvoja web aplikacije identificira određeni sigurnosni propust, ali taj propust nije popravljen sve dok ne postane problem. Bolja praksa je popravljati probleme čim su identificirani. Na taj način je identificirane probleme lakše i brže popraviti u odnosu na kasnije popravljanje. Kako bi se povećala šansa da sigurnosni propusti budu pronađeni i popravljeni na vrijeme, postoje pristupi koji se mogu koristiti tokom razvoja web aplikacije, poput pristupa provali i popravi (eng. Penetrate and patch) ili holistički pristup aplikacijskoj sigurnosti (eng. Holistic Approach to Application Security). Provali i popravi je pristup u kojem se koristi testiranje kako bi se pokušali otkriti sigurnosni propusti u izoliranom okruženju, te se onda ti propusti popravljaju. Kod holističkog pristupa se povećavanje sigurnosti ne shvaća kao jednokratan zadatak, već kao kontinuirani proces koji se provodi tokom cijelog razvoja aplikacije. [5]

3.1. Sigurnosne značajke u razvoju web aplikacija

Postoji mnogo različitih značajki koje se mogu implementirati kako bi se povećala sigurnost razvijene web aplikacije. Naravno, stranica nikada ne može biti u potpunosti sigurna

te uvijek postoji mogućnost da će potencijalna prijetnja pristupiti podacima, ali uvođenjem određenih značajki može se znatno smanjiti rizik da se dogodi neovlašteni pristup podacima.

Neke od najčešće korištenih i najvažnijih sigurnosnih značajki za implementirati su autentikacija, autorizacija, validacija korisničkih unosa, zaštita od SQL ubacivanja (eng. SQL injection), korištenje HTTPS protokola umjesto HTTP protokola, kodiranje podataka, itd.

3.1.1. Autentikacija

Autentikacija je proces koji koristi većina modernih web aplikacija. To je proces provjere identiteta korisnika. Koristi se kako bi se pristup aplikaciji ograničio samo na osobe kojima je ovlašten pristup. Autentikacija se sastoji od 2 koraka, identifikacije i potvrde ispravnosti. U koraku identifikacije se obično od korisnika traži unos podataka (npr. korisničkog imena i lozinke), te se u koraku potvrde ispravnosti provjeri jesu li ti podaci valjani te se na temelju toga utvrdi smije li korisnik dobiti pristup aplikaciji ili ne smije. [5]

3.1.2. Autorizacija

Autorizacija korisnika je proces usko vezan uz autentikaciju. Nakon što se korisnik autenticira, slijedi proces autorizacije. Proces autorizacije odnosi se na kontrolu pristupa nad resursima, odnosno određivanje čemu korisnik može pristupiti, a čemu ne može. Autorizacija se temelji na korisničkim ulogama i ovlastima, npr. korisnik koji je administrator aplikacije može pristupiti svim značajkama, dok korisnik koji nije administrator ima ograničeni pristup. [5]

3.1.3. Validacija korisničkih unosa

Kada se od korisnika traži unos bilo kakvih podataka, iznimno je važno unesene podatke validirati, odnosno provjeriti im ispravnost. Ako se ne provede validacija korisničkih unosa, mogu se dogoditi razni problemi. Jedan potencijalni problem je SQL ubacivanje. To je ranjivost koja omogućava napadaču da nad bazom podataka izvrši zlonamjerni SQL kod.

Kod validacije dobra je praksa definirati što unos treba biti te provjeriti zadovoljava li unos te uvjete, umjesto da definiramo što unos ne smije biti pa provjeravati je li nešto od toga. Na primjer, ako želimo da unos bude broj, bolje je provjeriti je li taj unos broj, umjesto da definiramo popis stvari koje unos ne smije biti te redom provjeravati je li unos nešto od toga. Ti pristupi se nazivaju validacija popisom dopuštenih vrijednosti (eng. Whitelist validation) i validacija popisom zabranjenih vrijednosti (eng. Blacklist validation). [5]

3.1.4. Korištenje HTTPS protokola

HTTPS (eng. Hypertext Transfer Protocol Secure) osigurava enkripciju podataka tijekom prijenosa između korisnika i poslužitelja. To štiti osjetljive informacije poput korisničkih imena i lozinki od mogućih napadača koji bi mogli pokušati presresti komunikaciju, te na taj način omogućuje korisnicima sigurnu komunikaciju s aplikacijom.

3.1.5. Kodiranje podataka

Kodiranje podataka je postupak pretvaranja osjetljivih informacija u oblik koji nije čitljiv bez odgovarajućeg dekodiranja. Enkripcija se koristi za šifriranje podataka prilikom pohrane i prijenosa, čime se osigurava da čak i ako podaci padnu u krive ruke, oni ostaju nečitljivi i neupotrebljivi.

4. Programski okvir React

Programski okviri na strani preglednika (eng. frontend frameworks) su alati koji olakšavaju izradu, organizaciju i upravljanje korisničkim sučeljem web aplikacija. Ovi okviri nude skup unaprijed definiranih komponenti, pravila i obrazaca koji omogućavaju programerima da efikasno razvijaju interaktivne, responzivne i moderne web aplikacije.

Glavni cilj programskih okvira na strani preglednika je pojednostaviti rad s HTML-om, CSS-om i JavaScriptom te omogućiti programerima da se usredotoče na logiku aplikacije umjesto na tehničke detalje manipulacije DOM-om. To dovodi do bržeg razvoja, veće produktivnosti i lakšeg održavanja koda.

Ovi okviri često uključuju koncepte poput virtualnog DOM-a za efikasno ažuriranje korisničkog sučelja, komponenta za modularnost i ponovnu upotrebu koda, i mehanizama za rukovanje događajima i interakcijama.

React je programski okvir otvorenog koda koji je 2013. godine kreirao Jordan Walke, softverski inženjer koji je radio za Facebook. React je opisan kao „V u MVC“. MVC (eng. Model View Controller) označava obrazac koji se koristi za razdvajanje komponenti i odgovornosti unutar aplikacije kako bi se olakšalo upravljanje, održavanje i razvoj. Model predstavlja podatke i logiku, prikaz (eng. View) predstavlja korisničko sučelje koje korisnik vidi, te kontroler upravlja interakcijom između modela i prikaza. Drugim riječima, React aplikacije fokusiraju se na komponente za prikaz podataka, te se ne fokusiraju na model i kontroler. [6]

React se temelji na reaktivnom pristupu, što znači da se korisničko sučelje ažurira automatski kada se promijeni stanje podataka. Ovaj reaktivni pristup omogućava programerima da se fokusiraju na ažuriranje podataka, dok se React brine o tome kako te promjene utječu na prikaz korisničkog sučelja.

Virtualni DOM je ključni koncept koji omogućava Reactu brzo ažuriranje korisničkog sučelja. Kada se stanje komponenata promijeni, React stvara novi virtualni DOM koji je apstraktna reprezentacija stvarnog DOM-a. Zatim se stari i novi virtualni DOM uspoređuju kako bi se pronašle promjene i ažurirale samo promijenjene dijelove stvarnog DOM-a. Ovaj pristup smanjuje potrebu za nepotrebnim ažuriranjem i poboljšava performanse aplikacije. [6]

4.1. Instalacija i kreiranje projekta

Prije kreiranja React projekta, potrebno je instalirati sam okvir. Za instalaciju i korištenje Reacta potrebno je imati instaliran npm. Kako bi se instalirao okvir React, u naredbenom retku potrebno je pokrenuti sljedeću naredbu:

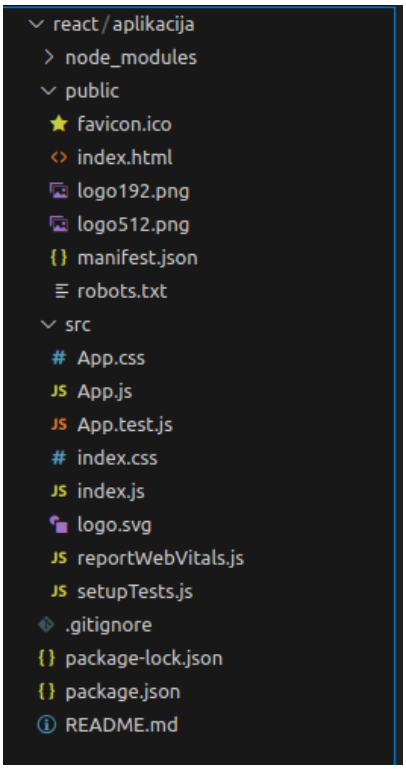
```
npm install react
```

Kada je React instaliran, može se kreirati novi projekt. Za kreiranje projekta koristi se sljedeća naredba:

```
npx create-react-app aplikacija
```

U tom primjeru, „aplikacija“ je naziv aplikacije koja će se kreirati te se taj naziv može promijeniti. Npx (eng. Node Package Execute) je alat koji dolazi s Node.js i omogućuje izvršavanje paketa iz Node.js ekosustava, bez potrebe za instalacijom tih paketa globalno na računalu. U ovom slučaju npx izvršava naredbu „create-react-app“ koja stvara novu React aplikaciju, bez potrebe da se ta naredba instalira globalno na računalu. [7]

Kreirani projekt ima sljedeću početnu strukturu:



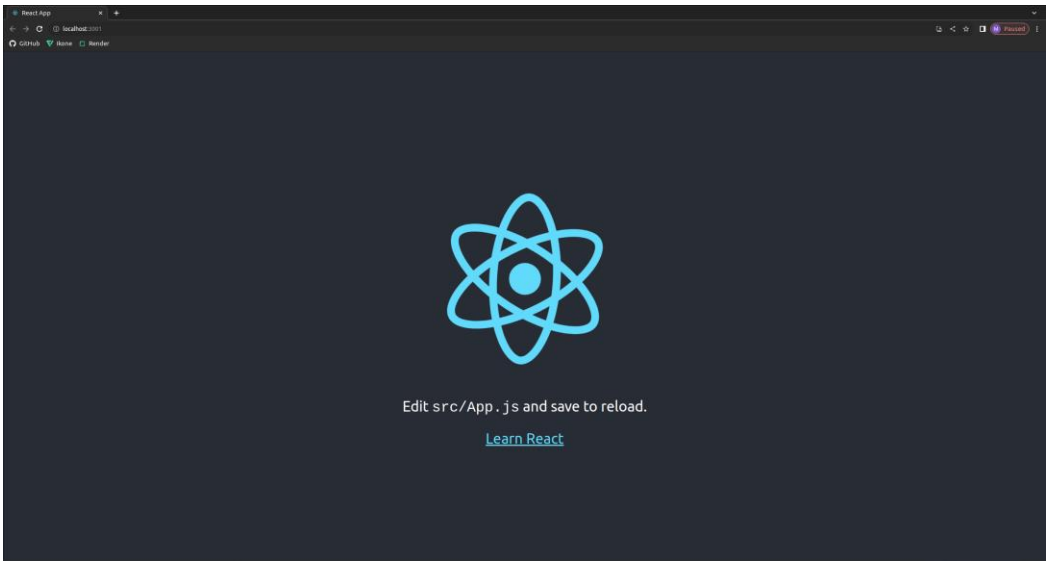
Slika 3. Početna struktura React projekta (vlastita izrada)

Direktorij „node_modules“ sadrži sve vanjske pakete i ovisnosti koje projekt koristi. Kada se instalira novi paket koristeći npm, datoteke tog paketa se preuzimaju i pohranjuju u taj direktorij. U direktoriju „public“ nalaze se statički resursi web aplikacije koji se izravno isporučuju korisniku. U direktoriju „src“ nalazi se sav izvorni kod aplikacije. U tom direktoriju pohranjuju se svi resursi, komponente, stranice i sl. Također se kreira datoteka „package.json“ te datoteka „package-lock.json“ koja sadrži dodatne informacije o ovisnostima koje projekt koristi.

Kako bi se pokrenuo projekt koristi se sljedeća naredba:

```
npm run start
```

Nakon pokretanja projekta, u pregledniku se može pristupiti projektu na definiranoj putanji. Uobičajena putanja je „localhost:3000“.



Slika 4. Prikaz početnog React projekta (vlastita izrada)

Svaka promjena koju želimo vidjeti na stranici mora se napraviti u datoteci „App.js“ koja se nalazi u direktoriju „src“.

4.2. JSX

React koristi JSX (JavaScript XML), specifičnu sintaksu koja omogućava pisanje koda sličnog HTML-u unutar JavaScripta. JSX olakšava definiranje strukture korisničkog sučelja unutar samog JavaScript koda. Iako je kod unutar JSX sličan HTML-u, razlika je što se JSX prilikom kompajliranja koda prevodi u čisti JavaScript kako bi se mogao izvršavati u pregledniku. [6]

Jednostavni primjer JSX sintakse:

```
const primjer = <h1>Pozdrav!</h1>;
```

U ovom primjeru, kreira se JSX izraz koji predstavlja naslov prve razine u HTML-u. Ovaj JSX izraz može se koristiti unutar JavaScript varijable „primjer“.

JSX također pruža mogućnost ugrađivanja JavaScript varijable. To se postiže vitičastim zagradama, te omogućuje dinamičko generiranje sadržaja korisničkog sučelja.

```
const ime = "Mladen";
const primjer = <h1>Moje ime je {ime}</h1>;
```

U navedenom primjeru, kreira se naslov prve razine koji koristi vrijednost varijable „ime“. Generirani naslov bi u ovom slučaju bio „Moje ime je Mladen“, te bi se sa svakom promjenom vrijednosti varijable „ime“ također ažurirao i naslov prve razine.

4.3. Komponente

Komponente u React-u su svi pojedini dijelovi od kojih je korisničko sučelje sastavljeno. One omogućuju ponovnu iskoristivost programskog koda te bolju i lakšu organizaciju aplikacije. Svaka komponenta može imati svoje stanje, svojstva i metode, čime se omogućava izolirano i efikasno upravljanje različitim dijelovima korisničkog sučelja.

Kako bi se kreirala komponenta, potrebno je prvo imati datoteku u kojoj će komponenta biti kreirana. Ekstenzija te datoteke treba biti „.jsx“. Za sljedeći primjer kreirana je datoteka „Komponenta.jsx“, te je u nju napisan sljedeći kod:

```
import React from 'react'

export const Komponenta = () => {
  return (
    <>
        <h1>Komponenta</h1>
        <p>Ovo je tekst komponente</p>
    </>
  )
}
```

U navedenom primjeru kreira se komponenta imena „Komponenta“ koja sadrži naslov prve razine te odjeljak teksta. Za kreiranje komponente koristi se JSX sintaksa. Kada je komponenta kreirana možemo ju postaviti na stranici tako da ju pozovemo u datoteci „App.js“, kao što je vidljivo u sljedećem primjeru:

```
import './App.css';
import { Komponenta } from './Komponenta/Komponenta';

function App() {
  return (
    <Komponenta />
  );
}

export default App;
```

Tada je komponenta vidljiva na stranici, te izgleda ovako:

Slika 5. Prikaz jednostavne komponente (vlastita izrada)

Komponenti se također može promijeniti stil koristeći CSS. Za to je potrebno kreirati novu datoteku. U ovom primjeru ta datoteka se naziva „Komponenta.scss“. U kreiranoj datoteci mogu se definirati stilovi koje komponenta treba poprimiti, te se prilikom uključivanja datoteke s stilom u komponentu promijeni stil kreirane komponente.

U sljedećem primjeru kreirana je klasa koja mijenja boje komponente, margine, širinu i punjenje komponente:

```
.stil-komponente {
    color: white;
    background-color: cadetblue;
    padding: 5px;
    margin: 5px;
    width: 200px;
}
```

Definirani stil se sada može primijeniti na komponenti tako da se uključi datoteka sa stilovima te se komponenti treba dodati kreirana klasa:

```
import React from 'react'
import './Komponenta.scss'

export const Komponenta = () => {
  return (
    <>
    <div className='stil-komponente'>
        <h1>Komponenta</h1>
        <p>Ovo je tekst komponente</p>
    </div>
    </>

  )
}
```

Nakon što su uvedene navedene provjere, kreirana komponenta sada izgleda ovako:

Slika 6. Prikaz komponente s promijenjenim stilom (vlastita izrada)

4.4. Stanje

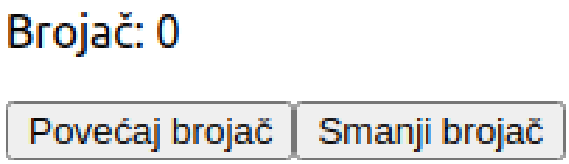
Stanje (eng. state) je lokalna podatkovna varijabla koja pripada komponenti i može se mijenjati tijekom vremena. Stanje je ključni koncept u React-u i koristi se za očuvanje informacija koje se mogu mijenjati i utjecati na prikaz komponente. Svaka komponenta može imati svoje vlastito stanje, što znači da je stanje ograničeno na okvir komponente. To osigurava izolaciju stanja između različitih komponenata. Kada se vrijednost varijable stanja promijeni, React će automatski ažurirati koponentu kako bi korisničko sučelje odražavalo promjene na varijabli stanja. Ovo omogućuje brz odgovor na promjene u podacima ili u korisničkoj interakciji. Stanje se mora inicijalizirati te mu se vrijednost mijenja pomoću metode za promjenu.

Kako bi se omogućilo upravljanje stanjem, koristi se tzv. „kuka“ (eng. hook) koja se zove „useState“. Kuke su uvedene u React verziji 16.8, te one omogućuju korištenje stanja i drugih React-ovih značajki bez potrebe da se komponenta napiše u obliku klase, dok se u prijašnjim verzijama nisu mogle koristiti te značajke ako komponenta nije bila napisana u obliku klase. Kuke čine komponente lakšima za čitanje i održavanje te su zbog toga postale ključni element za razvoj React aplikacija. [8]

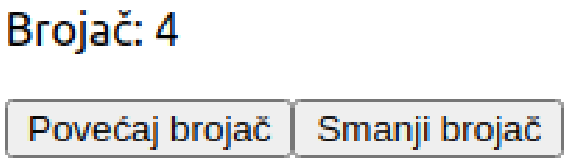
Kuka „useState“ koristi se kako bi se deklarirala varijabla stanja, te funkcija kojom se postavlja vrijednost te varijable stanja. To se može vidjeti u sljedećem primjeru:

```
import React, { useState } from 'react';

export const Brojac = () => {
  const [brojac, setBrojac] = useState(0);
  const povecaj = () => {
    setBrojac(brojac + 1);
  }
  const smanji = () => {
    setBrojac(brojac - 1);
  }
  return (
    <div>
      <p>Brojač: {brojac}</p>
      <button onClick={povecaj}>Povećaj brojač</button>
       <button onClick={smanji}>Smanji brojač</button>
    </div>
  );
}

export default Brojac;
```

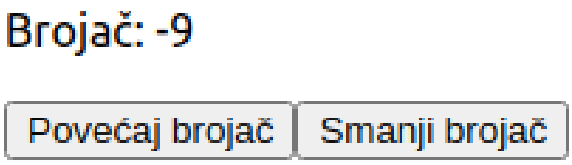
U navedenom primjeru, kreira se varijabla stanja imena „brojac“ te se u nju postavi početna vrijednost 0. Također se definira funkcija „setBrojac“, koja služi kako bi se mogla postaviti vrijednost varijable „brojac“. Na komponenti se zatim u odjeljku prikaže vrijednost tog brojača te se kreiraju 2 gumba, koja kada se pritisnu uvećaju ili umanje vrijednost varijable „brojač“. Kada se vrijednost varijable mijenja, automatski se ažurira odjeljak te je na stranici vidljiva promijenjena vrijednost varijable „brojac“.



Slika 7. Početna vrijednost varijable stanja (vlastita izrada)



Slika 8. Varijabla stanja nakon uvećavanja vrijednosti (vlastita izrada)



Slika 9. Varijabla stanja nakon smanjenja vrijednosti (vlastita izrada)

4.5. Svojstva

Svojstva (eng. props) su parametri koje komponenta prima od svoje nadređene komponente i ne mogu se mijenjati unutar same komponente. Svojstva omogućavaju prijenos podataka od roditeljske komponente prema djeci. Svaka komponenta može primati svojstva i koristiti ih za promjenu svog ponašanja ili prikaza. Svojstva koja roditelj proslijedi komponenti djetetu se ne mogu mijenjati unutar komponente djeteta, mogu se mijenjati samo unutar roditeljske komponente koja poziva dijete. Kada se svojstvo promijeni unutar roditeljske komponente, komponenta kojoj je to svojstvo proslijeđeno također će automatski ažurirati vrijednost tog svojstva te će se ta promjena odraziti na komponenti kojoj je proslijeđeno svojstvo.

Svojstvima se unutar komponente kojoj će biti proslijeđena može definirati koji tip podataka pojedino svojstvo treba imati te je li svako svojstvo obavezno definirati prilikom kreiranja komponente ili nije. Za to se koristi biblioteka „prop-types“.

Tip podataka	**Opis**
string	Očekuje vrijednost koja je string, odnosno tekst
number	Očekuje broj
boolean	Očekuje vrijednost tipa boolean, odnosno istina ili laž
array	Očekuje niz vrijednosti
object	Očekuje objekt kao vrijednost
func	Očekuje funkciju kao vrijednost
node	Očekuje bilo koju React komponentu ili JSX izraz
element	Očekuje React element (npr. <Komponenta/>)
isRequired	Koristi se kako bi se označilo je li to svojstvo obavezno
oneOf	Provjerava je li svojstvo jedna od vrijednosti navedenih u nizu
oneOfType	Provjerava je li tip podataka svojstva jedan od navedenih u nizu
arrayOf	Provjerava je li svojstvo niz koji sadrži samo elemente koji su određenog tipa podataka
objectOf	Provjerava je li svojstvo objekt čiji su svi atributi određenog tipa podataka

shape	Provjerava je li svojstvo objekt koje ima definiranu strukturu atributa
exact	Provjerava je li svojstvo objekt koje ima točno određene atribute, ne dopušta dodatne atribute
instanceOf	Provjerava je li svojstvo instanca određene klase ili konstruktora
enum	Koristi se za definiranje skupa dopuštenih vrijednosti koje svojstvo može poprimiti
customValidator	Omogućuje definiranje prilagođenih pravila koje svojstvo treba zadovoljiti

Tablica 3. Tipovi podataka koje svojstva mogu poprimiti [7]

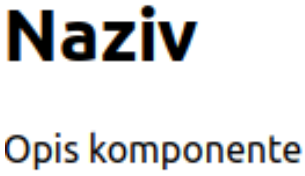
U sljedećem primjeru može se vidjeti kako se unutar komponente definira koja svojstva joj treba proslijediti te kako se koriste:

```
import React from 'react'
import PropTypes from 'prop-types'

export const Komponenta = ({ naziv, opis }) => {
  return (
    <div>
      <h1>{naziv}</h1>
      <p>{opis}</p>
    </div>
  );
}

Komponenta.propTypes = {
    naziv: PropTypes.string.isRequired,
    opis: PropTypes.string.isRequired
}

export default Komponenta
```

Kako je definirano da su svojstva „naziv“ i „opis“ obavezna, prilikom poziva komponente potrebno im je definirati vrijednost. Ukoliko im se ne definira vrijednost, komponenta se neće prikazati te će se u konzoli preglednika ispisati poruka greške. Kreirana komponenta poziva se na sljedeći način:

```
function App() {
  return (
    <Komponenta naziv='Naziv' opis='Opis komponente' />
  );
}
```

Komponenta s tako definiranim svojstvima prikazana je na stranici na sljedeći način:



Slika 10. Prikaz komponente s definiranim svojstvima (vlastita izrada)

Ako se u roditeljskoj komponenti promijeni vrijednost jednog od tih svojstava, komponenta će se automatski ažurirati. To se može vidjeti na sljedećem primjeru, gdje je promijenjen opis komponente te je na slici vidljivo kako se tekst prikazan na komponenti promijenio:

```
<Komponenta naziv='Naziv' opis='Novi opis komponente' />
```

Slika 11. Prikaz komponente s ažuriranim svojstvima (vlastita izrada)

4.6. Putanje

Putanja (eng. route) je način na koji se upravlja prikazom različitih dijelova korisničkog sučelja ovisno o URL-u. Putanje se u jednostraničnim aplikacijama koriste kako bi se korisnicima omogućilo da navigiraju kroz različite dijelove aplikacije bez potrebe za ponovnim učitavanjem stranica. React nema ugrađenu mogućnost kreiranja putanji, već je potrebno instalirati zasebnu biblioteku.

Često korištena biblioteka koja omogućuje korištenje putanji u React-u je „react-router-dom“. Ta biblioteka se instalira u projekt sljedećom naredbom:

```
npm install react-router-dom
```

Nakon instalacije biblioteke, u datoteci „index.js“ potrebno je uključiti korištenje putanja. To se čini na sljedeći način:

```
import React from 'react';
import ReactDOM from 'react-dom/client';
import './index.css';
import App from './App';
import { BrowserRouter } from 'react-router-dom';

const root = ReactDOM.createRoot(document.getElementById('root'));
root.render(
  <React.StrictMode>
    <BrowserRouter>
      <App />
    </BrowserRouter>
  </React.StrictMode>
);
```

Komponentu „App“ potrebno je postaviti kao dijete komponente „BrowserRouter“. To je potrebno učinit kako bi komponenta „BrowserRouter“ postavila mogućnost korištenja putanji unutar aplikacije, te se one mogu koristiti nakon te promjene.

Putanje se koriste unutar aplikacije na sljedeći način:

```
import { Route, Routes } from "react-router-dom";
import { TrajniZadaci } from "./Stranice/TrajniZadaci/TrajniZadaci";
import Projekti from "./Stranice/Projekti/Projekti";
import Biljeske from "./Stranice/Biljeske/Biljeske";
function App() {
  return (
    <>
        <Routes>
```

```
          <Route
            path="/trajni-zadaci"
            element={<TrajniZadaci />}
          />
          <Route
            path="/projekti"
            element={<Projekti/>}
          />
          <Route
            path="/biljeske"
            element={<Biljeske />}
          />
        </Routes>
    </>
  );
}
export default App;
```

U navedenom primjeru kreirane su 3 putanje. Kada korisnik u pregledniku otvori neku od navedenih putanja, biti će mu prikazana stranica koja je povezana s tom putanjom. Na primjer, ako je react aplikacija poslužena lokalno na portu 3000, te ako korisnik otvori putanju „localhost:3000/projekti“, u pregledniku će mu biti vidljiva stranica koja je definirana kao „Projekti“.

Kako bi putanje imale svrhu unutar aplikacije, potrebno je kreirati poveznice koje će korisniku promijeniti putanju te prikazati potrebnu stranicu. To se postiže na sljedeći način:

```
import React from 'react'
import './Navigacija.scss'
import { useNavigate } from "react-router-dom";

const Navigacija = () => {
  const navigacija = useNavigate();
  return (
    <div>
      <a onClick={() => {navigacija(`/trajni-zadaci`)}}>Trajni Zadaci</a>
      <a onClick={() => {navigacija(`/projekti`)}}>Projekti</a>
      <a onClick={() => {navigacija(`/biljeske`)}}>Bilješke</a>
    </div>
  )
}
export default Navigacija
```

U navedenom primjeru kreiraju se 3 poveznice. Kada korisnik pritisne neku od te 3 poveznice, trenutna putanja se promijeni na jednu od ranije navedenih putanja te se korisniku prikaže stranica povezana s tom putanjom. U React-u se ne koristi „href“ atribut elementa „a“, zato što se na taj način web stranica osvježi, što nije poželjno zato što se React-om kreiraju jednostranične aplikacije.

Kako bi se povećala sigurnost putanja, moguće je implementirati zaštićene putanje, koje će korisniku zabraniti da im izravno pristupi ako određeni uvjet nije ispunjen. Primjer situacije gdje je takva putanja korisna je ako aplikacija omogućuje prijavu korisnika u sustav, te ako želimo zabraniti pristup određenim putanjama ako korisnik nije prijavljen u sustav. Takva putanja može se kreirati na sljedeći način:

```
import React from 'react'
import PropTypes from 'prop-types'
import { Navigate } from 'react-router-dom';

const ZasticenaPutanja = ({ prijavljen, children }) => {
    if (!prijavljen) {
        return <Navigate to='/' replace />;
    } else {
        return children;
    }
}

ZasticenaPutanja.propTypes = {
   prijavljen: PropTypes.bool.isRequired,
   children: PropTypes.node
}

export default ZasticenaPutanja
```

Kreiranoj komponenti potrebno je proslijediti svojstvo koje sadrži informaciju o tome je li korisnik prijavljen ili nije. Ukoliko korisnik nije prijavljen, odnosno proslijeđeni atribut ima vrijednost laž, korisnik ne dobije pristup sadržaju koji je povezan s tom putanjom, već se korisnika preusmjeri na korijensku putanju. Takva komponenta koristi se na sljedeći način:

```
<Route
   path="/trajni-zadaci"
   element={
     <ZasticenaPutanja prijavljen={prijavljen}>
        <TrajniZadaci />
     </ZasticenaPutanja>
   }
/>
```

4.7. Usporedba programskih okvira

4.7.1. React

React je okvir koji je prvenstveno usmjeren na stvaranje komponenata koristeći JSX, koji omogućava kombiniranje HTML-a i JavaScripta. On razdvaja korisničko sučelje u komponente te omogućuje razvoj jednostraničnih web aplikacija. [6]

Prednosti korištenja React-a su što potiče razvoj web aplikacija korištenjem komponenti, čime se postiže ponovna iskoristivost napisanog koda te lakša organizacija koda, ima veliku zajednicu i podršku što olakšava razvoj te omogućuje lakši pronalazak resursa i dokumenata za rješavanje problema te lakše učenje, omogućuje razvoj jednostraničnih web aplikacija što poboljšava performanse aplikacija, te je fleksibilan, može se integrirati s drugim okvirima i bibliotekama. [6]

Neki od nedostataka korištenja React-a su što nema puno ugrađenih značajki što znači da je potrebno koristiti dodatne biblioteke, te što nema jasnu strukturu kojom se organizira projekt, što može otežati razvoj ako se projekt loše organizira. [6]

4.7.2. Angular

Angular je programski okvir koji također omogućuje razvoj web aplikacija koje koriste komponente. On koristi TypeScript kao glavni jezik, koji uvodi tipiziranje podataka u JavaScript. [9]

Prednosti korištenja Angular-a su što je opširniji u odnosu na React, odnosno mnoge značajke su ugrađene te se mogu koristiti bez potrebe za korištenjem dodatne biblioteke, koristi TypeScript kao osnovni jezik što uvodi određene prednosti u odnosu na JavaScript (npr. greške prilikom kompajliranja), te ima unaprijed definirane uzorke dizajna i smjernice koje olakšavaju strukturiranje aplikacije. [9]

Neki od nedostataka korištenja Angulara su što može biti složen za manje projekte, teže ga je naučiti u odnosu na neke druge programske okvire te veličina aplikacija može biti veća zbog ugrađenih alata i funkcionalnosti. [9]

4.7.3. Vue.js

Vue.js je programski okvir koji je poznat po svojoj jednostavnoj sintaksi i postupnom usvajanju, što znači da se može postupno uvoditi u postojeći projekt, umjesto naglog preseljenja na potpunu arhitekturu koju Vue.js nudi. [10]

Prednosti korištenja Vue.js-a su što ima nisku krivulju učenja i olakšava početak zbog svoje jednostavne sintakse i postupnog usvajanja, ima veliku brzinu izvođenja, fleksibilan je te postoje integrirane biblioteke za određene važne značajke, poput upravljanja stanjem i putanji.

Nedostatak korištenja Vue.js-a je što ima malu zajednicu što znači da je količina dostupnih resursa i biblioteka manja u odnosu na ranije navedene programske okvire.

5. Razvoj web aplikacije

5.1. Opis web aplikacije

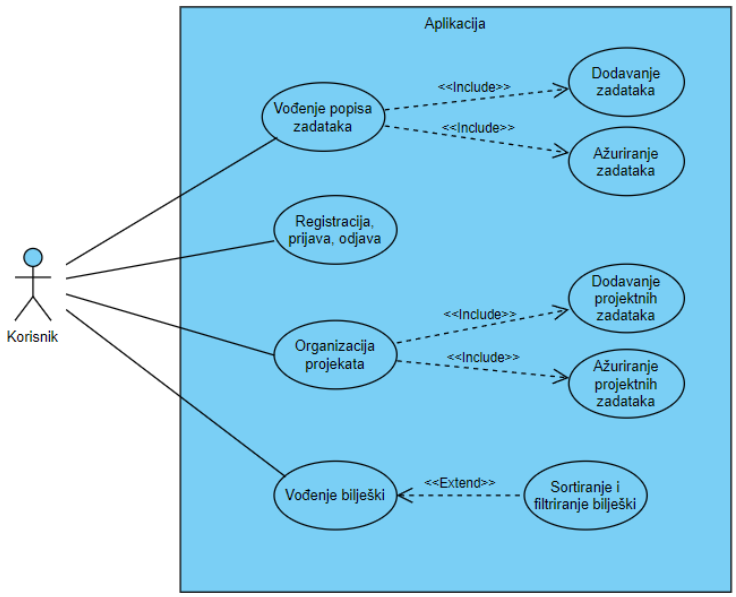
Web aplikacija izrađena za ovaj završni rad sastoji se od klijentskog i poslužiteljskog dijela. Za poslužiteljski dio koristi se Node.js koji pohranjuje podatke u SQLite bazu podataka.

Izrađena aplikacija služi za efikasno upravljanje svakodnevnim obavezama te organizaciju radnih projekata. Aplikacija ima funkcionalnosti vođenja popisa zadataka, organizaciju projekata, vođenje bilješki, te također ima funkcionalnosti prijave u sustav, registracije te odjave. Aplikacija ima responzivni dizajn kako bi bila jednostavna i intuitivna za korištenje na uređajima različitih veličina.

Aplikacija omogućuje korisnicima vođenje popisa zadataka kako bi organizirali svoje obaveze. Za svaki zadatak korisnici mogu unijeti naziv te opis zadatka. Korisnici svaki zadatak mogu označiti kao izvršen ili ga izbrisati. Postoje 3 vrste zadataka: trajni, tjedni i dnevni. Trajni zadaci se izvršavaju samo jednom te se ne ponavljaju, dok se tjedni ponavljaju svaki tjedan te dnevni svaki dan. Za tjedne i dnevne zadatke može se provjeriti koji zadaci su u prošlosti bili označeni kao završeni.

Aplikacija također korisniku omogućuje opciju kreiranja projekata, te se za projekte mogu kreirati zadaci koji se trebaju izvršiti. Svaki zadatak ima svoj naslov, opis, datum do kada se treba završiti te stanje izvršenosti. Postoji 5 stanja izvršenosti: „Nije započeto“, „Izvršava se“, „Završeno“, „Odgođeno“ te „Odbačeno“. Svaki zadatak spada u jedno od tih 5 stanja, te se može mijenjati stanje zadatka kako bi korisnik lakše organizirao rad na projektu.

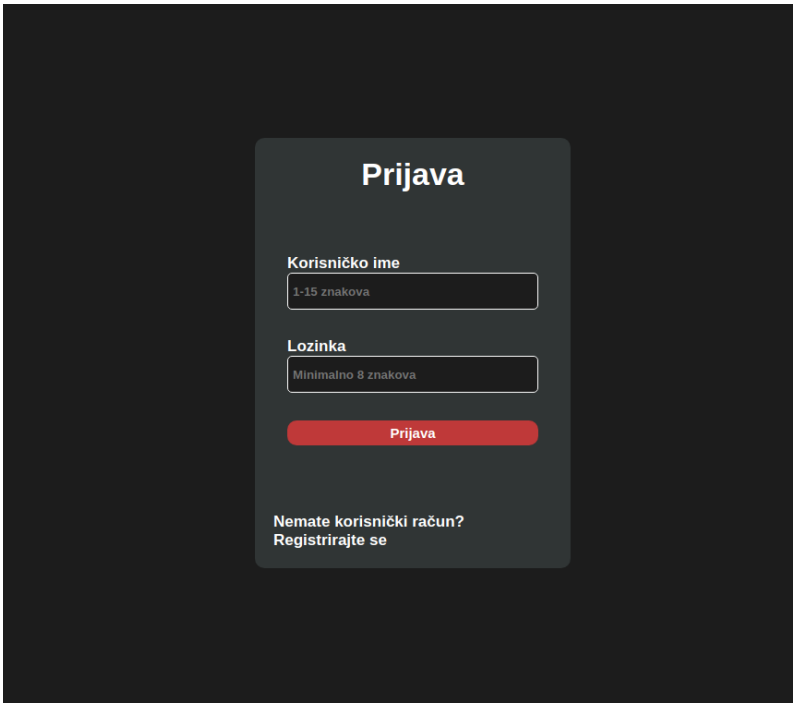
Korisnik ima opciju stvaranja bilješki gdje može zabilježiti željene informacije. Svaka bilješka sadrži naslov i sadržaj. Korisnik također svakoj bilješki može dodati kategorije kojima ta bilješka spada. Dodavanje kategorija bilješkama korisniku omogućuje lakše pronalaženje željenih bilješki. Svaka bilješka se može dodati u favorite, te su bilješke koje su favoriti korisniku vidljive na navigaciji stranice kako bi im korisnik lako mogao pristupiti.



Slika 12. Dijagram slučajeva korištenja aplikacije (vlastita izrada)

5.2. Prijava, registracija, odjava

Prva stranica koju korisnik vidi prilikom pokretanja aplikacije je stranica za prijavu u sustav. Korisniku se prikažu 2 polja za unos te je potrebno upisati korisničko ime i lozinku za prijavu. Izgled stranice za prijavu vidljiv je na sljedećoj slici:



Slika 13. Izgled stranice za prijavu korisnika (vlastita izrada)

Kako bi se kreirala polja za unos kreirana je komponenta „TekstualnoPolje“:

```
const TekstualnoPolje = ({ naziv, validacija, neispravanUnos, promjena, lozinka, poruka }) => {
  return (
    <input className={`tekstualno-polje ${neispravanUnos ? 'neispravan-unos' : ''}`}
        type={lozinka ? 'password' : 'text'}
        id={naziv}
        name={naziv}
        placeholder={poruka}
        onChange={(e) => promjena(e.target.value)}
        onBlur={validacija}
        required />
  )
}
```

Toj komponenti proslijeđeno je 6 svojstava: naziv, validacija, neispravanUnos, promjena, lozinka, poruka. Svojstvo naziv je ime tekstualnog polja. Svojstvo validacija je funkcija povratnog poziva koja se koristi kako bi se provjerila ispravnost podataka koji su upisani u tekstualno polje. Svojstvo neispravanUnos služi kako bi definirali komponenti da unesena vrijednost nije ispravna te da se treba promijeniti dizajn tekstualnog polja kako bi prikazalo da je unos neispravan. To svojstvo se u roditeljskoj komponenti određuje na temelju izvršenja funkcije validacija. Svojstvo promjena je također funkcija povratnog poziva, te ona služi kako bi se u roditeljskoj komponenti promijenila vrijednost koja se unosi u tekstualnom polju. Svojstvo lozinka ima tip boolean, te se na temelju toga određuje treba li tip polja biti „password“ ili „text“. Napokon, svojstvo poruka se koristi za atribut „placeholder“, odnosno definira se što treba pisati na tekstualnom polju kada korisnik nije ništa unesao.

Na stranici za prijavu sljedeći kod se koristi kako bi se kreiralo polje za unos

```
<div className='registracija-polje'>
  <label htmlFor="korime">Korisničko ime</label>
  <TekstualnoPolje
        naziv="korime"
        validacija={validacijaKorime}
        neispravanUnos={korimeGreska.length !== 0}
        promjena={(korime) => setKorime(korime)}
        poruka="1-15 znakova"
    />
  {korimeGreska && <div className="poruka-greske">{korimeGreska}</div>}
  {!korimeGreska && <div className="pozicija-greske"></div>}
</div>
```

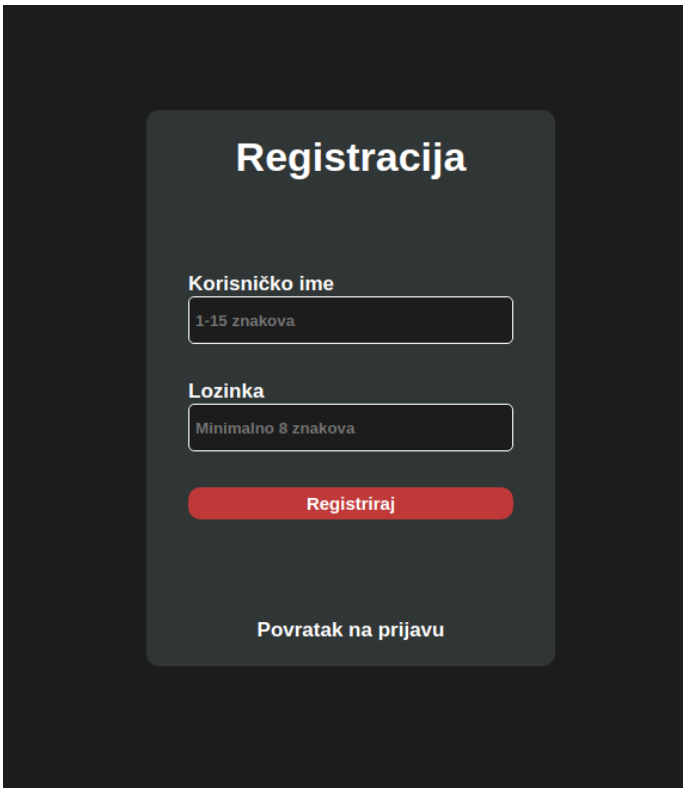
U navedenom kodu kreira se komponenta za tekstualno polje u kojoj se za validaciju koristi funkcija „validacijaKorime“. U toj funkciji se provjeri upisana vrijednost, te se na temelju te vrijednosti definira vrijednost varijable stanja „korimeGreska“, te ako ta varijabla stanja ima vrijednost, odnosno korisničko ime koje je upisano je neispravno, ispod tekstualnog polja se ispiše greška te se korisnik ne može prijaviti dok ne ispravi grešku s korisničkim imenom.

Na jednak način je kreirano i polje za unos lozinke. Kada korisnik pritisne gumb za prijavu, pozove se sljedeća funkcija koja šalje HTTP zahtjev klijentskoj strani aplikacije:

```
export const prijaviKorisnika = async (korime, lozinka) => {
  const odgovor = await fetch("/api/korisnici/prijava", {
    method: "POST",
    headers: {
      "Content-Type": "application/json",
    },
    body: JSON.stringify({
      korime,
      lozinka,
    }),
  });

  if (odgovor.ok) {
    const podaci = await odgovor.json(odgovor.body);
    localStorage.setItem("korisnik", podaci.korisnikId);
    localStorage.setItem("jwt", podaci.token);
    return true;
  } else {
    return false;
  }
};
```

U navedenoj funkciji se pošalje zahtjev klijentskoj strani aplikacije, te ukoliko klijentska strana pošalje odgovor u kojem se, ako su podaci ispravni, nalazi JWT token za korisnika te jedinstveni identifikator korisnika. Ti podaci se zatim spreme u lokalno spremište podataka, te se ti podaci šalju u svakom ostalom HTTP zahtjevu prema klijentskoj strani aplikacije.

Na stranici za prijavu nalazi se poveznica prema stranici za registraciju. Stranica za registraciju izgleda i funkcionira skoro identično stranici za prijavu. Dizajn stranice za registraciju je sljedeći:



Slika 14. Izgled stranice za registraciju korisnika (vlastita izrada)

Na klijentskoj strani registracija korisnika funkcionira jednako kao i prijava korisnika. Jedina bitna razlika je putanja na poslužiteljskoj strani koja se poziva prilikom registracije. Prilikom registracije klijentska strana također vraća JWT i identifikator korisnika koji se spremaju u lokalno spremište podataka, te se na poslužiteljskoj strani podaci pohrane.

5.3. Trajni zadaci

Trajni zadaci su zadaci koji se trebaju ispuniti jednom te se ne ponavljaju. Zadaci su na stranici podijeljeni na zadatke koji su završeni te na one koji nisu završeni. Svaki zadatak se može označiti kao završen te se može obrisati. Također postoji opcija kreiranja novog zadatka, prilikom čega se od korisnika traži unos naziva i opisa zadatka, te se kreirani zadatak doda u popis nezavršenih zadataka.

Dizajn stranice za trajne zadatke je sljedeći:

Slika 15. Izgled stranice za trajne zadatke (vlastita izrada)

Prvo što se učini prilikom učitavanja stranice je dohvat svih zadataka, te se dohvaćeni zadaci filtriraju u dva polja, jedno polje je za završene zadatke te drugo za nezavršene. Dohvat se čini sljedećom funkcijom:

```
const ucitajZadatke = async () => {
  const zadaci = await dohvatiTrajneZadatke();
  const zavrseniZadaci = zadaci.filter((zadatak) => zadatak.zavrsen);
  const nezavrseniZadaci = zadaci.filter((zadatak) => !zadatak.zavrsen);

  setZavrseniZadaci(zavrseniZadaci);
  setNezavrseniZadaci(nezavrseniZadaci);
}
```

Funkcija „dohvatiTrajneZadatke“ šalje HTTP zahtjev prema poslužiteljskoj strani te se u odgovoru dobiju svi zadaci povezani s trenutno prijavljenim korisnikom. Nakon dohvata, za svaki zadatak se kreira nova instanca komponente „Stavka“. Komponenta „Stavka“ ima sljedeći kod:

```
const Stavka = ({ naslov, opis, zavrsen, promijeniStanje, brisanje, klikPoziv,
bezPotvrdnogOkvira, bezOpisa, bezBrisanja}) => {
  const klasaStavke = `stavka ${klikPoziv ? 'promjena-pokazivaca' : ''}`;
  return (
    <div className={klasaStavke}>
      {!bezPotvrdnogOkvira &&
        <input type="checkbox" checked={zavrsen} onChange={promijeniStanje}/>
      }
```

```
      <div className="stavka-detalji" onClick={klikPoziv}>
        <span className="stavka-naslov">{naslov}</span>
        {!bezOpisa &&
          <span className="stavka-opis">{opis}</span>
        }
      </div>
      {!bezBrisanja &&
        <img src={ikonaBrisanje} alt='Ikona za brisanje' className='ikona-
brisanje' onClick={brisanje}/>
      }
    </div>
  );
};
```

Ta komponenta prima više svojstva na temelju kojih se definira koji dijelovi komponente trebaju biti prikazani a koji ne. Svojstva koja to omogućuju su „bezPotvrdnogOkvira“, „bezOpisa“, „bezBrisanja“. Ta svojstva omogućuju da se komponenta prilagodi potrebama te omogućuju ponovnu iskoristivost te komponente za više različitih situacija.

Na sljedeći način se kreiraju instance komponente „Stavka“ za svaki nezavršeni zadatak:

```
{nezavrseniZadaci.map((zadatak) => (
      <Stavka
        key={zadatak.id}
        naslov={zadatak.naslov}
        opis={zadatak.opis}
        zavrsen={zadatak.zavrsen}
        promijeniStanje={() => promijeniStanje(zadatak.id, !zadatak.zavrsen)}
        brisanje={() => {
          setBrisanje(true);
          setZadatakZaBrisanje(zadatak.id);
        }}
      />
    ))}
```

U navedenom kodu se iterira kroz polje „nezavrseniZadaci“ te se za svaku stavku kreira nova instanca komponente kojoj se kao svojstva proslijede vrijednosti te stavke. Na identičan način se prikazuju završeni zadaci.

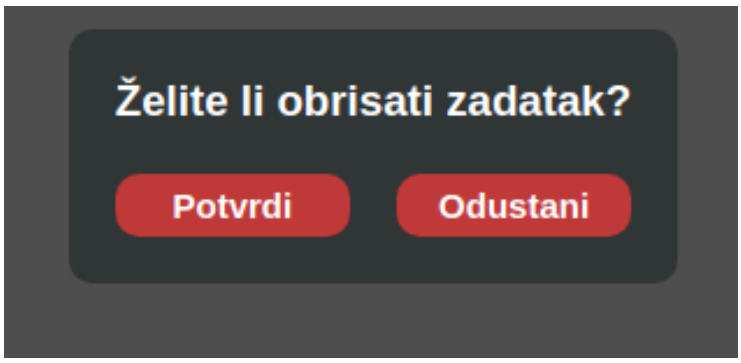
Pritiskom na potvrdni okvir komponente poziva se funkcija „promijeniStanje“ koja ima sljedeći kod:

```
const promijeniStanje = async (id, novoStanje) => {
  await promijeniStanjeTrajnogZadatka(id, novoStanje);
  const azuriraniZadaci = zavrseniZadaci.concat(nezavrseniZadaci)
    .map((zadatak) => {
```

```
      if (zadatak.id === id) {
        return { ...zadatak, zavrsen: novoStanje };
      }
      return zadatak;
    });
  const azuriraniZavrseniZadaci = azuriraniZadaci.filter((zadatak) =>
zadatak.zavrsen);
  const azuriraniNezavrseniZadaci = azuriraniZadaci.filter((zadatak) =>
!zadatak.zavrsen);

  setZavrseniZadaci(azuriraniZavrseniZadaci);
  setNezavrseniZadaci(azuriraniNezavrseniZadaci);
};
```

U toj funkciji se pozove funkcija koja šalje HTTP zahtjev poslužitelju za ažuriranje zadatka, te također ažurira varijable stanja u kojima su pohranjena polja završenih i nezavršenih zadaka, kako bi se zadatak kojem se mijenja stanje obrisao iz jednog polja te dodao u drugo polje, prilikom čega se automatski ažuriraju zadaci koji su prikazani korisniku.

Ukoliko korisnik pritisne gumb za brisanje zadatka, pozove se promjena varijable stanja „brisanje“ te se ona postavi na istinitu vrijednost, te se promijeni vrijednost varijable stanja „zadatakZaBrisanje“. Kada se to učini korisniku se prikaže potvrdni prozor, gdje korisnik treba potvrditi kako želi obrisati zadatak, kao što je vidljivo na sljedećoj slici:



Slika 16. Potvrdni prozor prikazan prilikom brisanja zadatka (vlastita izrada)

Ukoliko korisnik potvrdi kako želi obrisati zadatak, pozove se funkcija koja obriše odabrani zadatak iz varijabli stanja te također pozove funkciju koja šalje HTTP zahtjev za brisanje zadatka sa poslužitelja.

```
const obrisiZadatak = async () => {
    const id = zadatakZaBrisanje;
    await obrisiTrajniZadatak(id);
    const indeksZadatka = zavrseniZadaci.findIndex((zadatak) => zadatak.id === id);
    const zadatakZavrsen = indeksZadatka !== -1;
    if (zadatakZavrsen) {
```

```
      const azuriraniZavrseniZadaci = zavrseniZadaci.filter((zadatak) => zadatak.id
!== id);
      setZavrseniZadaci(azuriraniZavrseniZadaci);
    } else {
      const azuriraniNezavrseniZadaci = nezavrseniZadaci.filter((zadatak) =>
zadatak.id !== id);
      setNezavrseniZadaci(azuriraniNezavrseniZadaci);
    }
    setBrisanje(false);
  };
```

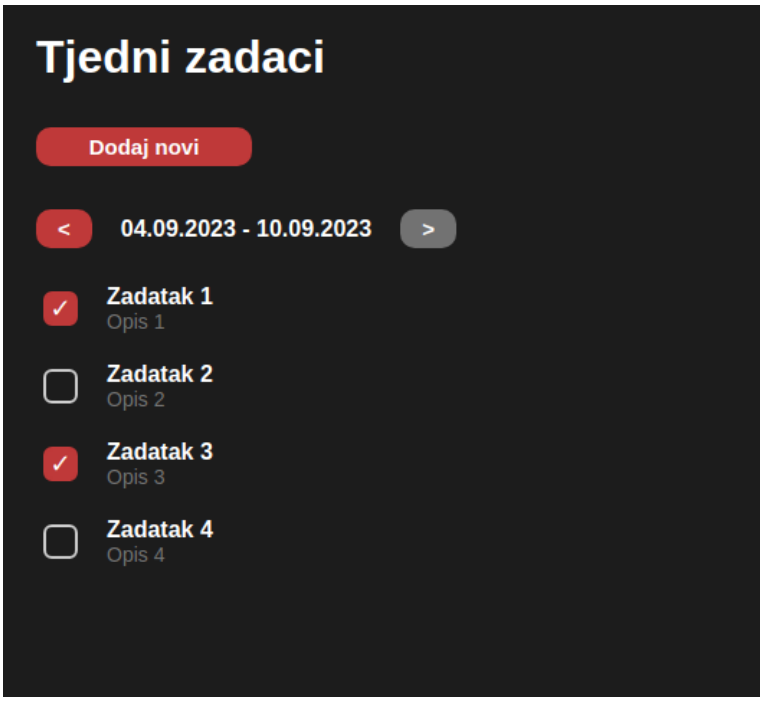
Pritiskom na gumb za kreiranje zadatka, korisniku se prikaže prozor gdje treba upisati naziv i opis zadatka. Taj prozor je vidljiv na sljedećoj slici:

Slika 17. Prozor za kreiranje novog zadatka (vlastita izrada)

Ispravnost unesenih podataka se provjerava na jednak način kao i prilikom prijave, te ukoliko su podaci ispravni pozove se funkcija koja doda zadatak s upisanim vrijednostima u varijablu stanja te pošalje HTTP zahtjev za kreiranje novog zadatka na poslužiteljskoj strani.

5.4. Tjedni zadaci

Tjedni zadaci su zadaci koji se ponavljaju svaki tjedan, što znači da za pojedini zadatak svaki tjedan postoji informacija je li ga korisnik označio završenim ili nije. Izgled stranice za tjedne zadatke je sljedeći:



Slika 18. Izgled stranice za tjedne zadatke (vlastita izrada)

Prilikom pokretanja stranice prvi tjedan koji je prikazan je trenutni tjedan, tako da prvo što se čini na stranici je dohvaćanje trenutnog datuma te određivanje koji datumi označavaju početak i kraj tog tjedna. Ti datumi se zatim formatiraju te se za navedeni tjedan dohvate podaci i pohrane se u varijablu stanja „spremljeniZadaci“.

Funkcija za dohvat tjedna ima sljedeći kod:

```
const dohvatiTjedan = async () => {
  const trenutniDatum = new Date();
  const trenutniDan = trenutniDatum.getDay();

  const doPonedjeljka = trenutniDan === 0 ? 6 : trenutniDan - 1;
  const pocetakTjedna = new Date(trenutniDatum);
  pocetakTjedna.setDate(trenutniDatum.getDate() - doPonedjeljka);

  const doNedjelje = 6 - doPonedjeljka;
  const krajTjedna = new Date(trenutniDatum);
  krajTjedna.setDate(trenutniDatum.getDate() + doNedjelje);

  const trenutniTjedan = `${formatirajDatum(pocetakTjedna)} -
${formatirajDatum(krajTjedna)}`;
  setTrenutniTjedan(trenutniTjedan);
  setTjedan(trenutniTjedan);
  setPocetakTjedna(pocetakTjedna);
  await dohvatiPodatkeZaTjedan(trenutniTjedan);
};
```

Navedena funkcija poziva funkciju „dohvatiPodatkeZaTjedan“ koja dohvaća podatke te ih pohranjuje u navedenu varijablu stanja. Pritiskom na gumb za prikaz podataka prethodnog ili sljedećeg tjedna poziva se funkcija koja izračuna kada je počeo i završio tjedan koji se treba prikazati, te ponovno poziva funkciju „dohvatiPodatkeZaTjedan“ gdje dohvaća podatke novog tjedna koji treba prikazati. Te dvije funkcije su skoro identične, jedina razlika je što se prilikom postavljanja prethodnog tjedna oduzima 7 dana od trenutno prikazanog tjedna, dok se kod postavljanja sljedećeg tjedna dodaje 7 dana.

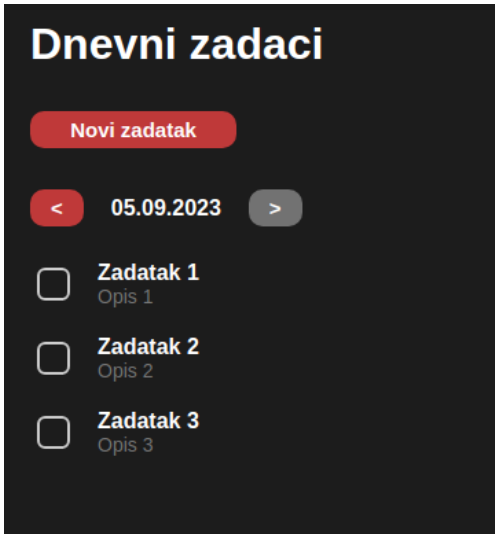
Kod funkcije za postavljanje prethodnog tjedna je sljedeći:

```
 const postaviPrethodniTjedan = async () => {
   const pocetakPrethodnogTjedna = new Date(pocetakTjedna);
   pocetakPrethodnogTjedna.setDate(pocetakPrethodnogTjedna.getDate() - 7);
   const krajPrethodnogTjedna = new Date(pocetakPrethodnogTjedna);
   krajPrethodnogTjedna.setDate(krajPrethodnogTjedna.getDate() + 6);
   const prethodniTjedan = `${formatirajDatum( pocetakPrethodnogTjedna)} -
${formatirajDatum(krajPrethodnogTjedna)}`;
   setTjedan(prethodniTjedan);
   setPocetakTjedna(pocetakPrethodnogTjedna);
   await dohvatiPodatkeZaTjedan(prethodniTjedan);
};
```

Tjedni zadaci također omogućuju brisanje zadataka, kreiranje novih zadataka te ažuriranje stanja. Navedene funkcionalnosti funkcioniraju na jednak način kao i kod trajnih zadataka.

5.5. Dnevni zadaci

Pored tjednih zadataka također postoje zadaci koji se ponavljaju svaki dan. Stranica za dnevne zadatke ima sljedeći dizajn:

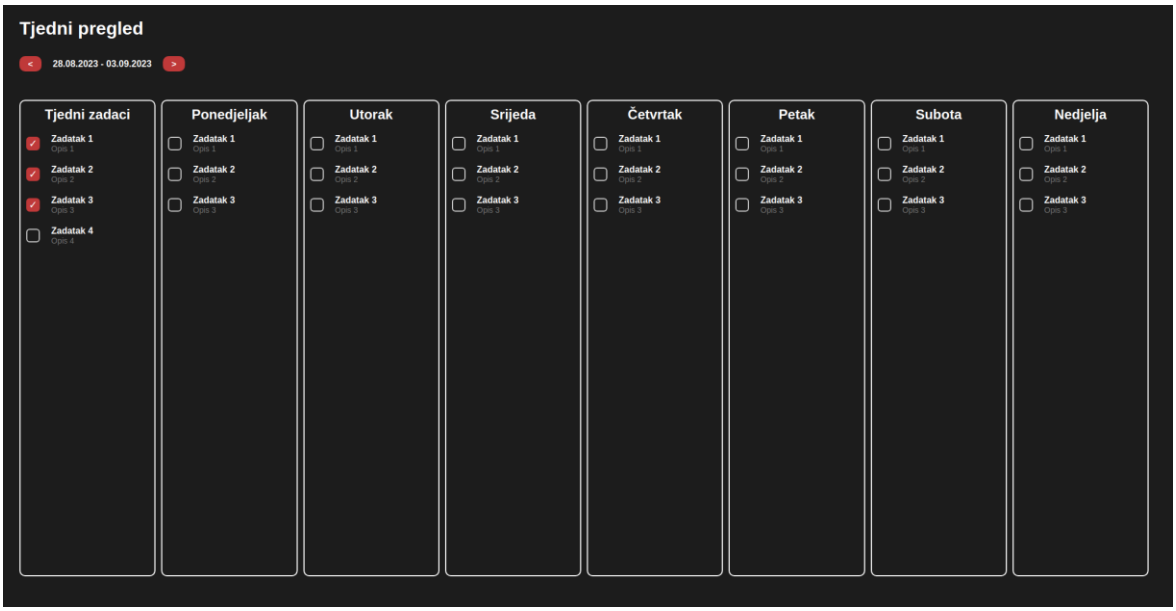


Slika 19. Izgled stranice za dnevne zadatke (vlastita izrada)

Stranica za dnevne zadatke ima skoro identičan dizajn kao i stranica za tjedne zadatke, te je i funkcionalno ista. Jedina značajna razlika je što se kod tjednih zadataka računao početak i kraj svakog tjedna, dok se kod dnevnih zadataka dohvati samo trenutni datum, te se promjenom dana za koji se prikazuju podaci ne mora računati cijeli novi tjedan već se samo prikaže novi dan.

5.6. Tjedni pregled

Stranica za tjedni pregled povezuje stranice tjednih zadataka i dnevnih zadataka te prikazuje te podatke na jednom mjestu. Dizajn stranice je sljedeći:



Slika 20. Izgled stranice za tjedni pregled (vlastita izrada)

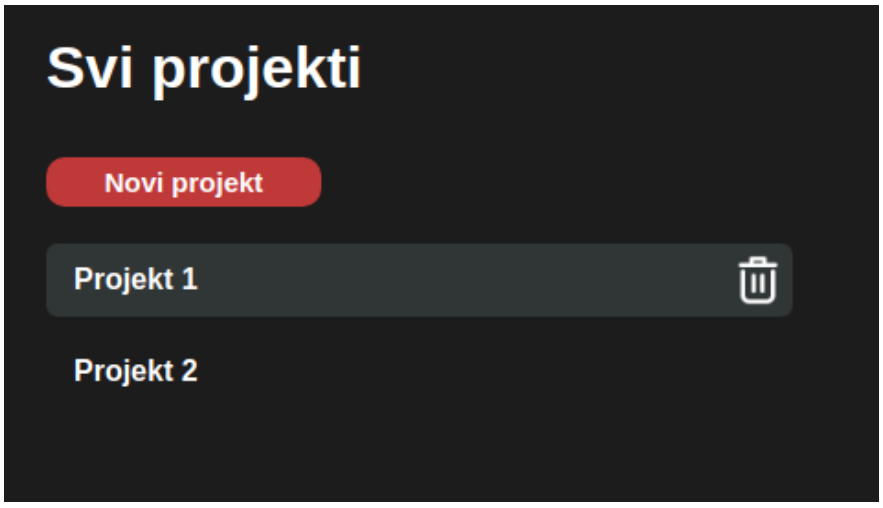
Na toj stranici prvo se dohvati trenutni dan te se definira kojim datumima trenutni tjedan počinje i završava, jednako kao kod tjednih zadataka. Za dohvaćeni tjedan se zatim dohvate dnevni zadaci te se prikaže izvršenost dnevnih zadataka za svaki od 7 dana tog tjedna. Kako bi se to postiglo, kreira se varijabla stanja u kojoj su pohranjeni svi dani u tjednu, te se prilikom dohvata datuma postave datumi svakog dana u tjednu.

```
  const [dani, setDani] = useState([
    {dan: "Ponedjeljak", datum: ''},
    {dan: "Utorak", datum: ''},
    {dan: "Srijeda", datum: ''},
    {dan: "Četvrtak", datum: ''},
    {dan: "Petak", datum: ''},
    {dan: "Subota", datum: ''},
    {dan: "Nedjelja", datum: ''},
]);
```

5.7. Projekti

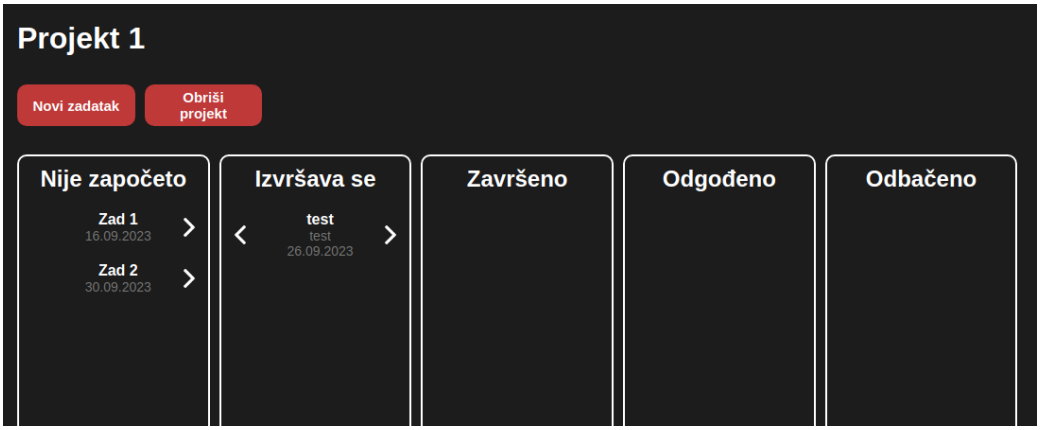
Funkcionalnost projekta uključuje dvije stranice: stranicu sa popisom svih projekata te stranicu sa informacijama o individualnom projektu.

Stranica sa popisom svih projekata je jednostavna. Na njoj se nalaze poveznice koje omogućuju pregled stranice o individualnom projektu za svaki projekt iz popisa te gumb za kreiranje novog projekta. Svaka stavka popisa projekata koristi instancu komponente „Stavka“ koja je korištena za popis zadataka. U ovom slučaju ta komponenta nema potvrdni okvir. Svaka stavka također ima opciju brisanja što omogućuje brisanje cijelog projekta. Brisanje projekta te kreiranje novog projekta funkcionira na jednak način kao što je to bilo kod trajnih, tjednih i dnevnih zadataka. Stranica sa popisom projekata ima sljedeći dizajn:



Slika 21. Izgled stranice sa popisom projekata (vlastita izrada)

Klikom na poveznicu za otvaranje stranice s individualnim projektom, korisniku se prikaže stranica sa sljedećim dizajnom:



Slika 22. Izgled stranice individualnog projekta (vlastita izrada)

Na toj stranici vidljivo je 5 odjeljaka koji označavaju 5 stanja završenosti zadataka. Svaki zadatak ima svoje stanje završenosti. Prilikom pokretanja stranice s poslužiteljske strane se dohvate sva stanja završenosti te svi zadaci, te se zatim dohvaćeni zadaci dodaju u odgovarajuću komponentu za svoje stanje završenosti.

```
const dohvatProjekta = async () => {
  const projekt = await dohvatiProjekt(id);
  setNazivProjekta(projekt.naziv);
  setZadaci(projekt.zadaci)
};

const dohvatStanjaIzvrsenosti = async () => {
  const stanja = await dohvatiStanjaZavrsenosti();
  setStanjaIzvrsenosti(stanja);
};
```

Dohvaćeni podaci se na sljedeći način koriste kako bi se prikazali zadaci sortirani na temelju svog stanja izvršenosti:

```
{stanjaIzvrsenosti.map((stanje, idStanja, polje) => (
  <Odjeljak naslov={stanje.naziv} sekundarni key={stanje.id}>
    {zadaci.length > 0 && (
      zadaci
        .filter((zadatak) => zadatak.stanje_id === stanje.id)
        .map((zadatak) => (
          <ProjektniZadatak
            key={zadatak.id}
            naslov={zadatak.naslov}
            opis={zadatak.opis}
            prikazanoLijevo={idStanja > 0}
            prikazanoDesno={idStanja < polje.length - 1}
            klikDesno={() => klikDesno(stanje.id, zadatak.id)}
            klikLijevo={() => klikLijevo(stanje.id, zadatak.id)}
            datum={zadatak.datum_zavrsetka}
            klikPoziv={() => {
              setZadatakDetalji(zadatak);
              setZadatakDetaljiPrikaz(true);
            }}
          />
        ))
    )}
  </Odjeljak>
))}
```

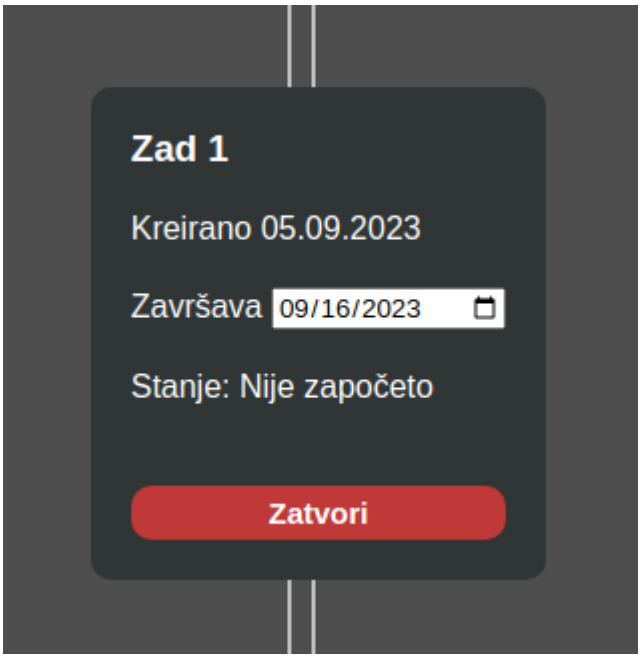
U navedenom kodu iterira se kroz sva stanja završenosti te se za svako prikaže komponenta „Odjeljak“, te se u svakom odjeljku prikažu svi zadaci koji imaju stanje koje pripada tom odjeljku.

Na komponenti „ProjektniZadatak“ definirana su svojstva „prikazanoLijevo“, „prikazanoDesno“, „klikLijevo“, „klikDesno“. Ta svojstva se koriste kako bi se omogućila promjena stanja izvršenosti pojedinog zadatka. Na slici 22. na svakom zadatku vidljive su strelice. Klikom na te strelice pozivaju se funkcije povratnog poziva „klikLijevo“ i „klikDesno“.

Te funkcije zatim promjene identifikator stanja tog zadatka na sljedeći ili prethodni identifikator koji se nalazi u listi „stanjaIzvrsenosti“.

```
const klikLijevo = async (stanjeId, zadatakId) => {
    const trenutnoStanje = stanjaIzvrsenosti.findIndex(stanje => stanje.id === stanjeId);
    const novoStanje = stanjaIzvrsenosti[trenutnoStanje - 1].id;

    await promijeniStanjeProjektnogZadatka(zadatakId, novoStanje);

    const noviZadaci = zadaci.map(zadatak =>
      zadatak.id === zadatakId ? { ...zadatak, stanje_id: novoStanje } : zadatak
    );
    setZadaci(noviZadaci);
  };
```

Kada korisnik pritisne na pojedini zadatak, prikaže se prozor na kojem su vidljive informacije o tom zadatku. Taj prozor ima sljedeći dizajn:



Slika 23. Prozor s informacijama o projektnom zadatku (vlastita izrada)

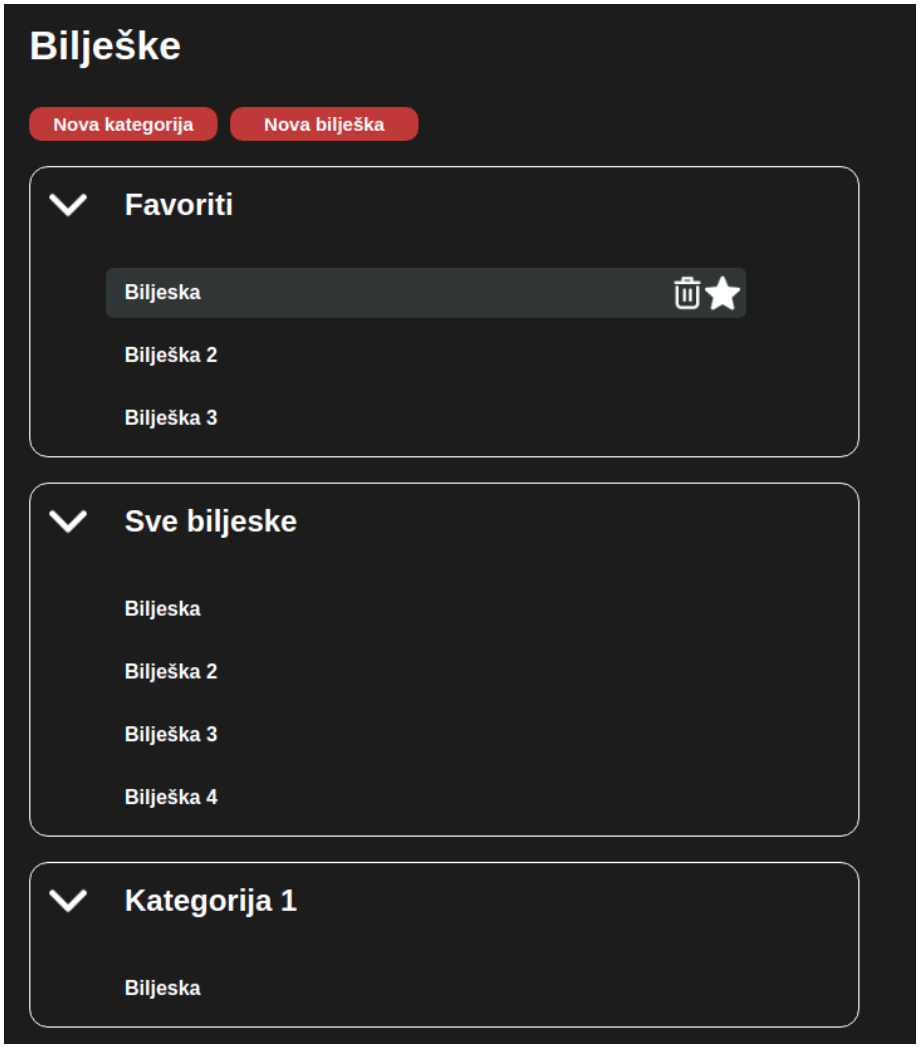
Na tom prozoru vidljive su informacije o tom zadatku te je također omogućena promjena datuma završetka tog zadatka.

Također je moguće kreirati novi projektni zadatak, što funkcionira na jednak način kao i kreiranje trajnih, tjednih te dnevnih zadataka.

5.8. Bilješke

Funkcionalnost bilješki se sastoji od dvije stranice: stranice sa popisom svih bilješki i kategorija, te stranica za pojedinu bilješku. Na stranici sa popisom bilješki prikazane su sve kategorije te su prikazane sve bilješke podijeljene po svojim kategorijama. Svaku bilješku može se obrisati te dodati u favorite. Na toj stranici također je moguće kreirati novu kategoriju te novu bilješku. Svaka bilješka u popisu služi kao poveznica na stranicu te pojedine bilješke, gdje se mogu mijenjati kategorije i sadržaj te bilješke.

Stranica s popisom kategorija i bilješki ima sljedeći dizajn:



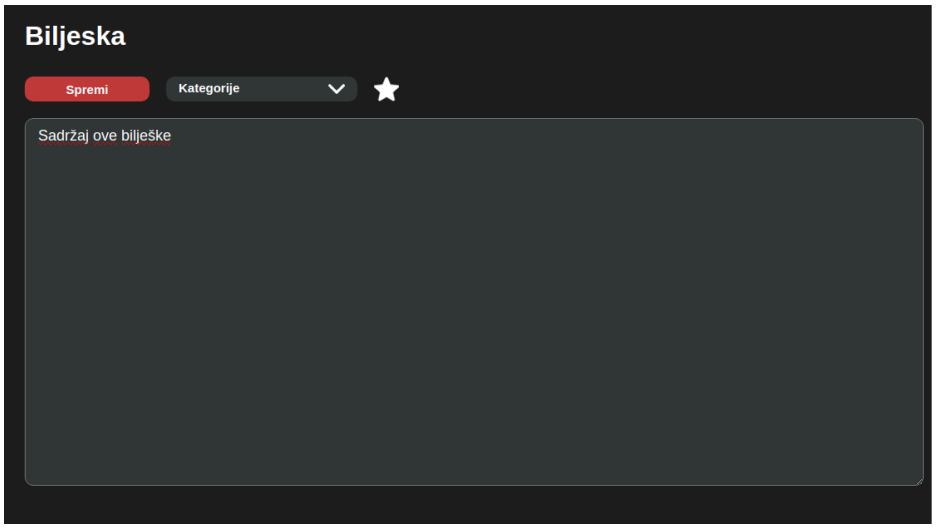
Slika 24. Dizajn stranice s popisom bilješki (vlastita izrada)

Na navedenoj stranici prvo se dohvaćaju sve kategorije i sve bilješke te se postavljaju u varijable stanja u obliku polja. Kroz dohvaćena polja se zatim iterira te se za svaku kategoriju kreira komponenta „PadajuciOdjeljak“, te se u svaki padajući odjeljak dodaju one bilješke koje u popisu svojih kategorija sadrže navedeni identifikator kategorije. To je vidljivo u sljedećem kodu:

```
{popisKategorija
  .map((kategorija) => (
    <PadajuciOdjeljak key={kategorija.id} naslov={kategorija.naziv}>
      {popisBiljeski
        .filter((biljeska) =>
          biljeska.kategorije.some(
            (kat) => kat.kategorija_id === kategorija.id
          )
        )
        .map((biljeska) => (
          <Biljeska
            key={biljeska.id}
            naslov={biljeska.naslov}
            biljeskaFavorit={biljeska.favorit}
            favoritKlik={() => {
              promijeniStanjeFavorita(biljeska.id, !biljeska.favorit);
            }}
            klikPoziv={() => {
              navigacija(`${biljeska.id}`);
            }}
            brisanje={() => {
              setBrisanje(true);
              setBiljeskaZaBrisanje(biljeska.id);
            }}
          />
        ))}
    </PadajuciOdjeljak>
  ))}
```

Pojedinoj bilješci moguće je promijeniti stanje favorita, što znači dodati ju u popis favorita ako nije favorit, a ukloniti iz popisa ako je. Kako su favoriti jedna od kategorija bilješki, da bi se to postiglo potrebno je dodati ili ukloniti identifikator te kategorije iz popisa kategorija pojedine bilješke.

Kada korisnik pritisne na jednu od bilješki iz popisa, prebaci ga se na stranicu te bilješke. Na toj stranici može se mijenjati sadržaj bilješke te kategorije bilješke, te ta stranica ima sljedeći dizajn:



Slika 25. Dizajn stranice pojedine bilješke (vlastita izrada)

Na toj stranici može se mijenjati stanje favorita bilješke, što funkcionira na jednak način kao i kod popisa svih bilješki. Također se mogu mijenjati kategorije bilješki, što funkcionira jednako kao i mijenjanje stanja favorita, zato što su favoriti zapravo jedna od kategorija.

Također se može mijenjati sadržaj bilješke, te se pritiskom na gumb spremi šalje HTTP zahjtev prema klijentskoj strani gdje se ažurira sadržaj te bilješke.

5.9. Poslužitelj

Poslužiteljska strana aplikacije ima 38 definiranih putanja koje služe za interakciju s klijentskom stranom aplikacije. Funkcije koje su definirane za izvršavanje prilikom slanja HTTP zahtjeva na te putanje imaju sličnu strukturu te sve funkcioniraju na isti princip. Prvo što se izvrši je provjera zahtjeva, provjerava se ima li proslijeđeni zahtjev u svom zaglavlju JWT token za autentikaciju te je li taj token ispravan. Prilikom kreiranja tokena koristi se jedinstveni identifikator korisnika, te se taj identifikator također treba poslati u tijelu zahtjeva, zato što se na temelju tog identifikatora utvrđuje je li token ispravan. Za provjeru tokena koriste se sljedeće funkcije:

```
exports.provjeriJWT = function (token, korisnikId) {
  try {
    const dekodirano = jwt.verify(token, konfiguracija.tajniKljuc);
    return dekodirano.korisnikId == korisnikId;
  } catch (error) {
    return false;
  }
};
```

```
exports.provjeriZahtjev = function (zahtjev) {
  const token = zahtjev.headers.authorization;
  const id = zahtjev.params.id;
  const provjera = exports.provjeriJWT(token, id);
  return provjera;
};
```

Ukoliko je zahtjev ispravan, izvrši se SQL upit nad bazom podataka koristeći podatke koji su također proslijeđeni unutar zahtjeva. Za interakciju s bazom podataka kreirane su dvije funkcije. Te funkcije nazivaju se „dohvati“ i „izvrši“. Obje funkcije kao parametre primaju SQL upit te podatke koji se koriste unutar tog upita, te je razlika između njih u tome što funkcija „dohvati“ služi za SQL upite koji dohvaćaju podatke te ih vraćaju korisniku, dok „izvrši“ služi za SQL upite koji ne vraćaju podatke već samo izvršavaju promjene nad bazom podataka.

```
const sqlite3 = require("sqlite3");
const db = new sqlite3.Database("./baza/baza.sqlite");

exports.izvrsi = function (sql, vrijednosti) {
  return new Promise((resolve, reject) => {
    db.run(sql, vrijednosti, function (greska) {
      if (greska) {
        console.error(greska);
        reject(greska);
      } else {
        resolve(this);
      }
    });
  });
};
exports.dohvati = async function (sql, vrijednosti) {
  return new Promise((uspjeh, greska) => {
    db.all(sql, vrijednosti, (gr, rez) => {
      if (gr) greska(gr);
      else {
        uspjeh(rez);
      }
    });
  });
};
```

Na kraju, svaki od tih upita definira statusni kod odgovora, gdje se postavlja kod „401“ ako korisnik šalje neispravan zahtjev, kod „500“ ukoliko je došlo do greške tokom izvršavanja funkcije, te kod „200“ ako je izvršavanje uspješno, te funkcije koje trebaju vratiti podatke korisniku također u odgovoru vrate te podatke.

Primjer jedne definirane putanje:

```
server.post('/api/zadaci/trajni-zadaci/:id', zadaci.kreirajTrajniZadatak);
```

Funkcija koja se poziva kada korisnik pošalje HTTP zahtjev s POST metodom na navedenu putanju:

```
exports.kreirajTrajniZadatak = async function (zahtjev, odgovor) {
  const id = zahtjev.params.id;
  if (pom.provjeriZahtjev(zahtjev)) {
    const zadatak = zahtjev.body;
    const upit = `INSERT INTO trajni_zadaci (naslov, opis, zavrsen, korisnik_id)
VALUES($naslov, $opis, 0, $id)`;
    const vrijednosti = {
      $naslov: zadatak.naslov,
      $opis: zadatak.opis,
      $id: id
    }
    try {
      const rezultat = await BP.izvrsi(upit, vrijednosti);
      const noviId = rezultat.lastID;
      odgovor.status(201).json(noviId);
    } catch (error) {
      odgovor
        .status(500)
        .json({ message: "Greška prilikom kreiranja zadatka" });
    }
  } else {
    odgovor.status(401).json({ message: "Niste autorizirani" });
  }};
```

6. Zaključak

Ovaj završni rad pružio je duboki uvid u proces razvoja web aplikacije koristeći React kao programski okvir. Tema rada obuhvaća ključne aspekte web tehnologija i programskih jezika koji su temelj za razvoj modernih web aplikacija. Također, rad se fokusira na React kao jedan od vodećih okvira za razvoj korisničkog sučelja.

U radu su analizirani jezici i specifikacije koji su bitni za razumijevanje osnova razvoja web aplikacija te su opisane kategorije programskih jezika koji se koriste za razvoj na strani preglednika. Naglasak je stavljen na React kao popularan okvir za izgradnju korisničkog sučelja zbog njegove fleksibilnosti i efikasnosti.

Sigurnost web aplikacija je istaknuta kao ključna tema, te su opisane osnovne sigurnosne značajke koje su neophodne za razvoj sigurnih aplikacija. Ovo je posebno važno u današnjem digitalnom okruženju.

Praktična primjena teorijskih koncepata u radu ostvarena je kroz izradu web aplikacije pomoću Reacta. Kroz ovaj praktični primjer, rad je demonstrirao kako se koriste ključni koncepti i tehnologije za razvoj modernih, brzih i skalabilnih web aplikacija.

U zaključku, ovaj završni rad pruža duboko razumijevanje procesa razvoja web aplikacije uz korištenje Reacta i naglašava važnost web tehnologija i sigurnosti. Također, praktični primjer izrade web aplikacije doprinosi boljem razumijevanju kako se teorijski koncepti primjenjuju u stvarnom svijetu.

Popis literature

[1] E. A. Scott Jr., „*SPA Design and Architecture: Understanding single-page web applications*“, Manning Publications Co., 2016.

[2] S. Howe, „*Learn to code HTML and CSS: Develop and style websites*“, New Riders, 2014.

[3] D. Flanagan, „*JavaScript: The Definitive Guide, Seventh edition*“, O'Reilly, 2020.

[4] S. Springer, „*Node.js: The Comprehensive Guide*“, Rheinwerk Computing, 2022

[5] B. Sullivan i V. Liu, "*Web Application Security, A Beginner’s Guide*", McGraw Hill Professional, 2011.

[6] A. Banks, E. Porcello, „*Learning React: Functional Web Development with React and Redux.“,* O'Reilly Media, 2020.

[7] Facebook Inc. (bez datuma), React [Na internetu]. Dostupno: https://react.dev/ [pristupano 31.08.2023.].

[8] M. Bertoli, „*React Design Patterns and Best Practices.*“, Packt, 2021.

[9] A. Lerner, F. Coury, N. Murray, C.Taborda „*Ng-Book: The Complete Guide to Angular*“, Fullstack.io, 2018.

[10] G. Chau, „*Vue.js 2 Web Development Projects*“,Packt Publishing, 2017.

Popis slika

Popis tablica