Usporedba razvojnih okvira React i Angular

Vuk, Luka

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Vuk

Usporedba razvojnih okvira React i Angular

DIPLOMSKI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Luka Vuk

Matični broj: 0016129997

Studij: Organizacija poslovnih sustava

Usporedba razvojnih okvira React i Angular

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Neven Vrček

Varaždin, rujan 2023.

Luka Vuk

Izjava o izvornosti

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

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Svrha ovog rada je usporediti razvojni okvir Angular i biblioteku React. Rad se sastoji od teoretskog i praktičnog dijela, gdje su u teoretskom dijelu rada obrađene i opisane tehnologije korištene u radu. Neke od ključnih tehnologija su JavaScript i TypeScript na kojem se baziraju React i Angular, te Firebase koji se koristi u svrhu autentifikacije korisnika i spremanja podataka korištenih unutar aplikacije. Nakon pojašnjenja korištenih tehnologija opisani su pogledi i funkcionalnosti aplikacije kako bi čitatelji ovog rada lakše razumjeli kompletnu aplikaciju. U praktičnom dijelu, izrađena je aplikacija naziva *Realtime Food Recipes* i u Angular okviru i u React biblioteci. Proces izrade aplikacije olakšao je razumijevanje oba razvojna okvira/biblioteke što je omogućilo da usporedba bude lakša i smislenija. Ključni dio ovog rada je sama usporedba Reacta i Angulara koja se fokusira isticanje njihovih sličnosti i različitosti. Usporedba je koncipirana tako da se pomoću primjera iz koda čitateljima objasni kako su neke značajke implementirane koristeći React, a kako su implementirane koristeći Angular. Za usporedbu su korišteni isječci kodova iz aplikacije, te ilustracije koje su popraćene dodatnim objašnjenima. U svrhu izbjegavanja zabune kod čitatelja, primjeri kodova koji su korišteni za usporedbu su prilagođeni i nisu prikazani u cijelosti. No, pored prilagođenog koda, bit će prikazan i puni kod kako bi se čitateljima ovog rada pobliže dočaralo kako izgleda implementacija neke značajke koja je vidljiva na ekranu korisnika.

Ključne riječi: React; Angular; usporedba; Firebase; JavaScript; TypeScript; jednostranična aplikacija;

Sadržaj

1. Uvod

Veliki i brzi napredak današnjeg digitalnog doba doveo je do toga da su web stranice i aplikacije postale neizbježni dio bilo kojeg današnjeg poduzeća. Svako poduzeće, ako želi ostati konkurentno na tržištu, zahtijeva web stranicu, barem kao dio svog vizualnog identiteta. Unutar ovog konteksta, razvojni okviri kao što su React i Angular postali su ključni alati za izgradnju tih digitalnih platformi. Moje iskustvo s Reactom započelo je prije otprilike dvije godine i od tada ga koristim svakodnevno. Međutim, iz znatiželje sam želio naučiti još barem jedan okvir pored Reacta kako bih proširio svoje znanje u polju web razvoja. Iz te znatiželje proizlazi i motivacija ovog rada, a to je usporedba Reacta i Angulara kako bih se uvjerio u njihove razlike, ali i sličnosti vezane uz arhitekturu, sintaksu, usmjeravanje i navigaciju, vrstu DOM-a te cjelokupnog razvojnog procesa. Također, osim upoznavanja i usporedbe Reacta i Angulara temeljenje na teoriji, volio bih iz osobnog iskustva znati kako je to izraditi vlastiti projekt baziran na Reactu i na Angularu. To bi dovelo do toga da iz prve ruke mogu reći kakvo je korisničko iskustvo u radu s ovim tehnologijama, te dati svoje zaključno mišljenje vezano uz implantaciju identične aplikacije i u Reactu i u Angularu.

Aplikacija na kojoj će se temeljiti usporedba ovih tehnologija naziva se *Realtime Food Recipes.* Ova aplikacija pružit će korisnicima istraživanje recepata gdje će svaki korisnik moći dodati svoj vlastiti recept koji će biti vidljiv svim korisnicima aplikacije. Na ovaj način ljubitelji kuhanja mogli bi isprobati nove recepte drugih korisnika, jer svaki uneseni recept ima svoj opis i nabrojane sastojke uz objašnjene korake pripreme recepta. Korisnici također mogu filtrirati recepte po različitim parametrima što omogućava lakši pronalazak traženog recepta. Također, korisnicima se pruža mogućnost komentiranja svih recepta čime se potiče razmjena iskustava. Firebase će biti korišten za autentifikaciju korisnika i spremanje podataka korištenih unutar aplikacije. Obje aplikacije, React i Angular, će koristiti jednaku bazu podataka. Fokus ovog rada temelji se na usporedbi Reacta i Angulara, dok će Firebase biti samo teoretski obrađen.

2. Uvod u razvojne okvire

Razvoj modernih web aplikacija zahtijeva korištenje različitih tehnologija, alata i okvira kako bi se postigla visoka funkcionalnost, performanse i korisničko iskustvo. U skladu s tim, ovo poglavlje ima za cilj pružiti pregled ključnih razvojnih okvira i tehnologija koji će biti korišteni za usporedbe između Reacta i Angulara. Fokus će biti na tehničkim aspektima i karakteristikama svakog pojedinog razvojnog okvira, u svrhu stvaranja bazičnog razumijevanja vezano za funkcionalnost, primjenu i prednosti svakog od alata.

Jedan od ključnih jezika koji će biti korišten u razvoju je JavaScript, koji služi kao osnova za većinu modernih web aplikacija, pa tako i za React i za Angular. On omogućava interaktivnost i dinamičnost na strani korisničkog sučelja, što je postalo neophodno za današnji izgled web stranica. Pored toga, TypeScript, nadogradnja JavaScripta, donosi statičku tipizaciju koja olakšava otkrivanje programskih grešaka i poboljšava održavanje koda. [6]

Nakon toga, bit će opisane dvije ključne tehnologije ovog rada, React i Angular. Oba se okvira koriste za izradu jednostraničnih aplikacija (eng. *Single-Page Applications)*. Također, bit će opisane i objašnjene razlike između jednostraničnih i višestraničnih aplikacija (eng. *Multi-Page Applications).* [7]

Kako bi razvoj React i Angular aplikacije bio moguć, koristit će se Node.js, izvršno okruženje koje omogućava pokretanje JavaScripta izvan web preglednika. Node.js je postao ključni alat u modernom razvoju zbog svoje skalabilnosti i efikasnosti . Za upravljanje eksternim bibliotekama i modulima, bit će korišten sustav za upravljanje bibliotekama NPM (eng. *Node Package Manager*), koji će biti uspoređen s njegovim konkurentom Yarn. Sustavi za upravljanje bibliotekama omogućavaju brzu instalaciju i upravljanje različitim paketima potrebnim za razvoj. [8]

Također, u svrhu olakšavanja razvoja i brže implementacije funkcionalnosti bit će korišten Firebase, platformu koja pruža razne alate i servise za razvoj web aplikacija. Firebase omogućava jednostavnu integraciju autentifikacije putem različitih metoda kao što su e-pošta, Google i Facebook, čime se ubrzava proces kreiranja sigurnih korisničkih računa. Također, Firebase nudi različite baze podataka poput *Cloud Firestore*, koje omogućavaju brz pristup podacima i jednostavno skaliranje. [3]

2.1. JavaScript

JavaScript je ključni jezik u svijetu modernog web razvoja, koji igra ključnu ulogu u omogućavanju dinamičnih i interaktivnih korisničkih iskustava na web stranicama i aplikacijama. Kroz svoju evoluciju, JavaScript je postao osnovna tehnologija za razvoj sučelja mnogih poduzeća, omogućujući programerima da kreiraju responzivna, intuitivna i bogata korisnička sučelja. Ovaj uvodni dio će razmotriti povijest, osnovne karakteristike i značajke JavaScripta, istražujući njegovu ključnu ulogu u razvoju modernih web aplikacija. Povijest JavaScripta datira unatrag u 1995. godinu, kada je *Brendan Eich*, programer tvrtke *Netscape Communications*, stvorio ovaj jezik kako bi omogućio interaktivno korištenje web stranica. Zanimljivo je napomenuti da je za razvoj JavaScript okvira bilo potrebno samo deset dana. Prvobitno nazvan *Mocha*, a zatim preimenovan u *LiveScript*, jezik je konačno dobio ime JavaScript kako bi iskoristio popularnost tadašnjeg jezika Java. Ovaj novi jezik brzo je postao ključna tehnologija za izradu dinamičnih korisničkih sučelja, dodajući interaktivnost i mogućnost manipulacije dokumentima u web preglednicima. [9] [10]

Jedna od ključnih karakteristika JavaScripta je njegova sposobnost izvođenja na klijentskoj strani (u web preglednicima), što omogućava brz odziv i interakciju s korisnicima. JavaScript omogućava izvođenje naredbi u realnom vremenu, omogućujući programerima da odgovaraju na korisničke akcije poput klikova mišem, unos teksta u tekstualna polja i slično.[11] Ova sposobnost dinamičnog izvođenja čini JavaScript nezamjenjivim za razvoj modernih jednostraničnih aplikacija gdje se cijela aplikacija učitava jednom, a sadržaj se mijenja dinamički bez potrebe za ponovnim učitavanjem cijele stranice. Jedna od ključnih prednosti JavaScripta je njegova integracija s HTML-om i CSS-om, koji uz JavaScript predstavljaju temelj za razvoj web aplikacija. Zahvaljujući JavaScriptu, programeri mogu dinamički mijenjati strukturu i stilove stranica (npr. nakon što korisnik klikne na dugme, promijeni boju cijele aplikacije u crveno).[12] Osim toga, JavaScript nudi bogat ekosustav biblioteka i okvira u koje spadaju i React i Angular, koji omogućavaju brži i efikasniji razvoj aplikacija.

Napredak JavaScripta nije stao samo na klasičnom programiranju u web preglednicima. Razvoj tehnologije poput Node.js-a omogućio je programerima da izvode JavaScript izvan preglednika, omogućujući izradu serverske logike i pozadinskog sustava (eng. *Backend)* aplikacija. To bi značilo da se za razvoj cjelovite web aplikacije može koristiti samo jedan jezik i ekosustav, a to je JavaScript. [13]

2.2. TypeScript

TypeScript je nadogradnja nad jezikom JavaScript koja donosi statičku tipizaciju i dodatne značajke programskog jezika kako bi olakšala razvoj, održavanje i skalabilnost složenih web aplikacija. U ovom poglavlju bit će opisan TypeScript kroz primjere koda, u svrhu razumijevanja razloga za uvođenje TypeScripta, njegove glavne značajke i doprinos modernom web razvoju. Jezik TypeScript nastao je kao odgovor na potrebu za jačim alatima za razvoj web aplikacija u JavaScriptu. Dok je JavaScript dinamički tipiziran jezik koji ne zahtijeva određivanje tipova varijabli prilikom deklaracije, TypeScript uvodi statičku tipizaciju koja omogućava programerima da definiraju kojeg će tipa biti korištenja varijabla. Ovo donosi niz prednosti, uključujući ranu detekciju grešaka, bolju dokumentaciju koda i poboljšanu podršku za refaktoriranje koda (eng. *Code refactoring*). [14]

Primjer 1: JavaScript bez tipizacije

```
function zbrojDvaBroja(broj1, broj2) {
    return broj1 + broj2;
}

const zbroj = zbrojDvaBroja (10, "5");
console.log(zbroj); // 105
```

U ovom JavaScript primjeru 1, funkcija *zbrojDvaBroja* prima dva argumenta *broj1* i *broj2*. Međutim, nema jasnog načina da se osigura da su oba argumenta brojevi. Kada je drugi argument *„5“* proslijeđen kao tip *string* umjesto tip *number*, rezultira u neintuitivnom ponašanju i potencijalnim pogreškama u aplikaciji. Rezultat *zbroj* će nakon izvođena funkcije *zbrojDvaBroja* iznositi *105*, a ne *15* kao što je očekivano. JavaScript u ovom slučaju neće izbaciti grešku, što može činiti otkrivanje ovakve pogreške veoma teško ako se ona desi u stvarnom projektu. [14]

Primjer 2: TypeScript sa tipizacijom

```
function zbrojDvaBroja(broj1: number, broj2: number): number {
    return broj1 + broj2;
}

const zbroj: number = zbrojDvaBroja (10, "5"); // Greška pri kompilaciji
console.log(zbroj);
```

U ovom TypeScript primjeru 2, funkcija *zbrojDvaBroja* je definirana s tipovima za argumente i povratnu vrijednost. Kada se pokuša proslijediti vrijednost *5* tipa *string* umjesto tipa *number* kao drugi argument, TypeScript će pri kompilaciji prijaviti sljedeću grešku:

```
Argument of type '"5"' is not assignable to parameter of type 'number'.
```

Ova greška se javlja tijekom kompilacije, što znači da se potencijalne greške otkrivaju prije nego što se aplikacija izvede. Time se smanjuje vjerojatnost neočekivanih problema u produkcijskom okruženju. Ovaj primjer jasno pokazuje kako TypeScript pomaže u prevenciji grešaka tako da osigurava da se tipovi podataka pravilno podudaraju. Statička tipizacija omogućava ranu detekciju potencijalno opasnih situacija i grešaka koje bi mogle biti teže identificirati u dinamički tipiziranom jeziku poput JavaScripta. [14]

Jedna od ključnih karakteristika TypeScripta je mogućnost definiranja tipova za varijable, funkcije, objekte i druge elemente koda. To omogućava razvijateljima (eng. *Developers)* da jasno komuniciraju namjenu svakog dijela koda te minimiziraju potencijalne pogreške koje mogu nastati zbog neočekivanih tipova podataka. Osim toga, TypeScript omogućava automatsko zaključivanje (eng. *Inferring)* tipa podataka, što olakšava proces tipizacije bez potrebe za eksplicitnim deklaracijama tipova. [14]

Primjer 3: Automatsko zaključivanje tipa podataka

```
let poruka = "Pozdrav, ovo je primjer automatskog zaključivanja tipa
              podataka";

poruka = 42; // Pogreška: Tip 'number' nije dodijeljiv tipu string'
```

U primjeru 3, TypeScript automatski zaključuje da je tip varijable *poruka* tipa *string*, na temelju inicijalne vrijednosti. Pokušaj dodijele broja *42* varijabli *poruka* rezultirat će greškom jer se tipovi podataka ne podudaraju

Dopuna JavaScripta statičkom tipizacijom nije jedina prednost koju TypeScript nudi. Ovaj jezik također omogućava i sučelja (eng. *Interfaces*), generičke tipove (eng. *generic types*), dekoratore (eng. *decorators*) i ostale napredne značajke koje unapređuju sposobnost programiranja i izgradnje visokokvalitetnih aplikacija. Sučelja omogućavaju definiranje struktura podataka (npr. objekata) za različite dijelove koda, dok generički tipovi olakšavaju rad s višenamjenskim funkcijama i komponentama. [14]

Primjer 4: TypeScript sučelja i generički tipovi

```
interface Korisnik {
    id: number;
    korisnicko_ime: string;
}

function dohvatiKorisnika<T extends Korisnik>(id: number): T {
    // Povratna vrijednost tipa T, koji nasljeđuje sučelje Korisnik
}
```

U ovom primjeru, definirali smo sučelje *Korisnik* koje opisuje strukturu podataka za korisnika. Zatim smo definirali funkciju *dohvatiKorisnika* koja prima *id* korisnika i vraća objekt

tipa *T*, gdje *T* nasljeđuje sučelje *Korisnik*. Ovo ograničenje govori TypeScriptu da će tip *T* biti neki tip podatka koji ima sve članove (svojstva i metode) definirane u sučelju *Korisnik*, ali ne nužno samo te članove. To znači da će tip *T* moći imati dodatne članove osim onih definiranih u sučelju *Korisnik*.

Dekoratori su još jedna moćna značajka TypeScripta koja ima posebnu primjenu u razvoju okvira kao što je Angular. Dekoratori omogućavaju programerima da dodaju metapodatke klasama, funkcijama i drugim elementima koda. Ovo je posebno korisno u Angularu gdje dekoratori omogućavaju definiranje meta podataka za komponente, servise i druge konstrukte. [14]

Primjer 5: TypeScript Dekoratori

```
@Component({
    selector: 'app-root',
    templateUrl: './app.component.html'
})
export class AppComponent {
    // ...
}
```

U primjeru 5, dekorator @Component pridružuje metapodatke klasi AppComponent, omogućujući Angularu da razumije kako treba interpretirati ovu komponentu i kako je povezati s HTML-om iz *app.component.html* datoteke.

U kontekstu razvojnih okvira kao što su React i Angular, TypeScript igra ključnu ulogu u poboljšanju produktivnosti i kvalitete koda. Kako su oba okvira razvijena s komponentnom arhitekturom, tipizacija omogućava jasno definiranje svojstva i stanja komponenata, što smanjuje mogućnost grešaka i olakšava suradnju među programerima. U Angularu, TypeScript se koristi kao glavni jezik za razvoj, što dodatno doprinosi stabilnosti i strukturiranosti koda. [14]

2.3. Node.js i značaj za React i Angular

Node.js predstavlja iznimno bitan alat u svijetu modernog web razvoja, posebice kada je riječ o platformama kao što su React i Angular. Node.js predstavlja otvoreni izvor (eng. *Open-source)* okruženja omogućava izvođenje JavaScript koda izvan okvira web preglednika, otvarajući mogućnosti koje su ključne za razvoj inovativnih i složenih web aplikacija. Na prvi pogled, JavaScript se često percipira kao jezik koji radi unutar preglednika, olakšavajući interakciju s korisničkim sučeljem. Međutim, s pojavom Node.js-a, JavaScript se izdignuo iz okvira preglednika, postajući alat kojim se može izvoditi na serverima ili čak na računalu korisnika. [15]

Kada govorimo o značaju Node.js u razvoju React i Angular aplikacija, bitno je napomenuti nekoliko stavki:

1. Upravljanje paketima
2. Kompilacija i izvođenje koda
3. Lokalni server za razvoj

2.3.1. Upravljanje paketima (NPM i Yarn)

U kontekstu razvoja složenih modernih web aplikacija kao što su React i Angular, pravilno upravljanje ovisnostima (eng. *Dependencies)* i paketima je od velike važnosti za osiguranje stabilnog i učinkovitog razvojnog okvira. Dva ključna alata koja se koriste za ovu svrhu su NPM (eng. *Node Package Manager*) i Yarn. NPM se smatra standardnim alatom za upravljanje paketima u Node.js ekosustavu, te Yarn predstavlja njegovu alternativu. Bez obzira koji se od ova dva alata za upravljanjem paketa koristi, oba su od velikog značaja za izgradnju React i Angular aplikacija [15]

Kao što je već spomenuto, *Node Package Manager* predstavlja standardni alat za upravljanje paketima u Node.js ekosustavu. NPM pruža programerima sposobnost preuzimanja, instalacije i upravljanja različitim paketima i bibliotekama koje se koriste u razvoju aplikacija. Ovisno o specifičnim potrebama projekta, NPM omogućava precizno definiranje ovisnosti putem datoteke *package.json*. U *package.json* datoteci izlistani su sve ovisnosti koji se koriste unutar projekta i pomoću nje osigurava se dosljednost i kompatibilnost među različitim verzijama paketa. [15]

Yarn je razvijen kao odgovor na određene izazove s brzinom i pouzdanošću prilikom preuzimanja paketa putem NPM-a, predstavlja alternativu koja donosi naprednije metode upravljanja ovisnostima i ubrzan proces instalacije paketa. Yarn koristi naprednije algoritme za rješavanje ovisnosti i osigurava brže preuzimanje paketa zahvaljujući istovremenom preuzimanju i korištenju datoteke y*arn.lock* za održavanje konzistentnosti. Osim toga, Yarn također pruža bolje mehanizme za rješavanje konflikata ovisnosti, čime se smanjuje vjerojatnost problema s kompatibilnošću između različitih paketa. [16]

Iako oba alata imaju isti cilj - olakšavanje upravljanja paketima i ovisnostima - razlike u performansama i pouzdanosti često vode do razmatranja koje alate koristiti u određenom projektu. U kontekstu razvoja React i Angular aplikacija, odabir između NPM-a i Yarn-a ovisi ponajviše o osobnim preferencijama tima za razvoj.

2.3.2. Kompilacija i izvođenje koda

Kompilacija i izvođenje koda predstavljaju značajan aspekt razvoja modernih web aplikacija, uključujući i one temeljene na Reactu i Angularu. Pri razvoju React aplikacija, često se koristi suvremeni JavaScript (ES6+) i JSX, specifičan sintaktički šećer (eng. *Syntactic sugar)* za definiranje i razvoj korisničkog sučelja. No, kako bi ovi moderni konstrukti bili pravilno interpretirani od strane preglednika, potrebno ih je prevesti u stariji JavaScript format. U ovom procesu, Node.js pokazuje svoju važnost omogućujući upotrebu alata poput *Babel-a*. Babel je besplatan JavaScript kompajler (eng. *Compiler)* otvorenog koda koji pruža mogućnost pretvorbe i kompilacije JavaScript koda napisanog u suvremenim standardima (ECMAScript 2015+) u JavaScript kod koji je kompatibilan s različitim preglednicima. Node.js tako služi kao platforma za izvršavanje ovih alata, osiguravajući da se kod pravilno prevede i pripremi za interpretaciju na strani klijenta . [17] [19]

S druge strane, Angular koristi TypeScript koji predstavlja prošireni jezik standardnog JavaScripta koji donosi strožu tipizaciju i dodatne funkcionalnosti za olakšavanje razvoja. No, budući da preglednici razumiju samo JavaScript, kod koji je napisan u TypeScriptu treba biti kompiliran u JavaScript kako bi se omogućilo izvršavanje u preglednicima. U ovom procesu, Node.js igra ključnu ulogu pružanjem podrške za alate kao što je TypeScript kompilator (tsc). Ovaj kompilator prevodi složen TypeScript kod u standardni JavaScript koji preglednici mogu interpretirati, omogućujući da se aplikacija izvodi bez problema kompatibilnosti. Ovi procesi kompilacije i izvođenja koda postaju nezaobilazni koraci u razvoju modernih web aplikacija. U kontekstu React i Angular aplikacija, Node.js se nameće kao ključna platforma koja omogućava upotrebu alata za kompilaciju i pripremu koda za interpretaciju u preglednicima, osiguravajući time dosljedno i učinkovito izvođenje aplikacija. [18]

2.3.3.Lokalni razvojni poslužitelj

Node.js lokalni razvojni poslužitelj ima ključnu ulogu u razvoju React i Angular aplikacija. Na primjer, kako bi React ili Angular aplikacija bila pokrenuta i vidljiva u pregledniku te kako bi programer mogao vidjeti što se uopće u aplikaciji događa, prvo mora izvršiti skriptu za pokretanje lokalnog razvojnog poslužitelja. Naredba *npx react-scripts start* za React ili *npx ng serve* za Angular, Node.js poslužitelj se pokreće kako bi služio aplikaciju tijekom razvojnog procesa. Ovaj poslužitelj omogućava brzu izmjenu, testiranje i interakciju sa živim prikazom aplikacije, što značajno olakšava rad razvijateljima. [17]

Primjerice, brza izmjena i trenutačna povratna informacija postaju ključne tijekom razvoja React ili Angular aplikacija, jer se tako uvelike olakšava i ubrzava razvojni proces. Pri implementaciji i razvoju novih funkcionalnosti, ispravljanje grešaka ili mijenjanje dizajna

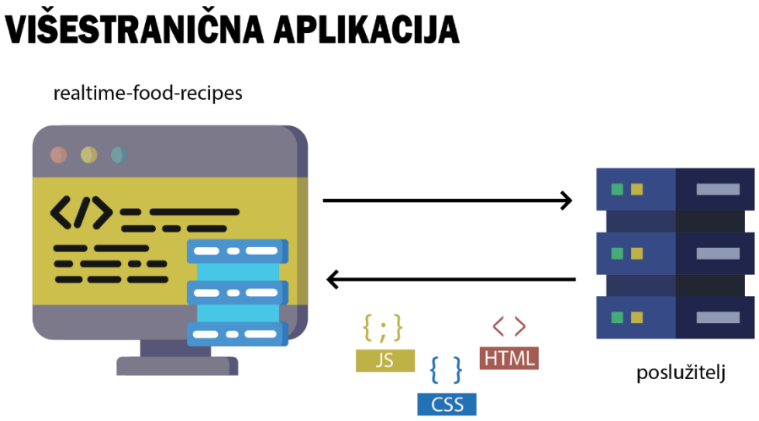
potrebne su učestale promjene u kodu, a automatsko osvježavanje aplikacije koje omogućava Node.js to radi izvanredno. Ova mogućnost omogućava automatsko osvježavanje samo dijelova aplikacije koji su promijenjeni, umjesto potpunog osvježavanja cijele stranice. Na primjer, ako se promijeni samo stil CSS-a ili dio komponente, ta će se promjena primijeniti bez ponovnog učitavanja cijelog sučelja. Ovo štedi vrijeme i čini razvojni proces još učinkovitijim. Uz to, lokalni razvojni poslužitelj osigurava okruženje slično produkcijskom, omogućujući razvijateljima da unaprijed identificiraju moguće probleme koji bi se mogli pojaviti kad aplikacija postane dostupna online. [17]

2.4. Jednostranične aplikacije

U svijetu brzih internetskih veza i sveprisutne upotrebe pametnih uređaja, web aplikacije su doživjele revoluciju. Jedan od ključnih tehnoloških pomaka koji je omogućio stvaranje bogatih, brzih i interaktivnih web aplikacija je koncept jednostranična aplikacija (eng. *Single Page Application - SPA*). SPA je inovativan pristup razvoju web aplikacija koji se suprotstavlja tradicionalnom modelu višestrukih HTML stranica. U svrhu boljeg razumijevanja što su to jednostranične aplikacije, te kako su jednostranične aplikacije promijenile način na koji korisnici doživljavaju web stranice, bit će navedeni konkretni primjeri i usporedit će ih se s klasičnim pristupom. [7]

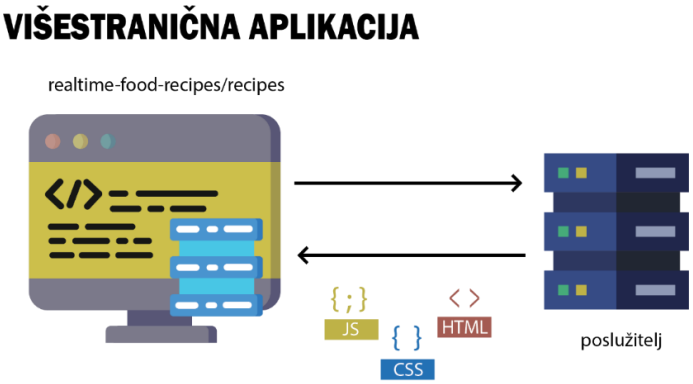
2.4.1. Klasični model: Višestruke HTML Stranice

Ako se pretpostavi da korisnik želi izraditi web stranicu s tri različita pogleda: *Početna stranica, Lista recepata i Moj profil*, u klasičnom modelu, svaki od ovih pogleda bila bi predstavljena zasebnom HTML stranicom. Na primjer, u slučaju kada korisnik klikne na *Početna stranica*, preglednik bi poslao zahtjev poslužitelju za dohvaćanje potrebnih datoteka za prikaz stranice, kao što je *lista-recepata.html*. Poslužitelj bi generirao tu stranicu i poslao je natrag pregledniku. [7]



Slika 1: Primjer za višestraničnu aplikaciju 1/2

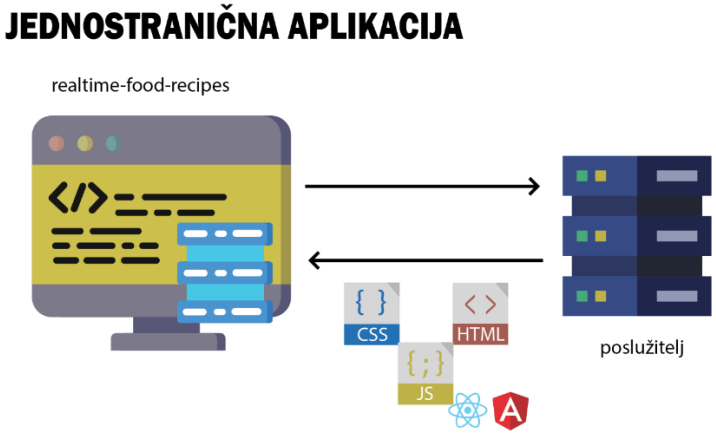
Ovaj cijeli proces bi rezultirao potpunim učitavanjem novog pogleda, uključujući sve slike, stilove i skripte, što bi naravno trošilo vrijeme i resurse. Ako korisnik nakon toga želi ići na neki drugi pogled, primjerice *Lista recepata*, proces sa slike 1 bi se ponovio, uz dodatno učitavanje svih potrebnih resursa za prikaz pogleda *Lista recepata*, vidi sliku 2. [7]



Slika 2: Primjer za višestraničnu aplikaciju 2/2

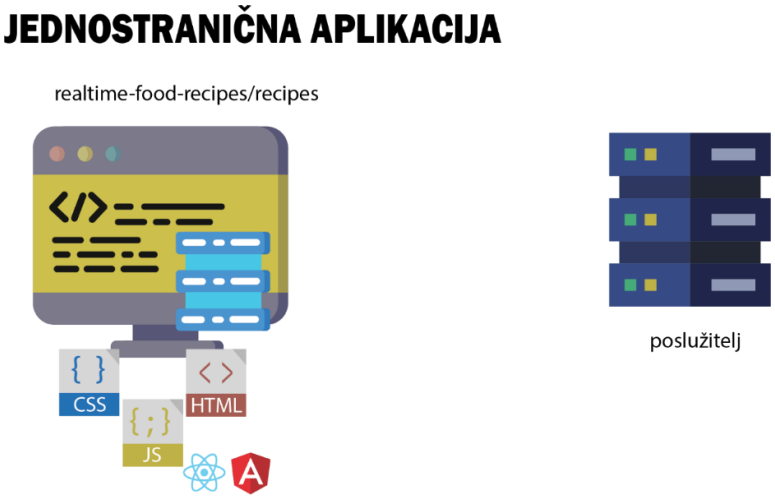
2.4.2. Jednostranični model

Za usporedbu bit će prikazan isti primjer kao na slikama 1 i 2, ali implementiran koristeći React ili Angular. U slučaju jednostraničnih aplikacija, prva posjeta stranici i dalje uključuje preuzimanje svih resursa, ali nakon toga sve interakcije su brze i glatke, vidi sliku 3.



Slika 3: Primjer za jednostraničnu aplikaciju ½

Važno je spomenuti da se u slučaju jednostraničnih aplikacija nakon početnog učitavanja svih potrebnih resursa s poslužitelja uz JavaScript datoteku nalazi i React ili Angular kod. Zbog toga, u slučaju kada korisnik klikne na *Početna stranica*, umjesto učitavanja nove stranice, JavaScript (React ili Angular) aplikacija dinamički mijenja sadržaj prikazan na ekranu. To bi značilo da web preglednik ne mora dodatno zatražiti resurse s poslužitelja kako bi se na ekran korisnika prikazao neki drugi pogled, npr. *Lista recepata*. [7]



Slika 4: Primjer za jednostraničnu aplikaciju 2/2

Primjerice, ako su slike koje se nalaze u *Početna stranica* već preuzete prilikom prvog učitavanja stranice, Jednostranična aplikacija će jednostavno prikazati pogled početne stranice na istom ekranu bez potrebe za novim zahtjevom poslužitelju ili učitavanjem resursa. Isto vrijedi i za ostale stranice. [7]

2.4.3. Prednosti i izazovi koje donose jednostranične aplikacije

Može se zaključiti da jednostranične aplikacije predstavljaju revoluciju u svijetu web razvoja, te se mogu istaknuti nekoliko glavnih prednosti: [20]

1. **Brži korisnički doživljaj**: Jer se većina resursa (HTML, CSS, skripte) učitava samo jednom pri početnom učitavanju, a daljnje promjene se vrše dinamički bez potrebe za ponovnim učitavanjem cijele stranice.
2. **Manje opterećenje servera**: Zbog smanjenog broja zahtjeva prema serveru, jer se samo podaci mijenjaju, ne i osnovna struktura stranice.
3. **Interaktivnost**: Može omogućiti interaktivniji korisnički doživljaj, budući da se može dinamički mijenjati sadržaj bez osjećaja prekida.
4. **Reaktivnost**: Zahvaljujući bibliotekama i okvirima poput React ili Angular, razvoj SPA-a može postati lakši jer se fokusira na komponente i reaktivno ažuriranje korisničkog sučelja.

,ali i izazova: [20]

1. **SEO izazovi**: Pretraga nije uvijek lako optimizirana jer se sadržaj često generira dinamički putem JavaScripta, što može otežati indeksiranje stranica od strane pretraživača.

2. **Inicijalno učitavanje**: Početno učitavanje SPA-a može biti sporije nego kod klasičnih višestraničnih aplikacija, jer se moraju preuzeti sve potrebne skripte i resursi prije nego korisnik vidi bilo kakav sadržaj.
3. **Sigurnost**: SPA-ovi često koriste veće količine JavaScripta koji se izvršavaju na korisničkim uređajima, što može otvoriti vrata za sigurnosne ranjivosti poput XSS, tj. *Cross-Site-Scripting* napada.
4. **Povratna navigacija**: Povratna navigacija u pregledniku može biti problematična, jer se korisnik može izgubiti unutar iste stranice umjesto da se vrati na prethodnu stvarnu stranicu s drugačijom web-lokacijom.

2.5. React

React.js, ili jednostavno React, je JavaScript biblioteka razvijena od strane Facebooka. Opisana je kao deklarativna, efikasna i fleksibilna platforma. Prva verzija Reacta je puštena u svibnju 2013. godine. React se smatra bibliotekom jer, suprotno od Angulara, ima uži opseg, fokusirajući se samo na prikaz korisničkog sučelja aplikacije. Prednost toga je lagana struktura biblioteke, što olakšava učenje i upotrebu Reacta. Međutim, to također znači da se React u određenim kontekstima naziva korisničkom sučeljnom bibliotekom, a ne okvirom. Općenito gledajući, ipak ga se može smatrati okvirom, jer se koristi u istu svrhu kao Angular ili neki drugi razvojni okviri. Početno je razvijen kao JavaScript verzija *XHP-a*, PHP biblioteke stvorene od strane Facebooka. *XHP* je bio modifikacija PHP-a koja je omogućila *XML* sintaksu u svrhu stvaranje prilagođenih i ponovno iskoristivih komponenti, što je također moguće i u Reactu. Taj se razvoj može smatrati kao važan korak u općem smjeru razvoja weba, gdje je JavaScript odabran kao osnovna web tehnologija umjesto PHP-a, koji je bio dominantni standard tijekom 2000-ih. [21] Jedan od razloga za uspjeh Reacta je taj što je bio prvi koji je optimizirao svoju funkcionalnost prema DOM-u - *Document Object Model*. Budući da je manipulacija DOM-om poprilično zahtjevna u smislu računalnih resursa, React je dizajniran da vrši što manje manipulacija DOM-om, koristeći upravljanje stanjem i virtualni DOM za kontroliranje te manipulacije. Upotreba virtualnog DOM-a omogućuje brže ažuriranje Reacta, ali uz veću potrošnju memorije: kako bi brzo ažurirao DOM preglednika, React čuva kopiju virtualnog DOM stabla u memoriji, što povećava potrošnju memorije. No više o tome u poglavlju **5.9.1. Virtualni DOM u Reactu**. [22] [4]

Popularnost Reacta se također proširuje i na brojne biblioteke koje su proizašle iz Reacta, na primjer postoji oblik za razvoj mobilnih aplikacija nazvan *React Native*. Glavni razvojni tim Reacta, Facebook, koristio je *React Native* za razvoj dijelova vlastite mobilne Facebook aplikacije. [1]

2.6. Angular

Angular je okvir koji postoji u dvije verzije, često nazvane *AngularJS* i *Angular 2+.* AngularJS je starija verzija temeljena na JavaScriptu te ju Angular više ne ažurira i preporučuje se korištenje Angular 2+ verzije. Angular 2+ je novija verzija temeljena na TypeScriptu, te je ova verzija Angulara korištena za izradu ovog diplomskog rada. Angular 2+ je pušten u uporabu 2016. godine i razlikuje se od prethodnika AngularJS-a i većine drugih okvira jer se temelji na TypeScriptu. Moguće je koristiti Angular i bez TypeScripta, ali ta se opcija ne preporučuje. Danas je Angular 2+ popularnija verzija Angulara, te kada se govori o korištenju Angulara podrazumijeva se na Angular 2+ verziju. Ova verzija sadržava poboljšanja u performansama i druge prednosti u usporedbi s AngularJS-om. Angular je dizajniran za razvoj većih aplikacija i jedan je od većih i višenamjenskih JavaScript okvira, kako po pitanju programskih mogućnosti tako i po veličini datoteka. [23] [5]

Najčešće spominjani pozitivni aspekti Angulara bili su broj značajki, stil programiranja i dokumentacija. Najčešće spominjani negativni aspekti bili su percipirana pretrpanost, složenost i zahtjevan stil razvoja, te se nije preporučivao za manje razvojne projekte. Također je istaknuto da ima nešto zahtjevniju krivulju učenja. [24]

2.7. Firebase

Firebase predstavlja sveobuhvatan mobilni i web razvojni okvir stvoren od strane *Googlea* kako bi pojednostavio proces stvaranja aplikacija. Firebase spada u skupinu *Backend as a Service* - Baas platformi i pruža integrirani skup alata i usluga koji pomažu razvojnim timovima da učinkovito kreiraju visokokvalitetne aplikacije. Firebase nudi uobičajene servise koji mogu biti potrebni za razvoj aplikacija, a neki od njih su baza podataka, autentifikacija, obavijesti u obliku poruke prema korisnicima i tako dalje. Ove značajke uvelike mogu ubrzati sam proces razvoja čitave aplikacije. Uzmemo li za primjer da određena aplikacija zahtjeva autentifikaciju korisnika, razvojni timovi bi potrošili uvelike više resursa da takvu značajku razviju od same nule u odnosu na rješenje koje Firebase omogućava. Tako razvojni timovi ne moraju ulagati resurse za razvitak složene značajke kao što je autentifikacija korisnika, nego svoje resurse mogu usmjeriti prema razvoju ostatka aplikacije na kojoj se radi. [3] Potpuni set značajki koje Firebase nudi se može podijeliti u dva dijela, a to su:

- Razvojne značajke
- Značajke za rast poduzeća

Kao što i samo ime govori, razvojne značajke pridonose i olakšavaju razvoj aplikacija, dok se u suprotnom značajke za rast poduzeća fokusiraju na klijente i rasta broja korisnika aplikacije. U ovom diplomskom radu bit će korištene razvojne značajke, točnije Firebase autentifikacija (eng. *Firebase Authentication)* i baza podataka u stvarnom vremenu (eng. *The Realtime Database)*. Firebase se ističe svojom sposobnošću upravljanja bazama podataka u stvarnom vremenu, omogućujući trenutačno sinkroniziranje podataka na različitim uređajima - ključno za aplikacije koje zahtijevaju suradnju među korisnicima. Firebase baza podataka u stvarnom vremenu je NoSQL baza podataka na oblaku koja sprema podatke u obliku JSON (eng. *JavaScript Object Notation)* datoteke. Osim toga, ističe se i po naprednom sustavu autentifikacije korisnika koji olakšava siguran pristup aplikacijama, čime se štite osjetljivi podaci i privatnost korisnika. Sa strane razvojnih timova, razvoj značajke poput autentifikacije je kompleksno jer se treba paziti na omogućavanje prijave novih korisnika, prijave postojećih korisnika, upravljanje korisničkim podacima i na kraju svega treba se paziti da se osjetljivi podaci zadrže sigurnima. Isto tako, sa strane korisnika aplikacije, autentifikacija predstavlja jednu dozu zabrinutosti jer autentifikacija zahtjeva osjetljive podatke od korisnika kao što su korisničko ime, šifra, sigurnosna pitanja ili bilo što bi im moglo naštetiti u slučaju da treća strana dođe do njegovih osobnih podataka. Zbog toga, neki od korisnika će lakše pristati na takozvanu autentifikaciju pomoću treće strane, odnosno autentifikaciju pomoću računa kao što su Google, Facebook, Twitter ili na primjer GitHub. Imajući to na umu, Firebase autentifikacija je dizajnirana tako da omogući korisniku prijavu i pomoću treće strane, kao i uobičajenu prijavu koristeći elektroničku poštu i šifru. [3]

2.7.1. Firebase SDK

Firebase također dolazi s nizom SDK-ova (eng. *Software Development Kit*) za različite platforme kao što su iOS, Android i web. Ovi SDK-ovi su komplet alata, biblioteka i resursa koji olakšavaju integraciju Firebase funkcionalnosti u aplikacije. To znači da developeri mogu koristiti već izgrađene komponente umjesto da razvijaju sve funkcionalnosti ispočetka. Za potrebe ovog diplomskog rada koristili su se SDK-ovi poput:

- *createUserWithEmailAndPassword*
- *signInWithEmailAndPassword*
- *signOut*
- *…*

Ova jednostavna integracija, korisničko sučelje i detaljna dokumentacija čine Firebase popularnim izborom među developerima svih razina vještina koji žele stvoriti visokokvalitetne aplikacije brzo i učinkovito. [3]

3. Opis aplikacije

U svrhu ovog diplomskog rada izrađena je aplikacija naziva *Realtime Food Recipes*. Tema aplikacije proizlazi iz potreba kulinarstva s ciljem omogućavanja korisnicima da istražuju, dijele i stvaraju recepte. Funkcionalnost ove platforme osigurava pristup raznovrsnim receptima, čime korisnici mogu proširiti svoj kulinarski repertoar i pronaći nove izazove za svoje kreativne sposobnosti kuhanja.

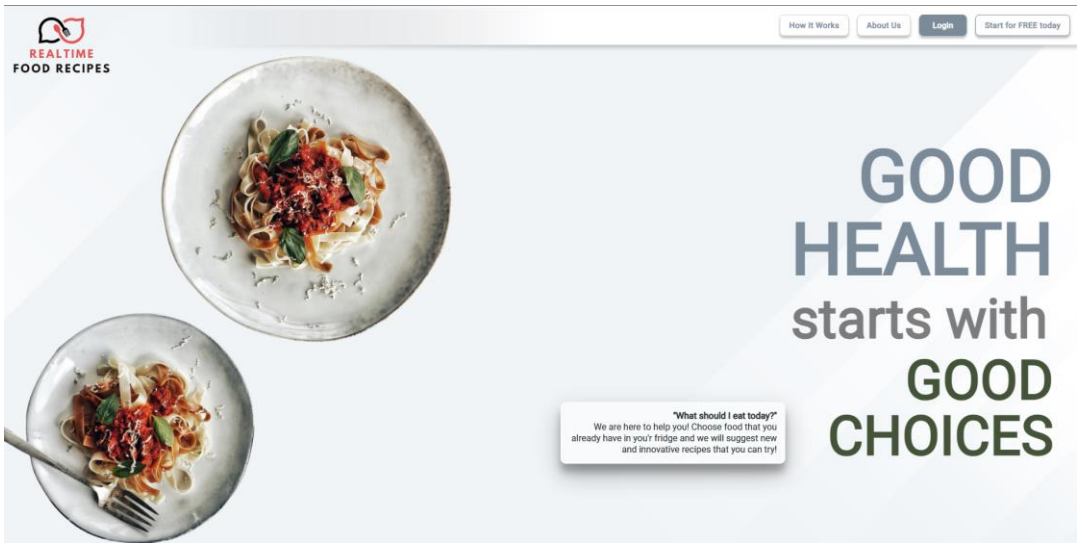
Posebno se ističe mogućnost filtriranja i pretraživanja recepata prema različitim kriterijima kao što su porijeklo jela, tip kuhinje i kategorije. To omogućava korisnicima da usmjere svoju pretragu prema svojim osobnim preferencijama, pojednostavljujući proces odabira odgovarajućih recepata. Ključna dodana vrijednost *Realtime Food Recipes* aplikacije leži u mogućnosti korisnika da unesu sastojke koje već imaju kod kuće, te će im aplikacija filtrirati sve one recepte koji se mogu skuhati koristeći unesene sastojke. Time se naglašava praktičnost i kreativnost u kuhinji, dok se istovremeno promiče svijest o iskorištavanju dostupnih sastojaka umjesto da one propadnu.

Aplikacija također potiče društvenu interakciju putem kreiranja osobnih profila korisnika. Kroz svoje profile, korisnici mogu spremiti i dijeliti vlastite recepte te istraživati jela drugih korisnika. Ova zajednica kulinarskih zaljubljenika potiče razmjenu iskustava, komentiranje recepata, kao i *lajkanje* te dijeljenje omiljenih jela putem društvenih mreža.

Kako bi se ostvarilo personalizirano iskustvo, korisnici moraju izvršiti autentifikaciju. Time se osigurava zaštita korisničkih podataka i omogućava praćenje aktivnosti unutar aplikacije. Dodatno, korisnici mogu birati između nekoliko dostupnih tema aplikacije koje primarno mijenjaju paletu korištenih boja u aplikaciji.

3.1. Pogled Odredišna stranica

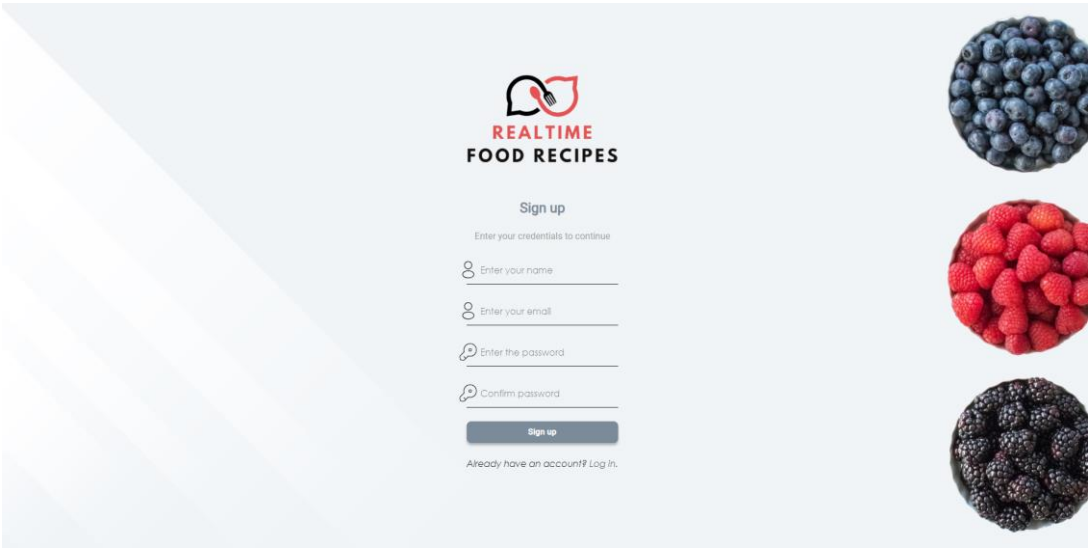
U slučaju da korisnik za vrijeme pristupanja aplikaciji *Realtime Food Recipes* nije prijavljen, prikazat će se takozvana odredišna stranica (eng. *Landing page),* slika 5. Na odredišnoj stranici cilj je privući pozornost korisniku te je iz tog razloga odabran minimalistički dizajn s fotografijom jela u tanjuru s lijeve strane, i s tekstom velikog fonta na desnoj strani. U lijevom gornjem kutu nalazi se identifikacijski znak (eng. *Logo)* aplikacije, a alatna traka proteže se gornjom stranom ekrana. Na alatnoj traci nalaze se interaktivni gumbi pomoću kojih korisnici mogu pristupiti drugim pogledima aplikacije kao što su *How It Works, About Us, Login, Start for FREE today*. Također, klikom na odjeljak s tekstom *What should I eat today?* korisnika se usmjerava na pogled registracije korisnika.



Slika 5: Pogled odredišne stranice

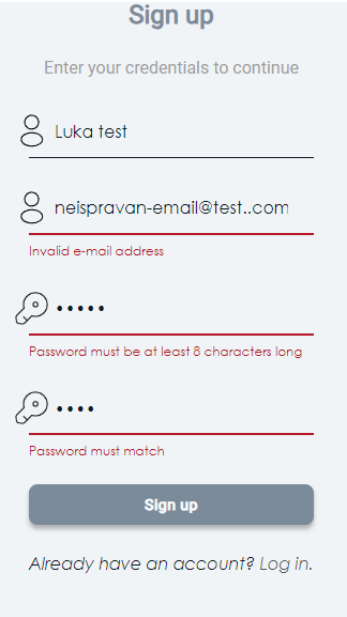
3.2. Pogled Registracija, Prijava, Zaboravljena lozinka

U slučaju da korisnik nije prijavljen u aplikaciju, neće moći pristupiti svim pogledima koje aplikacija nudi, na primjer pogled liste recepata. Pogled registracije, prijave i zaboravljene lozinke koriste se kako bi se korisniku omogućio potpuni pristup svim značajkama i pogledima aplikacije. Na slici 6 prikazan je pogled registracije korisnika koji se sastoji od identifikacijskog znaka, te kratkog opisa korisniku aplikacije. S desne strane se nalazi fotografija, a u samom središtu ekrana nalazi se forma za registraciju korisnika. Za registraciju, korisnik treba unijeti svoje ime, elektroničku adresu, te lozinku. Također, od korisnika se zahtjeva da ponovi već unesenu lozinku u svrhu osiguravanja da je unesena željena lozinka bez greške u tipkanju.



Slika 6: Pogled registracije korisnik

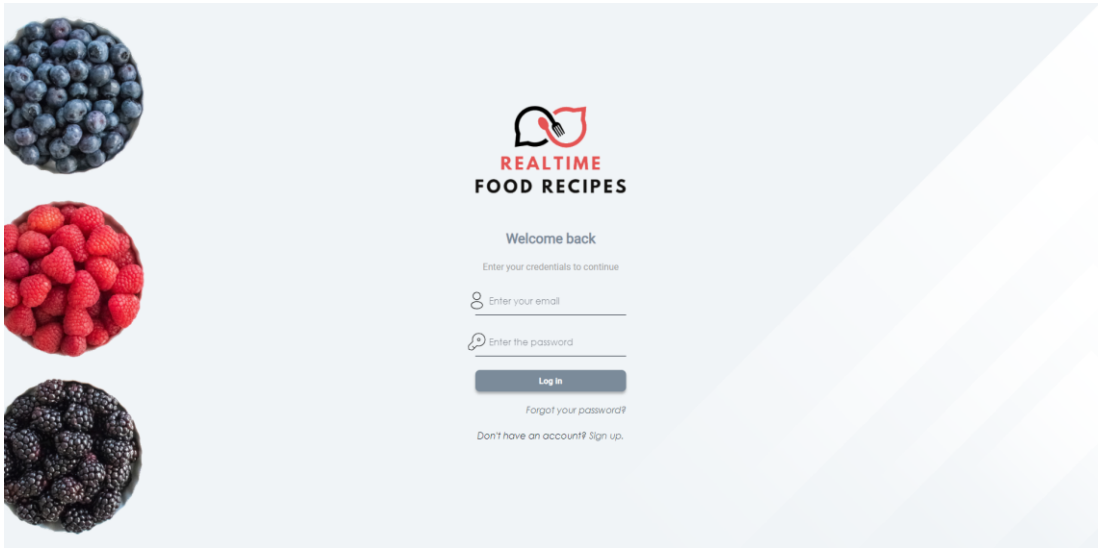
Također, od korisnika se zahtjeva da unese ispravan format elektroničke pošte, te da se lozinka sastoji od minimalno 8 znakova. U primjeru na slici 7 unesena je pogrešna elektronička pošta, te lozinka s nedovoljnim brojem znakova. Također, unesene lozinke se ne poklapaju. U slučaju unosa podataka koji ne zadovoljava kriterij za unos u tekstualno polje, obrub ispod tekstualnog polja bit će prikazan crvenom bojom, a ispod obruba nalazit će se povratna poruka korisniku.



Slika 7: Primjer validacije registracije

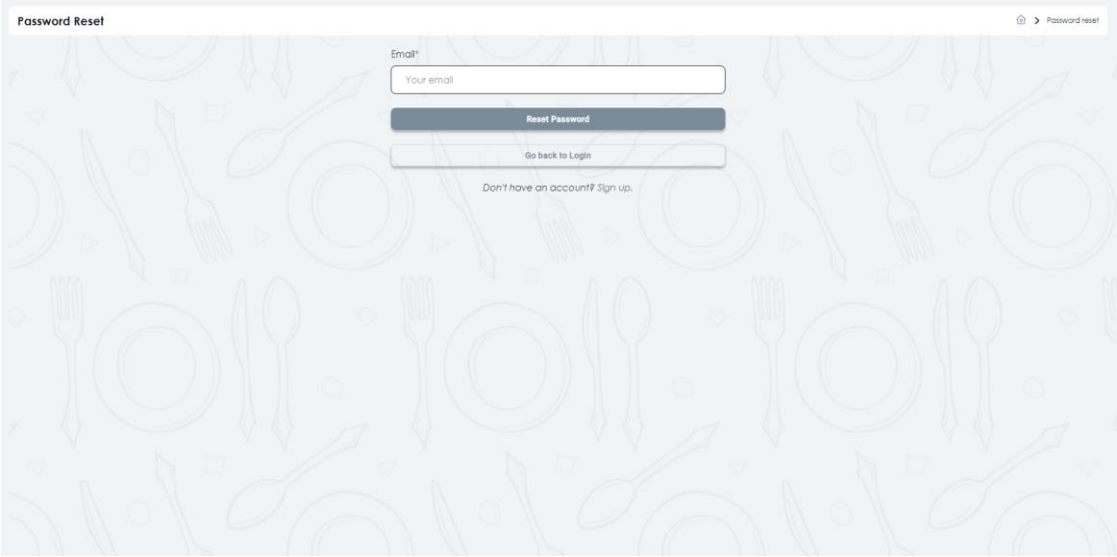
Korisniku je također ponuđena opcija za promjenu pogleda, odnosno za preusmjeravanje na pogled prijave korisnika. Ova značajka je od koristi u slučaju da korisnik već posjeduje korisnički račun, te se klikom na tekst *Log in* može brže preusmjeriti na pogled prijave korisnika. Nakon pravilnog unosa svih podataka u tekstualna polja za registraciju, korisnik klikom na dugme *Sign up* uspješno provodi registraciju.

Pogled prijave korisnika vidljivog sa slike 8 je veoma sličan pogledu registracije korisnika. Osim fotografije koja više nije na desnoj strani, nego lijevoj, korisniku je prikazana forma za prijavu s dva tekstualna polja. Princip validacije funkcionira identično kao i kod registracije, od korisnika se zahtjeva da unese ispravni oblik elektroničke pošte, te lozinku koja se sastoji od minimalno osam znakova. U slučaju da validacija nije zadovoljena bit će prikazana odgovarajuća povratna poruka crvene boje, i crvenim obrubom. Osim mogućnosti navigacije na pogled registracije korisnika klikom na tekst *Sign up.*, korisniku je također omogućena navigacija na pogled resetiranja lozinke klikom na tekst *Forgot your password?*.



Slika 8: Pogled prijave korisnika

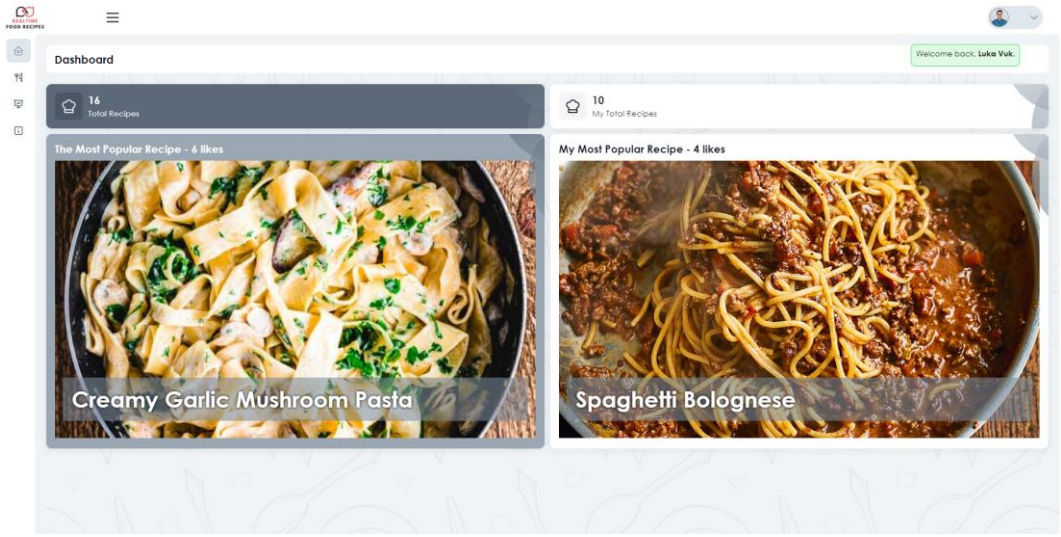
Pogled zaboravljena lozinka sastoji se od navigacijske trake na vrhu ekrana, te sadržaja na sredini ekrana, vidi sliku 9. U ovom pogledu korisnik ima mogućnost unosa elektroničke pošte koju je koristio za registraciju na aplikaciju *Realtime Food Recipes*. Nakon unosa, klikom na dugme *Reset Password* bit će poslana elektronička pošta s uputama za resetiranje lozinke. Dugme *Go back to Login* korisnika će preusmjeriti na stranicu za prijavu.



Slika 9: Pogled zaboravljena lozinka

3.3. Raspored/kostur aplikacije

Nakon uspješne prijave u aplikaciju, bit će prikazan pogled sa slike 10, na kojoj je primaran fokus na nadzornu ploču. Važno je istaknuti da se ovakav raspored aplikacije proteže kroz čitavu aplikaciju u slučaju da je korisnik uspješno prijavljen u aplikaciju. U prijašnjim poglavljima bili su prikazani pogledi registracije, prijave, resetiranja lozinke te odredišne stranice i one su se razlikovale od pogleda vidljivog sa slike 10. Kostur aplikacije sastoji se od zaglavlja, bočne trake, te sadržaja koju određen pogled prikazuje. U ovom slučaju sadržaj kojeg pogled prikazuje je takozvana *Nadzorna ploča* (eng. *Dashboard).*

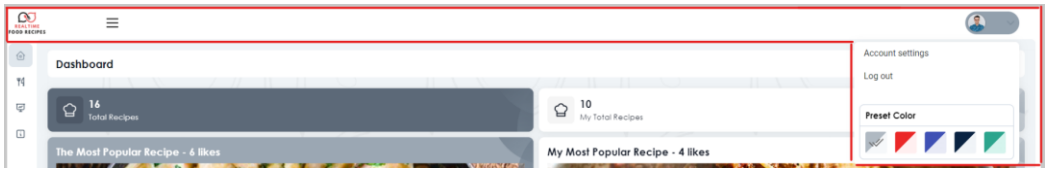


Slika 10: Pogled nakon uspješne prijave u aplikaciju

3.3.1. Zaglavlje i padajući izbornik

Zaglavlje aplikacije nalazi se na samom vrhu ekrana te se na njemu nalazi identifikacijska oznaka aplikacije s lijeve strane, te ikone s tri crtice koja je u svijetu internetskog razvoja poznata kao ikona izbornika. Na desnoj strani zaglavlja nalazi se komponenta koja prikazuje sliku profila prijavljenog korisnika, te strelicom koja je usmjerena prema dolje.

Klikom na spomenutu komponentu, otvara se padajući izbornik s dodatnim opcijama kao što su opcija *Account settings* koja prilikom klika na nju odvodi na pogled moj račun, opcija *Log out* koja klikom na nju izvršava odjavu korisnika s aplikacije, te opciju odabira palete boja koja mijenja boju cijele aplikacije prema boji koju je korisnik odabrao. U ovom slučaju odabrana je siva boja te je ona najdominantnija u aplikaciji. Pored sive boje, dostupne su još četiri boje, a to su crvena, plava, tamno plava i zelena. Na slici 11 vidljivo je zaglavlje s padajućim izbornikom.

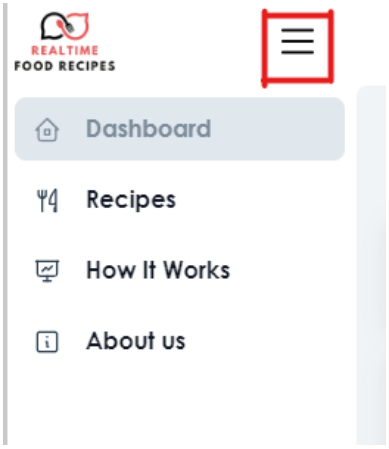


Slika 11: Zaglavlje i padajući izbornik

3.3.2. Bočna navigacijska traka

Bočna navigacijska traka nalazi se s lijeve strane ekrana i sastoji se od ukupno četiri ikone. Klikom na neku od ikona, korisnika se preusmjerava na odgovarajući pogled. Dakle, može se reći da je svaka ikona povezana s jednim pogledom. Prva ikona predstavlja ikonu kućice, te klikom na nju korisnika se preusmjerava na početnu stranicu, u ovom slučaju nadzornu ploču. Pozadina kontejnera u kojoj se nalazi ikona kućice je ofarbana u sivu boju što je korisniku jedna od naznaka da se nalazi u pogledu *Dashboard,* tj. u pogledu nadzorne ploče. Kontejneri preostalih ikona nemaju pozadinu.

Slika 12: Bočna navigacijska traka

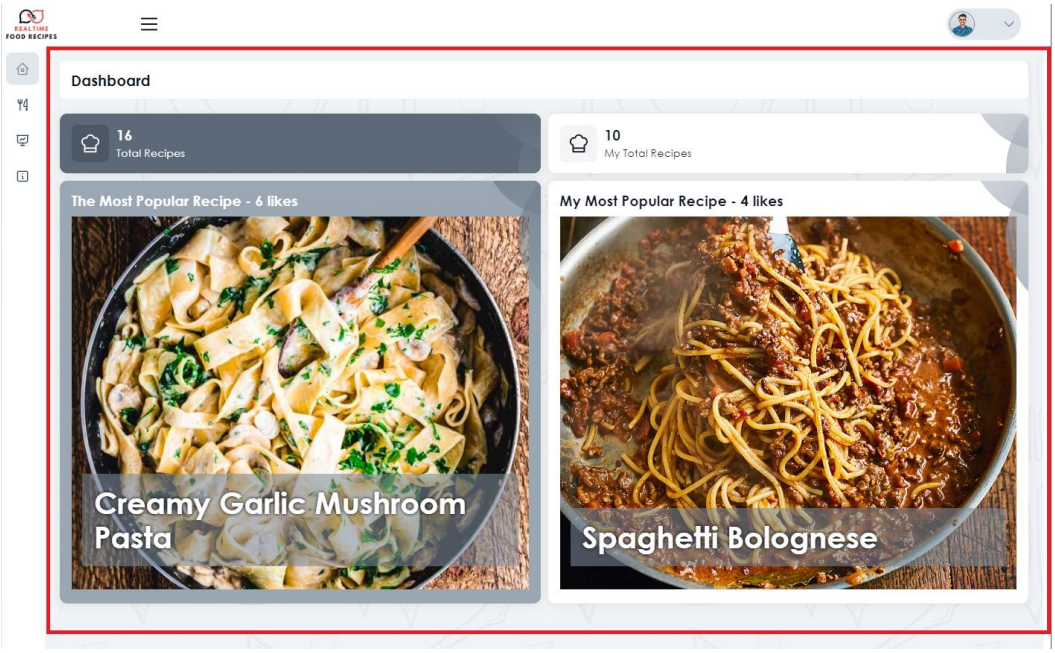


Slika 13: Proširena bočna navigacijska traka

Klikom na ikonu menija koji je na slici 13 označen crvenom bojom, može se mijenjate stanje bočne alatne trake, tj. bočna alatna traka se može proširiti i suziti. U slučaju kada je bočna alatna traka proširena, uz ikonu je također prikazan tekst koji dodatno opisuje na koji će se pogled stranica navigirati u slučaju da korisnik pritisne na nju. Sa slike 12 vidljivo je da su uz ikonu kućice vidljive još i preostale ikone koje navigiraju na poglede *Recipes*, *How It Works* i *About us.*

3.3.3. Kostur sadržaja

Kostur sadržaja se može interpretirati kao kontejner za sav sadržaj koji se trenutno prikazuje na ekranu. Gledajući za primjer sliku 14, crvenom je bojom označen dio na kojemu se prikazuje trenutni sadržaj, a u ovom slučaju to je pogled *Dashboard.*

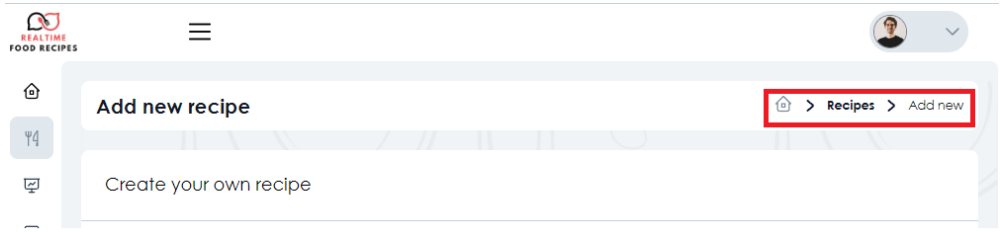


Slika 14: Kostur sadržaja

3.3.4. Krušne mrvice

Krušne mrvice (eng. *Breadcrumbs)* predstavljaju povijest navigacije te korisnicima pruža vizualan trag puta koji su prošli kroz aplikaciju. *Breadcrumbs* se obično prikazuju na vrhu stranice ili aplikacije te prikazuju hijerarhijsku strukturu web mjesta i tako pomažu korisnicima da razumiju svoju trenutnu lokaciju unutar strukture aplikacije, a ujedno i korisnicima olakšavaju navigaciju.

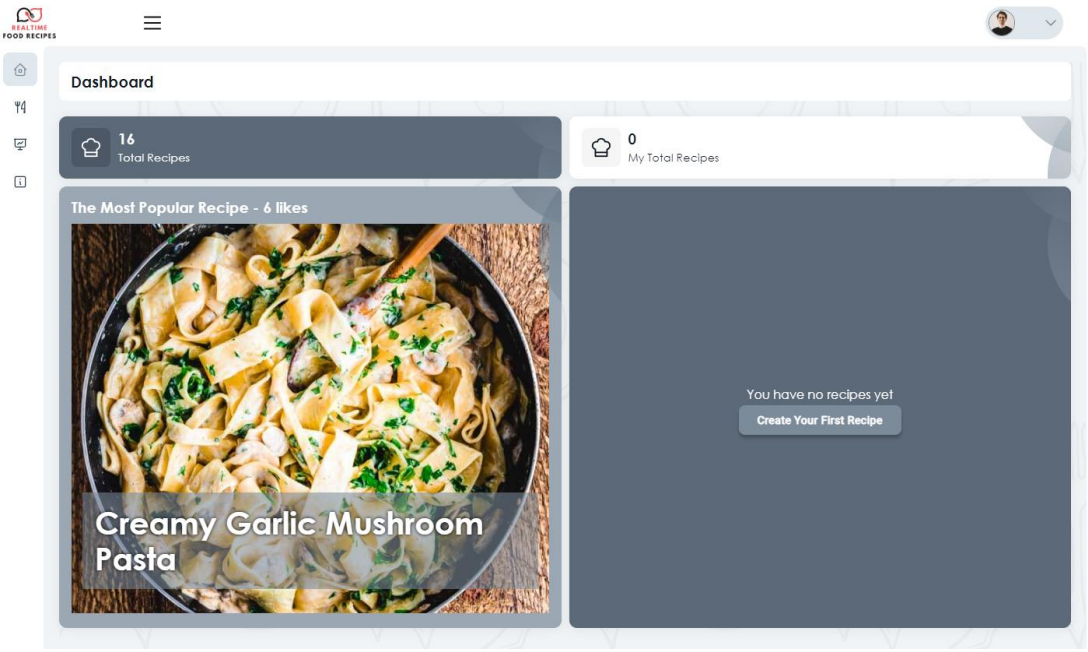
Iz primjera sa slike 15 vidljive su krušne mrvice koje su označene crvenim obrubom. Iz primjera je lako primijetiti da se korisnik trenutno nalazi na pogledu za dodavanje novog recepta. U ovoj aplikaciji *breadcrumbs* su dio kostura sadržaja jer su uvijek prikazani na svakoj stranici, osim u pogledu *Dashboard* jer je taj pogled ujedno i početni pogled te ne zahtijeva prikaz krušnih mrvica. U slučaju da se korisnik želi vratiti na prijašnji pogled, *Recipes*, sve što treba učiniti je kliknuti na tekst *Recipes* koji se nalazi u krušnim mrvicama.



Slika 15: Krušne mrvice

3.4. Pogled Nadzorna ploča

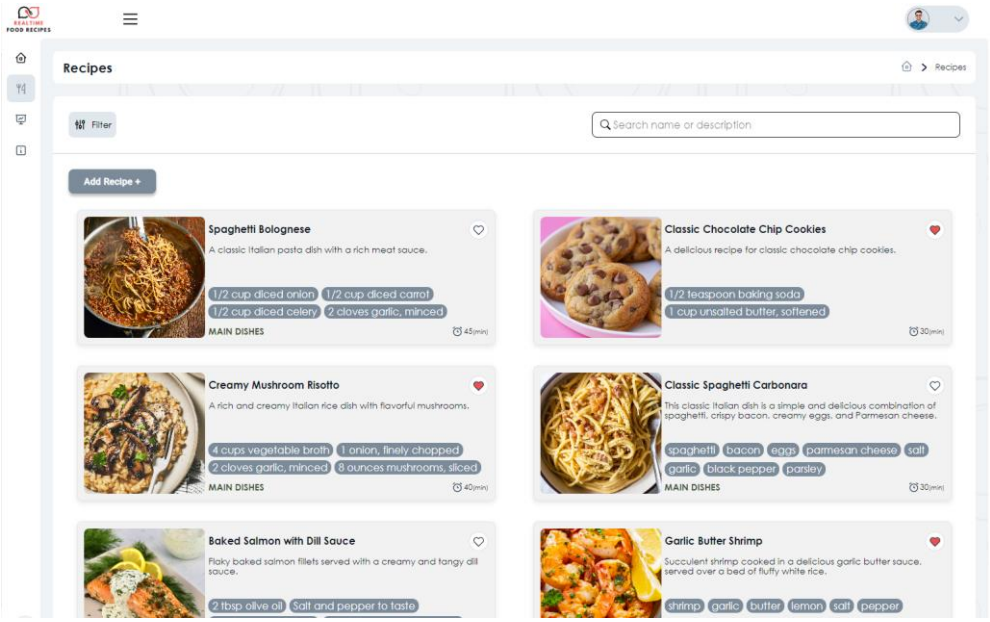
Pogled nadzorne ploče (eng. *Dashboard*) je pogled koji se prvi prikazuje nakon korisničke prijave u aplikaciju *Realtime Food Recipes*, a može se vidjeti sa slike 16. Na ovom pogledu korisniku je prikazan mali statistički prikaz broja sveukupnih recepata te najpopularnijeg recepata. U primjeru sa slike 16 može se vidjeti da trenutni broj sveukupnih recepata na ovoj aplikaciji iznosi 16, dok je najpopularniji recept *Creamy Garlic Mushroom Pasta* s ukupnih 6 oznaka sviđa mi se (eng. *Likes).* Osim toga, prijavljenom korisniku je prikazan i broj recepata koje je unio u aplikaciju, a u ovom slučaju iznosi 10 recepata, od kojih se izdvaja *Spaghetti Bolognese* sa sveukupne 4 oznake sviđa mi se. U slučaju kao na primjeru sa slike 16 gdje je u aplikaciju ulogiran korisnik koji trenutno nema kreiran niti jedan recept, broj ukupnih recepata korisnika bit će jednak nuli, a na mjestu gdje je inače prikazan korisnikov najpopularniji recept bit će prikazana poruka koja informira korisnika da trenutno nema kreiran niti jedan recept, te se ispod poruke nalazi dugme koje korisnikovim klikom vodi na pogled kreiranja novog recepta.



Slika 16: Pogled nadzorna ploča bez korisnikovih recepta

3.5. Pogled *Recepti*

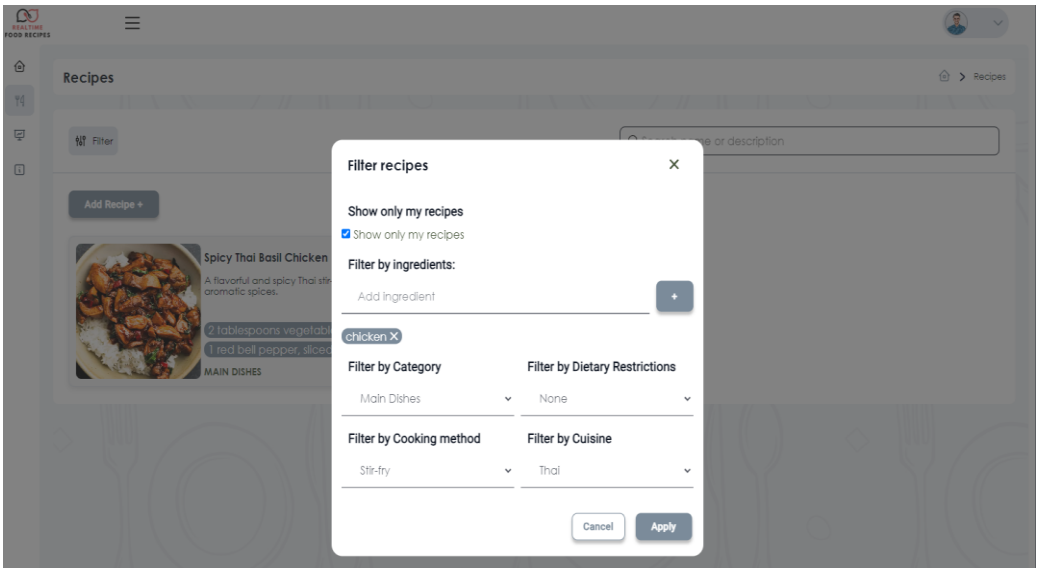
Na slici 17 prikazan je pogled recepata. Ovaj pogled može se smatrati najbitnijim pogledom ove aplikacije jer su ovdje izlistani svi recepti koji su dostupni prijavljenom korisniku. Recepti su prikazani u obliku kartica pri čemu svaka kartica posjeduje naziv recepta, opis recepta, neke od namirnica koje recept zahtjeva, kategoriju kojoj recept pripada te vrijeme kuhanja iskazano u minutama. U gornjem desnom kutu kartice recepta prikazana je ikona srca, a klikom na nju korisnik je označio recept sa sviđa mi se. Oni recepti koje je korisnik označio sa sviđa mi se imaju ikonu srca obojenu u crvenu boju. Kartica recepta također sadrži ilustraciju recepta s lijeve strane. Korisnik ima mogućnost klika na gumb *Add Recipe+* pri čemu se otvara pogled za dodavanje novog recepta.



Slika 17: Pogled recepti

3.5.1. Skočni prozor *Filtriranje recepta*

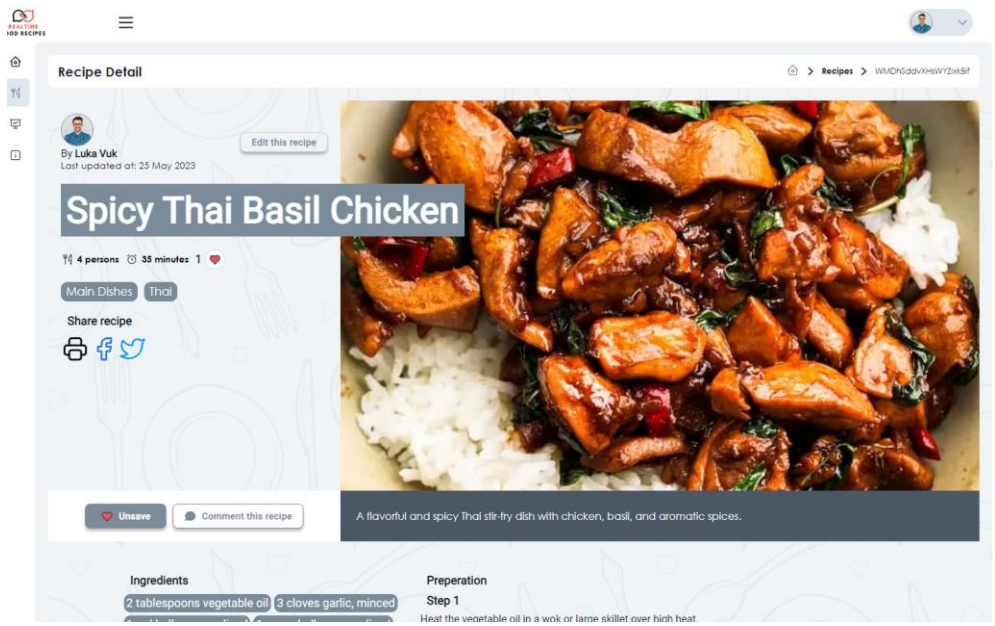
Klikom na dugme *Filter* koje se nalazi u gornjem lijevom kutu u pogleda recepata, otvara se skočni prozor za filter recepata. Recepti mogu biti filtrirani po raznim parametrima, a iz primjera sa slike 18 može se vidjeti da je korisnik odabrao filtere koje će prikazati samo one recepte koji su u njegovom vlasništvu, koji imaju piletinu (eng. *Chicken)* kao jedan od svojih sastojaka, koji spadaju u kategoriju glavnih jela (eng. *Main Dishes)*, koji nemaju dijetalna ograničenja, koji se kuhaju načinom prženja uz miješanje (eng. *Stir-fry)*, te koji pripadaju tajlandskoj kuhinji. Prema odabranim filterima, korisniku je prikazan samo jedan recept koji je također vidljiv sa slike 18, a to je *Spicy Thai Basil Chicken*.



Slika 18: Prikaz filtriranja recepta

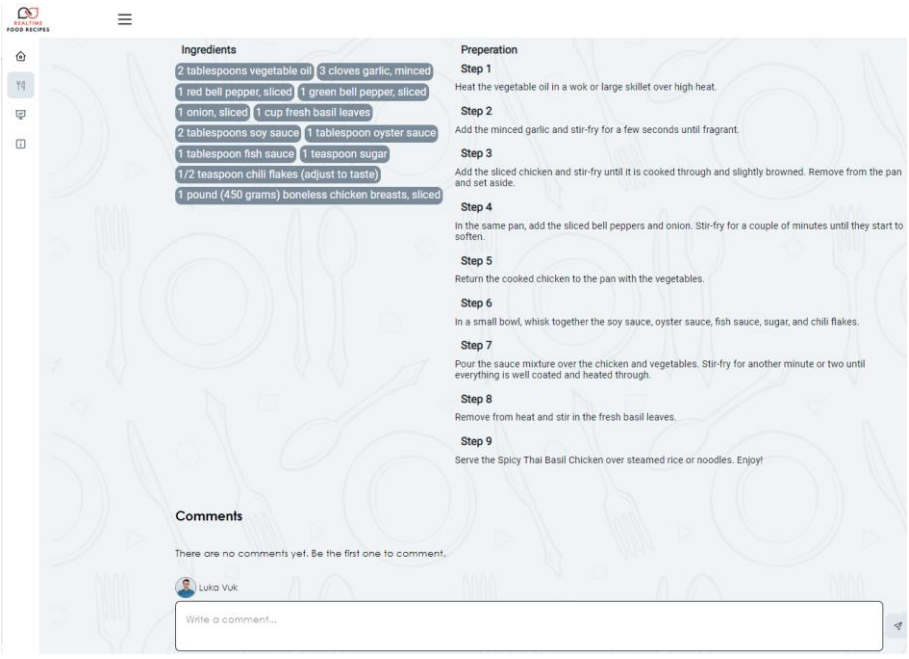
3.6. Pogled Detalji recepta

Nakon filtriranja recepata po kriterijima koji su odabrani s primjera na slici 18, korisnik ima mogućnost klika na karticu recepta pri čemu će aplikacija navigirati na pogled vidljiv sa slike 19. Na slici 19 prikazan je pogled detalja recepta naziva *Spicy Thai Basil Chicken*. U ovom pogledu nalaze se svi detalji recepta počevši s detaljima vlasnika recepta u gornjem lijevom kutu. Osim korisničkog imena prikazan je i podatak kada je recept dodan u aplikaciju, ili u ovom slučaju kada je recept zadnji puta uređivan. Primjer sa slike 19 ukazuje na to da je recept uređivan 25.05.2023. godine, a desno od korisničkog imena nalazi se i dugme *Edit this recipe*. Ovo dugme vidljivo je samo vlasniku recepta, odnosno u slučaju da se korisnik koji nije vlasnik ovog recepta nalazi na pogledu detalja ovog recepta, dugme za uređenje recepta tom korisniku neće biti vidljivo. Na pogledu detalji recepta prevladava ilustracija recepta koja se proteže od desne strane ekrana prema sredini. Isto tako, upadljiv je i sam naziv recepta koji se proteže od lijeve strane ekrana prema sredini, a detalj koji se lako uoči je preklapanje naziva recepta s ilustracijom recepta. Ispod samog naziva recepta vidljive su i ostale informacije recepta kao što su broj porcija jela, broj minuta potrebnih za pripremu jela, broj ukupnih sviđanja ovog jela, te ostale naznake kao što su kategorija recepta, porijeklo recepta i slično. Isto tako, korisniku je omogućeno dijeljenje recepta na društvenim mrežama Facebook i Twitter, te i mogućnost ispisa recepta što korisnik može izvršiti klikom na odgovarajuću ikonu. Ispod same ilustracije recepta vidljiv je i opis recepta, a lijevo pored opisa recepta prikazana su dva gumba pomoću kojih korisnik recept može označiti sa sviđa mi se, ili klikom na dugme *Comment this recipe* pokrenut će se automatsko skrolanje (eng. *Scroll)* do sekcije komentara gdje korisnik može ostaviti svoj komentar na recept ako to želi.



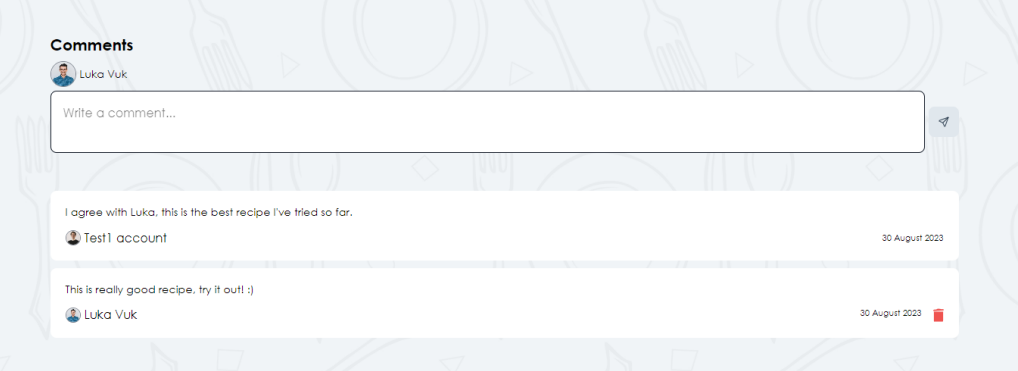
Slika 19: Pogled detalji recepta ½

Na slici 20 vidljiv je ostatak pogleda detalji recepta, gdje su s lijeve strane prikazane namirnice potrebne za pripremu ovog jela, a s desne strane koraci pripreme jela. U ovom specifičnom primjeru jelo se priprema u devet koraka koji su izlistani jedni ispod drugih počevši od koraka 1 pa sve do koraka 9. Na samom podnožju pogleda detalji recepta nalazi se sekcija komentara. Na slici 20 vidljiv je primjer u slučaju kada recept nema niti jednog komentara, te je korisniku prikazana odgovarajuća poruka koja ga o tome informira. Također, korisnik ima mogućnost pisanja u tekstualno polje te klikom na ikonu koja se nalazi desno od tekstualnog polja korisnik će uspješno obaviti komentiranje recepta.



Slika 20: Pogled detalji recepta 2/2

Na slici 21 prikazan je primjer u slučaju kada postoje komentari recepta, te se iz navedenog primjera mogu vidjeti dva komentara. Jedan komentar pripada korisniku *Test1 account,* dok drugi komentar pripada korisniku koji je trenutno prijavljen u aplikaciju, a to je korisnik s korisničkim imenom *Luka Vuk.* Vlasnici komentara imaju mogućnost brisanja komentara, što je omogućeno klikom na crvenu ikonu smeća koja se nalazi na desnoj strani kartice komentara.



Slika 21: Primjer komentara recepta

3.7. Pogled Dodavanje novog recepta

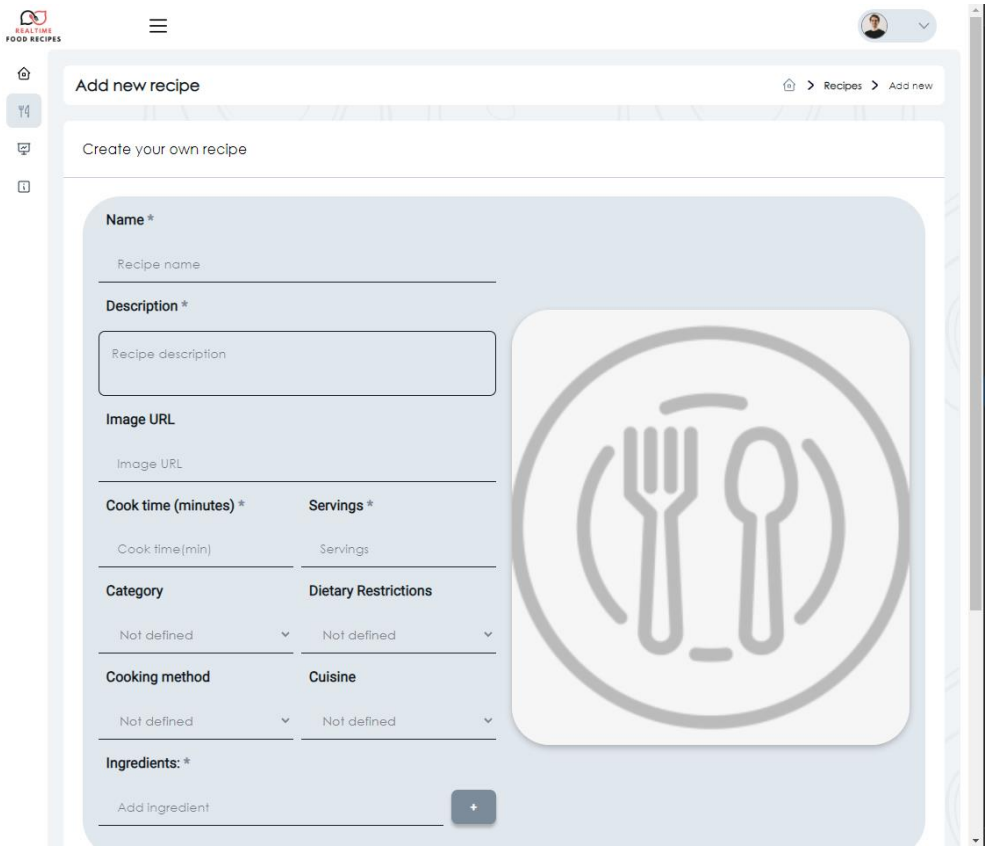
Korisnik ima mogućnost dodavanja novog recepta koristeći pogled *Dodavanje novog recepta* (eng. *Add new recipe).* Ovaj pogled sastoji se od kartice s naslovom koji upućuje korisnika na dodavanje vlastitog recepta, a ispod naslova nalazi se forma s lijeve strane i mjesto za prikaz slike s desne strane, vidi sliku 22. Kako bi korisnik uspješno unio recept u bazu podataka potrebno je popuniti obavezna polja, odnosno ona polja koja na kraju svog naziva imaju znak zvjezdice (eng. *Asterix).* Uz popunjavanje obaveznih polja, potrebno je zadovoljiti i uvjete validacije za pojedina polja.

Validacija polja:

- **Naziv (eng. *Name)*:**
  - Dozvoljeni svi znakovi
  - Minimalan broj znakova - 1
  - Maksimalan broj znakova - 60
- **Opis (eng. *Description)*:**
  - Dozvoljeni svi znakovi
  - Minimalan broj znakova - 30
  - Maksimalan broj znakova - 1500
- **Vrijeme kuhanja (eng. *Cook time)*:**

- Dozvoljeni pozitivni cijeli brojevi
- Minimalni broj - 1
- Maksimalan broj - 1440

- **Broj porcija (eng. *Servings)*:**
  - Dozvoljeni pozitivni cijeli brojevi
  - Minimalni broj - 1
  - Maksimalan broj – 50
- **Sastojci (eng. *Ingredients)*:**
  - Dozvoljeni svi znakovi
  - Minimalan broj unesenih sastojaka – 1
  - Maksimalan broj unesenih sastojaka - 20
  - Minimalni broj znakova po sastojku - 1
  - Maksimalan broj znakova po sastojku – 100
  - Imena sastojaka se ne smiju ponavljati

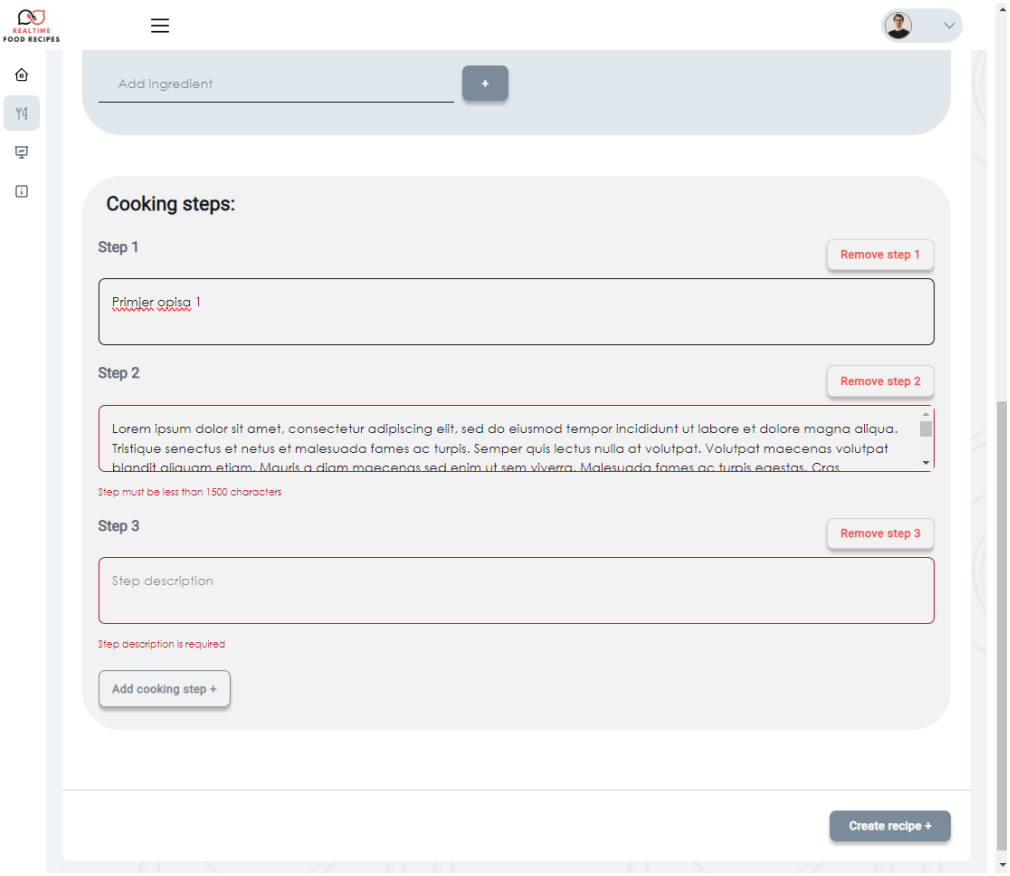


Slika 22: Pogled Dodavanje novog recepta ½

Na slici 23 može se vidjeti drugi dio pogleda *Dodavanje novog recepta* gdje korisnik može dodavati korake kuhanja recepta. Koraci kuhanja služe kao bi se detaljnije opisalo kako se jelo priprema. Koraci kuhanja ne spadaju u obavezna polja, no ako se korisnik odluči dodati korak kuhanja mora zadovoljiti uvjete validacije.

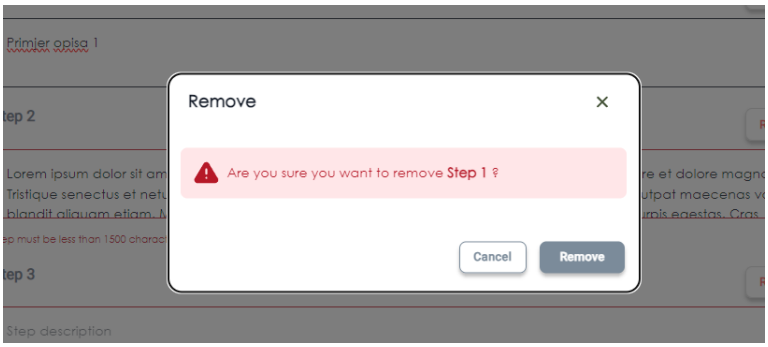
Validacija koraka kuhanja recepta

- **Koraci kuhanja (eng. *Cooking steps)*:**
  - Dozvoljeni svi znakovi
  - Minimalan broj unesenih koraka kuhanja – 0
  - Maksimalan broj unesenih koraka kuhanja - 20
  - Minimalni broj znakova po koraku kuhanja - 1
  - Maksimalan broj znakova po koraku kuhanja – 1500



Slika 23: Pogled "Dodavanje novog recepta" 2/2

Korisnik klikom na dugme *Remove step*, vidljivog sa slike 23, ima mogućnost brisanja svakog koraka kuhanja pojedinačno, a nakon klika na ekranu se prikazuje skočni prozor gdje korisnik ima mogućnost prihvaćanja ili odbijanja akcije brisanja koraka kuhanja, vidi sliku 24.

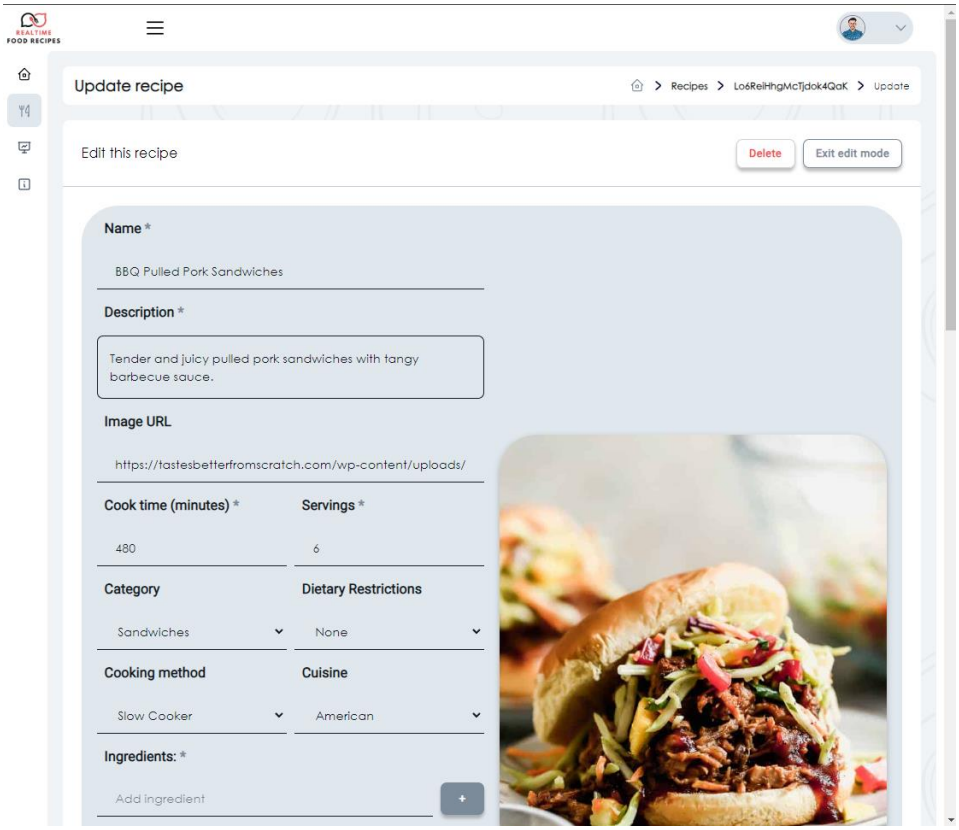


Slika 24: Skočni prozor za brisanje koraka kuhanja

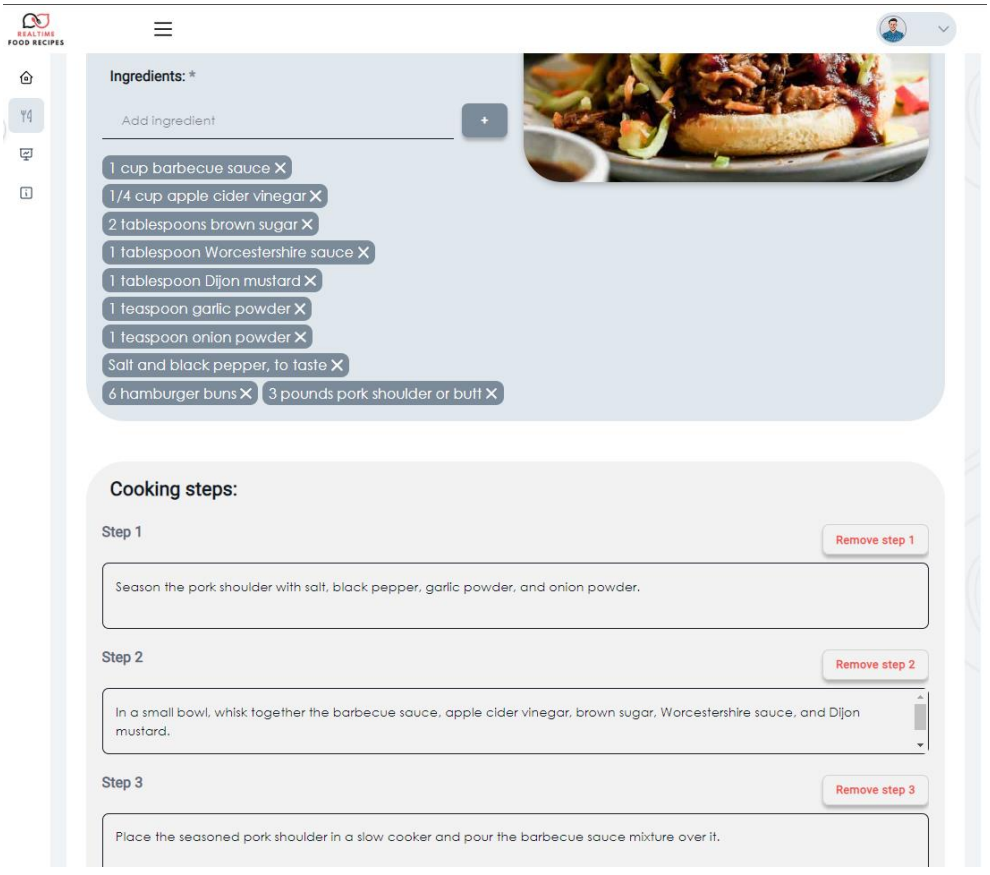
Nakon popunjavanja svih potrebnih polja za dodavanje recepta, korisnik ima mogućnost klika na dugme *Create recipe*+ nakon čega se prikazuje skočni prozor za prihvaćanje ili odbijanja akcije kreiranja recepta. Ako korisnik prihvati akciju kreiranja recepta, recept će ili biti uspješno kreiran nakon čega se aplikacija automatski usmjerava na pogled *Detalji recepta* za dodan recept, ili će se kreiranje recepta odbiti zbog nezadovoljavanja svih uvjeta validacije polja korištenih za dodavanje recepta.

3.7.1. Pogled *Uređivanje recepta*

Aplikacija korisnicima pruža mogućnost uređivanja recepta ako je korisnik vlasnik recepta. Do pogleda *Uređivanje recepta* moguće je navigirati klikom na dugme *Edit this recipe* koje se nalazi na pogledu *Detalji recepta*, vidi sliku 19. Na slikama 25 i 26 prikazan je pogled *Uređivanje recepta* koji ima jednaku strukturu kao i pogled ***Dodavanje novog recepta*** opisano u poglavlju 3.7. Korisnik ima mogućnost uređivanja svih polja, od mijenjanja naziva, opisa, slike, vremena kuhanja, broja porcija, kategorije recepta, dijetalnih ograničenja, metode kuhanja, porijeklo jela, do dodavanja ili brisanja sastojaka recepta i dodavanja, brisanja ili izmjene koraka kuhanja recepta.

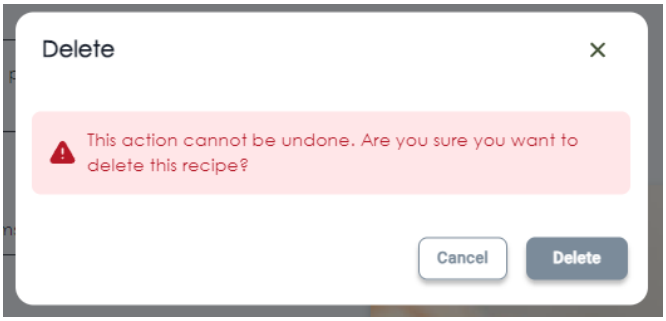


Slika 25: Pogled "Uređivanje recepta" ½



Slika 26: Pogled "Uređivanje recepta" 2/2

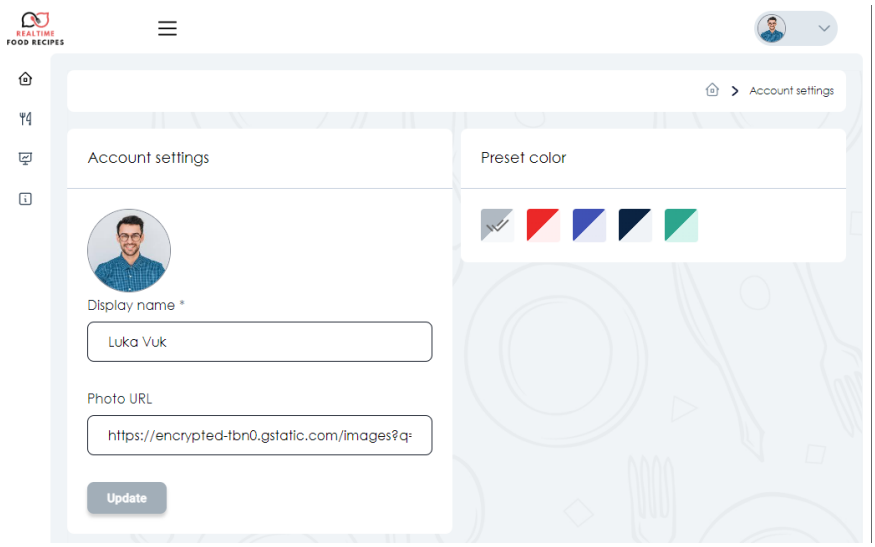
Recept se također može i obrisati, što bi uklonilo njegov zapis u bazi podataka. Klikom na dugme *Delete* vidljivo na slici 25, na ekranu se prikazuje skočni prozor prikazan na slici 27 gdje se od korisnika očekuje da prihvati ili odbije akciju brisanja recepta. Skočni prozor se sastoji od teksta i ikone upozorenja crvene boje uokvirenog u crvenu boju, a tekst jasno upućuje na posljedice brisanja recepta. Na ovaj način korisnika se dodatno osigurava od neželjenih akcija.



Slika 27: Skočni prozor za brisanje recepta

3.8. Pogled Moj profil

Na slici 28 prikazan je pogled *Moj profil*, a do pogleda se može navigirati preko padajućeg izbornika spomenutog u poglavlju **3.3.1. Zaglavlje i padajući izbornik**, slika 11. *Moj profil* sastoji se od dvije kartice. S lijeve strane nalazi se kartica o detaljima korisnika, točnije fotografija profila, te dva polja s informacijama o imenu za prikaz (eng. *Display name)* i jedinstveni lokator izvora (eng. *Uniform Resource Locator)* fotografije. Ime za prikaz je ono ime koje je vidljivo svim ostalim korisnicima aplikacije *Realtime Food Recipes*. Fotografija profila nije obavezna za postavljanje.



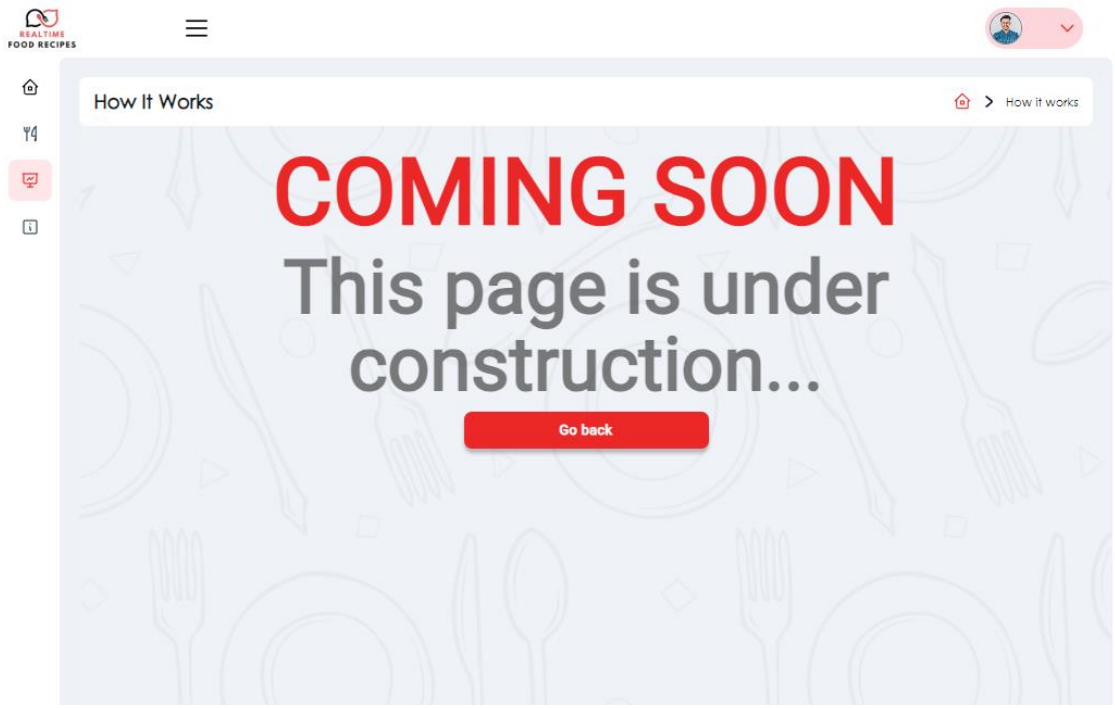
Slika 28: Pogled "Moj profil"

3.8.1. Mijenjanje teme

S desne strane pogleda *Moj profil* nalazi se kartica za odabir unaprijed postavljenje boje (eng. *Preset color)* aplikacije, vidi sliku 28. Unaprijed postavljena boja aplikacije mijenja određuje temu koja se koristi u aplikaciji. Moguće je odabrati između ukupno šest različitih tema aplikacije, a trenutno je odabrana siva tema.

3.9. Pogledi Kako funkcionira i O nama

Pogled *Kako funkcionira* (eng. *How It Works)* i pogled *O nama* (eng. *About Us)* prikazuju *Uskoro dolazi* stranicu. Stranica *Uskoro dolazi* predstavlja privremenu stranicu koja primarno služi za obavijest korisnicima aplikacije da se trenutno razvija na novoj značajki ili u ovom slučaju čitavom pogledu, vidi sliku 29.



Slika 29: Stranica "Uskoro dolazi"

3.10. Pogled Pogreška 404

U slučaju da se korisnik nađe na stranici koja ne postoji, na ekranu će biti prikazan pogled vidljiv na slici 30. Ovaj pogled služi nekoliko važnih svrha i korisna je i za korisnike aplikacije i za vlasnika aplikacije:

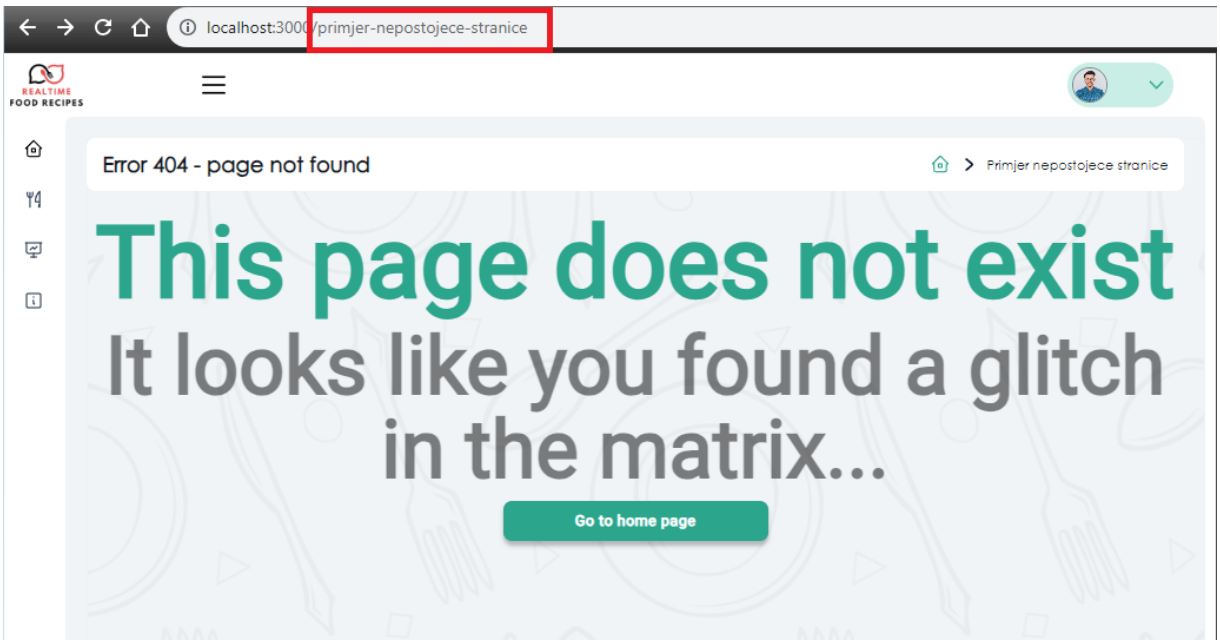
1. **Korisno rukovanje pogreškama**: 404 stranice su dizajnirane kako bi pružile korisniku prijateljsko iskustvo kada nešto pođe po zlu. Umjesto da korisnicima prikažu kriptičnu

poruku o pogrešci ili praznu stranicu, dobro osmišljena 404 stranica može prenijeti da je došlo do problema na način koji je lako čitljiv i često humoristični ili informativan, čime se korisnik osjeća ugodnije i manje frustrirano.

2. **Poboljšano korisničko iskustvo**: Kada se 404 stranica pažljivo oblikuje, može pomoći korisnicima da se lako vrate na glavni sadržaj web mjesta ili pronađu alternativne resurse. To može poboljšati ukupno korisničko iskustvo i potaknuti posjetitelje da ostanu na web mjestu umjesto da odu zbog nepostojeće veze.

3. **Smanjenje stope odbijanja**: Odbijanje se događa kada korisnik posjeti web stranicu i odmah napusti bez daljnjeg interesiranja s aplikacijom. Pružanjem korisne 404 stranice možete smanjiti stope odbijanja jer su korisnici skloniji istraživanju drugih dijelova vašeg web mjesta ili pronalaženju sadržaja koji su tražili putem navigacijskih veza ili značajki pretrage.

4. **Zadržavanje korisnika**: Zanimljiva 404 stranica može zadržati interes korisnika i zadržati ih na vašem web mjestu čak i kada naiđu na pogrešku. To može pomoći u održavanju njihovog angažmana.



Slika 30: Pogled "Pogreška 404"

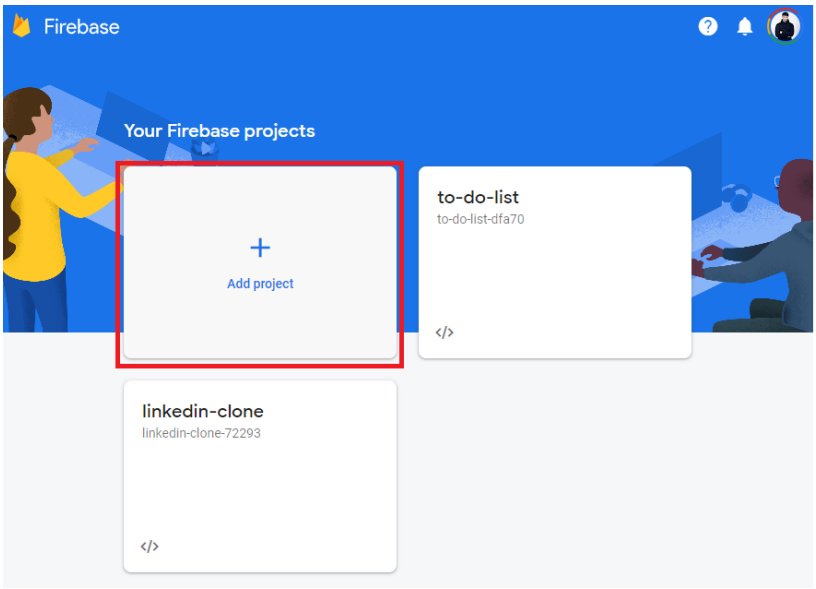
4. Postavljanje Firebase projekta

Prije same usporedbe Reacta i Angulara, u ovom poglavlju bit će opisano kako postaviti Firebase projekt. Iako React i Angular koriste različite pristupe i arhitekture za izradu web aplikacija, mogu koristiti istu Firebase aplikaciju za autentifikaciju i spremanje podataka u Firestore. Pretpostavlja se da korisnik već ima kreiran Firebase račun, te će biti opisani koraci kreiranja projekta bez opisa o kreiranju Firebase računa. Postavljanje Firebase projekta se može podijeliti u tri jednostavna koraka, a to su:

1. Stvaranje Firebase projekta
2. Konfiguracija Firebase projekta
3. Integracija Firebase projekta u React ili Angular

4.1. Kreiranje Firebase projekta

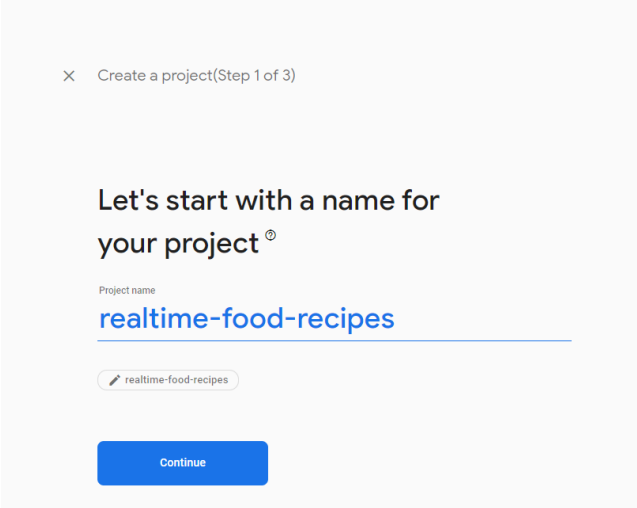
Dakle, pretpostavlja se da korisnik već ima postojeći Firebase račun. Za kreiranje projekta potrebno je navigirati u *Firebase konzolu*, a nakon otvaranja konzole korisniku će biti prikazan ekran kao na slici 31. Klikom na karticu označenom crvenim obrubom, otvara se pogled u kojem je potrebno unijeti naziv projekta kojeg kreiramo.



Slika 31: Firebase konzola

Sa slike 32, može se vidjeti da je projekt je nazvan *realtime-food-recipes*. Klikom na dugme *Continue* nastavlja se na sljedeće korake potrebne za kreiranje Firebase projekt u kojemu se pruža opcija korištenja *Google Analytics* u projektu. Ako korisnik odluči omogućiti

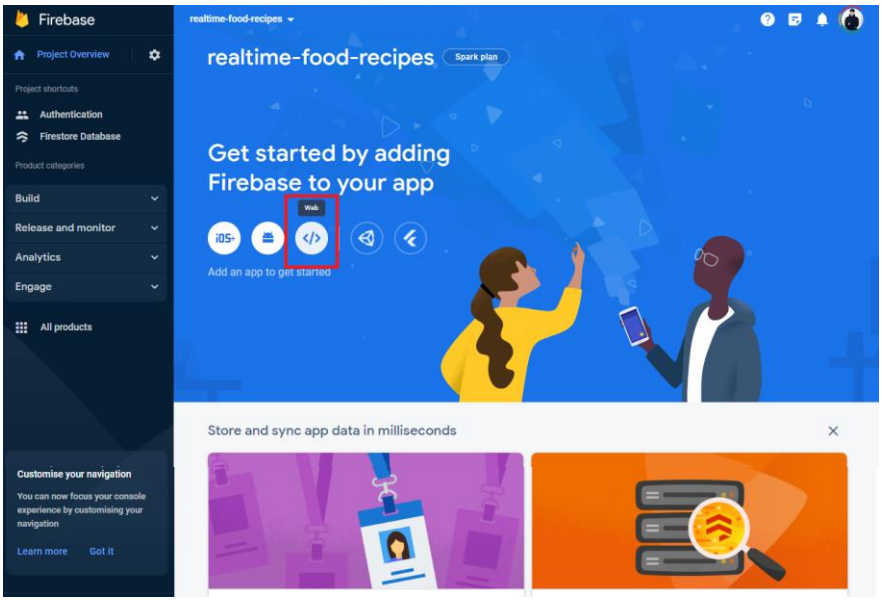
korištenje *Google Analytics* alata, bit će dostupne značajke koje u svrhu projekta ovog diplomskog rada nisu potrebne, tako da je u ovom koraku odabrana opcija bez korištenja *Google Analytics* alata. Klikom na dugme *Create project* kreira se novi projekt.



Slika 32: Kreiranje Firebase projekta 1/2

4.2. Konfiguracija Firebase projekta

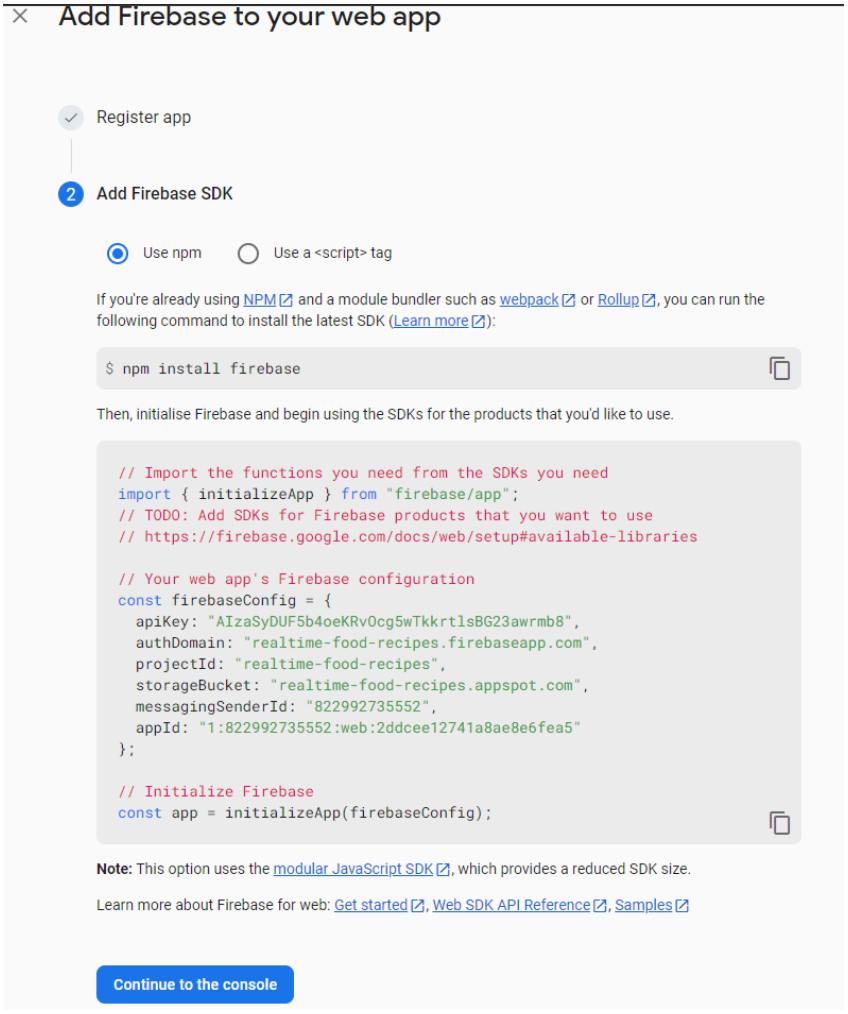
Nakon stvaranja projekta, prikazat će se pogled vidljiv na slici 33. Sljedeći korak bila bi konfiguracija Firebase projekta kako bi se Firebase projekt mogao integrirati u React ili Angular aplikaciju. Klikom miša na tipku označenu crvenim obrubom otvorit će se novi pogled u kojem je potrebno unijeti naziv naše aplikacije, i nakon unosa naziva potrebno je kliknuti na tipku *Register app*, ovim korakom se završava s konfiguracijom Firebase projekta.



Slika 33: Konfiguracija Firebase projekta

4.3. Integracija Firebasea u React ili Angular

Nakon konfiguracije Firebase projekta, Firebase je izgenerirao podatke koji će biti korišteni za pristup Firebase projektu koristeći React ili Angular aplikaciju. Sa slike 34 mogu se vidjeti daljnje upute koje su predložene sa strane Firebasea kako izvršiti integraciju. Ponuđene su dvije opcije, jedna koristeći *Node Package Manager* – NPM , a druga koristeći <script> tag pomoću koje bi se kreirao novi *modul* za integraciju sa Firebase-om. Potpuna integracija Firebase projekta u React i Angular aplikaciju bit će opisana u poglavlju ***5.11. Firebase API***.



Slika 34: Podaci za integraciju Firebase projekta s React i Angular aplikacijom

5. Komparativna analiza Reacta i Angulara

Ovo poglavlje fokusirat će se na usporedbu razvojnih okvira React i Angular. Usporedba će se provoditi na temelju identične aplikacije koja je zasebno implementirana i u Reactu i u Angularu. Bitno je istaknuti da React i Angular imaju različitosti u načinu kako pristupaju razvoju aplikacije te ih je teško staviti direktno jedno pored drugog i usporediti ih tako. Iz tog razloga potpoglavlja ovog poglavlja sadržavat će šturu strukturu gdje će se pomoću malih isječaka iz koda objasniti osnovni koncept funkcioniranja Angulara i Reacta iz kojih će se lako uočiti njihove sličnosti i razlike. Također, unutar poglavlja ***5.12. Primjer koda komponente RecipeCard*** bit će prikazan kod koji je zadužen za prikaz kartice recepta unutar okvira Angular i biblioteke React. Za početak, sama činjenica da React nije razvojni okvir nego biblioteka dovodi do prve razlike Reacta i Angulara koja je opisana u sljedećem poglavlju.

5.1. Razlika između biblioteke (React) i okvira (Angular)

Kao što je već napomenuto, React je biblioteka, a Angular je razvojni okvir, no što to zapravo znači? U smislu razvijanja aplikacije ovog diplomskog rada, to bi značilo da se uz biblioteku React moraju koristiti neke od dodatnih biblioteka kako bi finalna React aplikacija bila upotrebljiva, na primjer biblioteka za navigiranje i usmjeravanje između različitih pogleda. React u pravilu predstavlja biblioteku za razvoj korisničkog sučelja, a ne cijele aplikacije. No, već je postao standard koristiti React uz neke druge potrebne biblioteke kako bi se dobio cjelokupan set potreban za razvoj aplikacije od nule pa do finalne verzije. Dok u drugu ruku, korištenje Angulara za razvoj aplikacije ovog diplomskog rada podrazumijeva dostupnost svih značajki koje dolaze integrirane u Angularu, te cjelokupan set omogućava punu funkcionalnost aplikacije bez preuzimanja dodatnih biblioteka i slično. To bi značilo da od same instalacije Angulara programer dobiva sve ono što mu je potrebno za razvoj aplikacije od nule pa do finalne verzije, uključujući razvoj korisničkog sučelja, navigiranja i usmjeravanja i tako dalje.

5.1.1. Biblioteka

Biblioteka se može zamisliti kao skup unaprijed napisanog koda koji se može koristiti u svrhu olakšavanja nekog zadatka. Na primjer, React se može koristiti umjesto običnog JavaScripta za manipulaciju DOM-om na lakši i efikasniji način, uz pisanje manje linija koda. To je moguće jer uz React biblioteku dolaze razne predefinirane funkcije koje su na raspolaganju programerima u svrhu pojednostavljenja posla i ubrzanja procesa razvoja aplikacije. [25]

Na primjer, ako se razvija aplikacija koja zahtjeva implementaciju karusela (eng. *Carousel*), programeri imaju na raspolaganju dvije opcije, a to su:

1. Implementirati vlastiti kod za karusel
2. Iskoristiti postojeću biblioteku za karusel (npr. *react-responsive-carousel*)

Prva opcija bi zahtijevala od programera da samostalno implementiraju kompletan HTML, CSS i JS kod za karusel, uz to da moraju paziti i na responzivnost na različitim dimenzijama ekrana, funkcionalnosti karusela i slično. Kod druge opcije, programer bi trebao samo instalirati i uvesti (eng. *Import)* instaliranu biblioteku s gotovim funkcionalnostima i logikom, te s gotovim responzivnim dizajnom, a uz sve to ga programer može dodatno prilagoditi, ako je to uopće i potrebno, prema svom specifičnom slučaju korištenja promjenom nekoliko linija koda. [26]

5.1.2. Razvojni okvir

Razvojni okvir predstavlja temelj prema kojima programeri razvijaju aplikacije. To bi značilo da razvojni okvir osigurava cjelokupan eko-sistem (eng. *Eco-system*) i strukturu uz iskoristive dijelove koda kako bi se lakše pristupilo kompleksnijim izazovima. Drugim riječima, razvojni okvir pruža svoje okruženje oko kojeg programer razvija aplikaciju. To bi značilo da programeri nemaju totalno slobodnu o tome kako će razvijati aplikaciju, već je bitno pratiti neka od pravila koja dolaze uz određeni razvojni okvir. [26]

Razvojni okviri funkcioniraju na specifičnom uzorku dizajna, poput MVC (eng. *Model-View-Controller)* kojeg je pri razvoju aplikacije potrebno poštivati. Ako se ne poštuju principi određenog uzorka dizajna, vrlo je lako doći do pogrešne implementacije koje će na kraju voditi do nečistog koda s puno grešaka i smanjenih performansi. Razvojni okviri se sastoje od velikog broja programskih sučelja aplikacije (eng. *API – Application Programming Interface)*, skupova alata, kompajlera, programa za podršku, te čak i biblioteka. [25]

5.2. Struktura projekta i instalacija

Proces instalacije Reacta i Angulara veoma je jednostavan. Za početak potrebno je preuzeti *Node.js* i *npm* s njihove web stranice te ga instalirati na računalo. Nakon toga, za instalaciju Reacta potrebno je otvoriti konzolu i izvršiti sljedeću naredbu: `npm install -g create-react-app` [1]

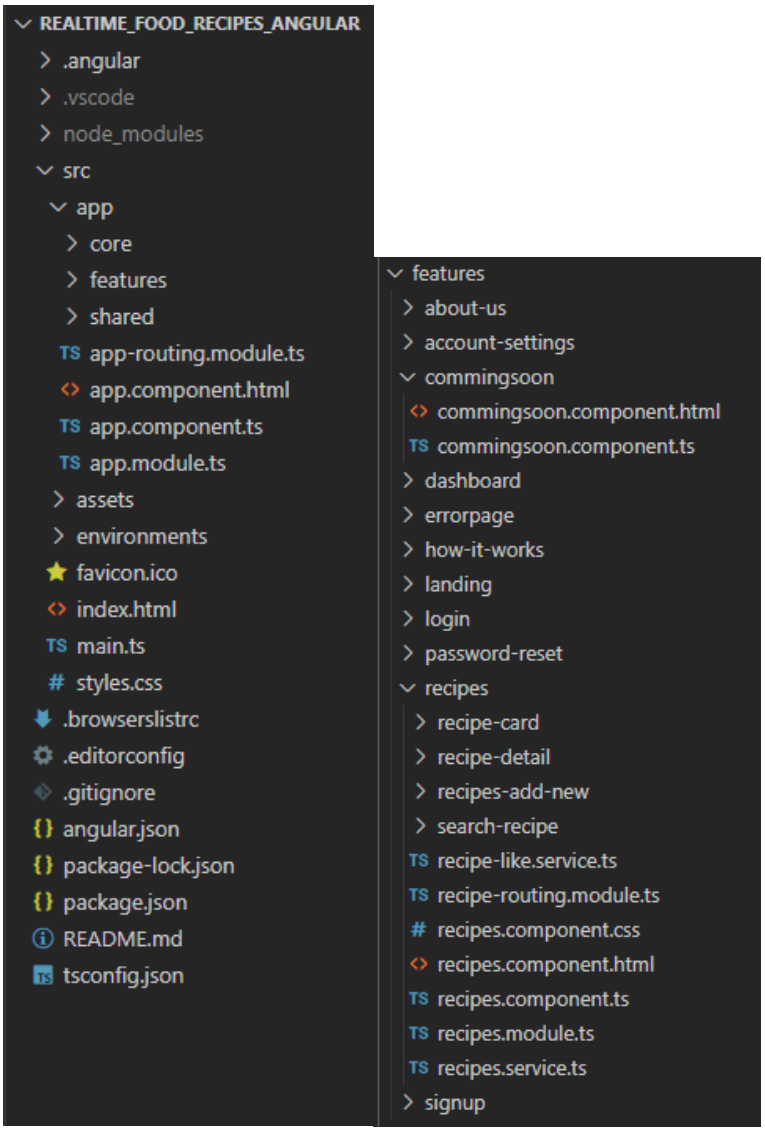
Za instalaciju Angulara, u konzoli je potrebno upisati sljedeću naredbu: `npm install -g @angular/cli` [2]

Nakon kreiranja projekta pomoću naredbe `ng new realtime-food-recipes` za Angular, i pomoću naredbe `npx create-react-app realtime-food-recipes` za React, generira se početna struktura mapa i datoteka. Na slikama 35 i 36 prikazane su finalne strukture projekta za React i Angular koje u odnosu na početnu strukturu imaju puno više dodatnog sadržaja.

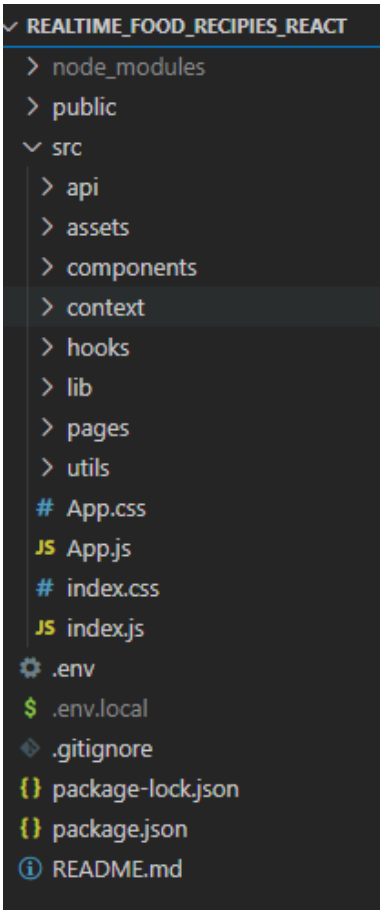
Gledajući strukturu React i Angular projekta možemo vidjeti da imaju neke sličnosti, ali i dosta različitosti. Razlika se može primijetiti u samoj organizaciji mapa, a glavni razlog tome je činjenica da se Angular bazira na modularnoj arhitekturi pomoću koje se aplikacija razdvaja na više odvojenih dijelova te se time postiže odvajanje odgovornost. Ovakva arhitektura se smatra standardom i ideja je ta da mora postojati korijenski (eng. *Root)* modul koji unutar sebe sadrži i ostale module koji se dodatno mogu dinamički učitati, više o dinamičkom učitavanju modula u poglavlju *Usmjeravanje i navigacija*. Ako se radi na nekom manjem projektu, možda neće ni biti potrebe za odvajanje aplikacije na module, te je dovoljno imati aplikaciju unutar *src* mape. No, to nije slučaj aplikacije ovog diplomskog rada, te je ova aplikacija podijeljena u više modula. Iz tog razloga dodatno su kreirane mape *features* unutar koje se nalaze mape svih pogleda naše aplikacije i modula, vidi sliku 35. [2] Sa slike 35 također se može primijetiti da mapa *recipes* posjeduje puno više datoteka nego na primjer mapa *commingsoon*. Razlog tome je taj što se mapa *recipes* smatra odvojenim modulom cjelokupne aplikacije kojoj će biti svrha pružiti specifične funkcionalnosti korisniku aplikacije vezane uz recepte, na primjer prikaz recepata u što spadaju specifične komponente komponenti pa i cijeli pogledi, npr. pogled *Lista recepata* ili pogled *Detalji recepta*. Dakle u Angular projektu mogu se naći sljedeće mape: [27]

- **core –** datoteke za ključne značajke aplikacije, a u njih spadaju servisi za autorizaciju, autorizacijski čuvari (eng. *Guards)* i temeljni modul (eng. *Core module)*
- **features –** mape i datoteke svih pogleda aplikacije, uz što mogu spadati i zasebni moduli kao što je modul *recipes* u slučaju ove aplikacije
- **shared -** sadrži sve one stvari koje su korištene od cijele aplikacije, na primjer servisi i komponente
- **assets –** statičke datoteke - slike, ikone, predefinirani scss stilovi
- **environments –** sadrži osjetljive podatke kao što su informacije o bazi podataka (API ključevi, domena baze, ime baze i tako dalje)

Angular projekt također sadrži i *tsconfig.json* koja sadrži konfiguraciju o tome kako transpilirati (eng. *Transpiling)* TypeScript kod u JavaScript kod.



Slika 35: Angular struktura projekta



Slika 36: React struktura projekta

Može se reći da se React s druge strane temelji na arhitekturi komponenti, gdje komponenta predstavlja jedan dio aplikacije unutar određenog pogleda. Ako se gleda slika 36, može se primijetiti da React aplikacija posjeduje sve elemente aplikacije unutar prve razine *src* mape. Dakle, u React projektu *src* mapa se sastoji od sljedećih mapa:

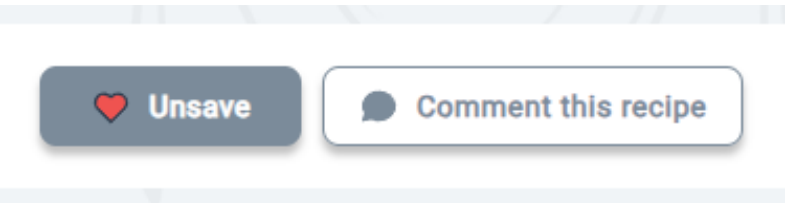
- **api** – integracija Firebasea i logika za poziv na bazu podataka
- **assets** – slike, ikone, predefinirani scss stilovi
- **components** – komponente koje se koriste unutar aplikacije (*Button*, *Chip*, ...)
- **context** – spremanje globalnog stanja aplikacije
- **hooks** – funkcije za upravljanje stanjem aplikacije
- **lib** – spremanje konstanti, globalni stilovi
- **pages** – pogledi aplikacije
- **utils** – spremanje ponovno iskoristivih funkcija

Osim već spomenutih mapa, također je vrijedno istaknuti i neke datoteke koje se mogu smatrati važnijima i nalaze se u korijenskoj mapi oba projekta. Datoteke i mape zajedničke i Reactu i Angularu su:

- **package.json** – sadrži informacije za sve potrebne ovisnosti i pakete potrebne za rad aplikacije
- **package-lock.json** – automatski generirana datoteka koja se generira nakon završetka instalacije svih ovisnosti od strane npm klijenta. Sadrži detalje i informacije svih instaliranih ovisnosti i paketa koje se koriste za rad aplikacije
- **node_modules/ -** aplikaciji pruža sve potrebne npm pakete za njen rad
- **.gitignore –** specificira koje datoteke Git treba zanemariti
- **README.md –** uvodna dokumentacija za aplikaciju

5.3. Komponente i njihova organizacija

Komponente se mogu smatrati glavnim značajkama i React i Angular aplikacija pomoću kojih se gradi korisničko sučelje aplikacije. Pomoću komponenti, korisničko sučelje se može razdvojiti u neovisne i ponovno iskoristive dijelove aplikacije. Primjer komponente koja se koristi u aplikaciji ovog diplomskog rada je *Button* komponenta. Komponenta *Button* koristi se, odnosno vidljiva je u gotovo svakom pogledu aplikacije, od početne stranice, prijave, registracije pa do liste recepata. Važno je istaknuti da je npr. *Button* komponenta izgledom i funkcionalnostima u svakom korištenom pogledu identična, od njene boje, obruba, razmaka, visine i tako dalje. [1] To u pravilu i je ideja koja stoji iza korištenja arhitekture komponenti, da se komponenta kreira jednom sa svim potrebnim dizajnom i funkcionalnostima, te da se kasnije poziva po potrebi uz zadržavanje dizajna i funkcionalnosti. Time se uvelike smanjuje napisan kod i mogućnost pogreške, te se postiže konzistentnost i ponovna iskoristivost. Na slici 37 mogu se vidjeti dvije *Button* komponente. *Button* komponenta je implementirana tako da programer može definirati varijaciju komponente. U primjeru sa slike 37 lijeva *Button* komponenta je varijacije *isPrimary*, dok je druga komponenta varijacije *isTertiary*.

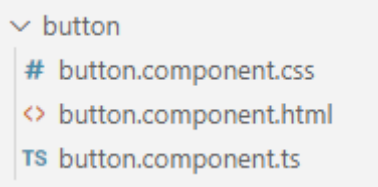


Slika 37: "Button" komponente unutar aplikacije

5.3.1. Angular komponente i sintaksa

Komponente se u Angular aplikaciji sastoje od tri glavne datoteke, pri čemu svaka datoteka ima svoju svrhu:

1. TypeScript klasa koja definira ponašanje komponente
2. HTML predložak koji definira što će se prikazati na stranici
3. CSS selektor koji definira izgled komponent



Slika 38: Datoteke za Angular komponentu "Button"

Unutar *button.component.ts* datoteke, vidljive sa slike 38, definirana je cijela *ButtonComponent* komponenta. U sljedećem isječku koda prikazan je kompletan sadržaj *button.component*.ts datoteke, te je lako uočljiva anotacija ***@Component*** koja se nalazi odmah iznad definiranja klase *ButtonComponent*. Ova anotacija Angularu označava da prvu klasu ispod treba interpretirati kao komponentu i tako joj dodijeliti posebna svojstva komponente. Unutar anotacije *@Componen*t definirana su dodatna svojstva: [2]

- **selector** – upućuje Angularu da instancira ovu komponentu gdje god pronađe odgovarajuću oznaku u HTML predlošku unutar cijele aplikacije
- **templateUrl** – relativna putanja do HTML datoteke povezanu s ovom komponentom
- **styleUrls –** lista relativnih putanja do CSS datoteka povezanih s ovom komponentom

button.componnet.ts:

```
import { Component, Input } from '@angular/core';
@Component({
  selector: 'app-button',
  templateUrl: './button.component.html',
  styleUrls: ['./button.component.css'],
})
export class ButtonComponent {…}
```

Međutim, osim *button.component.ts* datoteke koja je zadužena za logiku komponentne *ButtonComponent*, potrebno je i implementirati HTML predložak *button.component.html* koji će biti prikazana u pregledniku ako se komponenta *Button* pozove unutar koda. Isječak koda ispod prikazuje HTML predložak komponente *ButtonComponent*.

button.component.html:

```
  <button
    [disabled]="disabledProp"
    class="Button"
    [ngClass]="{
      'isTertiary': isTertiary,
    }"
 > <ng-content></ng-content>
</button>
```

5.3.2. React komponente i sintaksa

JSX predstavlja sintaktičko proširenje za JavaScript koje programerima omogućava korištenje HTML oznaka unutar JavaScript datoteke. Aplikacije i internetske stranice se implementiraju koristeći HTML, CSS i JavaScript, i godinama unazad koristila se praksa odvajanja sadržaja u HTML datoteku, stilova za sadržaj u CSS datoteku i logike u JavaScript datoteku. No, dolaskom modernijih web stranica koje zahtijevaju kompleksnije sadržaje, animacije i interakcije, logika je postala dio koji je najbitniji u određivanju sadržaja stranice. Može se reći da JavaScript upravlja HTML sadržajem, te iz tog razloga u React aplikacijama logika prikazivanja i HTML oznake žive zajedno u istom mjestu, to jest u komponentama. Svaka React komponenta je JavaScript funkcija koja može sadržavati HTML oznake koje React prikazuje u pregledniku, na ekranu korisnika. React komponente koriste sintaksu koja se naziva JSX, objašnjena u odlomku iznad. JSX izgledom liči na HTML, no zapravo sadrži dodatna pravila u odnosu na HTML i može prikazivati dinamičan sadržaj. [1] [28]

Dakle, komponente se u Reactu sastoje od jedne glavne datoteke, a to je *.jsx* datoteka unutar koje se nalazi sav sadržaj potreban za prikaz komponente, vidi sliku 39. Uz to, dobra je praksa odvojiti stilove u zasebnu datoteku kako se ne bi gomilao kod sa stilovima unutar jedne datoteke.

Slika 39: Datoteke za React komponentu "Button"

Button.jsx

```
import React from "react";
import { Button as ButtonWrapper } from "./ButtonStyle";

const Button = ({…})

export default Button;
```

5.4. Slanje podataka komponenti djetetu

Komponenta *Button* je implementirana tako da može imati više stilova ovisno o tome koje će podatke programer proslijediti komponenti pri pozivanju komponente. Ovi podaci nazivaju se svojstva (eng. *Properties)* komponente.

5.4.1. *@Input* dekorator (Angular)

Specifično u ovom primjeru, komponenta *ButtonComponent* može od svog roditelja zaprimiti sveukupno dva različitih svojstva. ***@Input()*** dekorator unutar komponente djeteta označava da specifično svojstvo može poprimiti vrijednost od njegovog roditelja. [2]

TypeScript klasa komponente (*button.component.ts)*:

```
export class ButtonComponent {
  @Input() disabledProp?: boolean = false;
  @Input() isTertiary?: boolean = false;
  …
}
```

Svojstva klase *ButtonComponent* mogu se koristiti unutar HTML predloška komponente, te je u isječku koda ispod vidljivo kako se svojstvo *isTertiary* koje dolazi od komponente *ButtonComponent* koristi kao svojstvo HTML elementa <*button*> unutar HTML predloška. Na primjer, ukoliko programer odluči iz komponente roditelj pozvati komponentu *ButtonComponent*, može joj dodijeliti svojstvo i*sTertiary=true* koje će se direktno koristiti kao svojstvo *[ngClass]* HTML elementa <*button*>. *[ngClass]* predstavlja Angular direktivu koja omogućava dinamičko postavljanje CSS klasa na HTML element te će se prema tome elementu <*button*> dodijeliti nova CSS klasa naziva *isTertiary*. Ova klasa odnosi se na stil komponente, te će se sukladno tome primijeniti stilovi za prikaz tercijarnog gumba na ekran korisnika. Na isti način HTML element <*button*> može primiti i ostala svojstva, čime se postiže dinamičnost. Na primjer, možda se želi onemogućiti klik na dugme ako validacija polja nije uspješna, sve što je potrebno učiniti kod poziva komponente *ButtonComponent* je proslijediti joj svojstvo *disabledProp* koje će biti *true* ili *false* ovisno o rezultatu validacije. Sukladno tome gumb će biti omogućen ili onemogućen. Bitno je istaknuti da element <*ng-content*> specificira

gdje će proslijeđeni sadržaj biti prikazan unutar predloška komponente. Proslijeđeni sadržaj predstavlja dijete elementa *ButtonComponent*.

HTML predložak komponente: (*button.component.html)*

```
  <button
    [disabled]="disabledProp"
    class="Button"
    [ngClass]="{
      'isTertiary': isTertiary,
    }"
 > <ng-content></ng-content>
</button>
```

5.4.2. Svojstva (React)

Slanje podataka komponenti djetetu unutar Reacta funkcionira slično kao i kod Angulara. Komponenta *Button* koja je definirana kao funkcija prima argumente koji definiraju komponentu *Button*, i ti argumenti zove se *props*, skraćeno od svojstva (eng. *Properites)*. *Ova* svojstva predstavljaju podatke koji dolaze od roditelja. Isto tako, ako se komponenti *Button* proslijedi svojstvo *disabled=true*, gumb će biti onemogućen, a ako joj se proslijedi svojstvo *isTertiary=true* komponenti *Button* bit će promijenjeni stilovi koji su prilagođeni za prikaz tercijarnog gumba. [1]

Button.jsx

```
import React from "react";
import { Button as ButtonWrapper } from "./ButtonStyle";

const Button = ({disabled=false, isTertiary=false, children}) => {
  return (
    <ButtonWrapper
      isTertiary={isTertiary}
      disabled={disabled}
    >
      {children}
    </ButtonWrapper>
  );
}

export default Button;
```

5.5. Pozivanje komponenti

Kako bi sve kreirane komponente bile vidljive na korisničkom sučelju unutar preglednika moraju biti dio HTML strukture stranice. Dakle, komponenta *Button* se unutar React ili Angular aplikacije mora pozvati kako bila prikazana na ekranu korisnika kao na slici 37.

5.5.1. Angular sintaksa pozivanja

Pozivanje komponente *ButtonsComponent* u Angular aplikaciji prikazano je u isječku koda ispod. Komponenta *ButtonComponent* se poziva koristeći HTML selektor <*app-button*> </*app-button*, ovaj selektor definiran je unutar *@Component* dekoratora koji je objašnjen u prijašnjim poglavljima. Može se primijetiti da se pri pozivu komponente prosljeđuje svojstvo *isTertiary* koje komponenta <*app-button*> koristi za promjenu stilova gumba prikazanog u aplikaciji. [2]

```
…
  <app-button [isTertiary]="true">
    Comment this recipe
  </app-button>
…
```

5.5.2. React sintaksa pozivanja

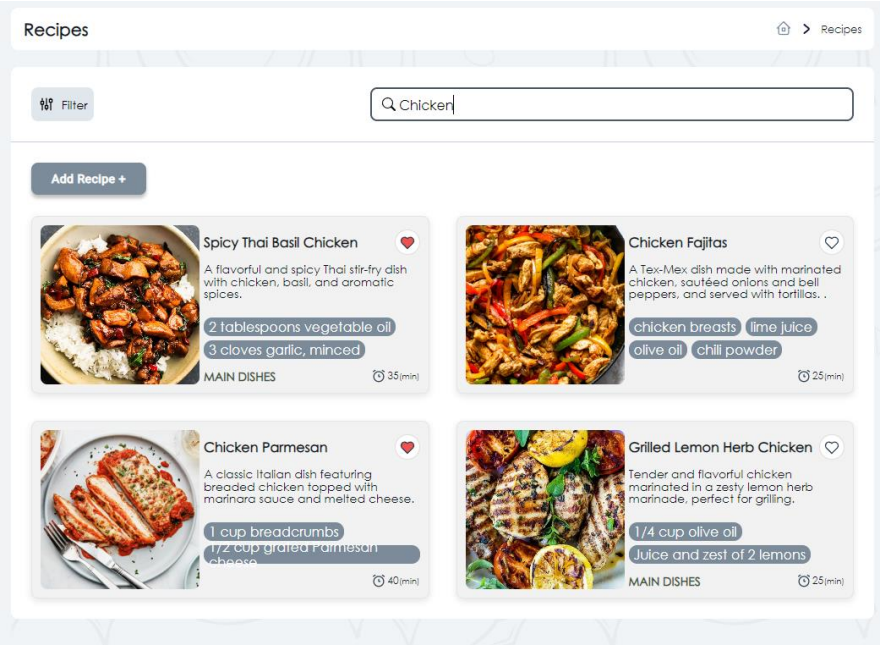
Slično kao i kod Angulara, ako želimo prikazati komponentu *Button* moramo je pozvati unutar koda. No u slučaju kod Reacta pozivamo je pomoću selektora <*Button*> </*Button*>. Kao i u Angular primjeru, komponenti <*Button*> proslijeđeno je svojstvo *isTertiary* koje će promijeniti stilove gumba prikazanog na ekranu. [29]

```
…
  <Button
    isTertiary
  >
    Comment this recipe
  </Button>
…
```

5.6. Vezanje podataka

Vezanje podataka (eng. *Data binding*) predstavlja ključni koncept u modernim web okvirima i bibliotekama kao što su React i Angular. Pomoću vezivanja podataka omogućuje se automatsko ažuriranje korisničkog sučelja u slučaju promjene podataka što na kraju olakšava interakciju između korisnika i aplikacije. Iako React i Angular dijele neke sličnosti u vezi s vezanjem podataka, imaju različite pristupe i metode. Vezanje podataka između Reacta i Angulara bit će opisano na temelju funkcionalnosti pretrage recepta, vidi sliku 40.

React koristi jednosmjerno vezanje podataka što bi značilo da podaci mogu putovati samo u jednom smjeru, od roditelja prema djetetu, dok s druge strane Angular koristi dvosmjerno vezanje podataka gdje podaci mogu putovati u oba smjera, čak i od djeteta prema roditelju. [30]



Slika 40: Vezanje podataka - pretraživanje recepata

5.6.1. Dvosmjerno vezanje podataka (Angular)

Pomoću dvosmjernog vezanja podataka olakšava se dijeljenje podataka između komponenti unutar aplikacije. Koristeći dvostruko vezanje podataka mogu se istovremeno pratiti događaji i ažurirati vrijednosti između roditeljskih i dječjih komponenti, a to se postiže kombiniranjem:

- Povezivanje svojstva – postavlja određeno svojstvo elementa
- Povezivanje događaja – praćenje promjena događaja elementa

@Output predstavlja dekorator pomoću kojeg se proslijeđeno svojstvo automatski ažurira ako se detektira promjena. Ovaj dekorator označava svojstvo djeteta kao „vrata“ preko kojih podaci mogu putovati od djeteta prema roditelju. Komponenta dijete koristi klasu *EventEmitter* koja je dio *@angular/core* biblioteke. [2] [30]

U ovom primjeru bit će prikazano kako postaviti *@Output* dekorator *searchRecipe*. Svojstvo *searchRecipe* predstavlja listu pretraženih recepata, odnosno listu recepata čije ime ili opis koji odgovaraju korisničkom unosu pretraživanog teksta.

Komponenta dijete – *search-recipe-component.ts*

```
import { Component, EventEmitter, Input, Output } from '@angular/core';

@Component({
```

```
  selector: 'app-search-recipe',
  templateUrl: './search-recipe.component.html',
  styleUrls: ['./search-recipe.component.css',
'../../../shared/shared.styles.css']
})
export class SearchRecipeComponent {
  @Input() filteredRecipes!: any;
  @Output() searchRecipe = new EventEmitter<any>();

  handleSearchInput = (e) => {
    const searchValue = e.target.value.toLowerCase();
    if (searchValue === "") {
      this.searchRecipe.emit(this.filteredRecipes)
    }
    const searchRecipeChanged = this.filteredRecipes.filter(
      (recipes) =>
        recipes.name.toLowerCase().includes(searchValue) ||
        recipes.description.toLowerCase().includes(searchValue)
    );

    this.searchRecipe.emit(searchRecipeChanged);
  };
}
```

Komponenta dijete se sastoji od jednog HTML elementa <input> kojem su pridružena razna svojstva, a svojstvo koje je bitno za ovaj primjer je događaj „(keyup)“. Svaki put kada korisnik pritisne i pusti dugme na tipkovnici, desi se događaj *(keyup)* koji poziva metodu *handleSearchInput*. [2]

Komponenta dijete – *search-recipe-component.html*

```
<input
  headingElement
  type="text"
  id="searchBar"
  class="FieldStyleType1 SearchBar"
  placeholder="Search name or description"
  (keyup)="handleSearchInput($event)"
/>
```

Unutar predloška komponente roditelj poziva se a*pp-search-recipe* komponenta te joj se prosljeđuju svojstvo *filteredRecipes* te povezuje događaj *searchRecipe* = *$event*. Varijabla *$event* sadrži podatke od događaja *SearchRecipe*.*searchRecipe*. Angular dodjeljuje vrijednost varijable *$event* varijabli *RecipesComponent*.*searchRecipe* svaki put kada korisnik pritisne i otpusti dugme na tipkovnici. [2]

Komponenta roditelj – *recipes-component.html*

```
    <app-search-recipe
      headingElement
      [filteredRecipes]="filteredRecipes"
      (searchRecipe)="searchRecipe = $event"
    ></app-search-recipe>
```

5.6.2. Jednosmjerno vezanje podataka (React)

React se za razliku od Angulara oslanja samo na jednosmjerno vezanje podataka, što bi značilo da se podaci prenose kroz komponente od roditelja prema djeci. No, gledajući primjer sa slike 40, što u slučaju da komponenta dijete treba promijeniti podatke koji se nalaze u roditeljskoj komponenti?

U slučaju kao što je ovaj, dobra je praksa koristiti takozvane funkcije uzvratnog poziva (eng. *Callback functions)* koje će biti proslijeđene kao svojstvo od komponente roditelj prema komponenti dijete. U ovom primjeru, potrebno je ažurirati podatke iz roditeljske komponente svaki put kada se pritisne tipka na tipkovnici unutar komponente dijete, pa će se unutar komponente dijete implementirati funkcionalnost u kojoj će se proslijeđena *callback funkcija* pozivati svaki put kada korisnik pritisne tipku tipkovnice. [1]

Komponenta dijete – *RecipesSearchBar.jsx*

```
import React from 'react';
import { SearchBar } from '../../lib/style/generalStyles';

const RecipesSearchBar = ({ callback }) => {
    return (
        <SearchBar
            placeholder='Search name or description'
            onChange={callback}
        />
    );
};
export default RecipesSearchBar;
```

Nakon implementacije komponente dijete, potrebno je unutar komponente roditelj pozvati komponentu dijete i proslijediti joj odgovarajuću funkciju koja će ažurirati podatke u komponenti roditelj. Naziv te funkcije je *handleSearchInput* i ona će u obliku svojstva biti proslijeđena komponenti dijete *RecipeSearchBar.jsx* kao *callback funkcija*.

Komponenta roditelj – *Recipes.jsx*

```
const Recipes = () => {
    const [searchRecipe, setSearchRecipe] = useState([]);
    const [filteredRecipes, setFilteredRecipes] = useState([]);

    const handleSearchInput = (e) => {
        const searchValue = e.target.value.toLowerCase();
        if (searchValue === "") {
            setSearchRecipe(filteredRecipes);
        }
        setSearchRecipe(
            filteredRecipes.filter(
                (recipes) =>
                    recipes.name.toLowerCase().includes(searchValue) ||
                    recipes.description.toLowerCase().includes(searchValue)
            )
        );
```

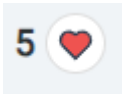
```
    };

    return (
        <Layout title="Recipes">
            <RecipesSearchBar
                callback={handleSearchInput}
            />
        </Layout>
    );
};
export default Recipes;
```

Činjenica da React ne podržava dvosmjerno vezivanje podataka ne znači da se ne može implementirati ažuriranje podataka roditeljske komponente pomoću događaja unutar komponente dijete i proslijeđene *callback funkcije* od komponente roditelj do komponente dijete.

5.7. Upravljanje podacima

Upravljanje podacima jedan je od važnijih sastava u razvoju modernih web aplikacija, jer ovisno o podacima i stanju aplikacije, korisničko iskustvo može biti izuzetno fluidno i intuitivno ili u drugu ruku naporno i frustrirajuće. Razliku upravljanja podataka između Angulara i Reacta bit će objašnjena na primjeru komponente koja je zadužena za označavanje recepta s oznakom *sviđa mi se*, vidi sliku 41.



Slika 41: Komponenta za označavanje recepta s oznakom "sviđa mi se"

5.7.1. Upravljanje podacima u Angularu

Servisi predstavljaju instancu klase koja može biti dostupna bilo kojem dijelu Angular aplikacije. Na primjer, potrebno je implementirati mogućnost gdje korisnik može označiti recept sa *sviđa mi se* (eng. *Like).* Kako bi ove informacije unutar korisničkog sučelja bile spremljene, potrebno je implementirati servis koji može spremiti i pružiti informacije kada je to potrebno. Servis naziva *RecipeLikeService* pratit će koliko specifičan recept ima oznaka *sviđa mi se* i je li prijavljen korisnik označio recept sa *sviđa mi se*.

Ove informacije bit će spremljene unutar varijable *isLikedByUser* i *recipeLikes*. Pomoću metoda *getIsLikedByUser* i *getRecipeLikes* može se pristupiti podacima varijabli. Metoda *handleLikeRecipe* zadužena je za označavanja recepta s oznakom *sviđa mi se/ne sviđa mi se*. [28]

recipes-like-service.ts:

```
import { Injectable } from '@angular/core';

@Injectable()
export class RecipeLikeService {
  private isLikedByUser = false;
  private recipeLikes = 0;

  init(recipe: any, userData: any) {
    this.isLikedByUser = recipe?.likedBy?.includes(userData.uid) || false;
    this.recipeLikes = recipe?.likedBy?.length || 0;
  }

  getIsLikedByUser(): boolean {
    return this.isLikedByUser;
  }

  getRecipeLikes(): number {
    return this.recipeLikes;
  }
  async handleLikeRecipe(isLikedByUser: boolean) {
      this.isLikedByUser = !isLikedByUser;
      if(isLikedByUser)
        this.recipeLikes -= 1;
      else
        this.recipeLikes += 1;
  }
}
```

Sada kada je implementiran servis za spremanje podataka o tome je li korisnik označio recept sa *sviđa mi se* te sveukupan broj oznaka *sviđa mi se* pojedinog recepta, možemo ga koristiti unutar komponenti. U sljedećem isječku koda prikazan je primjer komponente *RecipeCardComponent* unutar kojeg su inicijalizirani lokalne varijable *isLikedByUser* i *recipeLikes*. Pomoću Angular metode *ngOnInit* , komponenta inicijalizira *recipeLikeService* s podacima o receptu i korisniku te također od servisa *recipeLikeService* dobavlja podatke o tome je li korisnik već označio ovaj recept sa *sviđa mi se* i ukupan broj oznaka *sviđa mi se* ovog recepta. [2]

Recipe-card.component.ts

```
import { Component,  OnInit } from '@angular/core';
import { AuthService } from 'src/app/core/auth.service';
import { RecipeLikeService } from '../recipe-like.service';

@Component({
  selector: 'app-recipe-card',
  templateUrl: './recipe-card.component.html',
  styleUrls: ['./recipe-card.component.css']
})
export class RecipeCardComponent implements OnInit {
  isLikedByUser: boolean;
  recipeLikes: number;
  userData = null;
```

```
  constructor(
    private recipeLikeService: RecipeLikeService,
    private authService: AuthService,
  ) {
    this.userData = this.authService.userData;
  }

  ngOnInit() {
    // Initialize the service with recipe and user data
    this.recipeLikeService.init(this.recipe, this.userData);
    this.isLikedByUser = this.recipeLikeService.getIsLikedByUser();
    this.recipeLikes = this.recipeLikeService.getRecipeLikes();
  }

  handleFavoriteClick() {
    this.recipeLikeService.handleLikeRecipe(this.isLikedByUser);
    this.isLikedByUser = this.recipeLikeService.getIsLikedByUser();
    this.recipeLikes = this.recipeLikeService.getRecipeLikes();
  }
}
```

Metoda *handleFavoriteClick* poziva se kada korisnik klikne na ikonu omiljenog recepta. Ova metoda poziva odgovarajuću metodu unutar servisa *recipeLikeService* kako bi ažurirala stanje *lajka* za trenutnog korisnika i ukupan broj *lajkova* za recept. Nakon toga ažurira lokalne varijable *isLikedByUser* i *recipeLikes* kako bi se promjene prikazale na korisničkom sučelju.

recipe-card.component.html

```
<div class="FavoriteIconWrapper">
  <p>{{recipeLikes}}<p>
  <svg-icon
    [class]="isLikedByUser ? 'AddFavorite isFavorite' : 'AddFavorite'"
    [applyClass]="true"
    src="../../../../assets/img/add-favorite.svg"
    (click)="handleFavoriteClick()"
  >
  </svg-icon>
</div>
```

5.7.2. Upravljanje podacima u Reactu

Unutar React aplikacije ovog diplomskog rada za upravljanje podacima korišten je *Context API* za globalno spremanje podataka i *useState* kuka za lokalno spremanje podataka. *Context API* predstavlja značajku koja omogućava spremanje podataka kojima se može pristupiti unutar aplikacije. Generalno se *Context API* koristi kako bi se izbjeglo prosljeđivanje svojstva od roditeljske komponente kroz nekoliko komponenti dijete. *useState* predstavlja funkciju koja lokalno sprema podatke unutar varijable neke komponente. [1] [31]

Za implementaciju funkcionalnost označavanja recepta sa *sviđa mi se*, koristit će se kuka *useRecipeLike*. Kuka prima dva argumenta, a to su *recipe* i *userData* te se prema tome

određuje vrijednost stanja *isLikedByUser* i stanja *recipeLikes*. Nakon toga, definirana je *handleLikeRecipe* metoda koja se koristi za promjenu statusa *lajkanja* i broja *lajkova* recepta. Na kraju, *useRecipeLike* kuka vraća objekt s tri vrijednosti: *isLikedByUser*, *recipeLikes* i *handleLikeRecipe* koje će moći biti korištene unutar komponente gdje je pozvana *useRecipeLike* kuka.

useRecipeLike.js:

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

export const useRecipeLike = (recipe, userData) => {
    const [isLikedByUser, setIsLikedByUser] = useState(false);
    const [recipeLikes, setRecipeLikes] = useState(0)

    useMemo(() => {
        setIsLikedByUser(recipe?.likedBy?.includes(userData.uid) || false);
        setRecipeLikes(recipe?.likedBy?.length || 0)
    }, [recipe?.likedBy, userData?.uid]);

    const handleLikeRecipe = () => {
        setIsLikedByUser(!isLikedByUser);
        setRecipeLikes((prev) => isLikedByUser ? prev - 1 : prev + 1)
    };

    return { isLikedByUser, recipeLikes, handleLikeRecipe };
};
```

Sljedeći kod predstavlja komponentu *RecipeCard* unutar koje će biti korištena kuka *useRecipeLike* kako bi se upravljalo oznakom ***sviđa mi se*** nekog recepta. Unutar komponente, dobavljaju se tri vrijednosti putem *useRecipeLike* kuke, a to su stanja *isLikedByUser*, *recipeLikes* i funkcija *handleLikeRecipe*. Sučelje komponente sadrži ikonu u kojoj je definiran događaj *onClick* kojemu je pridružena funkcija *handleLikeRecipe*. Broj lajkova prikazan je unutar sučelja komponente koristeći vitičaste zagrade *{ recipeLikes }.*

RecipeCard.jsx:

```
import React, { useContext } from "react";
import { AuthContext } from "../../context/AuthContext";
import { useRecipeLike } from "../../hooks/useRecipeLike";
import {
  AddFavorite,
  FavoriteIconWrapper,
  LikesNumber,
} from "../../components/RecipeCard/RecipeCardStyle";

const RecipeCard = () => {
  const { userData } = useContext(AuthContext);
  const { isLikedByUser, recipeLikes, handleLikeRecipe } = useRecipeLike(
    recipe, userData);

  return (
      <FavoriteIconWrapper>
      <LikesNumber>{recipeLikes}</LikesNumber>
```

```
        <AddFavorite
          onClick={handleLikeRecipe}
          isfavorite={isLikedByUser}
        />
      </FavoriteIconWrapper>
  );
};

export default RecipeCard;
```

5.8. Usmjeravanje i navigacija

Nakon implementacije svih potrebnih komponenti programeri ih koriste kako bi pomoću njih implementirali kompletan pogled aplikacije. U slučaju ovog diplomskog rada, potrebno je implementirati više pogleda, a neki od njih su pogled *Odredišna stranica, Prijava, Registracija, Zaboravljena lozinka, Nadzorna ploča* i tako dalje. No, iako su uspješno implementirani svi pogledi, potrebno ih je dodatno povezati i svakom pogledu dodijeliti web-lokaciju (eng. *URL – Unifrom Resource Locator)* na kojoj će biti prikazan. Na ovaj se način postiže učitavanja onog pogleda na specifičnoj web-lokaciji, te cjelokupna aplikacija postaje smislenija i funkcionalnija.

Kako bi se postiglo dodjeljivanje specifične web-lokacije svakom pogledu, Angular koristi svoje gotovo rješenje, odnosno servis usmjernik (eng. *Router)* koji je dio biblioteke ***@angular/router***. [2] S druge strane, kao što je objašnjeno u prijašnjim poglavljima, programeri Reacta primorani su potražiti biblioteke treće strane (eng. *Third party)* kako bi implementirali funkcionalnost usmjeravanja unutar React aplikacije. U izradi ovog diplomskog rada korišten je biblioteka *React Router DOM* u svrhu rješavanja problema s usmjeravanjem i navigacijom kroz aplikaciju. *React Router DOM* je široko poznata biblioteka koju je unutar React projekta moguće instalirati naredbom ***npm install react-router-dom*.** [32]

Aplikacija ovog diplomskog rada osmišljena je tako da korisnicima koji nisu prijavljeni u aplikaciju onemogući korištene web-lokacija na kojoj se nalaze specifični pogledi, na primjer pogled *Lista svih recepata* ili pogled *Nadzorna ploča*. U izradi ove značajke bit će korišten usmjernik.

5.8.1. Usmjeravanje i navigiranje u Angularu

Kao što je već objašnjeno u prijašnjim poglavljima, Angular funkcionira na temelju modula, te kako bi implementirali funkcionalnost usmjeravanja unutar Angular aplikacije, potrebno je implementirati modul usmjeravanja. U sljedećem isječku koda prikazan je modul usmjeravanja naziva *AppRoutingModule* koji, ukoliko bi funkcionirao, se mora dodati u *import* listu unutar korijenskog modula *AppModule*.

Za implementaciju usmjeravanja, potrebno je koristiti *Router* i registrirati *RouterModule* iz ***@angular/router*** paketa. Nakon toga definira se lista web-lokacija, *routes*, koja se proslijedi u *RouterModule.forRoot()* metodu. Ova metoda vraća modul koji sadrži konfiguriran pružatelj usluga usmjerivač naziva *Router*, te uz njega i ostale pružatelje usluga potrebnima za rad usmjerivača. Nakon toga, *Router* je zadužen izvršavanje usmjeravanja i navigiranja kroz aplikaciju temeljem trenutne web-lokacije unutar preglednika korisnika. [2] Zadnji element liste *routes* definira rutu „ ****** “ i time predstavlja rutu koja nije važeća, tj. ako korisnik pristupi ruti koja nije definirana unutar liste ruta, prikazat će mu se komponenta *ErrorpageComponent* koja ustvari predstavlja pogled *Nepostojeća stranica*. Na ovaj se način osigurava da se korisnik zadrži unutar aplikacije čak i kada pristupi nepostojećoj ruti.

app-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { PreloadAllModules, RouterModule, Routes } from '@angular/router';
//guard
import { NotAuthorizedGuard } from './core/not-authorized.guard';
import { AuthorizedGuard } from './core/authorized.guard';
//components
import { ErrorpageComponent, } from
'./features/errorpage/errorpage.component';
import { DashboardComponent } from
'./features/dashboard/dashboard.component';
import { LandingComponent } from './features/landing/landing.component';
import { AboutUsComponent } from './features/about-us/about-us.component';
import { HowItWorksComponent } from './features/how-it-works/how-it-
works.component';

const routes: Routes = [
  {
    path: '',
    component: LandingComponent,
    pathMatch: 'full',
    canActivate: [NotAuthorizedGuard]
  },
  {
    path: 'dashboard',
    component: DashboardComponent,
    canActivate: [AuthorizedGuard],
  },
  {
    path: 'about-us',
    component: AboutUsComponent,
  },
  {
    path: 'how-it-works',
    component: HowItWorksComponent,
  },
  {
    path: 'recipes',
    loadChildren: () =>
      import('./features/recipes/recipes.module').then((mod) =>
mod.RecipesModule),
  },
  {
```

```
    path: 'login',
    loadChildren: () =>
      import('./features/login/login.module').then((mod) =>
mod.LoginModule),
    canActivate: [NotAuthorizedGuard]
  },
  {
    path: 'signup',
    loadChildren: () =>
      import('./features/signup/signup.module').then((mod) =>
mod.SignupModule),
    canActivate: [NotAuthorizedGuard]
  },

  { path: '**', component: ErrorpageComponent },
];

@NgModule({
  imports: [RouterModule.forRoot(routes, { preloadingStrategy:
PreloadAllModules })],
  exports: [RouterModule],
})

export class AppRoutingModule { }
```

Iz isječka koda iznad, može se primijetiti da je unutar liste definiranih ruta također definirana i ruta *recipes*, no nigdje nema definiranih ostalih ruta vezane uz recepte, kao što su *recipes/add-new-recipe* ili *recipes/edit-recipe*. U poglavlju **5.2. Struktura projekta i instalacija**, objašnjeno je arhitektura modula koju Angular koristi te je navedeno da su sve značajke koje su povezane s receptima implementirane u zasebnom modulu. Isto tako se i odnosi na rute povezane s receptima. Ove rute nisu definirane direktno unutar modula *AppRoutingModel*, već u zasebnom modulu naziva *RecipesModule*.

Sljedeći isječak koda prikazuje implementaciju usmjeravanja za rute povezane s receptima. Način implementacije identičan je kao i implementacija ruta objašnjenog iznad, a jedina razlika odnosi se na način na koji se registriraju rute unutar *RouterModule* gdje se umjesto *forRoot()* koristi *forChild().* [2]

recipe-routing.module.ts:

```
import { NgModule } from '@angular/core';
import { RouterModule, Routes } from '@angular/router';
//guard
import { AuthorizedGuard } from '../../core/authorized.guard';
//components
import { RecipesComponent } from './recipes.component';
import { RecipesAddNewComponent } from './recipes-add-new/recipes-add-
new.component';
import { RecipeDetailComponent } from './recipe-detail/recipe-
detail.component';
import { RecipesResolverService } from './recipe-detail/recipe-
detail.resolver';
```

```
const routes: Routes = [
  {
    path: 'recipes',
    component: RecipesComponent,
    canActivate: [AuthorizedGuard],
  },
  {
    path: 'recipes/add-new-recipe',
    component: RecipesAddNewComponent,
    canActivate: [AuthorizedGuard],
  },
  {
    path: 'recipes/:id',
    component: RecipeDetailComponent,
    canActivate: [AuthorizedGuard],
    resolve: [RecipesResolverService]
  },
  {
    path: 'recipes/:id/edit',
    component: RecipeDetailComponent,
    canActivate: [AuthorizedGuard],
    resolve: [RecipesResolverService]
  },
];

@NgModule({
  imports: [RouterModule.forChild(routes)],
  exports: [RouterModule]
})

export class RecipeRoutingModule { }
```

5.8.1.1. Čuvari ruta

Kako bi se postigla funkcionalnost zabrane pristupa onim korisnicima koji nisu prijavljeni unutar aplikacije, koristi se takozvani čuvar ruta. Čuvar ruta može se implementirati u obliku servisa koja se kasnije može koristiti unutar drugih modula kao što je *RecipeRoutingModule* ili *AppRoutingModule*. Za implementaciju koristit će se *CanActivate* klasa od *@angular/router* paketa koja će proširiti klasu *AuthorizedGuard*. Kako bi se odredilo ako je korisnik prijavljen u aplikaciju ili nije, koristit će se servis *AuthService*. Ovaj servis implementiran je u svrhu autentifikacije korisnika te ovisno o tome je li korisnik autentificiran ili ne ažurira vrijednost varijable *isLoggedInSubject* u *true* ili *false*. Ako je korisnik uspješno prijavljen unutar aplikacije, čuvar straže vratit će vrijednost *true*, inače pomoću *Router* klase pozvat će metodu ***createUrlTree(['/login'])*** pomoću koje će se korisnika usmjeriti na pogled *Prijava*. [2]

authorized.guard.ts:

```
import { Injectable } from '@angular/core';
import {
  ActivatedRouteSnapshot,
  CanActivate,
  Router,
```

```
  RouterStateSnapshot,
  UrlTree,
} from '@angular/router';
import { Observable, map, take } from 'rxjs';
//services
import { AuthService } from '../core/auth.service';

@Injectable({ providedIn: 'root' })

export class AuthorizedGuard implements CanActivate {

  constructor(
    private router: Router,
    private authService: AuthService,
  ) { }

  canActivate(
    route: ActivatedRouteSnapshot,
    router: RouterStateSnapshot
  ):
    | boolean
    | UrlTree
    | Promise<boolean | UrlTree>
    | Observable<boolean | UrlTree> {

    return this.authService.isLoggedInSubject.pipe(
      take(1),
      map(isLoggedIn => {
        if (isLoggedIn) {
          return true;
        }
        return this.router.createUrlTree(['/login']);
      }
      )
    );
  }
}
```

5.8.2. Usmjeravanje i navigacija u Reactu

Kako bi se omogućilo usmjeravanje i navigiranje unutar React aplikacije, potrebno je preuzeti biblioteku treće strane *React Router DOM*. Nakon instalacije biblioteke, može se početi s implementacijom usmjeravanja pomoću web-lokacija. React se bazira na strukturi komponenti, te je tako implementirana i značajka usmjerivanja. Za početak potrebno je uvesti potrebne komponente iz biblioteke **react-router-dom** i sve poglede koji će biti korišteni u prikazu određene web-lokacije. Funkcija *App()* je glavna komponenta React aplikacije i u njoj se provjerava stanje *isLoggedin*. Pomoću vrijednosti ove varijable saznaje se je li korisnik prijavljen, tj. autentificiran unutar aplikacije. Definiranje ruta vrši se unutar komponente *Routes*. Svaka ruta se definira koristeći *Route* komponentu, a web-lokacija se pridružuje putem *path* atributa. Unutar atributa *element* potrebno je pridružiti pogled kojeg želimo spojiti s definiranom web-lokacijom. Zbog činjenice da React koristi JSX, možemo koristiti JavaScript kod te time

direktno provjeriti koju komponentu želimo prikazati ovisno o stanju varijable *isLoggedIn*. Na primjer, ako je korisnik već prijavljen unutar aplikacije i pokuša navigirati na web-lokaciju */login*, *React Router DOM* će korisnika automatski preusmjeriti na početnu stranicu aplikacije. Početna stranica predstavlja web-lokaciju „ / “, a ovisno o varijabli *isLoggedIn*, na ekranu korisnika će se prikazati ili pogled Nadzorna ploča ili pogled *Početna stranica*. [32]

App.jsx:

```
import './App.css';
import { useContext } from 'react';
import {
    BrowserRouter as Router,
    Routes,
    Route,
    Navigate,
} from 'react-router-dom';
import { AuthContext } from './context/AuthContext';
//components
import LogIn from './pages/LogIn/LogIn';
import SignUp from './pages/SignUp/SignUp';
import ErrorPage from './pages/ErrorPage/ErrorPage';
import ComingSoon from './pages/ComingSoon/ComingSoon';
import Dashboard from './pages/Dashboard/Dashboard';
import PasswordReset from './pages/PasswordReset/PasswordReset';
import Landing from './pages/Landing/Landing';
import Recipes from './pages/Recipes/Recipes';
import Recipe from './pages/RecipeDetail/RecipeDetail';
import RecipesAddNew from './pages/RecipesAddNew/RecipesAddNew';
import AccountSettings from './pages/AccountSettings/AccountSettings';

//protected route
import ProtectedRoute from './components/ProtectedRoute/ProtectedRoute';

function App() {
    const { isLoggedIn } = useContext(AuthContext);

    return (
        <Router>
            <Routes>
                <Route
                    path='/'
                    element={isLoggedIn ? <Dashboard /> : <Landing />}
                />
                {isLoggedIn && (
                    <>
                        <Route
                            path='/login'
                            element={<Navigate replace to='/' />}
                        />
                        <Route
                            path='/signup'
                            element={<Navigate replace to='/' />}
                        />
                    </>
                )}
                <Route path='/login' element={<LogIn />} />
                <Route path='/signup' element={<SignUp />} />
                <Route path='/password-reset' element={<PasswordReset />}/>
```

```
                <Route
                    path='/about-us'
                    element={<ComingSoon title='About Us' />}
                />
                <Route
                    path='/how-it-works'
                    element={<ComingSoon title='How It Works' />}
                />
                <Route element={<ProtectedRoute />}>
                    <Route
                        path='/dashboard'
                        element={<Navigate replace to='/' />}
                    />
                    <Route path='/recipes' element={<Recipes />} />
                    <Route path='/recipes/:id' element={<Recipe />} />
                    <Route
                        path='/recipes/:id/update'
                        element={<RecipesAddNew isEditRecipe={true} />}
                    />
                    <Route
                        path='/recipes/add-new'
                        element={<RecipesAddNew isEditRecipe={false} />}
                    />
                    <Route
                        path='/account-settings'
                        element={<AccountSettings />}
                    />
                </Route>
                <Route path='*' element={<ErrorPage />} />
            </Routes>
        </Router>
    );
}
export default App;
```

5.8.2.1. Zaštićene rute

Pojam zaštićene rute unutar React aplikacije predstavlja identičnu stvar kao pojam čuvar ruta unutar Angular aplikacije. Dakle, cilj zaštićenih ruta je zabraniti pristup definiranim web-lokacijama onim korisnicima koji nisu prijavljeni u aplikaciju. U isječku koda ispod, implementirana je logika za zaštitu određenih dijelova aplikacije od pristupa neprijavljenim korisnicima. Pomoću varijable *isLoggedIn* definira se stanje autentifikacije korisnika, te ovisno o tome korisnika se navigira na pogled *Prijava* ili se prikaže onaj pogled koji je proslijeđen kao svojstvo unutar komponente *ProtectedRoute*. [32]

ProtectedRoute.jsx:

```
import React, { useContext } from 'react';
import { Navigate, Outlet } from 'react-router-dom';
import { AuthContext } from '../../context/AuthContext';

const ProtectedRoute = ({ redirectPath = '/login', children }) => {
  const { isLoggedIn } = useContext(AuthContext);
```

```
  if (!isLoggedIn) {
    return <Navigate to={redirectPath} replace />;
  }

  return children;
};

export default ProtectedRoute;
```

5.9. Rad s DOM-om

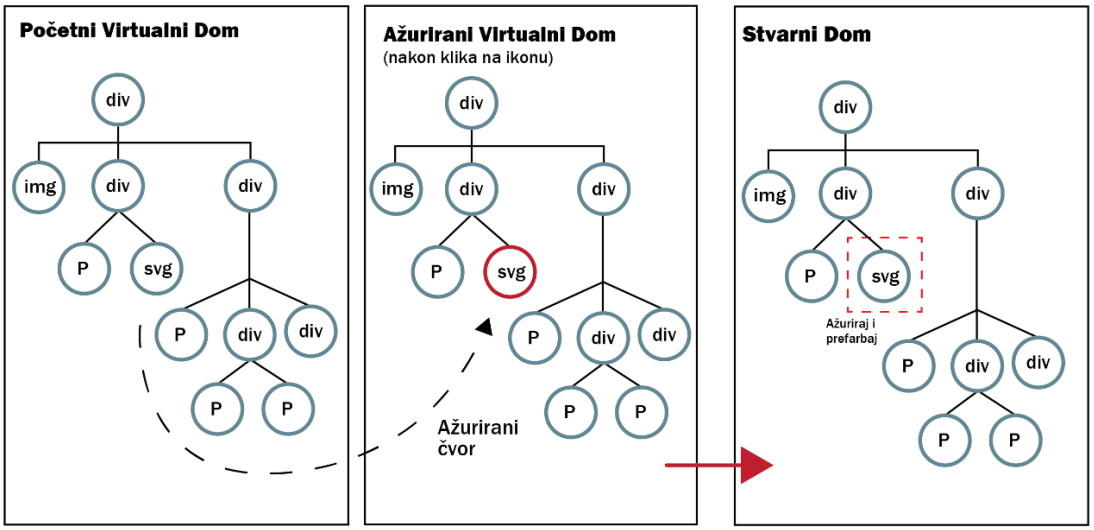
DOM (eng. *Document Object Model*) je ključan za prikaz elemenata implementiranih pomoću Reacta ili Angulara u korisnikovom pregledniku. Ukratko, DOM je programski sučelje koje predstavlja strukturirane čvorove (eng. *Nodes)* u HTML dokumentu. Kroz DOM, React i Angular mogu dinamički pristupiti i mijenjati sadržaj, strukturu i stil web stranice. React i Angular koriste različite pristupe za manipulaciju DOM-om, što uvelike utječe na performanse i učinkovitost web aplikacija.

Primjer prema kojem će biti uspoređeno funkcioniranje Reactovog i Angularovog DOM-a bit će kartica recepta, vidi sliku 42. Korisnik ima mogućnost označavanja recepta sa *sviđa mi se* klikom na ikonu srca označenu crvenim obrubom. Nakon klika, ikona mijenja svoj stil iz bijele boje u crvenu boju. U sljedećim poglavljima bit će objašnjeno kako se ove promjene izvršavaju koristeći Virtualni DOM i Stvarni DOM.

Slika 42: Kartica recepta - klik na ikonu

5.9.1. Virtualni DOM u Reactu

React koristi koncept nazvan Virtualni DOM (eng. *Virtual DOM*). Kada se stanje nekog čvora promijeni, React stvara novi virtualni DOM i uspoređuje ga sa starim. Ovaj proces naziva se *diffing*, a uključuje prolazak Reacta kroz stari i novi Virtualni DOM te ažurira Stvarni DOM samo na mjestima gdje postoje promjene. [33]

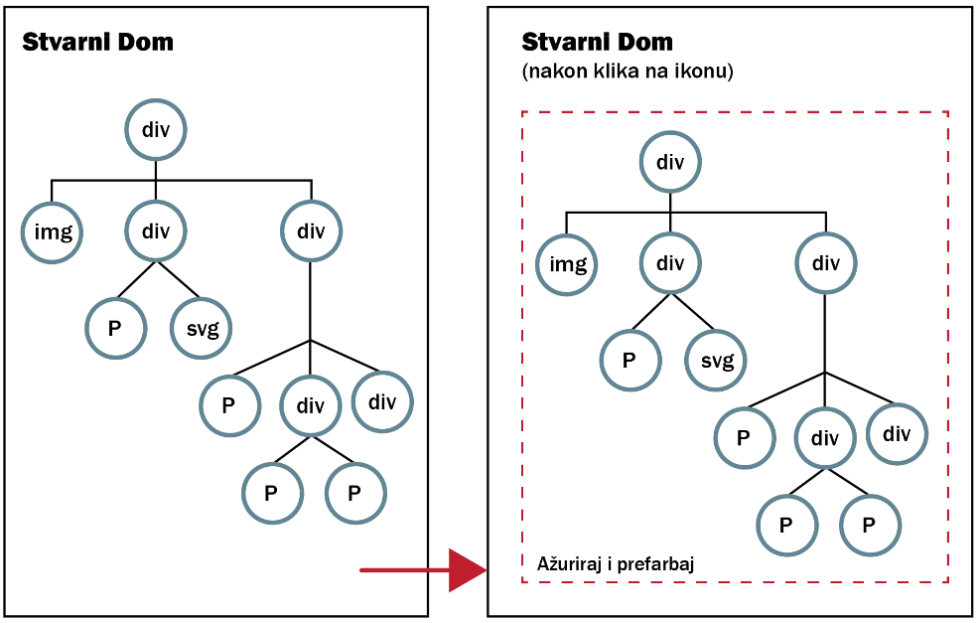


Slika 43: Primjer ažuriranja Stvarnog DOM-a u Reactu

Svaki dio aplikacije koji se prikazuje na ekranu korisnika dolazi od stvarnog DOM-a, pa tako i komponenta kartice recepta prikazana na slici 42. React stvara kopiju stvarnog DOM-a koja se naziva Virtualni DOM. Primjer na slici 43 prikazuje proces koji React obavlja u pozadini nakon što korisni pritisne na ikonu srca. Za početak, s lijeve strane je prikazan početni Virtualni DOM koji prikazuje trenutno stanje kartice recepta. Nakon što korisnik klikne na ikonu srca, stvara se novi Virtualni DOM te React uspoređuje početni i ažurirani Virtualni DOM kako bi pronašao koji se element unutar Virtualnog DOM-a promijenio. Bitno je napomenuti da promjene nisu vidljive na ekranu korisnika sve dok React ne ažurira Stvarni DOM. Dakle, Virtualni DOM predstavlja samo kopiju stvarnog DOM-a kojeg koristi za pronalazak onih elemenata koji su se promijenili. Ove razlike se zatim primjenjuju na Stvarni DOM, čime se minimizira manipulacija stvarnim DOM-om i time se postižu bolje performanse. [33]

5.9.2. Real DOM u Angularu

S druge strane, Angular radi izravno sa stvarnim DOM-om. Kada se podaci promijene, Angular ažurira cijelo stablo strukture HTML oznaka dok ne dođe do onih koje su se promijenile. Ovaj proces može biti sporiji i manje učinkovit od pristupa Virtualnog DOM-a, posebno za velike aplikacije gdje se odjednom može mijenjati mnogo elemenata. Kada se podaci promijene, Angular izravno ažurira Stvarni DOM. To znači da čak i ako se promijeni mali dio podataka, cijelo DOM stablo mora biti ažurirano da bi se odrazila ta promjena. Ovo može biti sporije i manje učinkovito, posebno u velikim aplikacijama ili kada su promjene učestale. [34]



Slika 44: Primjer ažuriranja Stvarnog DOM-a u Angularu

Slika 44 prikazuje promjene nad Stvarnim DOM-om koje se dešavaju nakon što korisnik klikne na ikonu srca koja označava *sviđanje* recepta, vidi sliku 42. Početno stanje komponente je zapisano unutar Stvarnog DOM-a prikazanog s lijeve strane slike 44, a nakon klika na ikonu srca Angular označava da se desila promjena pri čemu se stvara novi ažurirani Stvarni DOM nakon čega se DOM stablo u cijelosti ažurira. Iako se promjena nalazi samo u *svg* čvoru koji je promijenio svoju boju iz bijele u crvenu, promjena je utjecala na čitavo DOM stablo. [34]

5.10. Forme i validacija polja

Kada je riječ o formama Angular i React koriste različite pristupe. U Angularu se forme kreiraju koristeći *Angular Forms* modul koji dolazi uz Angular. U drugu ruku, React zahtijeva preuzimanje biblioteku treće strane za lakšu implementaciju formi i validacije. Implementacija formi koristeći React i Angular bit će objašnjena na primjeru registracije u aplikaciju, registracija se sastoji od četiri polja svaka sa svojim pravilima validacije, vidi sliku 6. [2]

5.10.1. Forme i validacija u Angularu

Za implementaciju formi u Angularu koristi se *Angular Form* modul koji pruža setove klasa i direktiva koje omogućavaju kreiranje formi .Pomoću klasa *FormGroup* i *FormControl.* Svaki *FormControl* koji se nalazi unutar *FormGroup* se prati pomoću dodijeljenog imena pri kreiranju grupe forme. Svakom *FormControl*-u definirane su i validacijske scheme. Kako bi se postigla funkcionalnost gdje korisnik mora dvaput unijeti svoju lozinku koja mora biti identična, mora se implementirati posebna metoda *matchpassword.* Validacijske poruke služe za

povratnu informaciju korisniku pri popunjavanju forme, a kako bi se prikazale na ekran korisnika koristi se funkcija *validationErrors.* [2]

signup.component.ts:

```
import { Component } from '@angular/core';
import { FormGroup, FormControl } from '@angular/forms';

@Component({
  selector: 'app-signup',
  templateUrl: './signup.component.html',
  styleUrls: ['../../shared/shared.styles.css'],
})
export class SignupComponent{
  validationMessages = {
    'name': {
      'required': 'Name is required.',
      'maxlength': 'Name cannot be more than 20 characters long.'
    },
    'email': {
      'required': 'Email is required.',
      'email': 'Please enter a valid email address.'
    },
    'password': {
      'required': 'Password is required.',
      'minlength': 'Password must be at least 8 characters long.'
    },
    'confirmPassword': {
      'required': 'Confirm password is required.',
      'matchpassword': 'Passwords must match.'
    }
  }

  signUpForm = new FormGroup({
    name: new FormControl('',[Validators.required,
Validators.maxLength(20)]),
    email: new FormControl('', [Validators.email, Validators.required]),
    password:newFormControl('',[Validators.required,Validators.minLength(8)
]), confirmPassword: new FormControl('', Validators.required)
  },{ validators: matchpassword });

  get validationErrors() {
    const errors = {}
    Object.keys(this.signUpForm.controls).forEach(key => {
      errors[key] = ''
      const control = this.signUpForm.controls[key]
      if (control && !control.valid) {
        const messages = this.validationMessages[key]
        Object.keys(control.errors).forEach(error => {
          errors[key] += messages[error] + ' '
        })
      }
    })
    return errors
  }
  onSubmit() {
    const { name, email, password } = this.signUpForm.value;
    await this.authService.signup(name, email, password)
  }
}
```

Nakon kreiranja forme, potrebno ju je povezati unutar HTML datoteke. *FormGroup* prati sve promjene koje se događaju nad *FormControls* i te promjene je potrebno implementirati unutar HTML predloška, a to se postiže korištenjem *[formGroup]* direktive unutar *<form>* elementa. Kako bi se postigla komunikacija između *FormControl* klase definirane u modelu, potrebno ju je dodijeliti onom *<input>* elementu koji će biti zadužen za ažuriranje vrijednosti modela kada korisnik upisuje podatke. Na posljetku, formi se također pridružuje događaj *(ngSubmit)* kojemu je dodijeljena metoda *onSubmit()*. Ovaj će se događaj aktivirati kada korisnik klikne na dugme *Sign up*. [2]

signup.component.html:

```
    <form [formGroup]="signUpForm" (ngSubmit)="onSubmit()">
        <input type="text" id="name" formControlName="name" />
        <span>{{ validationErrors["name"] }}</span>
        <input type="text" id="email" formControlName="email"/>
        <span>{{ validationErrors["email"] }}</span>
        <input type="password" id="password" formControlName="password"/>
        <span>{{ validationErrors["password"] }}</span>
        <input type="password" id="confirmPassword"
               formControlName="confirmPassword" />
        <span>{{ validationErrors["confirmPassword"] }}</span>
        <button type="submit">Sign up</button>
    </form>
```

5.10.2. Forme i validacija u Reactu

U praktičnom dijelu ovog rada korištena je biblioteka *Formik* i *Yup* koji u kombinaciji pružaju vrlo efikasno rješenje upravljanju formama i validacijom. Za kreiranje forme koristi se komponenta *Formik* unutar koje se definira objekt *initialValues* sa svim vrijednostima forme. Kako bi implementirali validaciju polja, unutar *Formik* svojstva *validationSchema* potrebno je proslijediti validacijsku shemu. U ovom slučaju, koristi se schema *SignUpSchema* koja je definirana pomoću biblioteke *Yup.* Unutar svojstva *onSubmit* proslijeđena je metoda koja će se izvršiti nakon klika na dugme *Sign up.* Polja forme definirani su pomoću HTM *<input>* elementa *Form.* [35] [36]

SignUp.jsx

```
const formFieldValues = [{type:"text", name:"name"}, {type:"email",
                         name:"email"}, {type:"password", name:"password"},
                         {type:"password", name:"confirmPassword"}]

const SignUpSchema = Yup.object({
    name: Yup.string().required("Name is required").max(20, "Name must be
less than 20 characters long"),
    email: Yup.string().email('Invalid e-mail address').required('Email is
required'),
    password: Yup.string().min(8, 'Password must be at least 8 characters
long').required('Password is required'),
```

```
    confirmPassword: Yup.string().oneOf([Yup.ref("password"), null],
"Password must match").required("Confirm password is required")
})

export default function SignUp() {
    const { signup } = useContext(AuthContext);
    <Formik
        initialValues={{
            name: "",
            email: '',
            password: '',
            confirmPassword: "",
        }}
        validationSchema={SignUpSchema}
        onSubmit={async (values, actions) => {
            await signup(values.email, values.password, values.name)
        }}
    >
        {(formik) => (
            <Form>
                {formFieldValues.map((item) => (
                    <Field
                        type={item.type}
                        name={item.name}
                    />
                ))}
                <Button type="submit">
                    Sign up
                </Button>
            </Form>
        )}
    </Formik>
}
```

5.11. Firebase API

Unutar ovog poglavlja bit će prikazano kako spojiti Firebase s React i Angular projektom. Nakon toga, bit će objašnjen primjer API (eng. *Application Programming Interface)* poziva na primjeru koji se nadovezuje na primjer iz prijašnjeg poglavlja, **5.10. Forme i validacija.** Nakon popunjavanja forme za registraciju, korisnik ima mogućnost pritiska tipke *Sign up* koja pokreće akciju registracije u aplikaciju koristeći Firebase autentifikaciju.

5.11.1. Integracija Firebasea u Angular

Spajanje kreiranog Firebase projekta s Angular aplikacijom započinje s pohranom *firebaseConfig* podataka, vidi sliku 34, unutar Angular projekta. Ovi podaci spremljeni su unutar datoteke ***environment.prod.ts***:

```
export const environment = {
  production: true,
  firebaseConfig: {
    apiKey: "AIzaSyDUF5b4oeKRvOcg5wTkkrtlsBG23awrmb8",
```

```
    authDomain: "realtime-food-recipes.firebaseapp.com",
    projectId: "realtime-food-recipes",
    storageBucket: "realtime-food-recipes.appspot.com",
    messagingSenderId: "822992735552",
    appId: "1:822992735552:web:2ddcee12741a8ae8e6fea5"
  },
};
```

Sada kada su konfiguracijski podaci Firebase projekta spremljeni, potrebno je inicijalizirati Firebase aplikaciju unutar korijenskog modula Angular projekta, *App module.* Metodi *initializeApp* prosljeđuju se konfiguracijski podaci Firebase projekta. Na ovaj način inicijalizira se Firebase aplikacija koja postaje dostupna za korištenje Angularu. [3]

app.module.ts:

```
//firebase
import { AngularFireModule } from '@angular/fire/compat';
import { AngularFireAuthModule } from '@angular/fire/compat/auth';
import { AngularFireStorageModule } from '@angular/fire/compat/storage';
import { AngularFirestoreModule } from '@angular/fire/compat/firestore';
import { AngularFireDatabaseModule } from '@angular/fire/compat/database';
//environment
import { environment } from 'src/environments/environment';

@NgModule({
  imports: [
    …
    AngularFireModule.initializeApp(environment.firebaseConfig),
    AngularFireAuthModule,
    AngularFirestoreModule,
    AngularFireStorageModule,
    AngularFireDatabaseModule,
  ]
  …
})
export class AppModule { }
```

5.11.1.1. API poziv na Firebase (Angular)

Nakon popunjavanja forme registracije vidljive na slici 6, korisnik klikom na dugome pokreće *(click)* događaj koji izvršava metodu *onSubmit()*. Metoda *onSubmit()* destrukturira (eng. *Destructuring*) objekt *signUpForm.value* unutar koje se nalaze vrijednosti koje je korisnik upisao u formu. Vrijednosti *name, email* i *password* prosljeđuju se kao argumenti funkcije *signup().* Funkcija *signup()* zadužena je za autentifikacija korisnika koristeći Firebase SDK *createUserWithEmailAndPassword*. Nakon autentifikacije, korisnika se navigira na web-lokaciju *dasbhoard*. [3]

auth.service.ts:

```
@Injectable()
export class AuthService {

  constructor(
```

```
    public afs: AngularFirestore, // Inject Firestore service
    public afAuth: AngularFireAuth, // Inject Firebase auth service
    public router: Router,
  ) {

  // Sign up with email/password
  async signup(displayName: string, email: string, password: string) {
    try {
      await this.afAuth.createUserWithEmailAndPassword(email, password);
      this.router.navigate(['dashboard']);
    }
    catch (error) {
        throw error;
    }
  }
```

5.11.2. Integracija Firebasea u React

Integracija Firebasea u React je slična kao i integraciji unutar Angular aplikacije. Za početak potrebno je spremiti konfiguracijske podatke Firebase projekta unutar React projekta. Konfiguracijski podaci spremljeni su unutar *firebase.js* datoteke iz koje se odmah i inicijalizira Firebase aplikacije pomoću naredbe *initializeApp* kojoj su također proslijeđeni konfiguracijski podaci Firebase projekta kao i u Angularu. [3]

firebase.js:

```
import firebase from 'firebase/app';
import "firebase/auth"
import "firebase/firestore"

const firebaseConfig = {
    apiKey: "AIzaSyDUF5b4oeKRvOcg5wTkkrtlsBG23awrmb8",
    authDomain: "realtime-food-recipes.firebaseapp.com",
    projectId: "realtime-food-recipes",
    storageBucket: "realtime-food-recipes.appspot.com",
    messagingSenderId: "822992735552",
    appId: "1:822992735552:web:2ddcee12741a8ae8e6fea5"
  };
const app = firebase.initializeApp(firebaseConfig)

export default app;
export const auth = firebase.auth();
export const firestore = firebase.firestore;
```

Nakon integracije Firebase aplikacije, programerima je na raspolaganju korištenje Firebase metoda. Nakon toga, funkcija *signup()* ima identičnu funkcionalnost kao i funkcija *signup()* u Angular primjeru, te se jedina razlika u sintaksi.

AuthProvider.js:

```
const AuthProvider = ({ children }) => {
    const navigate = useNavigate();
    const signup = async (email, password) => {
```

```
        try {
            await firebase.auth().createUserWithEmailAndPassword(email,
            password)
            navigate("/dashboard")
        } catch (err) {
            throw err
        }
    }
};
```

5.12. Primjer koda komponente *RecipeCard*

Isječci koda koji su korišteni u prijašnjim poglavljima usporedbe React i Angular aplikacije bili su pojednostavljeni i skraćeni kako bi se održao fokus na specifičnim stvarima koje je određeno poglavlje opisivalo, te u isto vrijeme izbjegla zabuna zbog nepotrebnih linija koda koji nisu bili od velike važnosti za usporedbu pojedinih značajki. Iz tog razloga, ovo poglavlje pružit će kompletan kod implementacije komponente *RecipeCard* u Angularu i Reactu. Temeljem ovih primjera, bit će jasno vidljiva implementacija identične značajke u Reactu i u Angularu.

5.12.1. Komponenta *RecipeCard* u Angularu

recipe-card.component.ts:

```
@Component({
  selector: 'app-recipe-card',
  templateUrl: './recipe-card.component.html',
  styleUrls: ['./recipe-card.component.css'],
  changeDetection: ChangeDetectionStrategy.OnPush

})
export class RecipeCardComponent implements OnInit {
  @Input() recipe = null;

  userData: any;
  imageSrc: string;
  isLikedByUser: boolean;
  recipeLikes: number;

  constructor(
    private recipeLikeService: RecipeLikeService,
    private authService: AuthService,
    private checkImageService: CheckImageService,
    private router: Router
  ) {
    this.userData = this.authService.userData;
  }

  async handleFavoriteClick(e) {
    e.stopPropagation();
    await this.recipeLikeService.handleLikeRecipe(this.recipe,
this.userData, this.isLikedByUser);
```

```
    this.isLikedByUser = this.recipeLikeService.getIsLikedByUser();
    this.recipeLikes = this.recipeLikeService.getRecipeLikes();
  }

  private checkImage(url: string, placeholderUrl: string): void {
    this.checkImageService.checkImage(url, placeholderUrl).then((src) => {
      this.imageSrc = src;
    });
  }

  handleRecipeClick = () => {
    this.router.navigate([`/recipes/${this.recipe.id}`]);
  }

  ngOnInit() {
    // Initialize the service with recipe and user data
    this.recipeLikeService.init(this.recipe, this.userData);
    this.isLikedByUser = this.recipeLikeService.getIsLikedByUser();
    this.recipeLikes = this.recipeLikeService.getRecipeLikes();
    this.checkImage(this.recipe.imgUrl, '../../../../assets/img/recipe-
image-placeholder.png');
  }
}
```

recipe-card.component.html:

```
<div class="RecipeCardWrapper" (click)="handleRecipeClick()">
  <div class="ImageWrapper">
    <img class="RecipeImage" src="{{ recipe.imgUrl }}" />
  </div>
  <div class="TextWrapper">
    <div class="NameFavoritesWrapper">
      <p class="RecipeName">{{ recipe.name }}</p>
      <div class="FavoriteIconWrapper">
        <svg-icon
          [class]="isLikedByUser ? 'AddFavorite isFavorite' :'AddFavorite'"
          [applyClass]="true"
          src="../../../../assets/img/add-favorite.svg"
          (click)="handleFavoriteClick($event)"
        >
        </svg-icon>
      </div>
    </div>
    <p class="RecipeDescription">{{ recipe.description }}</p>
    <div class="RecipeIngredientsWrapper">
      <app-chip *ngFor="let ingredient of recipe.ingredients">
        {{ ingredient }}
      </app-chip>
    </div>
    <div class="BottomTextWrapper">
      <p class="RecipeCategory">{{ recipe.category }}</p>
      <div class="CookTimeWrapper">
        <img class="CookTime" src="../../../assets/img/clock-icon.svg"/>
        <p class="CookTimeLabel">
          {{ recipe.cookTimeMin }}
        <span style="font-size:10px">(min)</span>
        </p>
      </div>
    </div>
  </div>
</div>
```

5.12.2. Komponenta *RecipeCard* u Reactu

RecipeCard.jsx:

```
const RecipeCard = ({ recipe }) => {
  const navigate = useNavigate();
  const { userData } = useContext(AuthContext);
  const imageSrc = useCheckImage(recipe.imgUrl, RecipeImagePlaceholder);
  const { isLikedByUser, handleLikeRecipe } = useRecipeLike(recipe,
userData);

  const handleRecipeClick = () => {
    navigate(`/recipes/${recipe.id}`);
  }

  return (
    <RecipieCardWrapper onClick={handleRecipeClick} >
      <ImageWrapper>
        <RecipeImage src={imageSrc} />
      </ImageWrapper>
      <TextWrapper>
        <NameFavoritesWrapper>
          <RecipeName className="RecipeName">{recipe.name}</RecipeName>
          <FavoriteIconWrapper>
            <AddFavorite
              onClick={(e) => handleLikeRecipe(e)}
              isfavorite={isLikedByUser}
            />
          </FavoriteIconWrapper>
        </NameFavoritesWrapper>
        <RecipeDescription>{recipe.description}</RecipeDescription>
        <RecipeIngredientsWrapper>
          {recipe.ingredients.map((ingredient, index) => {
            return <Chip key={index} name={ingredient} />;
          })}
        </RecipeIngredientsWrapper>
        {(
          <BottomTextWrapper>
            <RecipeCategory>{recipe.category}</RecipeCategory>
            <CookTimeWrapper>
              <CookTime />
              <CookTimeLabel>
                {recipe?.cookTimeMin || 0}
                <span style={{ fontSize: "10px" }}>(min)</span>
              </CookTimeLabel>
            </CookTimeWrapper>
          </BottomTextWrapper>
        )}
      </TextWrapper>
    </RecipieCardWrapper>
  );
};
export default RecipeCard;
```

5.13. Statistika Reacta i Angulara

U prijašnjim poglavljima obrađena je usporedba Reacta i Angulara vezano uz njihovu arhitekturu, sintaksu, usmjeravanje i navigaciju, vrsti DOM-a i tako dalje. Može se reći da se takva usporedba više fokusira na razvoj aplikacije koristeći ili React ili Angular. No, usporedba Reacta i Angulara može se obraditi i na temelju različitih faktora kao što su zajednica i ekosustav, brzina razvoja i produktivnosti, te korisničko iskustvo.

5.13.1. Zajednica i ekosustav

Prema *State of JS 2023*, React je i dalje jedan od najpopularnijih JavaScript okvira, ali Angular također ima snažnu prisutnost. Na GitHubu, React ima preko 210k zvjezdica i preko 44k forkova, dok Angular ima preko 90k zvjezdica i preko 24k forkova, što ukazuje na veliku aktivnost i doprinos zajednice za oba okvira. No, ipak se može reći da je zajednica Reacta neupitno veća i aktivnija u odnosu na zajednicu Angulara. No i dalje, oba okvira su podržana od strane velikih IT kompanija, što im pruža dodatnu prednost u odnosu na ostale JavaScript biblioteke i okvire. [37] [38] [39]

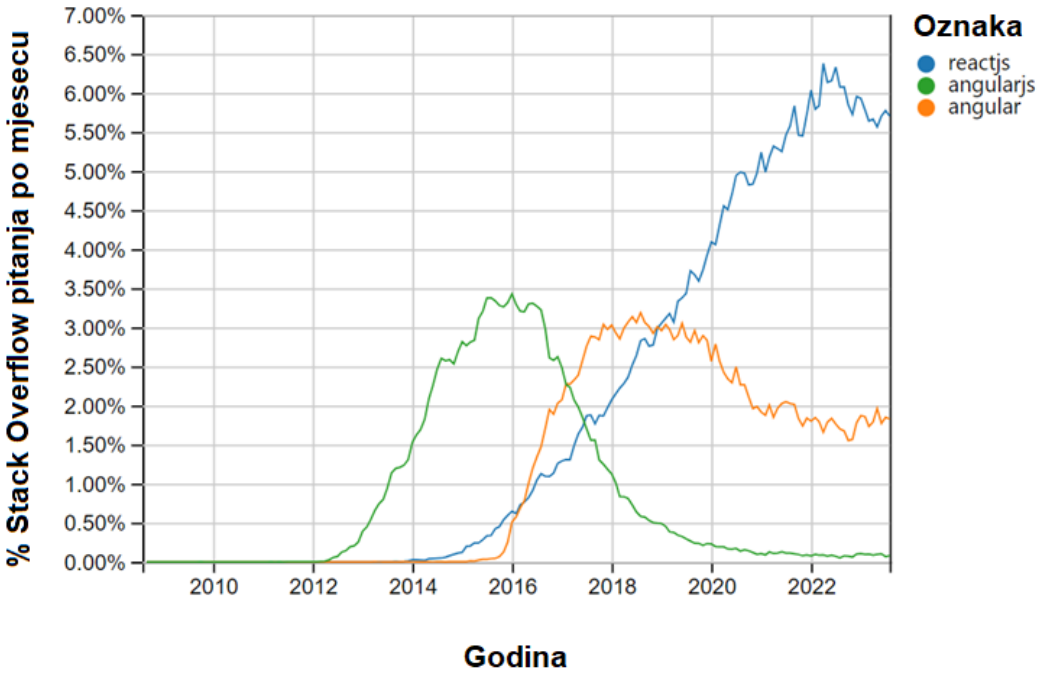
5.13.2. Brzina razvoja i produktivnost

Kada je riječ o brzini razvoja i produktivnosti, oba okvira imaju svoje prednosti. React, s virtualnim DOM-om, omogućuje brže ažuriranje i renderiranje komponenata u usporedbi s Angularovim stvarnim DOM-om. React je baziran na JavaScript-u i JSX-u, što omogućuje brži razvoj za iskusne programere. S druge strane, Angular je potpuni okvir koji dolazi s mnogo ugrađenih alata i značajki, što može povećati produktivnost za složene aplikacije.

5.13.3. Korisničko iskustvo

Kada je riječ o korisničkom iskustvu, oba okvira pružaju snažne mogućnosti za izradu responzivnih i interaktivnih sučelja. React koristi jednosmjerni tok podataka, što olakšava praćenje promjena stanja i *debugiranje*. Angular koristi dvosmjernu vezu podataka, što može olakšati sinkronizaciju modela i pogleda, ali može također dovesti do problema s performansama za velike aplikacije. Učenje i upotreba Reacta i Angulara također se razlikuju. React je minimalistički i prilično lak za razumijevanje, ako učenik već zna JavaScript. Međutim, potrebno je vremena da se nauči kako postaviti projekt, jer ne postoji predefinirana struktura projekta. Angular je sam po sebi ogromna biblioteka i učenje svih koncepata zahtijeva više vremena nego što je to slučaj s Reactom. Angular je kompleksniji za razumijevanje, postoji dosta nepotrebne sintakse, a upravljanje komponentama je nepotrebno zakomplicirano.

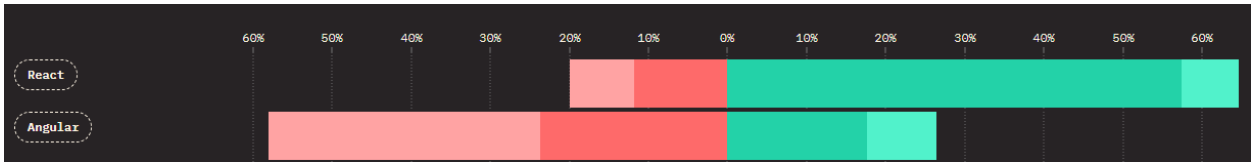
Prema *Stack Overflow Trends*, React ima konstantno visoku stopu interesa među programerima od 2016., dok je interes za Angular blago opadao. Ovo ukazuje na to da, iako oba okvira imaju veliku zajednicu, React može pružiti bolje korisničko iskustvo za neke programere, vidi sliku 45. [40]



Slika 45: Prikaz trenda pitanja na Stack Overflowu

Još jedna zanimljivost vrijedna spomena je prikazana na slici 45. Ova slika prikazuje omjer pozitivnih i negativnih korisničkih iskustava povezane uz React i Angular.

- Svijetlo crvena – Čuo sam za njega i ne zanima me
- Tamno crvena – Koristio sam ga i ne bi ga ponovno koristio
- Tamno zelena – Koristio sam ga i koristio bih ga ponovno
- Svijetlo zelena – Čuo sam za njega i htio bih ga naučiti



Slika 46: Prikaz pozitivnih i negativnih korisničkih iskustava

Čak 57% ispitanika je čulo za pojam *React* i koristili bi ga ponovno, dok za Angular taj broj iznosi 18%. Isto tako, 12% ispitanika je koristilo React i ne bi ga ponovno koristili, dok taj broj za Angular iznosi 24%. Nadalje, 34% ispitanika je čulo za pojam *Angular* i ne bi ga htjeli naučiti, dok taj broj za React iznosi samo 8%. [39]

6. Zaključak

Unutar ovog diplomskog rada obrađena je usporedba dva popularna okvira/biblioteke Angular i React kroz izradu iste aplikacije *Realtime Food Recipes* u oba okvira. React je razvijen od tvrtke *Meta,* poznatijeg kao *Facebook,* dok je Angular razvijan od tvrtke *Google.* Razvoj identične aplikacije omogućilo je razumijevanje oba okvira pri čemu se mogu istaknuti njihove sličnosti i različitosti isto kao i prednosti i mane. React i Angular koriste se za izgradnju modernih i skalabilnih web aplikacija. Oba se temelje na JavaScriptu sa time da se Angular bazira na TypeScriptu koji predstavlja superset JavaScripta. Također, i Angular i React se vode arhitekturom komponenti pri čemu se omogućava ponovna iskoristivost i održavanje koda. Komponente predstavljaju temelj za izgradnju cjelokupnog korisničkog sučelja. No, jedna od najvećih razlika leži u tome što je React biblioteka, dok je Angular potpuni okvir. To znači da React pruža samo osnovne funkcionalnosti za izgradnju korisničkog sučelja, dok Angular dolazi s mnogo više ugrađenih značajki, kao što su rutiranje, upravljanje stanjem unutar formi i slično. Također, to bi značilo da je React fleksibilniji i omogućava programerima da odaberu druge biblioteke i alate koje žele koristiti uz React, dok je Angular sveobuhvatan i od programera „zahtjeva“ da prati principe razvoja sve od strukture mapa i datoteka, pa do procesa implementacije određenih značajki. React koristi JSX (JavaScript XML) sintaksu za definiranje korisničkog sučelja, dok Angular koristi HTML predloške. React se također fokusira na jednosmjernu komunikaciju između komponenti, dok s druge strane Angular podržava i jednosmjernu i dvosmjernu komunikaciju.

Neke od prednosti koje se mogu izdvojiti vezane uz React su efikasno ažuriranje korisničkog sučelja koristeći Virtualni DOM, velika zajednica korisnika i veliki broj dostupnih biblioteka i komponenti što omogućava maksimalnu fleksibilnost pri izradi React projekta. Također, iz osobnog mišljenja mogu reći da je React jednostavan za učenje i korištenje. U drugu ruku, zato što je React biblioteka, potrebno je koristiti i pronaći dodatne alate za izradu funkcionalnog projekta pa se to može smatrati manom Reacta. Uz to, *JSX* sintaksa može biti zbunjujuća ako programeri nisu navikli na njeno korištenje. Od prednosti Angulara može se izdvojiti činjenica da je Angular potpuni okvir koji pruža sve što je potrebno za razvoj aplikacija, te pruža striktne smjernice prema kojima se razvija aplikacija. U teoriji, zbog striktnih smjernica i definirane strukture datoteka i mapa unutar projekta, programeri se mogu lagano prebaciti s jednog Angular projekta na drugi. Dvosmjerna komunikacija između komponenti može olakšati upravljanjem stanjem. No, iz osobnog mišljenja rekao bih da Angular ima veću krivulju učenja nego React, te arhitektura bazirana na modulima u kombinaciji sa striktnim smjernicama za izradu aplikacije nisu omiljena kombinacija programerima, što također potvrđuju statistike navedene u poglavlju **5.13.** ***Statistike Reacta i Angulara.***

Popis literature

[1] *React - Documentation* Dostupno na: *https://react.dev/learn* Pristupano: 14.06.2023.

[2] *Angular - Documentation* Dostupno na: *https://angular.io/docs* Pristupano: 27.05.2023.

[3] *Firebase - Documentation* Dostupno na: *https://firebase.google.com/docs* Pristupano: 02.04.2023.

[4] *InterviewBit – Angular Vs React: Difference Between Angular And React* Dostupno na: *https://www.interviewbit.com/blog/angular-vs-react/* Pristupano: 30.05.2023.

[5] *sitepoint – Angular vs React: Which is Better for Frontend Projects?* Dostupno na: *https://www.sitepoint.com/angular-vs-react/* Pristupano: 30.05.2023.

[6] *DZone - Why You Should Use TypeScript for Developing Web Applications* Dostupno na: *https://dzone.com/articles/what-is-typescript-and-why-use-it* Pristupano: 04.04.2023.

[7] *DZone - A Comparison of Single-Page and Multi-Page* Applications Dostupno na: *https://dzone.com/articles/the-comparison-of-single-page-and-multi-page-appli* Pristupano: 04.04.2023.

[8] *Medium – NPM vs. Yarn* Dostupno na: *https://medium.com/@aschittone/npm-vs-yarn-cc0001a2a4b8* Pristupano: 04.04.2023.

[9] *Medium – Fun JavaScript Facts to Share at Parties* Dostupno na: *https://javascript.plainenglish.io/javascript-fun-facts-to-share-at-parties-9721527591de* Pristupano: 05.04.2023.

[10] *Wikipedia – JavaScript* Dostupno na*: https://en.wikipedia.org/wiki/JavaScript* Pristupano: 06.05.2023.

[11] *MDN Web Docs – JavaScript - Dynamic client-side scripting* Dostupno na: *https://developer.mozilla.org/en-US/docs/Learn/JavaScript* Pristupano: 06.05.2023.

[12] *MDN Web Docs – What is JavaScript:* Dostupno na*: https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript* Pristupano: 07.05.2023.

[13] *codecademy – What is JavaScript Used For?* Dostupno na: *https://www.codecademy.com/resources/blog/what-is-javascript-used-for/* Pristupano: 07.05.2023.

[14] *TypeScript – Documentation* Dostupno na*: https://www.typescriptlang.org/docs/* Pristupano: 10.05.2023.

[15] *Node.js – Learn* Dostupno na*: https://nodejs.dev/en/learn* Pristupano: 14.05.2023.

[16] *Yarn – Documentation* Dostupno na*: https://yarnpkg.com/getting-started* Pristupano: 14.05.2023.

[17] *CronJ -Is Node.js still relevant in 2023? AngularJS, ReactJS & VueJS demystified* Dostupno na: *https://www.cronj.com/blog/is-nodejs-still-relevant-in-2019-nodejs-angularjs-reactjs-vuejs-comparision/* Pristupano: 29.08.2023.

[18] *Microsfot – Documentation* Dostupno na: *https://learn.microsoft.com/en-us/windows/dev-environment/javascript/react-overview* Pristupano: 04.07.2023.

[19] *Babel – Documentation* Dostupno na: *https://babeljs.io/docs/* Pristupano: 04.07.2023.

[20] *GeeksforGeeks – What is SPA(Single page application) in AngularJS?* Dostupno na: *https://www.geeksforgeeks.org/what-is-spa-single-page-application-in-angularjs/* Pristupano: 04.07.2023.

[21] *Wikipedia – React(software)* Dostupno na: *https://en.wikipedia.org/wiki/React_(software)* Pristupano: 06.07.2023.

[22] *GeeksforGeeks – What are the features of ReactJS?* Dostupno na: *https://www.geeksforgeeks.org/what-are-the-features-of-reactjs* Pristupano: 11.07.2023.

[23] *simplilearn – AngularJS vs. Angular 2 Vs Angular4: Key Differences* Dostupno na: *https://www.simplilearn.com/angularjs-vs-angular-2-vs-angular-4-differences-article* Pristupano: 21.08.2023.

[24] *altexsoft – The Good and the Bad of Angular Development* Dostupno na: *https://www.altexsoft.com/blog/engineering/the-good-and-the-bad-of-angular-development/* Pristupano: 28.08.2023.

[25] *TJ WEBDEV – Why React JS is a Library not Framework* Dostupno na: *https://www.tjwebdev.in/blogs/why-react-js-is-a-library-not-framework* Pristupano: 02.09.2023.

[26] *TJ WEBDEV – Framework VS Library* Dostupno na: *https://www.tjwebdev.in/blogs/framework-vs-library* Pristupano: 04.09.2023.

[27] *ekobit – Angular Architecture Tutorial – Prvi dio* Dostupno: *https://www.tjwebdev.in/blogs/framework-vs-library* Pristupano: 04.09.2023.

[28] *freeCodeCamp – Front End JavaScript Development handbook – React, Angular, and Vue Compared* Dostupno: *https://www.freecodecamp.org/news/front-end-javascript-development-react-angular-vue-compared/* Pristupano: 28.08.2023.

[29] *React Legacy – Docs* Dostupno: *https://legacy.reactjs.org/docs/getting-started.html* Pristupano: 28.08.2023.

[30] *Kinsta – Angular vs React: A Detailed Side-by-Side Comparison* Dostupno: *https://kinsta.com/blog/angular-vs-react/#8-state-management* Pristupano: 28.08.2023.

[31] *Syncfusion – React useState Vs. Context API: When to Use Them* Dostupno: *https://www.syncfusion.com/blogs/post/react-usestate-vs-context-api.aspx* Pristupano: 07.09.2023.

[32] *React Router – Docs* Dostupno: *https://reactrouter.com/en/main/start/tutorial* Pristupano: 07.09.2023.

[33] *LogRocket – What is the virtual DOM in React?* Dostupno: *https://blog.logrocket.com/virtual-dom-react/* Pristupano: 09.08.2023.

[34] *GeeksForGeeks – Difference between Virtual DOM and Real DOM* Dostupno: *https://www.geeksforgeeks.org/difference-between-virtual-dom-and-real-dom/* Pristupano: 13.08.2023.

[35] *Formik - Docs* Dostupno: *https://formik.org/* Pristupano: 27.06.2023.

[36] *NPM – Yup* Dostupno: *https://www.npmjs.com/package/yup* Pristupano: 27.06.2023.

[37] *GitHub – Angular* Dostupno: *https://github.com/angular/angular* Pristupano: 12.09.2023.

[38] *GitHub – React* Dostupno: *https://github.com/facebook/react* Pristupano: 12.09.2023.

[39] *State of JS 2022 – Front-End Framewroks* Dostupno: *https://2022.stateofjs.com/en-US/libraries/front-end-frameworks/* Pristupano: 12.09.2023.

[40] *Stack Overflow Trends* Dostupno: *https://insights.stackoverflow.com/trends?tags=reactjs%2Cangular%2Cangularjs* Pristupano: 12.09.2023.

Popis slika