

Sveučilište u Rijeci – Odjel za informatiku

Diplomski studij informatike - Informacijski i komunikacijski sustavi

Luka Matuzić

Izrada full stack e-commerce aplikacije pomoću tehnologija Vue.js i Firebase

Diplomski rad

Mentorica: prof. dr. sc. Nataša Hoić – Božić

Rijeka, rujan 2019.

Rijeka, 6.6.2019.

Zadatak za diplomski rad

Pristupnik: Luka Matuzić

Naziv diplomskog rada: Razvoj full stack web aplikacije u Vue.js i Firebase tehnologijama

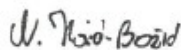
Naziv diplomskog rada na eng. jeziku: Development of full stack web application in Vue.js and Firebase technologies

Sadržaj zadatka:

Tehnologije Vue.js i Firebase za izradu i razvoj full stack web aplikacija potrebno je odgovarajuće povezati i uskladiti kako bi bilo moguće funkcioniranje ovakvih aplikacija. Zadatak diplomskog rada je prikazati na koji način funkcioniraju navedene tehnologije i kako ih treba uskladiti na primjeru izrade web aplikacije za online naručivanje hrane. Opisati će se čitavi postupak razvoja web aplikacije pomoću navedenih tehnologija te njihovo međudjelovanje i povezanost. Bit će navedeni svi koraci izrade same aplikacije, komponente i njihova povezivost, izrada baze podataka i njezina CRUD funkcionalnost, autentifikacija administratora koji će upravljati aplikacijom, izrada košarice za pohranu odabranih proizvoda te na kraju povezivanje navedenih tehnologija kako bi aplikacija funkcionirala.

Mentor:

Prof. dr. sc. Nataša Hoić – Božić



Voditeljica za diplomske radove:

Izv. prof. dr. sc. Ana Meštrović



Komentor:

Zadatak preuzet: 6.6.2019.

(potpis pristupnika)



Sadržaj

Sažetak.....	1
Uvod.....	2
1. Vue.....	3
1.1. Kratak pregled sintakse.....	4
2. Vue i ostali Javascript okviri.....	8
2.1. Angular.....	8
2.2. React.....	9
2.3. Vue.....	10
3. Usporedba s drugim bazama podataka.....	11
3.1. MongoDB.....	11
3.2. MySQL.....	12
3.3. Firebase.....	13
4. Opis projekta.....	16
5. Postavljanje projekta (eng. <i>setup</i>).....	17
6. Izrada komponenti.....	20
6.1. Header.....	21
6.2. Home.....	22
6.3. Menu.....	23
6.4. Košarica.....	25
6.5. Admin.....	29
6.5.1. Novi proizvod.....	31
6.5.2. Narudžbe.....	34
6.5.3. Login.....	35
7. Baza podataka, autentifikacija i validacija.....	37
8. Rute.....	46
9. Vuex.....	48
Iskustva.....	55
Zaključak.....	57
Popis slika.....	58
Literatura.....	60

Sažetak

Cilj ovog diplomskog rada je prikazati kako pojedinačno, ali i međusobno rade tehnologije Vue.js i Firebase na temelju izrade full stack aplikacije za online naručivanje hrane.

Vue.js je progresivan javascript okvir (eng. *framework*) za izgradnju korisničkih sučelja, a Firebase je Googleova mobilna platforma koja se koristi za razvijanje aplikacija.

Ova aplikacija sadržavati će komponentu za naručivanje proizvoda od strane korisnika aplikacije koji će moći birati veličinu, broj i cijenu proizvoda koji će se spremati u košaricu te će imati mogućnost brisanja proizvoda iz košarice. Također, ova aplikacija će, osim korisničke, imati i administracijsku stranu preko koje će se imaginarni vlasnik ili djelatnik zamišljene tvrtke moći ulogirati sa svojim e-mailom i lozinkom u administracijsku sekciju koja će sadržavati: komponentu za dodavanje ili brisanje proizvoda iz aplikacije koje korisnik može kupiti, komponentu za prikaz trenutnih narudžbi koje su poslane od strane korisnika te će administrator imati ovlasti za brisanje narudžbi koje su obavljene.

Za ovaj projekt koristiti će se Firebase za backend te će se preko njega pohranjivati podaci proizvoda koji su za prodaju zajedno sa podacima svih narudžbi. Također, Firebase će se koristiti i za autentifikaciju korisnika, odnosno u ovom slučaju administratora koji će se moći prijaviti u administracijsku sekciju. Napraviti će se lokalno Vue skladište (eng. *store*) koje će biti sinkronizirano s Firebaseom kako bi se mogla kontrolirati stanja (eng. *states*) same aplikacije što će omogućiti pristup i upravljanje svim informacijama potrebnima za rad aplikacije, a sve te informacije će se nalaziti na jednom mjesto i upravo iz tog razloga je Vue skladište vrlo korisna stvar.

Za navigaciju po aplikaciji koristiti će se Vue usmjerivač (eng. *Vue Router*) kako bi se postavile određene rute za stranice koje će se nalaziti na samoj aplikaciji. Osim toga, prikazati će se prikaz rada čuvara navigacije (eng. *navigation guards*) te će se definirati njihovo ponašanje.

Ključne riječi: Vue, Firebase, full stack, e-trgovina, states, router, navigacija

Uvod

Vue se smatra jednim od najjednostavnijih JavaScript okvira koji je veoma brz, ima malu veličinu i veoma detaljnu dokumentaciju. Veoma je fleksibilan i prilagodljiv različitim vrstama projekata. Primjenjiv je na male i srednje projekte, međutim Vue dodavanjem odgovarajućih službenih komponenti postaje punopravni okvir, i kao takav ima potencijal za izradu aplikacija na poduzetnih aplikacija na velikim razinama.

Iako trenutno Vue nije još uvijek popularan kao React (kojeg podržava Facebook) ili Angular (kojeg podržava Google), njegova popularnost raste iz dana u dan, a Laravel zajednica (eng. *community*) ga smatra za jedan od njihovih omiljenih frontend okvira.

Firebase Real Time Database je baza podataka u oblaku (eng. *cloud*) koja sve podatke pohranjuje u JSON (eng. *JavaScript Object Notation*) obliku koji predstavlja format razmjene podataka.

Radi se o bazi podataka u stvarnom vremenu, što znači da kad god se dogodi bilo kakva promjena unutar baze na poslužitelju – uređaj se automatski ažurira sinkronizirano i paralelno s tom bazom. Zahvaljujući tome, aplikacije koje koriste Firebase bazu podataka rade puno brže od onih “klasičnijih” koje koriste SQL baze podataka, jer se podaci preuzimaju sa servera paralelno tijekom korištenja aplikacije, tako da kada korisnik želi učitati određene podatke, oni se učitavaju direktno iz memorije uređaja, a ne sa poslužitelja. Ukoliko korisnik izgubi vezu s internetom, podaci se spremaju na uređaj te su dostupni bez obzira da li je na vezi ili ne.

Iz ovih razloga koristit ću ove tehnologije kako bi napravio aplikaciju za online naručivanje hrane te ću prikazati kako povezati i uskladiti ove tehnologije kako bi bi aplikacija funkcionirala. Opisati će se čitavi postupak razvoja web aplikacije kao i svi koraci izrade aplikacije uključujući izradu komponentata, baze podataka te CRUD funkcionalnosti, kao i autentifikacije i validacije administratorske sekcije unutar koje će administrator imati mogućnost izrade jelovnika koji će sadržavati proizvode koji će biti sinkronizirani s Firebase Real Time bazom podataka kako bi aplikacija cijelo vrijeme bila sinkronizirana i funkcionalna.

1. Vue

Vue je kreirao Evan You dok je radio u Googleu na AngularJS (Angular 1.0) aplikacijama, a nastao je iz potrebe za stvaranjem uspješnijih i efikasnijih aplikacija. Ono što Vue čini posebnim je to što taj okvir zapravo predstavlja mješavinu Angulara i Reacta te je to jedan od glavnih razloga zašto je ovaj Javascript okvir jedan od najpopularnijih među developerima.

Vue je progresivni okvir (eng. *framework*) za izgradnju korisničkih sučelja. Za razliku od drugih Javascript okvira, Vue je dizajniran od temelja da bude postupno prihvatljiv. Sama jezgra Vue-a fokusirana je samo na sloj prikaza te se lako uči i integrira s drugim bibliotekama (eng. *library*) ili postojećim projektima. S druge strane, Vue se može bez ikakvih problema koristiti u kombinaciji s modernim alatima i podržavajućim bibliotekama. [1]

Osim što uzima neke stvari od Angulara i Reacta (npr. virtualni DOM, komponente...), Vue također izbacuje sve ono što se smatra lošim unutar ta dva okvira, a to su, u najvećoj mjeri, JSX i TypeScript. Prvo, Vue ne zahtijeva učenje novog programskog jezika kao što to zahtijeva Angular s TypeScriptom već se sve odvija pomoću Javascripta. Drugo, JSX je puno kompleksniji za korištenje od običnog predloška (eng. *template*) unutar Vue-a te ne zahtijeva kombinaciju sintakse Javascripta i HTML-a već se njihova sintaksa odvaja unutar 3 oznake (eng. *tag*) : **template** – koristi se za pisanje HTML-a, **script** – koristi se za pisanje Javascripta te **style** – koristi se za pisanje CSS-a. Iz toga se također može primjetiti kako je Vue vrlo jednostavnog rasporeda te se lako navigira po samim komponentama.

Također, bitno je napomenuti upravljanje stanjima, što je temelj sva 3 navedena Javascript okvira. Vue je u tom slučaju više naklonjen Reactu te uzima njegovu arhitekturu i koncepte koji se toga tiču.

Osim navedenih stavki, vrlo bitna stvar je to što Vue nije podupiran od strane velikih korporacija kao što je to slučaj kod Angulara i Reacta, a prednost toga je to što je čitav koncept Vue-a potpomognut zajednicom developera koji ga koriste te oni svojim radom i kritikama sudjeluju u razvoju ovog okvira te sprečavaju da Vue-ov razvoj bude kontroliran od nekolicine ljudi. Situacija je dosta slična onoj između Windowsa i Linuxa – Windows predstavlja proizvod velike korporacije koji je napravljen unutar same korporacije te se razvija u smjeru koji ovisi o

korporaciji kao takvoj budući da se ne uvažavaju korisničke ideje, za razliku od Linuxa koji je u potpunosti otvorenog tipa te svatko može sudjelovati u pridonnošenju njegovog razvoja i u potpunosti je besplatan.

1.1. Kratak pregled sintakse

Vue ima vrlo dobar raspored prikazivanja elemenata. Iako se u jednoj datoteci mogu koristiti HTML, CSS i Javascript, sve je raspoređeno na elegantan način te ne dolazi do konfuzije oko toga što se gdje koristi unatoč uobičajenoj konvenciji kako bi datoteke različitog tipa trebale biti odvojene jedna od druge.

```
<template>
  <p>{{ greeting }} World!</p>
</template>

<script>
module.exports = {
  data: function () {
    return {
      greeting: 'Hello'
    }
  }
}
</script>

<style scoped>
p {
  font-size: 2em;
  text-align: center;
}
</style>
```

Slika 1: Prikaz rasporeda oznaka unutar Vue datoteke

Ova slika najbolje prikazuje prikaz rasporeda elemenata unutar 3 navedene oznake: **template**, **script** i **style**. **Template** oznaka označava područje za rad s HTML-om unutar kojeg se mogu postavljati dinamički generirani podaci pomoću dvostruke vitičaste zagrade iz **script** oznake unutar koje se odvija čitava logika svake komponente, odnosno rad s Javascriptom. Unutar **style** oznake koristi se CSS te CSS predprocesori poput SASS-a ili LESS-a te oni utječu na izgled kojeg korisnik vidi na ekranu.

Data instanca mora biti funkcija tako da svaka druga instanca može održavati neovisnu kopiju vraćenog podatkovnog objekta te ona mora sadržavati uobičajenu Javascript naredbu *return* koja vraća vrijednost funkcije u kojoj se ta naredba nalazi. [2]

Unutar **script** oznake, osim **data** instance, mogu se nalaziti i neke druge instance:

- **watch** - funkcije koje omogućuju nadgledanje jednog određenog entiteta i obavljanje određenih akcija nakon što se taj entitet promijeni nakon što se promijeni. Ove funkcije ne vraćaju nikakvu vrijednost. [3]
- **computed** – svojstva koja će se automatski ažurirati nakon što se njihove ovisnosti promijene. Ne prihvataju nikakve argumente i moraju vratiti najviše jednu vrijednost. [3]
- **methods** - statičke funkcije koje se pokreću jednom kada su pozvane. Mogu prenositi argumente, a oni mogu, ali i ne moraju vratiti vrijednost [3]

Osim navedenih instanci, jedna od najbitnijih stavki programiranja u Vue okvisu su direktive. Direktive su posebni HTML atributi koji počinju s **v-**, a koriste se za povezivanje Javascript izraza s HTML elementima. [4] Neke od najbitnijih direktiva su:

- **v-bind** - dinamički veže HTML atribut za Javascript izraz. To vezanje predstavlja zamjenu stvarne vrijednosti nakon što je ona sastavljena. [4]


```

<template>
  <div>
    <!-- binds src attribute to an expression -->
    

    <!-- v-bind shorthand syntax -->
    
  </div>
</template>

<script>
export default {
  data: function() {
    return {
      dogImage: "https://i.imgur.com/A8eQs1l.jpg"
    };
  }
};
</script>

```

Slika 2: Prikaz v-bind direktive

- **v-on** - koristi se za pridruživanje Javascript “slušatelja” događaja (eng. *event listener*) na HTML element, najčešće su to klik, prijelaz strelicom miša preko određenog elementa i sl. [4]

```

<template>
  <div>
    <h1>{{ title }}</h1>
    <!-- <button v-on:eventname="eventhandlerName">title</button> -->
    <button v-on:click="handleClick">Change title</button>
  </div>
</template>

<script>
export default {
  data: function() {
    return {
      title: "Hello"
    };
  },
  methods: {
    handleClick: function() {
      this.title = "Hello Vue";
    }
  }
};
</script>

```

Slika 3: Prikaz v-on direktive

- **v-model** - stvara dvostruko povezivanje podataka za elemente koji se nalazi unutar određene forme, obično HTML input elementa. [4]

```
<template>
  <div>
    <input v-model="name" placeholder="Name" />
    <p>{{ name }}</p>
  </div>
</template>

<script>
export default {
  data: function() {
    return {
      name: "King"
    };
  }
};
</script>
```

Slika 4.: Prikaz v-model direktive

- **v-if** – omogućuje da se elementi HTML-a dopune ovisno o istinitosti vrijednosti određenog izraza. [4]

```
<template>
  <div>
    <h1 v-if="isActive">Conditional rendering</h1>
    <button v-on:click="isActive = !isActive;">Show</button>
  </div>
</template>

<script>
export default {
  data: function() {
    return {
      isActive: false
    };
  }
};
</script>
```

Slika 5: Prikaz v-if direktive

- **v-for** - koristi se za rad petlje, odnosno iteriranje nad nizom elemenata te njihovo stavljanje u DOM. [4]

```

<template>
  <ul>
    <!-- list rendering starts -->
    <li v-for="user in users">{{user.name}}</li>
  </ul>
</template>

<script>
  export default{
    data:function(){
      return{
        users:[
          {id:1,name:"king"},
          {id:2,name:"gowtham"},
          {id:3,name:"oops"},
        ]
      }
    }
  }
</script>

```

Slika 6: Prikaz v-for direktive

2. Vue i ostali Javascript okviri

U ovoj sekciji navesti će se najpopularniji Javascript okviri i među njima Vue kao odabran za izradu ovog projekta. Ukratko će se opisati i međusobno usporediti sva tri okvira te će se na taj način bolje pojasniti što određeni okvir radi te koje su njihove prednosti i nedostaci.

2.1. Angular

Angular je razvijen od strane Google-a te je prvi put objavljen 2010. godine, što ga čini najstarijim frameworkom od spomenuta tri.

Prednosti: [8]

- Velika zajednica (eng. *community*)

- Kôd koji je u vlasništvu Google-a koji ga i održava
- Sadrži detaljnu dokumentaciju koja omogućuje dobivanje svih potrebnih informacija
- Jednosmjerno vezanje podataka koje omogućuje jedinstveno ponašanje aplikacije koje minimizira rizike mogućih pogrešaka
- MVVM (Model-View-View-Model) koji razvojnim programerima omogućuje da rade odvojeno u istom odjeljku aplikacije koristeći isti skup podataka
- Struktura i arhitektura posebno su stvoreni za veliku skalabilnost projekta
- Vrlo je tražen kod novih poslovnih prilika

Nedostaci:

Kod Angulara u svakom trenutku izrade aplikacije postoji više načina izrade određenih dijelova programa. To u nekim slučajevima može biti dobro, ali problem nastaje iz razloga što ne postoji standard rješavanja ponavljajućih problema te se programeri u većini slučajeva moraju oslanjati na dokumentaciju te kreiranje vlastitih rješenja umjesto standardiziranog načina. Osim toga, dodatan otežavajući faktor je to što Angular zahtijeva upotrebu Typescripta, što znači da je osim Javascripta potrebno naučiti još jedan programski jezik.

2.2. React

React je najpoznatija Javascript biblioteka s najvećom zajednicom i ekosustavom. U Reactu se koristi isključivo Javascript. Sve komponente izražavaju svoje korisničko sučelje u **render** funkcijama koristeći JSX (deklarativna XML-sintaksa koja se koristi unutar JavaScripta) za spajanje HTML strukture s logikom. [6] Omogućuje da se sastave složena korisnička sučelja iz malih i izoliranih dijelova kôda koji se nazivaju komponente (eng. *components*).

Prednosti:

- Velika zajednica
- Kôd napravljen i održavan od strane Facebooka

- Veliki ekosistem
- Fleksibilnost i skalabilnost
- Može se koristiti i za mobilne i za web aplikacije
- Srednja krivulja učenja (lakši za učiti od Angulara)
- Koristi virtualni DOM - programski koncept u kojem prikaz korisničkog sučelja čuva u memoriji i sinkronizira s "pravim" DOM-om [7]

Nedostaci:

Zahtijeva učenje JSX-a, a to je ekstenzija sintakse Javascripta koja se na jednostavan način može opisati kao sintaksa unutar koje se može pisati i Javascript i HTML unutar iste datoteke. Osim toga, najveći problem je sličan kao i kod Angulara – ne postoji standardiziran način rješavanja problema. [9] Ovaj problem najviše dolazi do izražaja u situaciji kada na određenoj aplikaciji radi novi tim ljudi, dakle tim koji nije sudjelovao u izradi originalne verzije aplikacije. Problem je u tome što svaki programer ima svoj smjer razmišljanja i stil pisanja te je vrlo teško uklopiti više stilova u jednu aplikaciju. Standardizirana metodologija je ono što nedostaje u takvim situacijama.

2.3. Vue

Vue je također jedna od najbrže rastućih Javascript biblioteka. Njegova brzina prikazivanja (eng. *render*), učinkovit rad komponenti i jednostavnost kôda neke su od njegovih glavnih prednosti. Vue je dizajniran da bude inkrementalno prihvatljiv što znači da je jezgra biblioteke usmjerena samo na sloj prikaza, ali još uvijek dopušta da se projekti savladavaju dodavanjem dodatnih paketa. Ono što odlikuje Vue je njegova jednostavnost. [7]

Prednosti:

- Ima podršku Laravela i Alibabe – velike tvrtke svakoga dana sve više daju priliku Vue-u
- Jednostavnost sintakse
- Lagan je za učiti, intuitivan
- Jednostavna struktura

- Sadrži mnogo koncepata iz Angulara i Reacta – može se reći kako je Vue mješavina onog najboljeg iz oba frameworka
- Fleksibilnost – može se koristiti i sa Typescriptom i JSX-om
- Kao i React, koristi virtualni DOM

Nedostaci:

Vue je relativno mlad Javascript okvir te evoluira nevjerovatno brzo. Upravo zato što toliko brzo evoluira, većina tutorijala koji se nalaze na internetu vrlo brzo postanu zastarjeli što otežava čitavi proces učenja te na kraju krajeva i same izrade aplikacije, neovisno o kojoj se radi. Iako je vrlo popularan, još uvijek nije raširen i popularan u tolikoj mjeri kao Angular i React.

3. Usporedba s drugim bazama podataka

U ovoj sekciji navesti će se najpopularnije baze podataka te među njima i Firebase koji je izadabran za izradu ovog projekta. Ukratko će se opisati i međusobno usporediti sve tri baze podataka.

3.1. MongoDB

MongoDB je baza podataka dokumenata sa skalabilnošću i fleksibilnošću s upitima i indeksiranjem. MongoDB pohranjuje podatke u fleksibilne dokumente slične JSON-u, što znači da se polja mogu razlikovati od dokumenta do dokumenta, a struktura podataka može se mijenjati tijekom vremena. Model dokumenta mapira objekte u kôdu aplikacije, što olakšava rad s podacima. Ad hoc upiti, indeksiranje i agregacija u stvarnom vremenu pružaju izvrsne načine za pristup i analizu podataka. MongoDB je u svojoj srži distribuirana baza podataka, tako da je ugrađena visoka dostupnost, horizontalno skaliranje i geografska distribucija te jednostavna upotreba. [11]

Prednosti

MongoDB je baza podataka bez sheme. To znači da može imati bilo koju vrstu podataka u zasebnom dokumentu. Ova stvar daje fleksibilnost i slobodu pohrane podataka različitih vrsta. Može se pohraniti velika količina podataka tako da ih se distribuira na nekoliko poslužitelja spojenih na aplikaciju. Ako poslužitelj ne može obraditi tako velike podatke, tada neće postojati uvjet kvara. Izraz koji se ovdje možemo upotrijebiti je "automatsko izoštravanje". MongoDB je također baza podataka orijentirana na dokumente. Lako je pristupiti dokumentima indeksiranjem, stoga pruža brz odgovor na upit. Brzina MongoDB-a 100 puta je veća od klasične relacijske baze podataka.[10]

Nedostaci:

MongoDB ne podržava spajanje podataka poput relacijske baze podataka. Ipak, može se pridružiti funkcionalnosti dodavanjem ručnim kodiranjem, ali također može usporiti izvršenje i utjecati na performanse. Ima ograničenu veličinu podataka koja iznosi 16MB. [10]

3.2. MySQL

MySQL je relacijska baza otvorenog tipa (eng. *open source*). To je najpoznatija i najkorištenija baza podataka na svijetu te ju koriste neke od najvećih svjetskih korporacija poput YouTube-a, Facebook-a, Google-a, Adobe-a, i sl.

Prednosti:

80% cijelog weba koristi MySQL bazu podataka te iz tog razloga ima izvrsnu podršku. Vrlo je jednostavan za korištenje. Većina zadataka može se obaviti u naredbenom retku, većinu vremena ne treba koristiti grafičko sučelje. Budući da u MySQL-u zaista nema mnogo složenih stvari koje je potrebno učiniti, ako se treba samo uvoziti i izvoziti podatke, uslužni program `mysqldump` je moćan i jednostavan za korištenje te se on koristi također u terminalu. Cjelokupna arhitektura baze podataka vrlo je jednostavna i kompaktna. MySQL općenito ima bolje performanse u jednostavnim upitima koji se svakodnevno koriste, kao što su pretrage primarnih ključeva, upiti raspona itd. [10]

Nedostaci:

MySQL nije osmišljen da bude skalabilan. Ako se očekuje da će se aplikacija koju netko koristi povećati, morat će razmotriti odluku o korištenju MySQL-a kao svoje baze podataka. Može odlično raditi na početku, ali s vremenom postane prekrcata. Možda se želi podijeliti podatke, tj. distribuirati podatke iz jedne tablice u više instanci i računala, ali MySQL ne podržava automatsko izoštravanje te se čvorovi moraju održavati ručno.

3.3. Firebase

Firebase je platforma za razvoj mobilnih i web aplikacija koju je 2011. razvila tvrtka Firebase, Inc., a zatim ga je Google kupio 2014. Od listopada 2018., Firebase platforma ima 18 proizvoda, koji koriste 1,5 milijuna aplikacija. [3]

Jedan od glavnih razloga zašto sam izabrao Firebase, a ne neku drugu bazu podataka, je to što Vue ima podršku za Firebase i obrnuto. Vrlo je jednostavno postaviti projekt koji je sinkroniziran sa Vue.js-om i Firebaseom upravo zbog modula koji omogućuju njihovo

jednostavno korištenje. Ostale baze se također mogu koristiti s Vue.js-om, međutim gubi se efikasnost, jednostavnost i brzina izvođenja upita.

Firebase omogućuje programerima da se usredotoče na kreiranje korisničkih iskustava umjesto da se bave podacima unutar tablice. Kada se koristi Firebase, ne mora se upravljati poslužiteljima, ne moraju se pisati API-jevi. Firebase je sam po sebi poslužitelj API i baza podataka te se sve može modificirati da odgovara većini potreba koje su potrebne tokom izrade aplikacije.

Firebase nudi niz usluga:

Analitika - omogućuje uvid u izvještaje do 500 slučajeva te zahvaća ključna korisnička svojstva te mjeri određene izvještaje prema unaprijed određenim parametrima:

- Firebase analitika (eng. *Firebase Analytics*) - besplatno rješenje za mjerenje aplikacija koje pruža uvid u korištenje aplikacije i angažman korisnika. [5]

Razvoj – odnosi se na razvoj aplikacija na različitim platformama: iOS, Web, Android...:

- Firebase cloud slanje poruka (eng. *Firebase Cloud Messaging*) - Prethodno poznat kao Google Cloud Messaging (GCM), Firebase Cloud Messaging (FCM) je cross-platformsko rješenje za poruke i obavijesti za Android, iOS i web aplikacije, koje se od 2016. mogu koristiti bez troškova. [5]
- Firebase Auth - Firebase Auth je usluga koja može autentificirati korisnike koji koriste samo klijentsku stranu. Podržava pružatelje društvenih usluga poput Facebooka, GitHuba, Twittera i Googlea (i Google Play igre). Osim toga, to uključuje sustav za upravljanje korisnicima kojim programeri mogu omogućiti provjeru autentičnosti korisnika putem e-pošte i prijave zaporke pohranjene s Firebaseom. [5]
- Firebase baza podataka u realnom vremenu (eng. *Firebase Realtime Database*) - Firebase pruža bazu podataka u stvarnom vremenu i backend kao uslugu. Servis pruža programerima API koji omogućuje sinkronizaciju podataka u aplikacijama između klijenata i pohranjuje ih u Firebaseov oblak (eng. *cloud*). Tvrtka nudi knjižnice (eng. *libraries*) klijenata koje omogućuju integraciju s aplikacijama rađenih u sljedećim

tehnologijama: Android, iOS, JavaScript, Java, Objective-C, Swift i Node.js. Baza podataka je također dostupna putem REST API-ja i vezama za nekoliko JavaScript okvira (*eng. framework*) kao što su AngularJS, React, Vue.js, itd. [5]

- Firebase Cloud Firestore - 31. siječnja 2019. Cloud Firestore službeno je postao službenim proizvodom Firebase linije. Nasljednik je Firebaseovog izvornog sustava za baze podataka i baze podataka u realnom vremenu te omogućuje ugniježdene dokumente i polja umjesto stabla prikazanog u bazi podataka u realnom vremenu. [5]
- Firebase Storage - pruža siguran prijenos datoteka i preuzimanja za Firebase aplikacije, bez obzira na kvalitetu mreže. Programer ga može koristiti za pohranjivanje slika, audiozapisa, videozapisa ili drugog korisničkog sadržaja. Pohrana vatrozida podržava Google Cloud Storage. [5]
- Firebase Hosting - statična i dinamična web hosting usluga koja je pokrenuta 13. svibnja 2014. Ona podržava hosting statičkih datoteka kao što su CSS, HTML, JavaScript i druge datoteke, kao i podršku putem Cloud funkcija. Usluga isporučuje datoteke preko mreže za isporuku sadržaja (CDN) putem HTTP Secure (HTTPS) i Secure Sockets Layer enkripcije (SSL). [5]

Stabilnost – grupiranje grešaka i bugova u aplikaciji i bazi podataka, njihova analitika te prikaz eventualnog rješenja pomoću raznovrsnih testiranja kroz sve platforme:

- Crashlytics - Izvješćivanje o “padu” stvara detaljna izvješća o pogreškama u aplikaciji. Pogreške su grupirane u skupove sličnih tragova stogova i prikazane su ozbiljnošću utjecaja na korisnike aplikacije. Osim automatskih izvješća, programer može zabilježiti i prilagođene događaje kako bi lakše uhvatio korake koji su doveli do rušenja. [5]
- Firebase testni laboratorij za Android i iOS - Firebase Test Lab za Android i iOS osigurava infrastrukturu temeljenu na oblaku za testiranje Android i iOS aplikacija. Programeri jednom operacijom mogu pokrenuti testiranje svojih aplikacija na širokom rasponu uređaja i konfiguracija uređaja. Rezultati ispitivanja, uključujući zapise, videozapise i snimke zaslona, dostupni su u projektu u Firebase konzoli. Čak i ako programer nije napisao nikakav testni kôd za svoju aplikaciju, testni laboratorij može

izvršiti aplikaciju automatski, tražeći padove (eng. *crash*). Test Lab za iOS trenutno je u beta fazi. [5]

Prednosti:

Jedna od najvećih prednosti Firebasea je to što se vrlo lako instalira i koristi u bilo kojoj situaciji. Lako se dolazi do podataka, datoteka, informacija, lako se obavlja autentifikacija i validacija te ima ogromnu količinu moguće pohrane. Također, napravljen je od strane Googlea što znači da ima sjajnu podršku, vrlo je siguran te nije potreban server koji treba konfigurirati nego je dovoljno instalirati, uključiti u aplikaciju i spreman je za korištenje. Firebase je jedno od najnaprednijih BaaS (eng. *Backend as a Service*) poslužiteljsko rješenje. [10]

Nedostaci:

Kada se jednom napravi aplikacija s Firebase bazom podataka, vrlo teško se prebaciti na neko drugo rješenje budući da je Firebase Real Time Database jedna od rijetkih koja ima podatke spremljene u JSON obliku. Također ima vrlo ograničenu mogućnost upita i indeksiranja što dovodi u pitanje brzinu i skalabilnost baze podataka, a samim time i aplikacije na koju se ta baza odnosi. Budući da koristi JSON format za pohranu podataka, Firebase ne posjeduje mogućnost agregacije skupine podataka kao što to imaju neke više standardizirane baze podataka.

4. Opis projekta

Kao praktični dio diplomskog rada odabran je projekt čije je ideja za izradu sljedeća: napraviti full stack web aplikaciju za online naručivanje hrane pomoću Vue frameworka i Firebasea za trajno pohranjivanje podataka te provjeru autentičnosti. Dakle, korisnik ove web aplikacije bi mogao naručivati proizvod, u ovom slučaju hranu, preko interneta. Također, aplikacija bi sadržavala administracijsku opciju koja bi predstavljala vlasnika tvrtke/restorana koji bi se mogao ulogirati te bi na taj način imao mogućnost dodavanja i brisanja proizvoda koji bi se

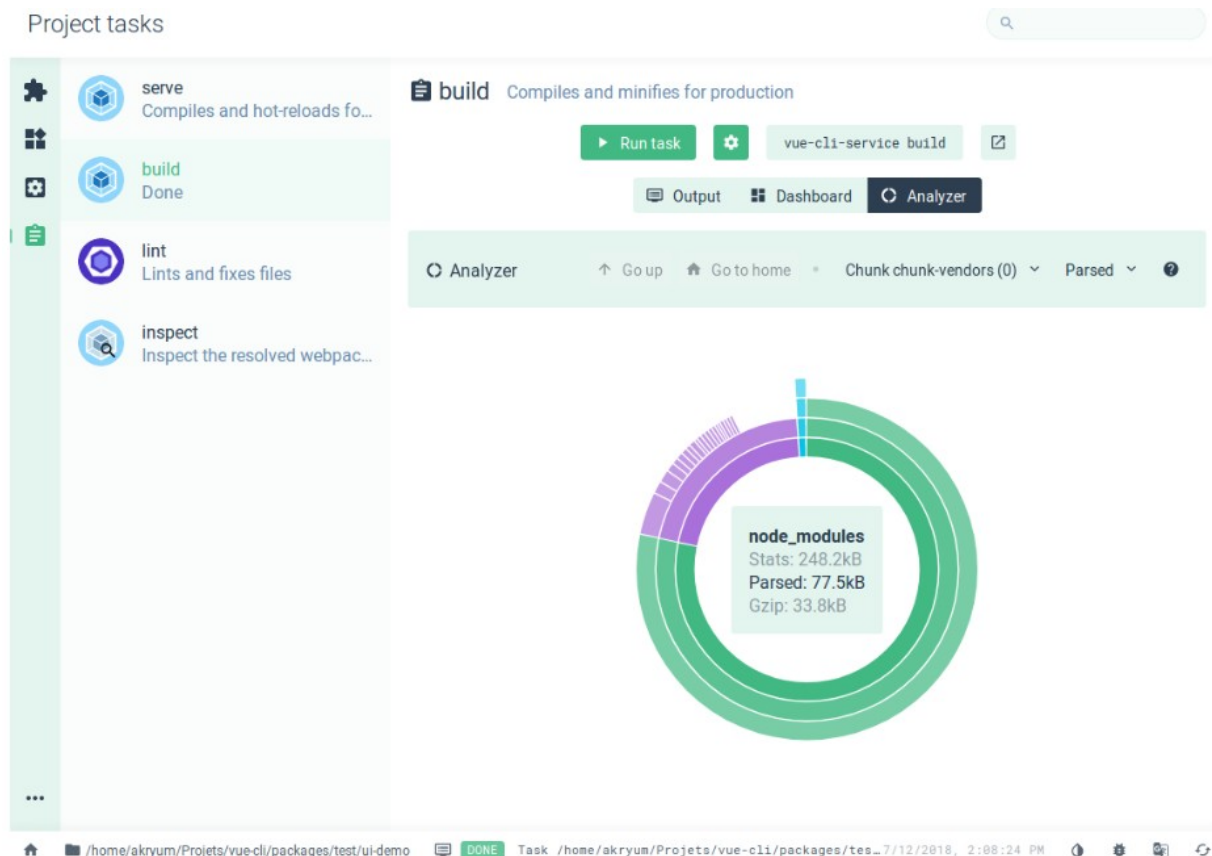
nalazili u samoj aplikaciji te koje bi korisnik mogao odabrati i spremiti u košaricu, vidjeti trenutne narudžbe korisnika koji su već naručili proizvode te informacije i količinu tih proizvoda te brisanje istih.

Za backend će se koristiti Firebase za bazu podataka, autentifikaciju, pohranu narudžbe te login, odnosno dopuštenje logiranja admina sa odgovarajućim podacima (e-mail i lozinka).

5. Postavljanje projekta (eng. *setup*)

Za početak, potrebno je instalirati Node.js. Node.js je open-source cross-platforma za JavaScript run-time okruženje koja izvršava JavaScript kôd izvan preglednika (eng. *browser*). Node.js omogućuje programerima da koriste JavaScript za pisanje naredbi i za skriptiranje na strani poslužitelja, odnosno omogućuje im pokretanje skripti na strani poslužitelja za izradu dinamičkog sadržaja web stranice prije slanja stranice web pregledniku korisnika. Prema tome, Node.js predstavlja "JavaScript svugdje" paradigmu, koja objedinjuje razvoj web aplikacija oko jednog programskog jezika umjesto različitih jezika za poslužiteljske i klijentske skripte. [10]

Nakon toga, u projektu (datoteci nazvanoj e-commerce-app) je potrebno instalirati Vue.js. To se čini preko Vue CLI-a (Command Line Interface). CLI je globalno instaliran **npm** paket i osigurava **vue** naredbu u terminalu. Pruža mogućnost brzog stvaranja novog projekta putem **vue** kreacije ili trenutnog prototipa novih ideja putem Vue servisa. Također može se upravljati vlastitim projektima koristeći grafičko korisničko sučelje preko Vue korisničkog sučelja. [12]



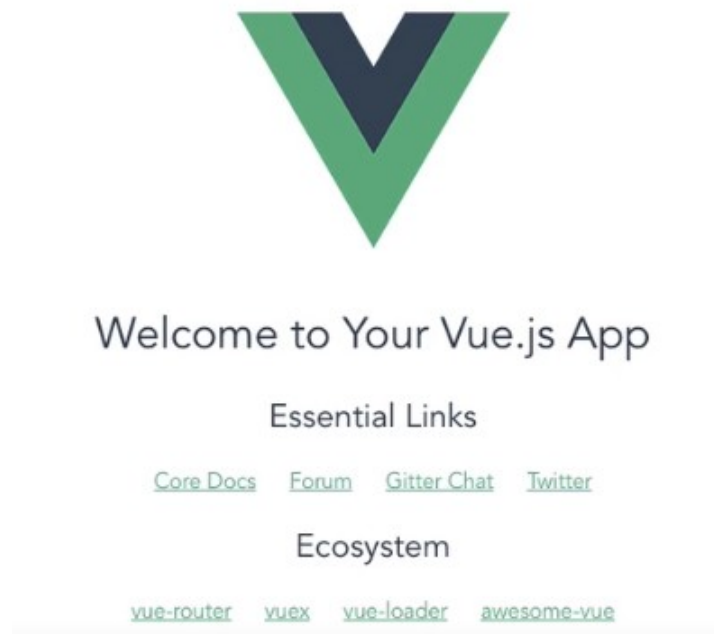
Slika 7: Vue grafičko korisničko sučelje (eng. Graphical User Interface, GUI)

Naredbom `vue create e-commerce` unutar CLI-a stvara se projekt naziva `e-commerce`. Nakon toga, Vue CLI traži da se odaberu unaprijed postavljene postavke koje se odnose na postavljanje projekta pomoću Babela i ESLinta. Babel je Javascript kompajler koji se uglavnom koristi za pretvaranje kôda ECMAScript 2015+ u unazad kompatibilnu verziju JavaScripta u trenutnim i starijim preglednicima ili okruženjima, a ESLint je priključiv (eng. *pluggable*) i konfigurabilan alat za identifikaciju i izvještavanje o uzorcima u JavaScriptu. Pomoću ovoga se može postaviti Vue projekt u samo nekoliko minuta.

```
Vue CLI v3.4.0
? Please pick a preset: (Use arrow keys)
> default (babel, eslint)
  Manually select features
```

Slika 8: Vue CLI - Odabir unaprijed odabranih postavki za novi Vue.js projekt

Kada Vue CLI postavi projekt te podesi sve module koji su potrebni za rad s napravljenim projektom, može se pokrenuti aplikacija pomoću naredbe `npm run serve`. Nakon izvršavanja ove naredbe, otvara se localhost na portu 8080 te aplikacija po defaultu izgleda ovako:



Slika 9: Defaultni izgled aplikacije prilikom stvaranja novog projekta

Za kodiranje, odnosno izradu ovog projekta korišten je Visual Studio Code editor koji je trenutno jedan od najpopularnijih na svijetu.

Unutar Vue projekta, postoje 3 dijela unutar kôda od kojih se sastoji gotovo svaka komponenta: **template**, **script** i **style**. Template oznaka se odnosi na HTML predložak, odnosno u njemu se nalazi HTML struktura pojedine komponente. Unutar script oznake odvijaju se Javascript aktivnosti koje se odnose na stvaranje objekata, nasljeđivanje komponenti, označavanje metoda,

stvaranje polja, funkcija, itd te se njime stvara logika koja utječe na korisničko iskustvo te efikasnost same aplikacije. Style oznaka koristi se za stiliziranje template dijela te se u njemu koristi CSS (Cascading Style Sheets).

Kako je plan bio da ova aplikacija bude responzivna, odnosno da se može prilagoditi svakom ekranu, koristio sam Bootstrap. Bootstrap je open source alat za razvoj s HTML, CSS i JS. Koristi se za brzu izradu prototipova, ideja ili za izgradnju cijele aplikacije pomoću Sass-a, odgovarajućim grid sustavom, već izgrađenim opsežnim komponentama i moćnim dodacima izgrađenima u jQuery-u. [14]

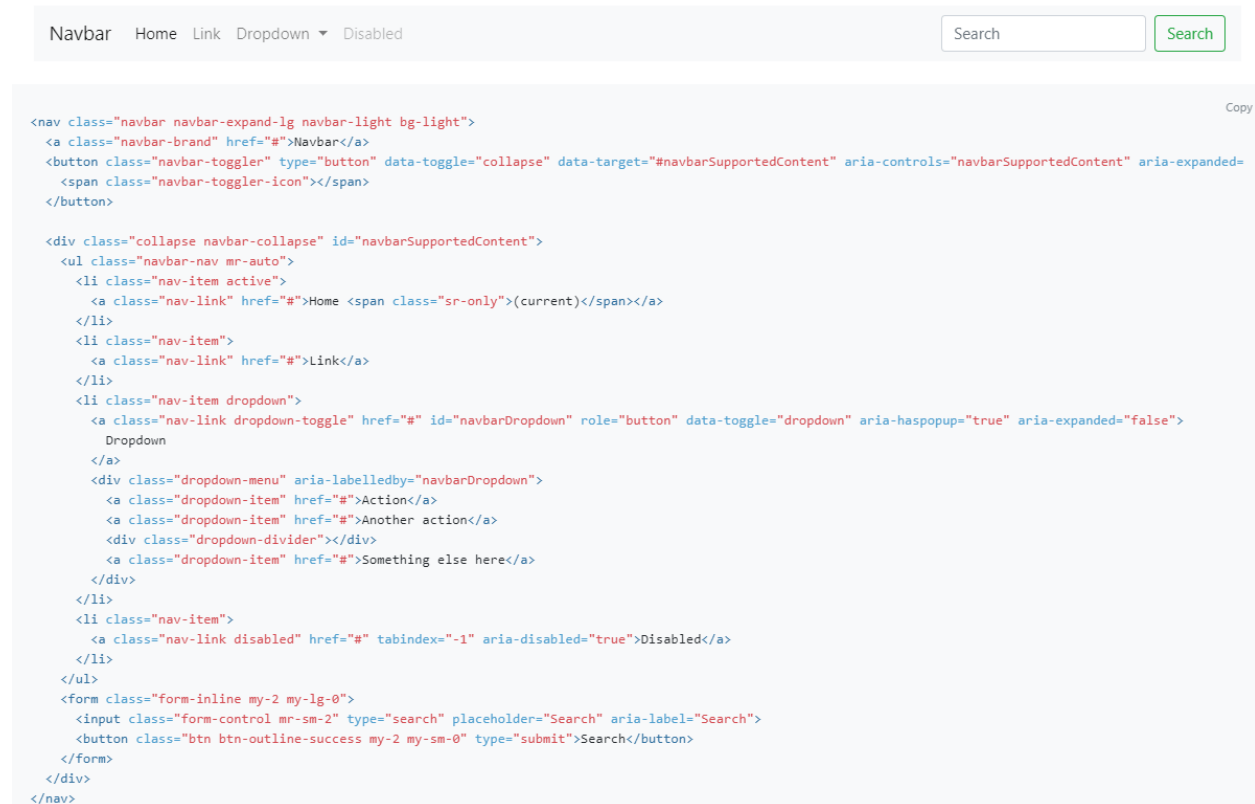
U nastavku će se objasniti izrada komponenti koje će međudobno komunicirati i izmjenjivati podatke kako bi aplikacija imala dinamičan i interaktivan osjećaj.

6. Izrada komponenti

Najbitnija komponenta ovog projekta bit će App.vue komponenta koja će biti jezgra funkcionalnosti te spajanja svih ostalih komponenata, kao i sinkroniziranja s Firebase bazom podataka te autentifikacijom administratora. Svaka pojedina komponenta mora biti uvedena u App.vue komponentu, inače same po sebi ne bi imale previše smisla jer ne bi od drugih komponenata dobivale potrebne podatke i informacije. Na početku projekta je svaka pojedina komponenta uvedena (eng. *import*) u App.vue datoteku jedna po jedna, a njihova hijerarhija kreće se od vrha prema dnu. Taj način se neće prikazivati u radu, nego će se kasnije opisati kako se to radi pomoću ruta, odnosno Vue routera, budući da je to konvencija kojom se vode gotovo svi programeri koji koriste Vue.js.

6.1. Header

Header komponenta će biti navigacijska traka koja će se pretvarati u ”hamburger” izbornik kada se veličina ekrana mijenja. Za navigacijsku traku koristi se sljedeći Bootstrap predložak:



Slika 10: Zadani Bootstrap predložak za navigacijsku traku

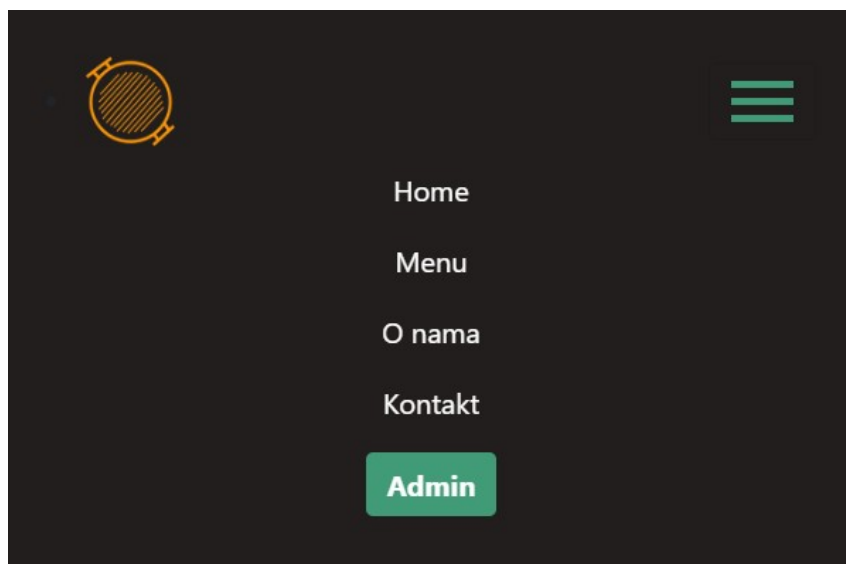
Iz ovog predloška se vidi HTML koji se koristi za prikaz navigacije te na vrhu sam izgled navigacijske trake. Uklonjene su određeni elementi poput dropdown izbornika, search gumba te search trake za pretraživanje budući da se neće koristiti u aplikaciji.

Nakon stiliziranja Header komponente, odnosno navigacijske trake, ona sada izgleda ovako:



Slika 11: Izgled navigacijske trake nakon stiliziranja

Za mobilnu verziju, navigacijska traka izgleda ovako:



Slika 12: Prikaz navigacijske trake u "hamburger" izborniku za mobilni pregled

6.2. Home

Home komponenta sastoji se od imena zamišljenog restorana, izmišljenog slogana, slike te najbitnije stavke home komponente – gumb "Naruči!" koji vodi korisnika u jelovnik restorana preko kojeg može naručiti hranu online.



Slika 1: Home komponenta nakon završetka stiliziranja

6.3. Menu

Menu komponenta će predstavljati jednu od, ako ne i najvažniju, komponentu u čitavoj aplikaciji unutar koje će korisnik moći pogledati koji se proizvodi nalaze u cjeniku, njihov opis, veličinu i cijenu. Menu komponenta će biti sinkronizirana s Firebase realtime bazom podataka kako bi se iz nje mogli povlačiti podaci ili u nju unositi novi podaci. Također, Firebase baza podataka će s druge strane biti sinkronizirana i sa Vuex store-om.

S lijeve strane jelovnika nalazit će se proizvodi koje će korisnik moći odabrati, odnosno spremiti u svoju košaricu koja će se nalaziti s desne strane jelovnika te će imati uvid u količinu odabranih proizvoda i ukupnu cijenu svih odabranih proizvoda. Nakon svog odabira, korisnik će moći izvršiti narudžbu klikom na gumb "Naruči!" te će se to izvršenje automatski poslati u Firebase te će na taj način administrator, u ovom slučaju vlasnik zamišljenog restorana, moći pregledati sve narudžbe koje su u tijeku te sve stavke koje se trenutno nalaze u jelovniku.

Menu komponenta podijeljena je u dva dijela što se tiče template oznake: menu i košarica (eng. *basket*).

```
<div class="col-sm-12 col-md-6">
  <table class="table table-hover">
    <thead class="thead-default">
      <tr>
        <th scope="col">Naziv</th>
        <th scope="col">Cijena</th>
        <th scope="col">Dodaj</th>
      </tr>
    </thead>

    <tbody v-for="item in getMenuItems" :key="item['.key']">
      <tr>
        <th>{{item.name}}</th>

        <tr>
          <td class="item-description">{{item.description}}</td>

          <tr v-for="option in item.options" :key="option['.key']">
            <td>{{option.size}}</td>
            <td>{{option.price}} kn</td>
            <td>
              <button class="btn btn-sm btn-outline-success" @click="addToBasket(item, option)">
                +
              </button>
            </td>
          </tr>
        </tbody>
      </table>
    </div>
```

Slika 2: Dio kôda koji se odnosi na Menu komponentu - menu - mjesto gdje korisnik bira proizvod

Za menu dio, odnosno dio u kojem korisnik može birati proizvode, koristi se **div** element s Bootstrap klasama **col-sm-12** i **col-md-6**, što označava svojevrsni responzivni kontejner, odnosno to znači da će jelovnik biti prikazan punom širinom na malim ekranima (sm, mobilni pregled), a na većim ekranima (md, srednja veličina) poput tableta, jelovnik će zauzimati jednu polovicu ekrana, a košarica s proizvodima drugu polovicu ekrana.

Za strukturu prikaza podataka koristioi se **table** element koji predstavlja tablični prikaz. U zaglavlju tablice (**thead**) biti će naslovi Naziv, Cijena i Broj. U tijelu tablice (**tbody**) nalazi se prikaz imena proizvoda, njegovog opisa, količine i cijene te gumb za dodavanje proizvoda u košaricu. U početku izrade aplikacije koristile su se statične vrijednosti za proizvode, npr. da određeni proizvod košta 20 kn te da je jumbo veličine, a razlog toga je testiranje funkcionalnosti prije prelaska na dinamične podatke iz baze podataka. Za prolaz kroz sve podatke kod statičnih podataka koristi se **for** direktiva u tijelu tablice na sljedeći način: **<tbody v-for="item in getMenuItems">**. Ova **for** petlja predstavlja klasični Vue.js način prolaska kroz podatke unutar određene strukture, u ovom slučaju objekta te se gotovo uvijek koristi **v-for** model. Za prolazak kroz opcije, odnosno za zamišljene veličine proizvoda, npr. normal i jumbo pizzu, koristi se sljedeća **for** direktivu: **<tr v-for="option in item.options">**.

Statični podaci nalazili su se u JSON objektu naziva **getMenuItems** sa svojim informacijama te je taj objekt predstavljao strukturu koja je odgovarala onoj u HTML predlošku iznad **script** oznake.

Objekt **getMenuItems** sa statičnim podacima je izgledao ovako:

```
1:{
  'name': 'Čevapi',
  'description': 'Juneće meso lepinja, ajvar, luk',
  'options': [{
    'size': 5,
    'price': 25
  }, {
    'size': 10,
    'price': 35
  }]
},
2: {
  'name': 'Burger',
  'description': 'Juneća pljeskavica 250g, pecivo sa sezamom, karamelizirani luk, Cheddar sir, umak po želji',
  'options': [{
    'size': 1x pljeskavica,
    'price': 35
  }]
```

```

      }, {
        'size': '2x pljeskavica',
        'price': 45
      }
    ],
  },
  3: {
    'name': 'Hot Dog',
    'description': 'Pecivo, kobasica, Cheddar sir, salata, umak po želji',
    'options': [{
      'size': 'normal',
      'price': 17
    }, {
      'size': 'jumbo',
      'price': 25
    }]
  }
}

```

6.4. Košarica

Dosadašnji dio odnosio se na proizvode koji će se nalaziti u jelovniku. Sada je potrebno napraviti košaricu koja će se nalaziti s desne strane u odnosu na jelovnik. Kada korisnik klikne na + gumb koji se nalazi do svakog pojedinog proizvoda, taj odabrani proizvod će se pojaviti u košarici.

Prvo je potrebno deklarirati prazno polje unutar kojeg će se spremati odabrani proizvodi i podaci o njima → **basket**: []. Nakon toga, potrebno je definirati klik upravitelj (eng. *click handler*) unutar dijela za jelovnik kako bi se omogućilo dodavanje proizvoda u košaricu:

```

<button class="btn btn-sm btn-outline-success" @click="addToBasket(item, option)">
  +
</button>

```

Slika 3: + gumb koji se odnosi na dodavanje proizvoda iz jelovnika u košaricu

Dakle, click handler je @click, a metodi koju ću koristiti dao sam naziv **addToBasket**, a argumenti su **item** i **option**. Metode se u Vue.js-u dodaju u **script** oznaku koja označava rad s Javascriptom:

```

methods: {
  addToBasket(item, option){
    this.basket.push({
      name: item.name,
      price: option.price,
      size: option.size,
      quantity: 1
    })
  },
}

```

Slika 4: Deklaracija metode unutar Vue.js-a: metoda za dodavanje proizvoda u košaricu

Metoda `push()` u Javascriptu dodaje nove stavke na kraj niza (polja) te vraća njegovu novu duljinu, ovisno o količini dodanih stavki. Budući da sam u početku radio sa statičnim podacima unutar objekta, košarica je bila samo polje s tim statičnim podacima iz objekta:

```

[
  {
    'name': 'Čevapi',
    'description': 'Juneće meso lepinja, ajvar, luk',
    'options': [
      { 'size': 5, 'price': 25 },
      { 'size': 10, 'price': 35 }
    ]
  }
]

```

Kako bi se riješio problem prikaza odabranih proizvoda, napravljeno je sljedeće:

```

<!-- SHOPPING BASKET -->
<div class="col-sm-12 col-md-6">
  <div v-if="basket.length > 0">
    <table class="table">
      <thead class="thead-default">
        <tr>
          <th scope="col">Količina</th>
          <th scope="col">Proizvod</th>
          <th scope="col">Ukupno</th>
        </tr>
      </thead>
      <tbody v-for="item in basket" :key="item.id">
        <tr>
          <td>
            <button class="btn btn-sm" type="button" @click="decreaseQuantity(item)"> - </button>
            <span> {{ item.quantity }} </span>
            <button class="btn btn-sm" type="button" @click="increaseQuantity(item)"> + </button>
          </td>
          <td>{{ item.name }}</td>
          <td>{{ item.price * item.quantity }} kn</td>
        </tr>
      </tbody>
    </table>
    <p>Total: {{ total }} kn</p>
    <button class="btn btn-success btn-block" @click="addNewOrder">Naruči!</button>
  </div>
  <div v-else>
    <p>{{ basketText }}</p>
  </div>
</div>

```

Slika 5: Prikaz koda za spremanje proizvoda u košaricu te sami izgled košarice

Dodan je kontejner koji će sadržavati sve podatke i informacije o odabranim proizvodima. Kao i kod jelovnika, za prikaz responzivnog kontejnera koriste se Bootstrap klase `col-sm-12` i `col-md-6`, `for` direktiva za prolaz kroz proizvode te tablični prikaz sa zaglavlјima Količina, Proizvod i Cijena koji će predstavljati informacije o odabranim proizvodima.

Ispod glavne `div` oznake nalazi se još jedan `div` s `if` direktivom koji se odnosi na to da ako korisnik nema nijedan odabran proizvod, da se prikaže tekst ‘Vaša košarica je prazna.’ Taj tekst nalazi se u objektu naziva `basketText` te se nalazi u `script` predlošku te unutar `data ()` funkcije koja vraća određene vrijednosti:

```

data () {
  return {
    options: '',
    basket: [],
    basketText: 'Vaša košarica je prazna.'
  }
},

```

Slika 6: Prikaz `data ()` funkcije, praznog polja košarice te teskta košarice ako korisnik nije odabrao niti jedan proizvod

Na slici 14 vide se dvije nove metode naziva `decreaseQuantity` i `increaseQuantity`. Kako im samo ime kaže, ove metode odnose se na smanjivanje ili povećanje količine odabranih proizvoda koji se nalaze u košarici. One su deklarirane na sljedeći način:

```
methods: {
  addToBasket(item, option){
    this.basket.push({
      name: item.name,
      price: option.price,
      size: option.size,
      quantity: 1
    })
  },
  increaseQuantity(item){
    item.quantity++;
  },
  decreaseQuantity(item){
    item.quantity--;
    if(item.quantity === 0){
      this.removeFromBasket(item);
    }
  },
  removeFromBasket(item){
    this.basket.splice(this.basket.indexOf(item), 1);
  },
}
```

Slika 7: Prikaz funkcija za povećanje i smanjivanje proizvoda unutar košarice

Također, na slici 16 vidi se i funkcija `removeFromBasket ()` unutar koje se nalazi Javascript metoda `splice ()` koja dodaje ili uklanja stavke u ili iz niza te vraća njihove vrijednosti. Dakle, metoda `removeFromBasket ()` se koristi kako bi se iz košarice uklonili proizvode koje korisnik želi ukloniti kada indeks tog proizvoda dođe na 1. Na slici 14 se unutar tijela tablice vidi i informacija o ukupnoj cijeni odabranih proizvoda, ovisno o njihovoj cijeni i količini:

`{{ item.price * item.quantity }}`.

Ispod toga nalazi se još jedan dio koji korisniku prikazuje kolika je ukupna vrijednost svih odabranih proizvoda koji se nalaze u košarici te ta vrijednost ima naziv “total”:

```
<p>Total: {{ total }} kn</p>
<button class="btn btn-success btn-block" @click="addNewOrder">Naruči!</button>
```

Slika 8: Kod za prikaz ukupne vrijednosti svih odabranih proizvoda u košarici

```

total() {
  let totalCost = 0;
  for(let items in this.basket) {
    let individualItem = this.basket[items];
    totalCost += individualItem.quantity * individualItem.price;
  }
  return totalCost;
}

```

Slika 9: Prikaz logike pomoću koje se dobiva ukupna cijena odabranih proizvoda u košarici

Funkcija `total ()` također se nalazi u `script` predlošku, ali također i unutar `computed` svojstva koji se odnosi na izračunatu vrijednost kao izvedenu vrijednost iz nečega koja će se automatski ažurirati kada se ažurira jedna od temeljnih vrijednosti korištenih za taj izračun (u ovom slučaju je to ukupna cijena odabranih proizvoda u košarici). `Computed` svojstvo se ne može pozivati te ono ne prihvaća nikakve parametre već se ono može samo referencirati.

6.5. Admin

Slično kao i `Menu`, ova komponenta sastoji se od dva glavna dijela: prvi, odnosno lijevi dio, sastoji se od dodavanja novih proizvoda (nove hrane) u jelovnik, njihovih opisa, količine i cijene, a drugi dio se sastoji od sažetog pregleda onoga što administrator postavi u jelovnik, odnosno s desne strane se nalazi mini jelovnik te administrator ima mogućnost jednim klikom izbaciti pojedine proizvode iz jelovnika.

Osim ta dva najbitnija dijela, `Admin` komponenta sastoji se od dodatna dva dijela: dio koji će prikazivati trenutne narudžbe koje je kupac odabrao, tj. poslao te sve informacije o narudžbi, a druga stvar je prijava (eng. *login*) samog administratora koji će mu omogućiti pristup gore opisanoj funkcionalnosti. Ova komponenta je strukturom slična komponenti `Menu`:


```

<template>
  <div class="container-fluid">
    <section v-if="currentUser">
      <div class="row">
        <div class="col-sm-12 col-md-6">
          <NewProduct />
        </div>

        <div class="col-sm-12 col-md-6">
          <h3>Menu</h3>
          <table class="table">
            <thead>
              <tr>
                <th scope="col">Item</th>
                <th scope="col">Remove fom menu</th>
              </tr>
            </thead>

            <tbody v-for="item in getMenuItems" :key="item['.key']">
              <tr>
                <td>{{item.name}}</td>
                <td><button class="btn btn-outline-danger btn-sm" @click="removeMenuItem(item['.key'])">x</button></td>
              </tr>
            </tbody>
          </table>
        </div>
      </div>
    </div>
  </div>
</template>

```

Slika 10: Struktura Admin komponente 1)

```

<div class="row">
  <div class="col-12">
    <h3>Current orders: {{ numberOfOrders }}</h3>
    <table class="table table-sm" v-for="(orders, index) in getOrders" :key="orders['.key']"> <!-- for petlja za loop kroz tablicu -->
      <thead>
        <div class="order-number">
          <strong><em> Broj narudžbe: {{ index + 1 }} </em></strong>
          <button class="btn btn-outline-danger btn-sm" @click="removeOrderItem(orders['.key'])"> x </button>
        </div>

        <tr>
          <th scope="col">Proizvod</th>
          <th scope="col">Veličina</th>
          <th scope="col">Količina</th>
          <th scope="col">Cijena</th>
        </tr>
      </thead>

      <tbody>
        <tr v-for="orderItems in orders['.value']" :key="orderItems.id">
          <td>{{ orderItems.name }}</td>
          <td>{{ orderItems.size }}</td>
          <td>{{ orderItems.quantity }}</td>
          <td>{{ orderItems.price }} kn</td>
        </tr>
      </tbody>
    </table>
  </div>
</section>

<hr>
<div class="row">
  <div class="col-sm-12 col-lg-12">
    <Login />
  </div>
</div>
</div>
</template>

```

Slika 11: Struktura Admin komponente 2)

Glavni div ima Bootstrap klasu `container-fluid`, a ta klasa označava kontejner pune širine koji se proteže cijelom širinom okvira prikaza, ovisno o veličini ekrana. Gotovo čitavi prikaz

Admin komponente nalazi se u sekciji, odnosno **section** elementu, koja je ujedno i uvjet, a uvjet je taj da korisnik (u ovom slučaju administrator) mora biti prijavljen kako bi imao pristup Admin komponenti i njezinim funkcionalnostima.

Unutar prvog Bootstrapovog responzivnog reda, odnosno Bootstrapove klase **row**, postavljena su dva vertikalna prikaza – prvi se odnosi na upisivanje novog proizvoda te sve potrebne informacije o njemu (lijeva strana), a drugi se odnosi na sažet prikaz upravo tih dodanih proizvoda u jelovnik (desna strana). Budući da Vue.js ima mogućnost postavljanja komponenti unutar određene hijerarhije, komponente se mogu postavljati bilo gdje unutar aplikacije. Ta situacija je iskorištena prilikom izrade prikaza prvog vertikalnog prikaza koji je naveden iznad, a to je dodatna komponenta naziva **NewProduct**.

6.5.1. Novi proizvod

```
<div class="row">
  <div class="col-sm-12 col-md-6">
    <NewProduct />
  </div>
</div>
```

Slika 12: Način uvođenja jedne komponente u drugu, u ovom slučaju NewProduct unutar Admin komponente

```

<div class="new-product">
  <form>
    <h3>Dodaj novi proizvod: </h3>

    <div class="form-group row">
      <label class="col-sm-3"> Naziv: </label>
      <div class="col-sm-9">
        <input type="text" class="form-control" v-model="newProduct.name">
      </div>
    </div>

    <div class="form-group row">
      <label class="col-sm-3"> Opis: </label>
      <div class="col-sm-9">
        <textarea type="text" class="form-control" rows="5" v-model="newProduct.description"></textarea>
      </div>
    </div>

    <p><strong> Opcija 1: </strong></p>
    <div class="form-group row">
      <label class="col-sm-3"> Veličina: </label>
      <div class="col-sm-9">
        <input type="text" class="form-control" v-model="newProduct.options[0].size">
      </div>
    </div>

    <div class="form-group row">
      <label class="col-sm-3"> Cijena: </label>
      <div class="col-sm-9">
        <input type="text" class="form-control" v-model="newProduct.options[0].price">
      </div>
    </div>
  </form>
</div>

```

Slika 22: Struktura koda za prikaz dijela Admin komponente – NewProduct komponente

Dodaj novi proizvod:

Naziv:

Opis:

Opcija 1:

Veličina:

Cijena:

Opcija 2:

Veličina:

Cijena:

Slika 23: Izgled NewProduct komponente u web pregledniku

Na slici 22 može se primijetiti nova stavka unutar Vue-a, a to je **v-model**. Direktiva v-model ažurira predložak kad god se mijenja model te ažurira podatkovni model kad god se mijenja predložak. V-model direktiva koristi se unutar Vue-a kada se radi s formama, a budući da je dodavanje novog proizvoda jedna velika forma, neophodno je da se koristi ta direktiva.

Rad s **v-modelom** unutar formi naziva se dvosmjerno povezivanje (eng. *two-way binding*). Ukratko, kada postoji potreba za sinkroniziranjem korisnikovog unosa, u ovom slučaju

administratora, sa nekim drugim podacima koje korisnik također unosi te njihovo spremanje unutar određenog objekta (newProduct u ovom slučaju), koristi se v-model.

```
data() {  
  return {  
    newProduct: {  
      'name': '',  
      'description': '',  
      'options': [  
        {  
          'size': '',  
          'price': ''  
        }, {  
          'size': '',  
          'price': ''  
        }  
      ]  
    }  
  }  
},
```

Slika 13: Objekt s praznim vrijednostima u kojega se spremaju vrijednosti koje unosi administrator - dvosmjerno povezivanje

NewProduct komponenta zauzima lijevi vertikalni dio Admin komponente, a desni dio zauzima jelovnik kojega slaže administrator:

```
<div class="col-sm-12 col-md-6">  
  <h3 class="menu">Menu</h3>  
  <table class="table">  
    <thead>  
      <tr>  
        <th scope="col">Proizvod</th>  
        <th scope="col">Izbaci iz jelovnika</th>  
      </tr>  
    </thead>  
    <tbody v-for="item in getMenuItems" :key="item['.key']">  
      <tr>  
        <td>{{item.name}}</td>  
        <td><button class="btn btn-outline-danger btn-sm" @click="removeMenuItem(item['.key'])">x</button></td>  
      </tr>  
    </tbody>  
  </table>  
</div>
```

Slika 14: Struktura koda za prikaz proizvoda i njihovih informacija nakon dodavanja novog proizvoda

U ovom slučaju koriste se iste Bootstrap klase kao i prije te se koristi sličan tablični prikaz. Zaglavlje tablice sadrži 2 naslova: “Proizvod” te “Izbaci iz jelovnika”. Tijelo tablice sadrži ime proizvoda te gumb koji služi za brisanje proizvoda s jelovnika te na završno brisanje proizvoda iz baze podataka.

Menu

Proizvod	Izbaci iz jelovnika
Čevapi	<input type="checkbox"/>
Grill Bill burger	<input type="checkbox"/>
Vegeburger	<input type="checkbox"/>
Fishburger	<input type="checkbox"/>
Grill Bill Hot Dog	<input type="checkbox"/>
Mesna plata	<input type="checkbox"/>

Slika 15: Sažeti prikaz jelovnika u web pregledniku unutar Admin komponente - proizvodi koje dodaje administrator

6.5.2. Narudžbe

Sljedeći vrlo bitan element, ponajviše što se tiče zamišljene dostave restorana, je prikaz trenutnih narudžbi te njihovih informacija. Struktura ovog elementa je sljedeća:

```

<div class="row">
  <div class="col-12">
    <h3 class="orders-title">Narudžbe: {{ numberOfOrders }}</h3>
    <table class="table table-sm" v-for="(orders, index) in getOrders" :key="orders['.key']"> <!-- for petlja za loop kroz tablicu -->
      <thead>
        <div class="order-number">
          <strong><em> Broj narudžbe: {{ index + 1 }} </em></strong>
          <button class="btn btn-outline-danger btn-sm" @click="removeOrderItem(orders['.key'])"> x </button>
        </div>
        <tr>
          <th scope="col">Proizvod</th>
          <th scope="col">Veličina</th>
          <th scope="col">Količina</th>
          <th scope="col">Cijena</th>
        </tr>
      </thead>
      <tbody>
        <tr v-for="orderItems in orders['.value']" :key="orderItems.id">
          <td>{{ orderItems.name }}</td>
          <td>{{ orderItems.size }}</td>
          <td>{{ orderItems.quantity }}</td>
          <td>{{ orderItems.price }} kn</td>
        </tr>
      </tbody>
    </table>
  </div>
</div>

```

Slika 16: Struktura koda za prikaz narudžbi i njihovih informacija nakon naručivanja od strane korisnika aplikacije

Kao i kod jelovnika, za glavni div element postavljena je Bootstrap klasa row unutar koje se nalazi još jedan div klase col-12 koji označava prikaz čitave strukture informacija pune širine preko bilo koje veličine ekrana.

Naslov ovog dijela Admin komponente je “Narudžbe” koje su dinamički povezane s Firebase-om, odnosno s Firebase bazom podataka. Unutar narudžbi nalaze se dvije for direktive: jedna za prolaz kroz čitavu tablicu za prebrojavanje ukupnog broja elemenata i narudžbi, a druga za prolaz kroz tijelo tablice koje će prikazivati ime, veličinu, količinu te cijenu odabranih proizvoda od strane korisnika.

Svaki element unutar ove tablice ima svoj identifikacijski broj koji je referenca na vrijednosti koje će se nalaziti u Firebase bazi podataka budući da će proizvodi biti povezani prema principu “ključ → vrijednost”.

Narudžbe: 2

Broj narudžbe: 1

Proizvod	Veličina	Količina	Cijena
Čevapi	5 komada	2	25 kn
Vegeb主rger	Mini	3	20 kn

Broj narudžbe: 2

Proizvod	Veličina	Količina	Cijena
Čevapi	10 komada	2	35 kn
Čevapi	5 komada	1	25 kn
Fishburger	Mini	1	22 kn

Slika 26: Prikaz narudžbi u web pregledniku

6.5.3. Login

Zadnja stavka unutar Admin komponente je Login komponenta koja je uvedena unutar Admin komponente na isti način kao i NewProduct:

```
<div class="row">
  <div class="col-sm-12 col-lg-12">
    <Login />
  </div>
</div>
```

Slika 27: Uvođenje Login komponente unutar Admin komponente

```

<template>
  <div class="container">
    <div class="row">
      <div class="col">
        <div>
          <p v-if="!currentUser">Molimo prijavite se za nastavak u administracijsku sekciju.</p>
          <p class="logged-in" v-else>Logged in as: <br> {{ currentUser }}</p>
        </div>

        <form>
          <div class="form-group">
            <label> E-mail: </label>
            <input type="email" class="form-control" id="email">
          </div>

          <div class="form-group">
            <label> Lozinka: </label>
            <input type="password" class="form-control" id="password">
          </div>

          <button type="button" class="btn btn-primary" @click.prevent="signIn"> Prijava </button>
          <button type="button" class="btn btn-danger" @click.prevent="signOut"> Odjava </button>
        </form>
      </div>
    </div>
  </div>
</template>

```

Slika 28: Prikaz strukture koda za Login komponentu

Struktura ove komponente je slična ostalima – sastoji se od glavnog **div-a** s uobičajenim Bootstrap klasama za responzivnost unutar kojega su podaci i informacije podijeljene u zasebne djelove. Kod ove komponente najbitnija je validacija i autorizacija te sinkronizacija s Firebase-om. Unutar prvog **div-a** ispod **div-a** s klasom **col** nalazi se paragraf **p** s **if** direktivom koja se odnosi na to da ako korisnik nije prijavljen kao administrator unutar Firebase-a, ne može nastaviti na administracijsku stranu aplikacije, odnosno Admin komponentu. Nadalje, u Login komponenti nalazi se klasična forma s upisom e-maila i lozinke te dva gumba koji se odnose na prijavu i odjavu korisnika s prikladnom autorizacijom, odnosno administratora.

Ova komponenta predstavlja ulaz u administratorsku sekciju sa svim navedenim funkcionalnostima te je neophodno da ona postoji upravo iz razloga da se zabrani pristup običnom korisniku aplikacije da promijeni podatke koji se nalaze u jelovniku te u bazi podataka. Validacija i autorizacija određenih korisnika je jedan od najbitnijih djelova svake aplikacije budući da se želi očuvati koncept aplikacije upravo onakvim kakav je i zamišljen, bez neočekivanih promjena koje bi mogle uzrokovati nepravilan rad aplikacije.

Molimo prijavite se za nastavak u administracijsku sekciju.

E-mail:

Lozinka:

Prijava

Odjava

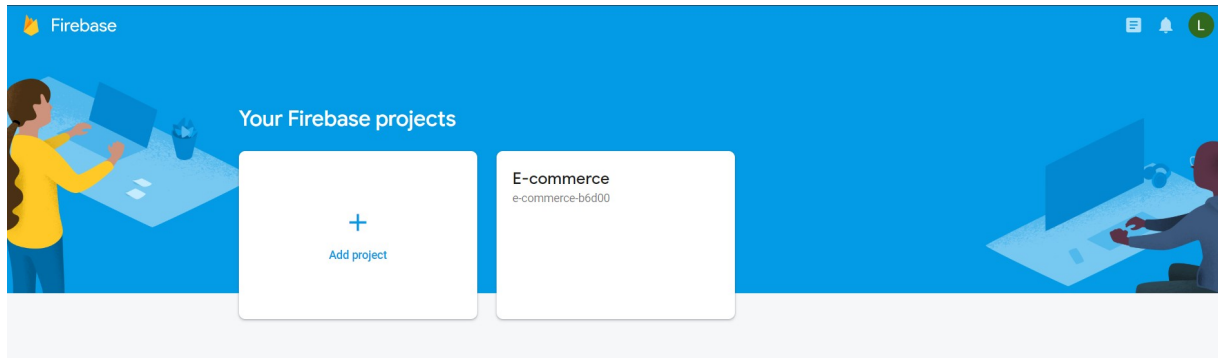
Slika 29: Prikaz prijave za administratora u web pregledniku

7. Baza podataka, autentifikacija i validacija

U ovom poglavlju se opisuje postupak izrade baze podataka, kako je spojiti na projekt u Vue-u, kako slati podatke u bazu podataka, kako iz nje izvlačiti podatke te kako ih prikazati i na kraju kako postaviti validaciju i autorizaciju kod prijave administratora.

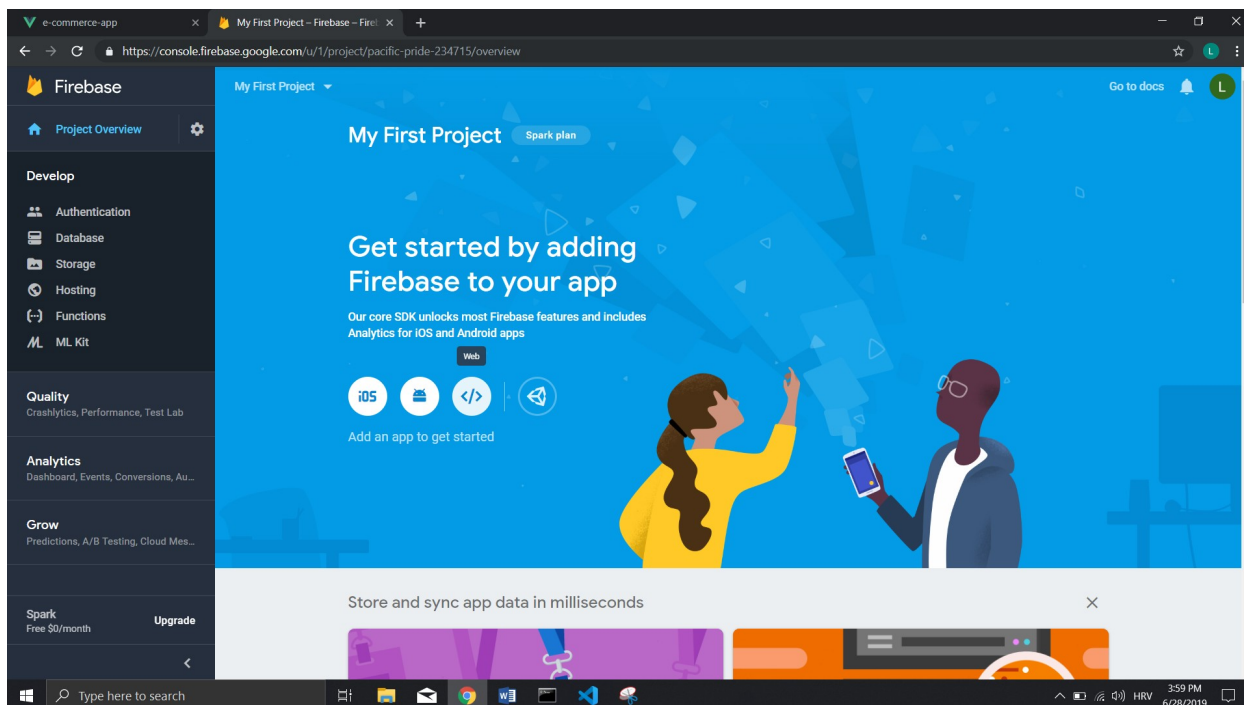
U prijašnjim djelovima ukratko je opisano što je Firebase, za što ću se koristiti i sl., a sada će se opisati kako se koristi te kako se spaja s Vue-om. Firebase će omogućiti pohranu proizvoda koje će administrator preko forme slati u jelovnik (menu) te pohranu svih narudžbi koje je korisnik odabrao, odnosno naručio. Ukratko, Firebase će služiti kao baza podataka koja će biti sinkronizirana s Vue-om.

Za početak, potrebno je napraviti račun (eng. *account*) na službenoj Firebase stranici. Nakon prijave, dostupna je konzola koja korisnicima Firebase pruža opciju stvaranja novog projekta.



Slika 30: Firebase konzola koja omogućuje stvaranje novog projekta

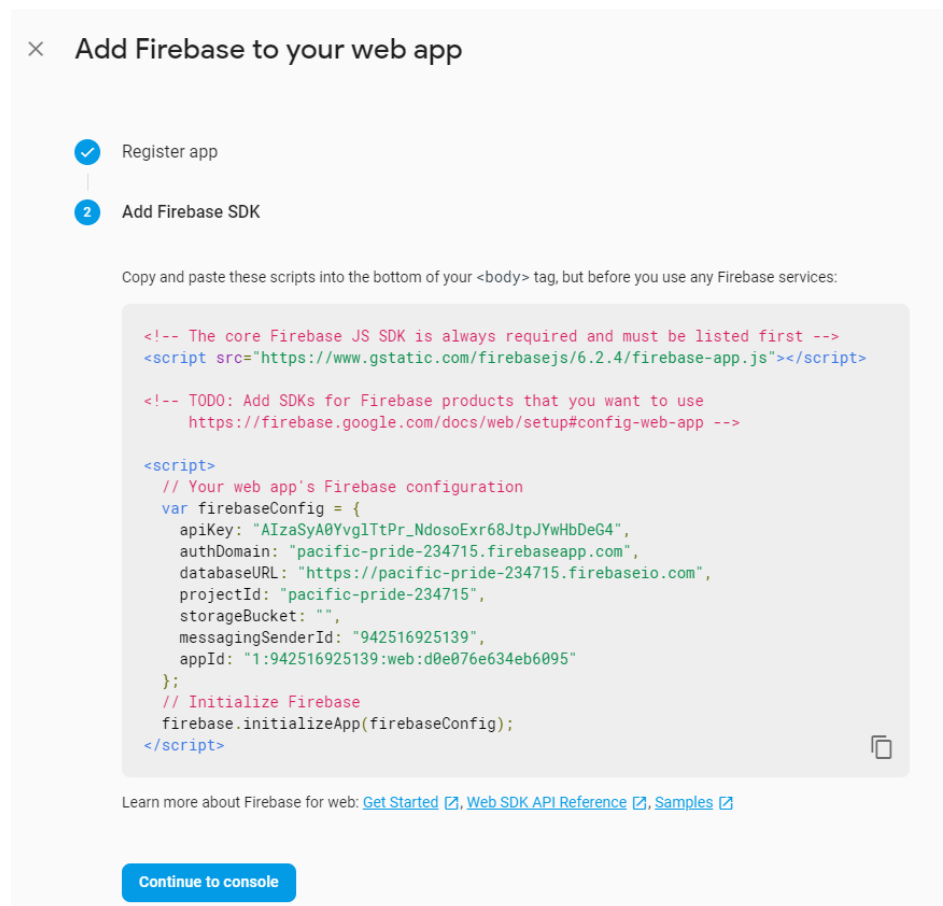
Nakon stvaranja projekta, konzola vodi korisnika do sučelja za korištenja svih mogućnosti za rad s projektom koje Firebase ima na raspolaganju.



Slika 31: Firebase sučelje za rad s projektom

Na slici se vidi da Firebase zahtijeva platformu za koju se postaviti projekt, a ima ih 4: iOS, Android, Web i Unity. Budući da je ovo projekt za full stack web aplikaciju, koristit će se skripta za web platformu.

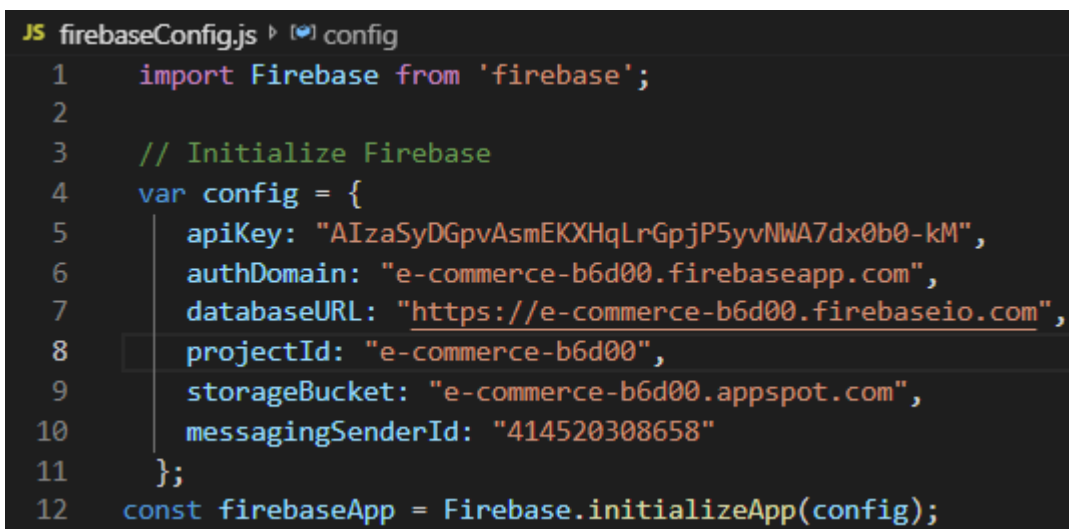
Nakon odabira web platforme, Firebase prikazuje na koji način ga povezati s web aplikacijom.



Slika 32: Prikaz koda koji je potreban za spajanje Firebase-a s web aplikacijom

Iz navedenog, potrebno je kopirati kod iz druge script oznake. Izvorni Firebase link iz prve script oznake nije potreban jer će se Firebase instalirati pomoću paket menadžera naziva npm, a to je zadani upravitelj paketa za Javascript runtime okruženje Node.js. Pristupa mu se preko komandne linije.

Kopirani kod iz druge script oznake kopiramo u novostvorenu Javascript datoteku naziva `firebaseConfig.js`:

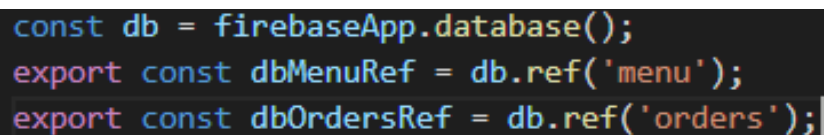


```
JS firebaseConfig.js ▸ config
1  import Firebase from 'firebase';
2
3  // Initialize Firebase
4  var config = {
5    apiKey: "AIzaSyDGpvAsmEKXHqLrGpjP5yvNWA7dx0b0-kM",
6    authDomain: "e-commerce-b6d00.firebaseio.com",
7    databaseURL: "https://e-commerce-b6d00.firebaseio.com",
8    projectId: "e-commerce-b6d00",
9    storageBucket: "e-commerce-b6d00.appspot.com",
10    messagingSenderId: "414520308658"
11  };
12  const firebaseApp = Firebase.initializeApp(config);
```

Slika 33: Potreban kod za spajanje web aplikacije s Firebaseom unutar Javascript datoteke `firebaseConfig.js`

Nakon toga, potrebno je instalirati Firebase preko `npm`-a unutar terminala pomoću naredbe `npm install --save firebase`. `Npm` će preko ove instalacije povući sve potrebne module za spajanje Firebase-a i web aplikacije te ih spremiti u datoteku `node_modules`. Kada `npm` obavi instalaciju, potrebno je uvesti Firebase u aplikaciju pomoću sljedeće naredbe: `import Firebase from 'firebase'`, što se vidi i na vrhu slike 33.

Sada je Firebase spojen na aplikaciju i sljedeći korak je napraviti referencu na bazu podataka u Firebase-u te nad njom napraviti export kako bi se mogla koristiti gdje god bude potrebna:



```
const db = firebaseApp.database();
export const dbMenuRef = db.ref('menu');
export const dbOrdersRef = db.ref('orders');
```

Slika 34: Referenciranje Firebase baze podataka

Iz slike se vidi kako je Firebase baza podataka referencirana preko konstante `db` te je izvedena preko `export` naredbe kako bi postojao pristup jelovniku i narudžbama koje će se nalaziti u bazi podataka kako bi se mogle koristiti u aplikaciji.

Kako će se Firebase koristiti za autentifikaciju i validaciju podataka, potrebno je promijeniti par postavki unutar same baze podataka. Dovoljno je unutar sučelja za rad s projektom kliknuti na karticu naziva Database te nakon toga klikom na karticu Realtime Database doći do pravila (eng. rules) koja utječu na rad s bazom podataka. Autentifikacija znači to da će samo autentificirani korisnik, u ovom slučaju administrator, imati pristup podacima baze podataka. Unaprijed zadana (eng. *default*) pravila Firebase baze podataka su sljedeća:

```
{
  "rules": {
    ".read": "auth != null",
    ".write": "auth != null"
  }
}
```

Slika 35: Unaprijed zadane vrijednosti pravila unutar Firebase baze podataka

Kako bi omogućili pristup samo administratoru, potrebno je promijeniti ta pravila na sljedeći način:

```
{
  "rules": {
    ".read": true,

    "orders": {
      ".write": true
    },

    "menu": {
      ".write": "auth != null"
    }
  }
}
```

Slika 36: Promijenjena pravila baze podataka za omogućenje pristupa administratoru

Ovom promjenom pravila administratoru je omogućeno čitanje te zapisivanje novih podataka u bazu podataka, te pristup podacima proizvoda s jelovnika.

Kako bi se Firebase baza podataka povezala s kodom, potrebno ju je uvesti u **NewProduct** komponentu jer se u toj komponenti odvijaju aktivnosti koje se tiču rada s bazom podataka: dodavanje, modificiranje, brisanje proizvoda s jelovnika te prikaz narudžbi u **Admin** komponenti unutar koje se nalazi **NewProduct** komponenta. Na sljedeći način se uvodi referenca na bazu podataka, specifično za jelovnik, unutar **script** oznake u **NewProduct** komponenti:

```
import { dbMenuRef } from '../firebaseConfig.js'.
```

Budući da je sada Firebase povezan s kodom, odnosno komponentom **NewProduct**, vrlo je jednostavno povezati to dvoje:

```
methods: {  
  addItem() {  
    dbMenuRef.push(this.newProduct);  
  }  
}
```

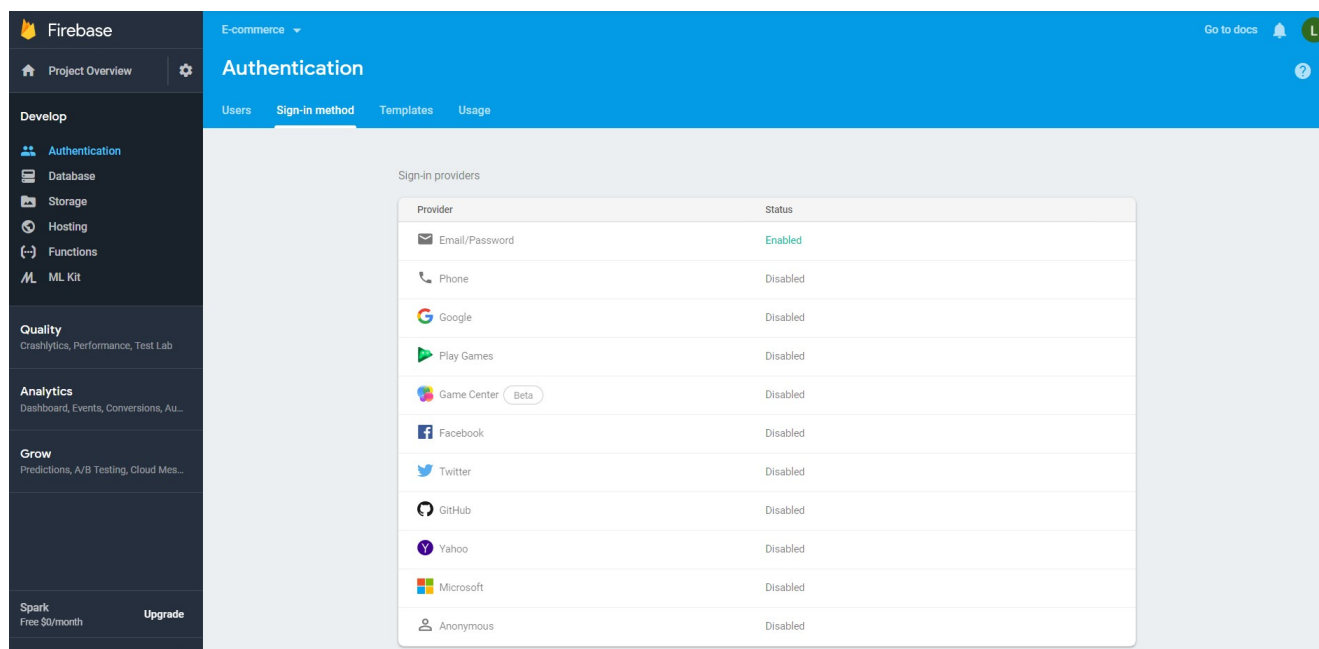
Slika 37: Dodavanje novih proizvoda u menu, tj. jelovnik

Dakle, **addItem ()** potrebno je deklarirati unutar sekcije za metode unutar **script** oznake, a samo odvijanje te funkcije se odvija klikom administratora na gumb “Dodaj”. To se čini pomoću Javascript funkcije **push ()** te ključne riječi **this** preko koje se pristupa objektu **NewProduct**. Informacije kojima se puni polje nalazi se unutar objekta **NewProduct** koji se nalazi iznad same metode sa slike 37, a to su ime proizvoda, opis te dvije opcije za veličinu i cijenu određenog proizvoda, ovisno o veličini samog proizvoda.

```
<div class="form-group row">
|   <button type="button" class="btn btn-success btn-block" @click="addItem"> Dodaj </button>
</div>
```

Slika 38: Okidač (eng. trigger) odvijanja funkcija addItem ()

Kako bi administrator imao mogućnost dodavanja novih proizvoda te uvida u trenutne narudžbe, potrebno mu je omogućiti pristup, odnosno autentifikaciju. Firebase ima vrlo jednostavnu proceduru za postavljanje posebnog pristupa određenih korisnika. U konzoli se nalazi kartica za autentifikaciju te mogućnost odabira izbora prijave:



Slika 39: Firebase izbor metoda za autentifikaciju

Izabrana je prva stavka, a to je autentifikacija pomoću e-maila i lozinke. Nakon odabira metode, potrebno je klikom na “Users” karticu, koja se nalazi odmah pored izbora metode za autentifikaciju, te nakon toga klikom na “Add User” gumb jednostavno dodati e-mail i lozinku koje će administrator koristiti kako bi imao pristup funkcionalnostima dodavanja novih proizvoda, uvida u narudžbe i sl.

Moguće je imati više administratora budući da Firebase nudi tu opciju, ali za potrebe ove postojati će samo jedan administrator.

Sada je potrebno uvesti Firebase unutar Login komponente, a to se čini sljedećom linijom koda: `import Firebase from 'firebase'` unutar `script` oznake. Kako bi se administrator mogao prijaviti i odjaviti, postavljena su dva gumba: Prijava i Odjava.

```
<button type="button" class="btn btn-primary" @click.prevent="signIn"> Prijava </button>
<button type="button" class="btn btn-danger" @click.prevent="signOut"> Odjava </button>
```

Slika 40: Gumbi za prijavu i odjavu administratora

Iz slike se može vidjeti klik aktivnost te `prevent` Javascript metoda koja sprečava osvježavanje stranice klikom na jedan od ova dva gumba. Administrator se, ovisno o gumbu, prijavljuje ili odjavljuje, a za to su potrebne dvije funkcije, odnosno metode naziva `signIn ()` za prijavu te `signOut ()` za odjavu.

```
methods: {
  signIn() {
    let email = document.getElementById('email').value;
    let password = document.getElementById('password').value;

    Firebase.auth().signInWithEmailAndPassword(email, password).catch(function(error){
      let errorCode = error.code;
      let errorMessage = error.message;

      if (errorCode === 'auth/wrong-password') {
        alert('Pogrešna lozinka, pokušajte ponovno.')
      }
      else {
        alert (errorMessage);
      }
    });
  },
  signOut() {
    Firebase.auth().signOut().then(function (){
      alert ('Odjavljeni ste.')
    }).catch(function(error){
      alert (error)
    })
  }
},
```

Slika 41: Prikaz metoda za prijavu i odjavu administratora

Budući da je kao metode autentifikacije izabran e-mail i lozinka, administratorovog unos (eng. *input*) sprema se u formu za e-mail i lozinku. To se čini preko čistog Javascripta pomoću `document.getElementById ().value` funkcije koja vraća rezultat, odnosno vrijednost unosa te se na taj način dohvaća ono što korisnik, u ovom slučaju administrator, upisuje.

Preko funkcije `Firebase.auth ()` koja služi za autentifikaciju dobiva se pristup mnoštvu drugih funkcija koje nudi Firebase i koje se odnose na autentifikaciju općenito. Budući da su odabrani e-mail i lozinka, koristi se funkcija `signInWithEmailAndPassword ()` koja za argumente uzima unos administratora unutar forme za prijavu. Naknadno, kako bi se osigurala točnost prijave, potrebno je na autentifikaciju postaviti validaciju podataka koji su spremljeni unutar Firebase konzole, a to se čini preko funkcije `catch ()` koja unutar sebe sadrži još jednu funkciju koja za argument ima vrijednost varijable `errorCode` koja predstavlja kod greške, ovisno o kojoj greški se radi.

Error codes for FCM failure conditions.

Enums	
UNSPECIFIED_ERROR	No more information is available about this error.
INVALID_ARGUMENT	(HTTP error code = 400) Request parameters were invalid. An extension of type <code>google.rpc.BadRequest</code> is returned to specify which field was invalid.
UNREGISTERED	(HTTP error code = 404) App instance was unregistered from FCM. This usually means that the token used is no longer valid and a new one must be used.
SENDER_ID_MISMATCH	(HTTP error code = 403) The authenticated sender ID is different from the sender ID for the registration token.
QUOTA_EXCEEDED	(HTTP error code = 429) Sending limit exceeded for the message target. An extension of type <code>google.rpc.QuotaFailure</code> is returned to specify which quota got exceeded.
APNS_AUTH_ERROR	(HTTP error code = 401) APNs certificate or auth key was invalid or missing.
UNAVAILABLE	(HTTP error code = 503) The server is overloaded.
INTERNAL	(HTTP error code = 500) An unknown internal error occurred.

Slika 42: Šifre koje predstavljaju tip geške prilikom autentifikacije i validacije

Osim šifre greške, funkcija `catch ()` će unutar sebe sadržavati poruku greške ako korisnik unese krivu lozinku, a to se dohvaća pomoću Firebase autentifikacijske opcije `auth/wrong-password` koja ima vrijednost `true` ako korisnik upiše krivu lozinku.

Za validaciju ovih podataka koriste se **if/else** uvjeti. Ako korisnik unese pogrešnu lozinku, na ekranu će mu se izbaciti upozoravajuća poruka ”Pogrešna lozinka, pokušajte ponovno” i ta poruka se nalazi u **if** uvjetu, dok se u **else** uvjetu izbacuje poruka greške ovisno o tome koje je ona vrste. To može biti bilo koja greška, npr. pogrešan format upisanog e-maila ili nešto slično.

Kod odjave se također poziva **Firestore.auth ()** metoda budući da se radi o autentifikaciji te se preko nje dolazi do implementirane Firestore metode **signOut ()**. Ako je odjava uspješna, tada se okida funkcija **then ()** koja će korisniku na ekranu prikazati poruku da je uspješno odjavljen.

I u ovom slučaju postavljena je validacija kako bi sve funkcioniralo kako treba– preko **catch ()** metode koja korisniku na ekran izbacuje upozoravajuću poruku koja sadržava grešku ovisno o tome koje je ona vrste.

8. Rute

Za potrebe postavljanja ruta u ovom projektu koristi Vue Router i on je službeni usmjerivač SPA-a (eng. *Single Page Applications*) za Vue.js.

Kako bi se počeo koristiti Vue Router, potrebno ga je instalirati unutar projekta. To se radi pomoću **npm-a** naredbom :

```
npm install - - save vue-router.
```

Nakon toga, potrebno ga je uvesti u sami projekt, a to se radi pomoću **import** metode unutar **main.js** datoteke na sljedeći način:

```
import VueRouter from 'vue-router'.
```

Može se deklarirati gdje će se koristiti Vue Router budući da se koristi unutar nekoliko komponenti, a deklarira se na sljedeći način:

```

Vue.use(VueRouter);

const router = new VueRouter({
  routes,
  mode: 'history',
  scrollBehavior() { // svaki put kad se neka komponenta loada, počinje od navigacije, tj. otpočetak
    return {
      x: 0,
      y: 0
    }
  }
})

```

Slika 43: Deklaracija Vue Router-a

Iz slike se može vidjeti kako konstanta **router** sadrži novi objekt **VueRouter** koji sadržava više stavki: **routes** varijabla koja predstavlja polje svih komponenti koje se koriste unutar projekta, **mode** varijable koja je definirana sa stringom 'history' koji koristi HTML5 History API koji omogućava da se promijeni URL adresa stranice bez osvježavanja (eng. *refresh*) stranice što dodatno ubrzava samu aplikaciju. Na kraju se nalazi metoda **scrollBehavior ()** koja svaki put kada određena komponenta loada stranicu vraća na početni dio, odnosno odozgo, a to je određeno s x i y osi koje su postavljene na 0.

Unutar routes.js datoteke može se vidjeti kako su definirane rute:

```

export const routes = [
  { path: '/', name: 'homeLink', component: Home },
  { path: '/menu', name: 'menuLink', component: Menu },
  { path: '/admin', name: 'adminLink', component: Admin,
    beforeEnter: (to, from, next) => {
      alert ('Ovo je područje samo za ovlaštene korisnike. Prijavite se kako biste nastavili. ');
      next();
    }
  },
  { path: '/contact', name: 'contactLink', component: Contact },
  { path: '/about', name: 'aboutLink', component: About,
    children: [ // nested routes (children)
      { path: '/history', name: 'historyLink', component: History },
      { path: '/delivery', name: 'deliveryLink', component: Delivery },
      { path: '/ordering-guide', name: 'orderingGuideLink', component: OrderingGuide }
    ]
  },
  { path: '*', redirect: '/' } // redirect na homepage za nepostojeće URL-ove
]

```

Slika 44: Način definiranja i deklariranja ruta

Routes polje je konstanta koja sadrži određene putanje za svaku pojedinu komponentu. Svaka komponenta sadrži putanju, svoje ime te te put do same sebe unutar projekta. Putanje su proizvoljne i u njih se može postaviti bilo koja vrijednost, a imena im se daju radi njihovog lakšeg i bržeg raspolaganja.

Na slici se također može vidjeti **beforeEnter ()** funkcija unutar Admin komponente koja upozorava korisnika da dolazi do Admin komponente te da se treba ulogirati s točnim autentifikacijskim podacima kako bi joj mogao pristupiti.

Nadalje, također se mogu primijetiti i komponente koje su potomci (eng. *children*) svojih roditelja, a u ovom slučaju to su komponente **History**, **Delivery** te **OrderingGuide** koje su potomci komponente **About**.

Na dnu slike 44 vidi se i preusmjeravanje (eng. *redirect*) na homepage kada se dogodi neka greška ili kada korisnik dođe do URL-a koji ne postoji.

9. Vuex

Vuex je obrazac upravljanja stanjima (eng. *state*) te ujedno i biblioteka za Vue.js aplikacije. Vuex služi kao centralizirano skladište (eng. *store*) za sve komponente u aplikaciji, s pravilima koja osiguravaju to da stanje može mutirati samo na predvidivi način. [15]

Instalacija Vuex-a se odvija na isti način kao i kod dodavanja ruta u aplikaciju – pomoću **npm install** naredbe, odnosno pomoću naredbe:

npm install vuex - - save.

Dodavanje unutar aplikacije je potpuno proizvoljno, međutim nepisana konvencija je da se u **src** folder doda još jedan folder pod imenom **store**. Unutar **store** foldera dodana je Javascript

datoteka imena **store.js** koja će služiti kao glavna datoteka zadužena za aktivnosti koje se odnose na skladište podataka.

Kako bi se moglo koristiti to centralizirano skladište, potrebno je uvesti Vue i Vuex module te **store.js** datoteku:

```
import Vue from 'vue'
import Vuex from 'vuex'
```

Slika 45: Dodavanje vue i vuex modula u centralizirano skladište

Potrebno je inicijalizirati pomoću **Vue.use()** kako bi Vuex bio dostupan svim komponentama i modulima. Skladište je kao varijablu postavljena kao konstanta te je postavljen njezin export kako bi je također mogle koristiti ostale komponente unutar aplikacije te je način deklariranja ove varijable veoma sličan deklaraciji varijable za rute:

```
import Vue from 'vue'
import Vuex from 'vuex'
import menu from './modules/menu.js'
import orders from './modules/orders.js'
import users from './modules/users.js'
import { firebaseMutations } from 'vuexfire'

Vue.use(Vuex)

export const store = new Vuex.Store ({
  mutations: {
    ...firebaseMutations
  },
  modules: {
    menu,
    orders,
    users
  }
})
```

Slika 46: Puni prikaz store.js datoteke

Ponekad će biti potrebno izračunati izvedeno stanje na temelju stanja centraliziranog skladišta, npr. filtriranje kroz popis stavki i njihovo brojanje ili u ovom slučaju prikaz svih proizvoda na jelovniku. Ako je to izračunato izvedeno stanje potrebno koristiti u više komponenti, tada je potrebno duplicirati funkciju ili je izvesti u neki pojedini modul te je importati na više lokacija, a to nije dobra programerska praksa jer dovodi do napuhavanja koda. Vuex u tom slučaju omogućuje definiranje “gettera” unutar skladišta na koje se može gledati kao na izračunata svojstva određenih stavki i koje možemo dohvatiti u danom trenutku kada su potrebne.

Rezultat gettera stavlja se u predmemoriju (eng. *cache*) na temelju njegovih ovisnosti i preispitat će se tek kada se promijene neke od njegovih ovisnosti. Dakle, getteri se koriste za poboljšavanje i pojednostavljivanje pristupa samom centraliziranom skladištu bez ponavljanja istih dijelova koda.

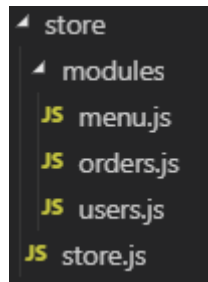
Getteri se koriste unutar sve 3 datoteke unutar centraliziranog skladišta, a odnose se na dohvaćanje proizvoda s jelovnika, dohvaćanje narudžbe i njezinog broja te na prikaz trenutnog korisnika, odnosno administratora. Svi getteri u ovom projektu bave se dohvaćanjem vrijednosti pojedinih elemenata iz polja. Npr. ako se gleda jelovnik, stanje jelovnika je prazno polje `menuItems: []` koje se puni preko administracijske sekcije putem gumba “Dodaj”, a puni se preko Javascript metode `push ()` koja se koristi kako bi se dodali elementi na kraj polja.

Kako bi se smanjila upotreba gettera, koristi se metoda Vuex-a naziva `mapGetters`. Koristi se pomoću novog Javascript operatora – `spread` operatora koji se označava kao trotočje (...), a taj operator se koristi kako bi se getteri mogli spojiti s izračunatim svojstvima (eng. *computed properties*) te međusobno komunicirati.

```
computed: {
  ...mapGetters ([
    'numberOfOrders',
    'getMenuItems',
    'getOrders',
    'currentUser'
  ])
},
```

Slika 47: Computed properties koja se nalaze unutar Admin komponente unutar kojih se nalaze svi getteri koji međusobno komuniciraju

Budući da se čitava aplikacija temelji na 3 najbitnije stavke: jelovnik, narudžbe i admina, odnosno korisnika, centralizirano skladište podijeljeno je prema ta 3 temeljna dijela unutar foldera naziva **modules**. Unutar tog folder, postavio sam 3 datoteke: **menu.js**, **orders.js** i **users.js**.



Slika 48: Prikaz hijerarhije centraliziranog skladišta te pripadnih datoteka i foldera

Na slici 46 također se vidi i objekt naziva **mutations**, odnosno mutacije. Jedini način da se promijeni stanje u centraliziranom skladištu Vuex-a je vršenje mutacija pomoću određenih metoda. Vuex mutacije su vrlo slične događajima (eng. *events*, JavaScript Events), što znači da se odvijaju preko određenog okidača, npr. klika mišom na određeni gumb. Funkcija koja sadrži taj okidač će primiti odabrano stanje kao prvi argument.

U slučaju ovog projekta, postoje dvije mutacije koje će se odvijati: mijenjanje stanja narudžbe u slučaju dodavanja ili brisanja novih proizvoda iz jelovnika te preusmjeravanje na administracijsku sekciju prilikom uspješne registracije korisnika koja je prošla autentifikaciju i validaciju.

```
const mutations = {  
  addOrder: (state, orders) => state.orders.push(orders)  
}
```

Slika 49: Mutacija koja se odnosi na dodavanje novih proizvoda u jelovnik

```
const mutations = {
  userStatus: (state, user) => {
    if(user){
      state.currentUser = user.email;
    }
    else {
      state.currentUser = null;
    }
  }
}
```

Slika 50: Mutacija koja se odnosi na korisnika koji želi pristupiti administracijskoj sekciji

Preduvjet odvijanja mutacije su akcije (eng. *actions*). Akcije su radnje koje su slične mutacijama uz dvije bitne razlike:

- 1) Umjesto mutiranja, odnosno mijenjanja samog stanja, akcije izvršavaju same mutacije – prvo se trebaju odvitati akcije pa tek onda mutacije
- 2) Akcije mogu sadržavati proizvoljne asinkrone operacije (asinkrone operacije se odnose na bilo koje funkcionalnosti koje se mogu odvijati bez ponovnog učitavanja stranice na kojoj se korisnik trenutno nalazi)

Za izradu ovog projekta potrebne su tri akcije, svaka za gore navedene 3 temeljne stavke (jelovnik, narudžbe i korisnika):

- 1) Akcija koja se odnosi na prikaz trenutnog korisnika te preusmjeravanje na administracijsku sekciju:

```
const actions = {
  setUser: ({commit}, user) => {
    commit('userStatus', user)
  }
}
```

Slika 51: Akcija za prikaz trenutnog korisnika

- 2) Akcija koja se odnosi na sinkronizaciju proizvoda iz jelovnika iz Vuex-a s Firebase bazom podataka koju sam objasnio u prijašnjim djelovima ovog projekta:

```
const actions = {
  setMenuRef: firebaseAction(({ bindFirestoreRef }, { ref }) => {
    bindFirestoreRef('menuItems', ref)
  })
}
```

Slika 52: Sinkronizacija proizvoda s jelovnika s Firebase bazom podataka

- 3) Akcija koja se odnosi na spajanje narudžbi iz Vuex-a s Firebase bazom podataka koju sam objasnio u prijašnjim djelovima ovog projekta:

```
const actions = {
  setOrdersRef: firebaseAction(({ bindFirestoreRef }, { ref }) => {
    bindFirestoreRef('orders', ref)
  })
}
```

Slika 53: Sinkronizacija narudžbi iz Vuex-a s Firebase bazom podataka

Kako bi gore navedena sinkronizacija bila moguća, potrebno je spojiti samu aplikaciju s Firebaseom. To se čini sljedeći način unutar glavne datoteke ovog projekta – **App.vue**:

```

<script>
import Header from './components/Header'
import { dbMenuRef, dbOrdersRef } from '../firebaseConfig.js'

export default {
  name: 'app',
  components: {
    Header
  },
  created() {
    this.$store.dispatch('setMenuRef', dbMenuRef)
    this.$store.dispatch('setOrdersRef', dbOrdersRef)
  }
}
</script>

```

Slika 54: Povezivanje Firebase s aplikacijom

U importu se mogu vidjeti konstante koje su deklarirane unutar datoteke firebaseConfig.js:

```

import Firebase from 'firebase';

// Initialize Firebase
var config = {
  apiKey: "AIzaSyDGpvAsmEKXHqLrGpjP5yvNWA7dx0b0-kM",
  authDomain: "e-commerce-b6d00.firebaseio.com",
  databaseURL: "https://e-commerce-b6d00.firebaseio.com",
  projectId: "e-commerce-b6d00",
  storageBucket: "e-commerce-b6d00.appspot.com",
  messagingSenderId: "414520308658"
};

const firebaseApp = Firebase.initializeApp(config);
const db = firebaseApp.database();
export const dbMenuRef = db.ref('menu');
export const dbOrdersRef = db.ref('orders');|

```

Slika 55: Prikaz cijele firebaseConfig.js datoteke i njezinih varijabli

Iskustva

Izrada ove aplikacije dala mi je vrlo detaljan uvid u Firebase, a pogotovo u Vue.js. Nisam imao prijašnjih iskustava u radu s ova dva alata te sam krenuo od samog početka učiti o izradi aplikacije proučavajući dostupnu dokumentaciju, gledajući raznorazne video uratke na Youtubeu i sličnim platformama te pokušavajući izraditi funkcionalnu aplikaciju.

Naučio sam kako postaviti projekt pomoću Vue CLI-a u terminalu, kako kreirati komponente te njihovo međusobno djelovanje i spajanje, autentifikaciju i autorizaciju preko Firebase metoda, kako raditi s bazom podataka (barem što se Firebase tiče), kako pravilno postaviti rute pomoću Vue Routera te kako ih pravilno povezati i imenovati, kako spremiti podatke unutar centraliziranog skladišta pomoću Vuex-a te njegovu sinkronizaciju s Firebaseom, kako raditi s getterima, akcijama i mutacijama te još mnogo stvari koje povezuju ovu čitavu aplikaciju.

S druge strane, Vue je relativno mlad Javascript okvir te evoluirao nevjerojatno brzo. Upravo zato što toliko brzo evoluirao, većina tutorijala koji se nalaze na internetu vrlo brzo postanu zastarjeli što otežava čitavi proces učenja te na kraju krajeva i same izrade aplikacije, neovisno o kojoj se radi. Tako je i u mom slučaju te se metode koje sam koristio u ovom projektu mogu smatrati zastarjelima u odnosu na prošla 3 mjeseca jer su izašle nove metode rješavanja, novi standardi te će krajem godine izaći i nova cjelokupna verzija Vue-a s novim mogućnostima. Naravno da se metode koje sam koristio u ovom projektu smatraju validnima jer aplikacija funkcionira onako kako bi trebala, ali kao developer uvijek treba težiti boljemu i biti u skladu s novim tehnologijama.

Najveće prednosti Vue-a su to što se lako uči te je vrlo intuitivan, ima jednostavnu integraciju u trenutno rađene projekte kao i one koji su već otprije napravljeni te je vrlo fleksibilan budući da omogućuje korisniku da piše HTML, CSS i Javascript u istoj datoteci bez uvođenja vanjskih datoteka. Također, za izradu ovog projekta je bilo vrlo korisno to što Vue i Firebase imaju međusobnu podršku te je vrlo laka integracija jedne tehnologije u drugu i obrnuto.

Trenutno najveći nedostatak Vue-a je to što ima najmanju zajednicu korisnika obzirom na veličinu zajednice koju imaju Angular i React. Manja zajednica znači da automatski postoji manje resursa, dokumentacije i tutorijala koje mogu koristiti novi korisnici Vue-a te to otežava

njegovo učenje. Međutim, Vue sadrži više prednosti nego nedostataka te smatram da će ubrzo biti na jednakoj razini popularnosti kao Angular i React.

Zaključak

Tema ovog diplomskog rada bila je prikazati na koji način se koriste Javascript okvir Vue te Googleova mobilna platforma Firebase funkcioniraju kada se koriste zajedno u izradi full stack web aplikacije. Opisao se postupak razvoja aplikacije te međudjelovanje i usklađenost navedenih tehnologija. Također su navedeni i opisani svi koraci potrebni za izradu aplikacije uključujući izradu komponenata, njihovo povezivanje, izrada baze podataka te njezina CRUD funkcionalnost, autentifikacija i validacija administratora prilikom prijave u aplikaciju te na kraju izrade košarice za pohranu odabranih proizvoda iz jelovnika.

Ovaj projekt sam odabrao iz razloga što me zanima Javascript i svaki framework koji se odnosi na Javascript jer smatram da je dobro imati uvid u svaki od njih te na temelju tih iskustva odabrati pravi framework ovisno o tome koji projekt bi se u budućnosti izrađivao.

S jedne strane Vue nema toliku popularnost kao Angular i React budući da nije toliko proširen na globalnoj razini (pogotovo na našim prostorima). Međutim s druge strane Vue je jedan od najbolje ocijenjenih Javascript okvira na Githubu, najpopularnijoj svjetskoj platformi za verzioniranje i dijeljenje koda, te me iz tog razloga zanimalo o čemu se točno radi te sam odlučio naučiti temelje ovog okvira kako bih mogao napraviti aplikaciju pomoću njega.

Vue je framework koji je, po mom mišljenju, najintuitivniji od najpopularnije trojke te ga je najlakše naučiti, što je vrlo bitno ponajprije za samu popularnost, a pogotovo za jednostavnost razvijanja web aplikacija.

Također, zanimalo me kako Firebase funkcionira te koje su njegove mogućnosti što se tiče backenda. Izrađena aplikacija nije možda toliko napredna da bi se mogla postaviti u produkcijsku verziju, ali je dovoljno kompleksna budući da uzima temeljne djelove gotovo svake veće aplikacije.

Vjerujem da ću u budućnosti koristiti Firebase prvenstveno zbog njegove jednostavnosti i prilagodljivosti određenim frameworksima, a pogotovo zbog njegovih mogućnosti kojih ima napretek. Budući da je Firebase u potpunosti besplatan alat koji je vrlo moćan te se može koristiti u gotovo svakoj aplikaciji na bilo kojoj razini, smatram da će se u budućnosti zasigurno sve više koristiti.

Popis slika

Slika 7: Prikaz rasporeda oznaka unutar Vue datoteke

Slika 8: Prikaz v-bind direktive

Slika 9: Prikaz v-on direktive

Slika 10: Prikaz v-model direktive

Slika 11: Prikaz v-if direktive

Slika 12: Prikaz v-for direktive

Slika 7: Defaultni izgled aplikacije prilikom stvaranja novog projekta

Slika 8: Zadani Bootstrap predložak za navigacijsku traku

Slika 9: Izgled navigacijske trake nakon stiliziranja

Slika 10: Prikaz navigacijske trake u "hamburger" izborniku za mobilni pregled

Slika 11: Home komponenta nakon završetka stiliziranja

Slika 12: Dio koda koji se odnosi na Menu komponentu - menu - mjesto gdje korisnik bira proizvod

Slika 13: + gumb koji se odnosi na dodavanje proizvoda iz jelovnika u košaricu

Slika 14: Deklaracija metode unutar Vue.js-a: metoda za dodavanje proizvoda u košaricu

Slika 15: Prikaz koda za spremanje proizvoda u košaricu te sami izgled košarice

Slika 16: Prikaz data () funkcije, praznog polja košarice te teksta košarice ako korisnik nije odabrao niti jedan proizvod

Slika 17: Prikaz funkcija za povećanje i smanjivanje proizvoda unutar košarice

Slika 18: Kod za prikaz ukupne vrijednosti svih odabranih proizvoda u košarici

Slika 19: Prikaz logike pomoću koje se dobiva ukupna cijena odabranih proizvoda u košarici

Slika 20: Struktura Admin komponente 1)

Slika 21: Struktura Admin komponente 2)

Slika 22: Način uvođenja jedne komponente u drugu, u ovom slučaju NewProduct unutar Admin komponente

Slika 23: Struktura koda za prikaz dijela Admin komponente – NewProduct komponente

Slika 24: Izgleda NewProduct komponente u web pregledniku

Slika 25: Objekt s praznim vrijednostima u kojega se spremaju vrijednosti koje unosi administrator - dvosmjerno povezivanje

Slika 26: Struktura koda za prikaz proizvoda i njihovih informacija nakon dodavanja novog proizvoda

Slika 27: Sažeti prikaz jelovnika u web pregledniku unutar Admin komponente - proizvodi koje dodaje administrator

Slika 28: Struktura koda za prikaz narudžbi i njihovih informacija nakon naručivanja od strane korisnika aplikacije

Slika 29: Prikaz narudžbi u web pregledniku

Slika 30: Uvođenje Login komponente unutar Admin komponente

Slika 31: Prikaz strukture koda za Login komponentu

Slika 32: Prikaz prijave za administratora u web pregledniku

Slika 33: Prikaz koda koji je potreban za spajanje Firebase-a s web aplikacijom

Slika 34: Potreban kod za spajanje web aplikacije s Firebaseom unutar Javascript datoteke firebaseConfig.js

Slika 35: Referenciranje Firebase baze podataka

Slika 36: Unaprijed zadane vrijednosti pravila unutar Firebase baze podataka

Slika 37: Promijenjena pravila baze podataka za omogućenje pristupa administratoru

Slika 38: Dodavanje novih proizvoda u menu, tj. jelovnik

Slika 39: Okidač (eng. trigger) odvijanja funkcija addMenuItem ()

Slika 40: Firebase izbor metoda za autentifikaciju

Slika 41: Gumbi za prijavu i odjavu administratora

Slika 42: Prikaz metoda za prijavu i odjavu administratora

Slika 43: Šifre koje predstavljaju tip geške prilikom autentifikacije i validacije

Slika 44: Deklaracija Vue Router-a

Slika 45: Način definiranja i deklariranja ruta

Slika 46: Dodavanje vue i vuex modula u centralizirano skladište

Slika 47: Puni prikaz store.js datoteke

Slika 48: Computed properties koja se nalaze unutar Admin komponente unutar kojih se nalaze svi getteri koji međusobno komuniciraju

Slika 49: Prikaz hijerarhije centraliziranog skladišta te pripadnih datoteka i foldera

Slika 50: Mutacija koja se odnosi na dodavanje novih proizvoda u jelovnik

Slika 51: Mutacija koja se odnosi na korisnika koji želi pristupiti administracijskoj sekciji

Slika 52: Akcija za prikaz trenutnog korisnika

Slika 53: Sinkronizacija proizvoda s jelovnika s Firebase bazom podataka

Slika 54: Sinkronizacija narudžbi iz Vuex-a s Firebase bazom podataka

Slika 55: Povezivanje Firebase s aplikacijom

Slika 56: Prikaz cijele firebaseConfig.js datoteke i njezinih varijabli

Literatura

- [1] Vue.js, *Introduction*, <<https://vuejs.org/v2/guide/>>. Pristupljeno 28. travnja 2019.
- [2] Vue.js, *Single File Components*, <<https://vuejs.org/v2/guide/single-file-components.html/>>. Pristupljeno 5. rujna 2019.
- [3] Dev, *What is the difference between Methods, Computed, and Watchers?* <<https://dev.to/blankbash/what-is-the-difference-between-methods-computed-and-watchers-4egn>>. Pristupljeno 5.9.2019.
- [4] Reactgo, *VueJS Directives List Tutorial*, <<https://reactgo.com/vuejs-directives-list/>>. Pristupljeno 5.9.2019.
- [5] Wikipedia, *Firebase*, <<https://en.wikipedia.org/wiki/Firebase>>. Pristupljeno 30. travnja 2019.
- [6] Shaumik Daityari (2019), codeinwp, *Angular vs React vs Vue: Which Framework to Choose in 2019*, <<https://www.codeinwp.com/blog/angular-vs-vue-vs-react/>>. Pristupljeno 2.5. 2019.
- [7] Medium, *React vs Angular vs Vue.js — What to choose in 2019? (updated)* <<https://medium.com/@TechMagic/reactjs-vs-angular5-vs-vue-js-what-to-choose-in-2018-b91e028fa91d>>. Pristupljeno 2. svibnja 2019.
- [8] Alfonso Barros (2018), Medium, *Angular/React/Vue pros and cons*. <<https://medium.com/@afonsobarros/angular-react-vue-pros-and-cons-75e161311e86>>. Pristupljeno 3. svibnja 2019.
- [9] React, *Virtual DOM and Internals*, <<https://reactjs.org/docs/faq-internals.html>>. Pristupljeno 5. svibnja 2019.
- [10] Prezi, *Firebase vs MongoDB vs SQL*, <<https://prezi.com/l8ixc0mtifhs/firebase-vs-mongodb-vs-mysql/>>. Pristupljeno 6. svibnja 2019.
- [11] MongoDB, <<https://www.mongodb.com/>>. Pristupljeno 6. svibnja 2019.
- [12] Vue CLI, <<https://cli.vuejs.org/guide/#cli>>. Pristupljeno 7. svibnja 2019.
- [14] Bootstrap, <<https://getbootstrap.com/>>. Pristupljeno 8. svibnja 2019.
- [15] Vuex, <<https://vuex.vuejs.org/>>. Pristupljeno 3. lipnja 2019.