Izrada poslužiteljskog dijela aplikacije koristeći Node ekosustav

Šomođi, Mario

VELEUČILIŠTE U VIROVITICI

MARIO ŠOMOĐI

IZRADA POSLUŽITELJSKOG DIJELA APLIKACIJE KORISTEĆI NODE EKOSUSTAV

ZAVRŠNI RAD

Virovitica, 2022.

VELEUČILIŠTE U VIROVITICI

Preddiplomski stručni studij Računarstva, smjer Programsko inženjerstvo

Mario Šomođi

IZRADA POSLUŽITELJSKOG DIJELA APLIKACIJE KORISTEĆI NODE EKOSUSTAV

ZAVRŠNI RAD

radi stjecanja akademskog zvanja stručnog prvostupnika računarstva

Virovitica, 2022.

Veleučilište u Virovitici

Preddiplomski stručni studij Računarstva - Smjer Programsko inženjerstvo

OBRAZAC 1b

ZADATAK ZAVRŠNOG RADA

Student/ica: **MARIO ŠOMOĐI** **JMBAG:** **0307016892**

Imenovani mentor: Ivan Heđi, dipl. ing., v. pred.

Imenovani komentor: -

Naslov rada:

Izrada poslužiteljskog dijela aplikacije koristeći Node ekosustav

Puni tekst zadatka završnog rada:

U vašem radu opišite Node ekosustav u svrhu izrade aplikacija. Kao praktični primjer implementacije teorijskog dijele osmislite i izradite aplikaciju za rezervaciju stolova ugostiteljskih objekata. Za pohranu podataka koristite Firebase. Detaljno se osvrnite za poslužiteljski dio aplikacije. Osim programskog rješenja, u pisanom dijelu završnog rada opišite ukratko korištene tehnologije, opišite detaljno arhitekturu sustava te opišite detaljno odabrane procese i/ili funkcije unutar same aplikacije. Prilikom opisivanja, osim neformalnih koristite i neke od formalnih metoda koje poznajete.

Datum uručenja zadatka studentu/ici: 28.07.2022..
Rok za predaju gotovog rada: 09.09.2022.

Mentor:

Ivan Heđi, dipl. ing., v. pred.

Dostaviti:
1. Studentu/ici
2. Povjerenstvu za završni rad - tajniku

Temeljna dokumentacijska kartica

IZRADA POSLUŽITELJSKOG DIJELA APLIKACIJE KORISTEĆI NODE EKOSUSTAV

Sažetak

Svrha ovog rada je osmisliti i kreirati poslužiteljski dio aplikacije. Drugim riječima, kreiranje Aplikacijskog programskog sučelja (API, eng. Application programming interface) koje bi klijentskom dijelu aplikacije vraćalo sve potrebne podatke u svrhu prikazivanja. API je podijeljen na četiri kategorije krajnjih točaka. Administracijske krajnje točke koje omogućuju dohvaćanje konfiguracijskih podataka potrebnih na klijentskom dijelu aplikacije. Krajnje točke za korisnike koje upravljaju s kreiranjem podataka o korisniku, ažuriranjem podataka te dohvaćanjem podataka korisnika. Krajnje točke za ugostiteljske objekte koje omogućuju kreiranje i brisanje ugostiteljskih objekata te dohvaćanje podatka o ugostiteljskim objektima, također postoji i mogućnost dohvaćanja podataka po određenim parametrima kao što su vlasnik objekta i pretraga po imenu objekta. Krajnje točke za rezervacije koje su zaslužene za kreiranje nove rezervacije, promjenu statusa rezervacije, dohvaćanja rezervacija po određenim parametrima kao što su vlasnik objekta za koji je rezervacija kreirana te po korisniku koji je zatražio rezervaciju. Zaslužene su i za slanje obavijesti o novim rezervacijama ugostitelju, promjeni statusa rezervacije korisniku te kreiranja QR koda pri odobrenju rezervacije koji sadrži podatke o rezervaciji. Tehnologije koje su korištene pri izradi API-a su JavaScript, Node.js, Express, Firebase-admin sdk, za bazu podataka korišten je Firestore Database i za spremanje slika korišten je Firebase Storage. Pristup API-u je osiguran pomoću JSON mrežnih tokena (JWT, eng. JSON Web Tokens).

(48 stranica, 18 slika, 5 programskih kodova)

Ključne riječi: REST API, Node, JavaScript, Firebase, Rezervacije, Express, Ugostiteljski objekti

Mentor: Ivan Heđi, dipl.int., v.pred.

Basic documentation card

DEVELOPMENT OF THE SERVER PART OF THE APPLICATION USING THE NODE ECOSYSTEM

Abstract

The purpose of this work is to design and create the server part of the application. That is, creating the Application Programming Interface (API) which will return all the necessary data to the client part of the application for display purposes. The API is divided into four categories of endpoints. Administrative endpoints that allow retrieving configuration data that is required on the client side of the application. Endpoints for users that manage the creation of user data, updating data and retrieving user data. Endpoints for establishments that enable the creation and deletion of establishments and the retrieval of their data, there is also the possibility of retrieving data by certain parameters such as the owner of the establishment and by searching by the name of the establishment. Reservation endpoints that are used for creating a new reservation, changing the status of a reservation, retrieving reservations by certain parameters such as the owner of the establishment for which the reservation was created and by the user who requested the reservation. These endpoints are also the ones that send notifications about new reservations to the establishment's owner, when reservation status changes they notify the user, and they also create a QR code if a reservation has been accepted that contains information about the reservation. The technologies used in creating the API are JavaScript, Node.js, Express, Firebase-admin sdk. Firestore Database was used for storing the data, and Firebase Storage was used for saving images. Access to the API is secured with JSON Web Tokens (JWT).

(48 pages, 18 figures, 5 code samples)

Keywords: REST API, Node, JavaScript, Firebase, Reservations, Express, Establishments

Supervisor: Ivan Heđi, dipl.int., v.pred.

Sadržaj

Uvod

Aplikacijsko programsko sučelje (API, eng. Application programming interface) je zapravo skup definicija i protokola za izgradnju i integraciju aplikacijskog softvera. Na API se može gledati kao posrednika on je zaslužen da komunicira s korisnicima tj. klijentskom stranom aplikacije te da im dostavlja potrebne podatke ili izvršava potrebne akcije. Recimo ako korisnik želi rezervirati stol u ugostiteljskom objektu poslat će zahtjev, a API će obraditi taj zahtjev provjeriti je li je on pravilan ako nije vratit će korisniku povratnu informaciju o tomu što treba popraviti. U slučaju da je sve uredu stupit će u kontakt s bazom podataka i spremiti rezervaciju te ju prikazati vlasniku ugostiteljskog objekta.

Svrha izrade ovog API-a je ta da klijentska strana ima lagani pristup svim podatcima i funkcionalnostima koje su joj potrebne. API je razvijen pomoću Čiste arhitekture (eng. Clean Architecture) te je podijeljen na 4 sloja koji su subjekti (eng. entities), slučajevi korištenja (eng. use cases), adapteri sučelja (eng. Interface Adapters) te na okvire i pokretače (eng. Frameworks and drivers). Izrađen je u Node.js tehnologiji s JavaScript jezikom uz Express okvir.

Ideja za aplikaciju je došla radi toga što svaki puta kada negdje želimo rezervirati stol moramo prvo tražiti na koji način vlasnik toga ugostiteljskog objekta želi da se to učini preko poruke, poziva, Facebook stranice, Instagram profila… Cilj aplikacije je da na jednom mjestu imamo sve ugostiteljske objekte te da u samo par dodira možemo rezervirati stol u bilo kojem željenom objektu.

U ovom radu postoje četiri dijela koja opisuju rad aplikacijskog programskog sučelja. Prvi dio opisuje izabrane tehnologije koje su se koristile prilikom izrade ovog aplikacijskog rješenja. Tehnologije koje će biti opisane su: Node.js, JavaScript, JSON, JWT, Express, Babel, Webpack, Swagger, Firebase, REST API. Drugi dio detaljno opisuje arhitekturno rješenje API-a i kako pojedinačni dijelovi komuniciraju jedni s drugima. U trećem dijelu bit će opisana struktura baze podataka. Četvrti dio opisuje funkcijske aspekte API-a.

1. Korištene tehnologije

Ovaj dio rada sadrži objašnjenja programskih tehnologija koje su bile korištene pri izradi ovoga aplikacijskog programskog sučelja.

1.1. JavaScript (JS)

JavaScript (JS) je lagani, interpretirani programski jezik s prvoklasnim funkcijama što znači da se funkcije u tom jeziku tretiraju kao bilo koja druga varijabla. Funkcije mogu biti proslijeđene kao argument drugim funkcijama, mogu biti vraćene te dodijeljene kao vrijednost varijabli. Iako je najpoznatiji kao skriptni jezik za web stranice, koriste ga još i mnoga okruženja kao što su Node.js, Adobe Acrobat.. Podržava objektno orijentirane, deklarativne te imperativne stilove. JS je dinamički jezik temeljen na prototipima, s više paradigmi i s jednom niti na webu se izvodi na strani klijenta. Jednostavan je za naučiti i jako koristan, naširoko se koristi za kontrolu ponašanja web stranica ali ima i druge svrhe.

1.2. REST API

REST API je aplikacijsko programsko sučelje koje je u skladu s ograničenjima arhitektonskog stila reprezentativnog prijenosa stanja (REST, eng. representational state transfer). Omogućuje interakciju s drugim web uslugama koje su također u skladu s ograničenjima arhitektonskog stila reprezentativnog prijenosa stanja.

1.3. JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) dizajniran je kao jednostavni standardni tekstualni format kako bi omogućio prijenos i razmjenu podataka koje je lagano iščitati ljudima te računalnim programima. Predstavlja strukturirane podatke na temelju sintaksi JavaScript objekata. Općenito se koristi za prijenos podataka u web aplikacijama (npr. slanje određenih podataka od poslužitelja do klijenta kako bi se onda ti podatci mogli prikazati klijentu u obliku web stranice, ili obrnuto). Iako jako podsjeća na sintaksu JavaScript objekata, moguće ga je koristiti neovisno i izvan JavaScript jezika, a mnoga programska okruženja imaju mogućnost čitanja i generiranja JSON-a.

1.4. Babel

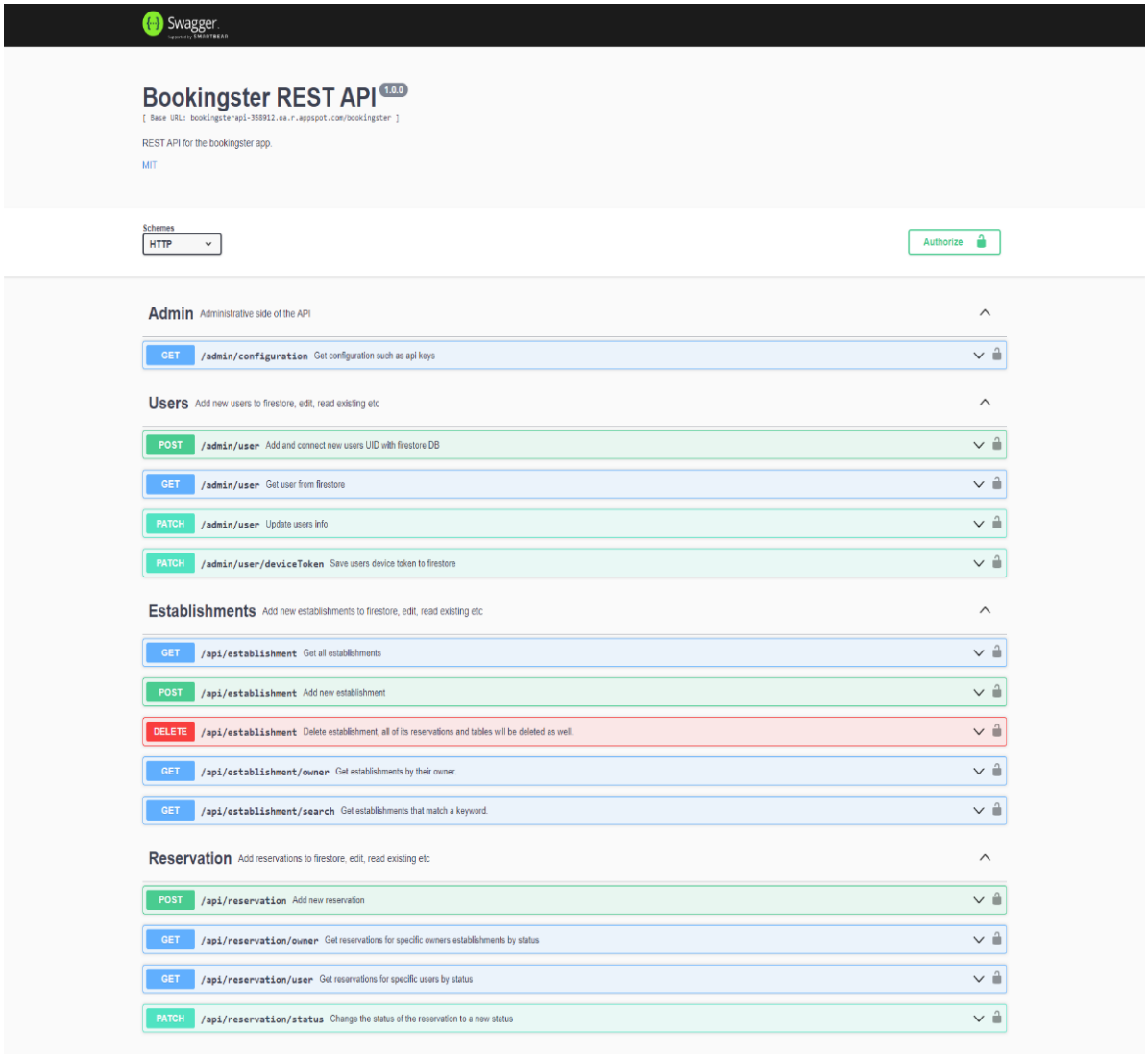
Babel je JavaScript kompajler koji se uglavnom koristi za pretvaranje novijeg standarda JS koda u onaj kompatibilan sa svim preglednicima i okruženjima. Babel može transformirati sintaksu i izvorni kod. Ima i podršku za transformaciju JSX i React sintaksi. Pokušava transformirati kod na najbolji način tako da ima što manje koda u krajnjoj datoteci.

1.5. Webpack

Webpack se koristi za kompiliranje JavaScript modula, u svojoj jezgri je on sakupljač statičkih modula za moderne JavaScript aplikacije. Tijekom obrađivanja aplikacije Webpack interno gradi grafikon ovisnosti iz jedne ili više ulaznih točki, a zatim kombinira svaki modul koji je vašem projektu potreban u jedan ili više paketa tj. statička sredstva za posluživanje kompiliranog sadržaja. U kombinaciji s Babel-om, Webpack kreira paket koji se može iskoristiti za posluživanje kompiliranog sadržaja na bilo kojem pregledniku ili okruženju.

1.6. Swagger

Swagger omogućuje opis strukture API-ja tako da ih strojevi mogu čitati. Uz pomoć toga Swagger može automatski izgraditi prekrasnu i interaktivnu API dokumentaciju. Uspijeva to tako što traži od API-a JSON ili YAML datoteku koja sadrži detaljan opis cijele strukture API-a. Ta datoteka sadrži opis resursa vašeg API-a koji se pridržava specifikacije OpenAPI.



Slika 1. Primjer izgleda dokumentacije generirane Swagger-om

1.7. JSON Web Tokens

JSON Web Token (JWT) je otvoreni standard koji definira samostalan i kompaktan način za sigurni prijenos informacija između dvije strane u obliku JSON objekt. On je digitalno potpisan. Može se potpisati korištenjem tajne ili parom javnih i privatnih ključeva pomoću RSA ili ECDSA. Naveliko se koristi za autorizaciju, nakon što se korisnik prijavi svaki sljedeći zahtjev uključivat će njegov JWT po kojemu će se onda dopuštati određeni pristup rutama, uslugama i resursima.

1.8. Node.js

Node.js je asinkroni JavaScript runtime vođen događajima. Dizajniran je za izgradnju skalabilnih mrežnih aplikacija tj. on je serversko okruženje i izvršava JavaScript kod izvan web preglednika. Node.js predstavlja petlju događaja kao konstrukciju runtime-a, on ulazi u petlju događaja nakon izvršavanja ulazne skripte te izlazi iz nje nakon što više nema povratnih poziva za izvršavanje. Omogućuje razvojnim programerima korištenje JavaScript-a za pisanje alata naredbenog retka, skriptiranje na strani poslužitelja. Omogućava pokretanje skripti na strani poslužitelja web stranice prije nego što se stranica pošalje web pregledniku korisnika. Predstavlja paradigmu „JavaScript posvuda“ spajajući razvoj web aplikacija oko jednog programskog jezika, umjesto više različitih jezika za skripte na strani poslužitelja i na stani klijenta.

1.9. Express

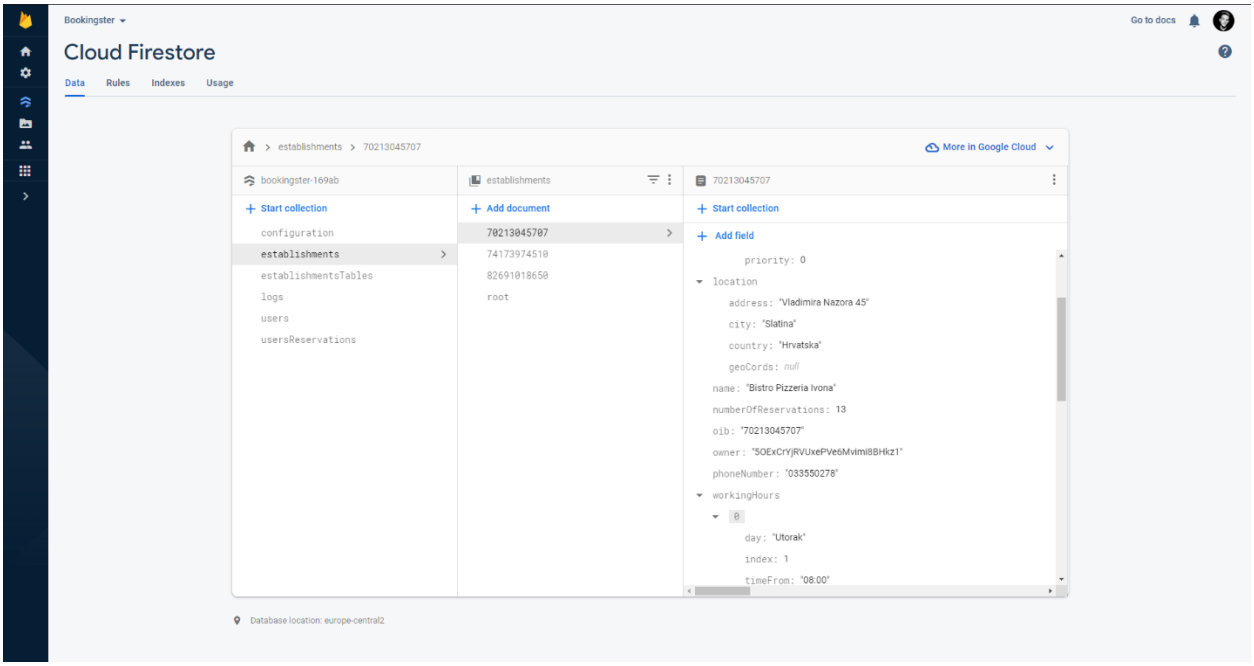
Express je samostalan, fleksibilan, minimalističan te veoma brzi web okvir za Node.js koji pruža skup značajki za web i mobilne aplikacije. Express sadrži mnoštvo HTTP uslužnih metoda i međuprograma koji su vam na raspolaganju za stvaranje API-ja.

1.10. Firebase

Firebase je platforma za razvoj aplikacija koja vam pomaže u izradi i razvoju aplikacija. U nastavku će biti opisani neki od Firebase-ovih programskih rješenja.

1.10.1. Firestore Database

Firestore je skalabilna, fleksibilna baza podataka za mobilne, web i poslužiteljske aplikacije od Google-a. Firestore održava sinkronizaciju vaših podataka i nudi izvanmrežnu podršku za mobilne uređaje i web kako biste mogli izraditi responzivne aplikacije koje rade bez obzira na latenciju mreže ili internetsku povezanost. To je NoSQL baza podataka smještena u oblaku. Podatke pohranjujete u dokumente (eng. document) koji sadrže mapirana polja u vrijednosti, a ti dokumenti su pohranjeni u zbirkama (eng. collection) koje su spremnici za dokumente te njih možete koristiti za organizaciju vaših podataka i izradu upita.



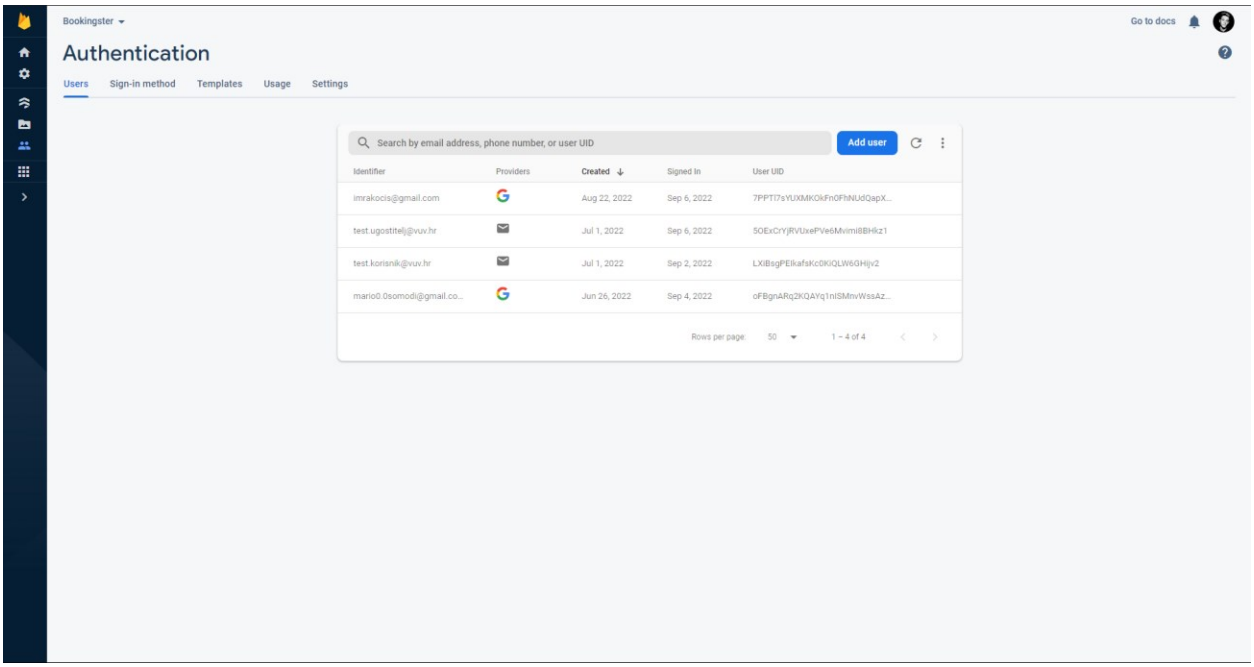
Slika 2. Primjer izgleda sučelja Firestore Database-a

1.10.2. Firebase Storage

Firebase Storage je jednostavna usluga za pohranu datoteka kao što su slike, audio zapisi, videozapisi ili drugi sadržaj. Ako je mreža loša, klijent može ponoviti operaciju tamo gdje je stao, štedeći klijentu vrijeme. Programeri mogu koristiti Firebase SDK-ove za pohranu u Firebase Storage.

1.10.3. Firebase Authentication

Firebase Authentication pruža usluge te SDK-ove jednostavne za korištenje, podržava autentifikaciju korištenjem e-pošte i lozinke, telefonskih brojeva, popularnih pružatelja identiteta kao što su Google, Facebook, Twitter...



Slika 3. Prikaz izgleda sučelja Firebase Authentication-a

1.10.4. Firebase Admin SDK

Firebase Admin SDK je skup poslužiteljskih biblioteka koje vam omogućuju interakciju s Firebase-om iz ovlaštenih okruženja radi izvođenja radnji s punim administratorskim pristupom.

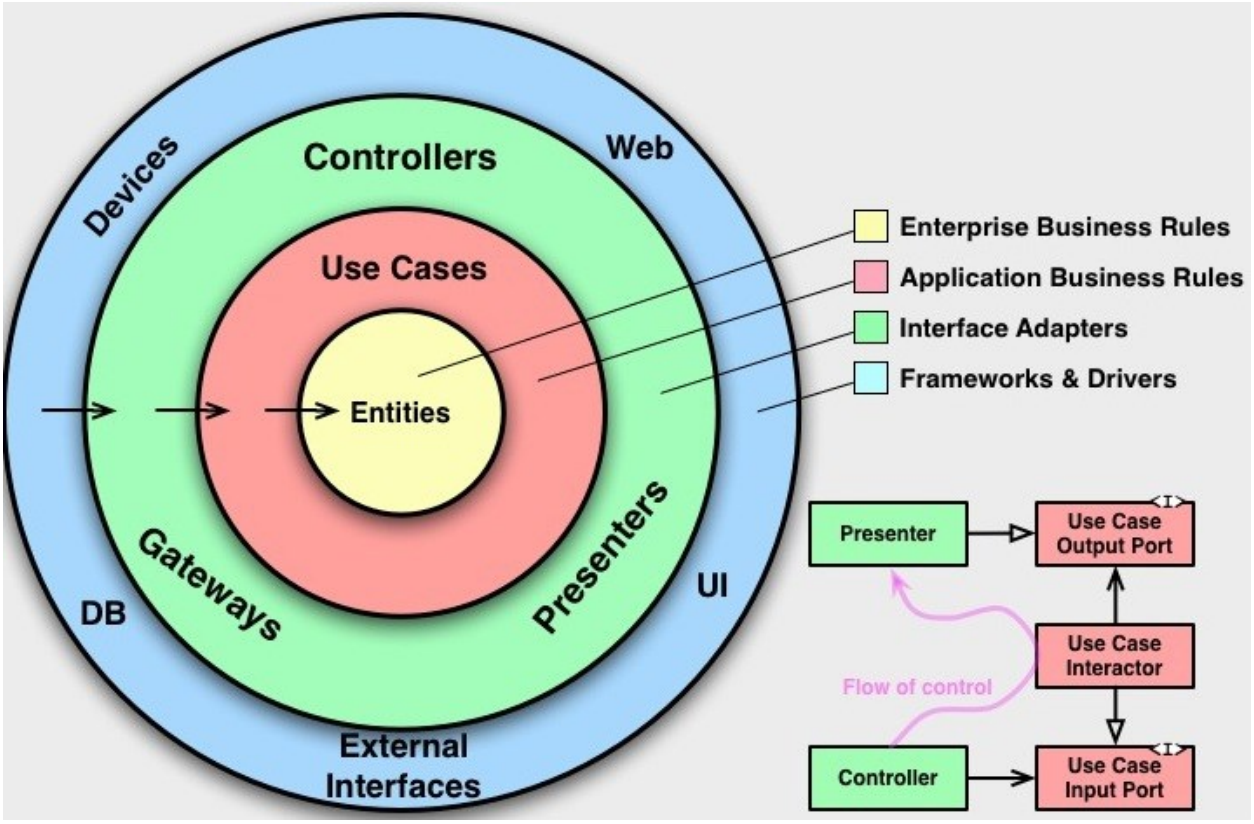
1.10.5. Firebase Cloud Messaging

Firebase Cloud Messaging je programsko rješenje za razmjenu poruka na više platformi koje vam omogućuje pouzdano slanje poruka bez ikakvih troškova.

2. Arhitektura aplikacijskog programskog sučelja

Ovaj dio rada će objasniti arhitekturu aplikacijskog programskog sučelja te kako dijelovi arhitekture komuniciraju jedni s drugima. U ovom radu korištena je Čista arhitektura (eng. The Clean Architecture).

2.1. Čista arhitektura (The Clean Architecture)

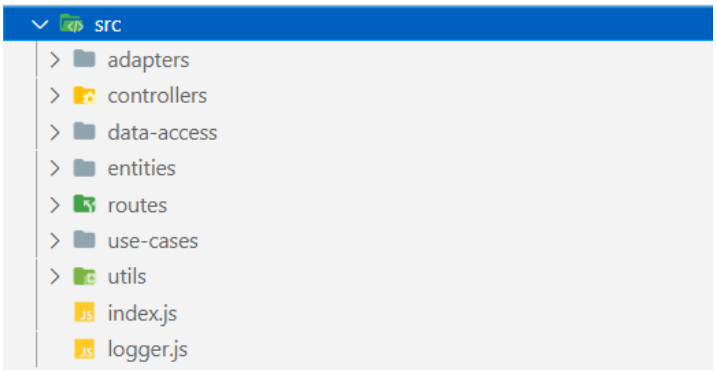


Slika 4. Prikaz Čiste arhitekture [21]

Sastoji se od 4 sloja. Subjekti (eng. entities) tj. glavna pravila aplikacije (eng. Enterprise Business Rules) koja sadrže opća te najbitnija pravila, za njih je najmanje vjerojatno da će se promijeniti kada se bilo što od vanjskih sustava ili pravila promjeni. Slučajevi korištenja (eng. use cases) tj. poslovna pravila specifična za aplikaciju (eng. Application Business Rules) ona orkestriraju protok podataka do i od subjekata i time onda usmjeravaju subjekte da koriste svoja pravila kako bi postigli ciljeve slučaja korištenja. Adapteri sučelja (eng. Interface Adapters) pretvaraju podatke iz formata koji je najprikladniji za slučajeve upotrebe i subjekte u

format koji je najprikladniji za vanjsku upotrebu, oni sadrže i kontrolere. Okviri i pokretači (eng. Frameworks and drivers) sastoje se od baze podataka, vanjskih usluga itd..

Cilj ove arhitekture je smanjenje problema do kojih može doći ako se pojedinačni dijelovi aplikacijskog programskog sučelja isprepletu. To postiže tako što razdvaja dijelove aplikacijskog programskog sučelja u slojeve. Najvažnije pravilo koje čini ovu arhitekturu funkcionalnom je Pravilo ovisnosti (eng. The Dependency Rule) ovo pravilo kaže da se ovisnosti koda mogu samo usmjeravati prema unutra (Slika 4.).



Slika 5. Prikaz arhitekture aplikacije

U donjem desnom dijelu dijagrama je primjer kako prelazimo granice kruga. Recimo imamo slučaj korištenja koji dodaje rezervaciju ali taj slučaj bi trebao pristupiti bazi podataka kako bi spremio tu rezervaciju, a baza podataka se nalazi u najvišem sloju. Pri tom slučaju pošto JavaScript nema Sučelja (eng. Interface) kao što slika pokazuje, to ćemo postići s Dependency Injection-om. Na programskom kodu 1. možemo vidjeti primjer Dependency Injection-a, unutar parametara funkcije „makeListEstablishments“ imamo objekt koji sadrži sve ovisnosti (eng. Dependency). Ovisnosti dolaze iz programskog koda 2. koji je zadužen za kreiranje funkcije listEstablishment te će onda on poslati sve njene ovisnosti funkciji „makeListEstablishments“.

Programski kod 1. Primjer Depenedency Injection-a

```
1.  import { makeEstablishment } from '../../entities';
2.
3.  export default function makeListEstablishments({
4.    establishmentsCollection,
5.    CRUDDb,
6.    establishmentsTablesCollection,
7.    usersReservationsCollection,
8.  }) {
9.    return async function listEstablishment() {
10.     const establishments = await CRUDDb.getAllFromCollection({
11.       collection: establishmentsCollection,
12.     });
13.     const listEstablishments = [];
14.     await Promise.all(
15.       ..
16.       ..
17.       ..
18.     );
19.     return listEstablishments;
20.   };
21. }
22.
```

Programski kod 2. Primjer izvora ovisnosti koje su Inject-ane

```
1.  import {
2.    usersCollection,
3.    CRUDDb,
4.    establishmentsCollection,
5.    establishmentsTablesCollection,
6.    storageActions,
7.    usersReservationsCollection,
8.  } from '../../data-access';
9.  import makeCreateEstablishment from './createEstablishment';
10. import makeFetchOwnersEstablishments from './fetchOwnersEstablishments';
11. import makeListEstablishments from './listEstablishments';
12. import makeRemoveEstablishment from './removeEstablishment';
13. import makeSearchEstablishments from './searchEstablishments';
14.
15. export default function makeEstablishmentUseCases() {
16.   const listEstablishment = makeListEstablishments({
17.     establishmentsCollection,
18.     CRUDDb,
19.     establishmentsTablesCollection,
20.     usersReservationsCollection,
21.   });
22.   ..
23.   ..
24.   ..
25.   return Object.freeze({
26.     createEstablishment,
27.     fetchOwnersEstablishments,
28.     listEstablishment,
29.     searchEstablishments,
30.     removeEstablishment,
31.   });
32. }
```

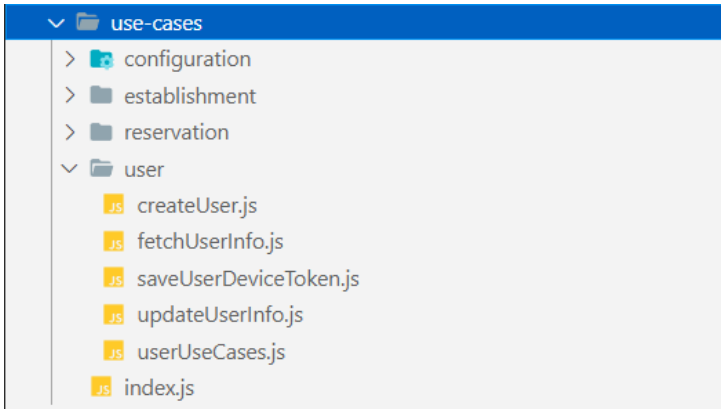
2.2. Glavna pravila aplikacije (eng. Enterprise Business Rules)

U ovom sloju se nalaze subjekti (Prilog 1.) aplikacije koji svaki pojedinačno sadrži sva pravila te validacije o tomu kako se oni trebaju kreirati i izgledati. Subjekti koji su definirani su konfiguracijski subjekt koji sadrži sve konfiguracijske podatke, subjekt ugostiteljskog objekta, subjekt konstrukcije kolekcije slika od ugostiteljskog objekta, lokacije, rezervacije, stola, korisnika, te kolekcije radnih sati ugostiteljskog objekta.

Slika 6. Struktura sloja glavnih pravila aplikacije tj. subjekata

2.3. Poslovna pravila specifična za aplikaciju (eng. Application Business Rules)

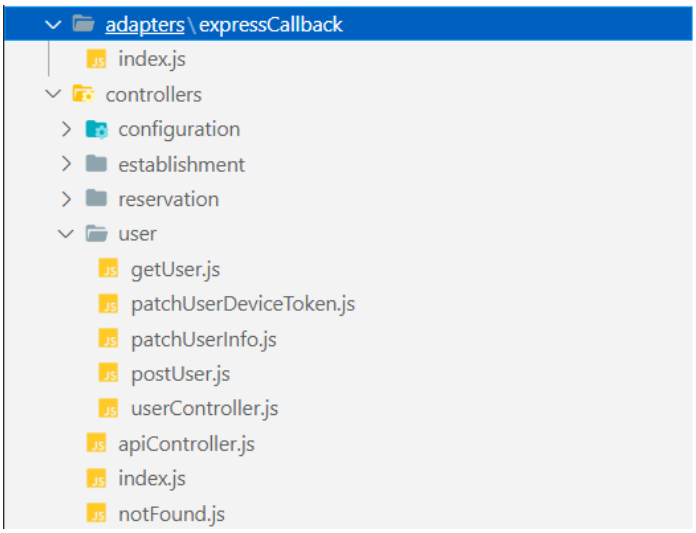
Ovaj sloj sadrži slučajeve korištenja (Prilog 2.) aplikacije koji su raspodijeljeni u 4 skupine koje su slučajevi korištenja za ugostiteljske objekte, rezervacije, konfiguraciju te korisnike. Oni komuniciraju sa subjektima tako što zatražuju od njih da provedu svoja pravila i validiraju podatke te vrate validan oblik sebe koji se dalje može koristiti u svrhu ispunjenja zadaće slučaja korištenja. Imaju komunikaciju i s višim slojevima pomoću Dependency Injection-a ako trebaju spremiti neki podatak u bazu ili pozvati neku vanjsku uslugu.



Slika 7. Prikaz strukture poslovnih pravila specifičnih za aplikaciju tj. slučajeva korištenja

2.4. Adapteri sučelja (eng. Interface Adapters)

U ovom sloju su kontroleri (Prilog 3.) i adapteri, kontroleri su podijeljeni na 4 skupine koje su kontroler za rezervacije, konfiguraciju, ugostiteljske objekte i korisnike. Svaki od njih sadrži sve svoje moguće akcije (Prilog 4.). Kontroleri koriste adapter za express callback (Primjer 5.) koji formatira svaku akciju u format koji express razumije. Komuniciraju sa slučajevima korištenja kako bi dobili podatke koje trebaju adaptirati za vanjsku uporabu.



Slika 8. Prikaz strukture adaptera sučelja te kontrolera

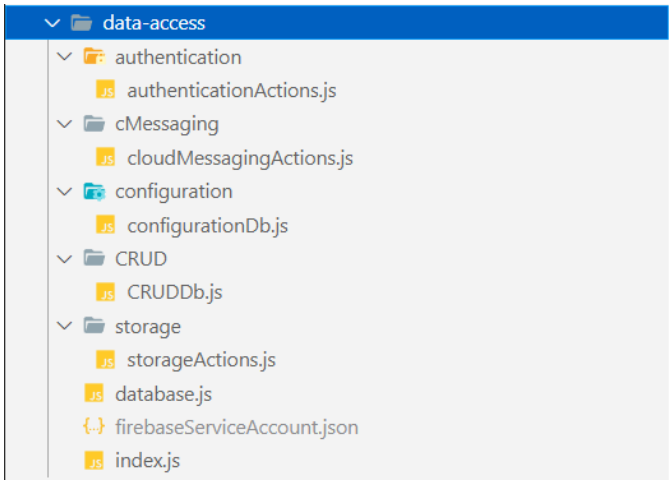
2.5. Okviri i pokretači (eng. Frameworks and Drivers)

Ovaj sloj je skup vanjskih usluga te akcija nad tim uslugama koje su API-ju potrebne. Sadrži konfiguraciju Firebase-a te spoj na bazu podataka za koju se koristi Firestore Database. Akcije nad bazom su definirane u objektu CRUDDb koji sadrži dinamičke funkcije s mogućnosti izvršavanja neke akcije na bilo kojoj kolekciji unutar Firestore-a.

Programski kod 3. Primjer jedne od funkcija koje sadrži objekt CRUDDb

```
1.     async function getDocumentsFromCollectionByPropertyValue({
2.       collection,
3.       propertyName,
4.       propertyValue,
5.     } = {}) {
6.       const results = [];
7.       const queryResult = await collection
8.         .where(propertyName, '==', propertyValue)
9.         .get();
10.      queryResult.forEach((doc) => {
11.        results.push(doc.data());
12.      });
13.      return results;
14.    }
15.
```

Uz konfiguraciju baze i akcija na nju ovaj sloj još sadrži spoj na autentifikaciju za koju se koristi Firebase Authentication pomoću kojega onda možemo dobiti pristup podatcima korisnika. On nam omogućava promjene korisnikovih podataka kao što su lozinke, e-pošte, ime i prezime korisnika. Firebase cloud messaging je također u ovom sloju i on nam omogućava kreiranje i slanje notifikacija korisnicima, uz njega još postoji i spoj na Firebase Storage pomoću kojeg spremamo i dohvaćamo slike u oblaku.



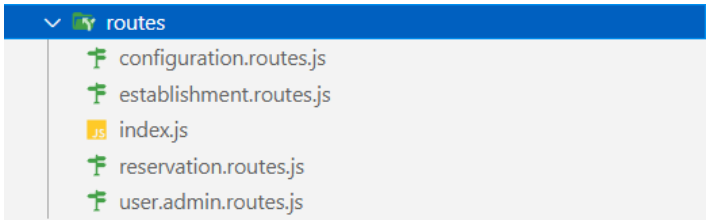
Slika 9. Prikaz strukture okvira i pokretača

2.6. Rute i razni alati

Arhitektura aplikacijskog programskog sučelja sadrži još rute koje su podijeljene u 4 grupe koje su konfiguracijska, ugostiteljski objekti, rezervacije i korisnici. Svaka od njih definira sve svoje krajnje točke te komuniciraju s kontrolerima kako bi ujedinili točnu krajnju točku s njenim kontrolerom.

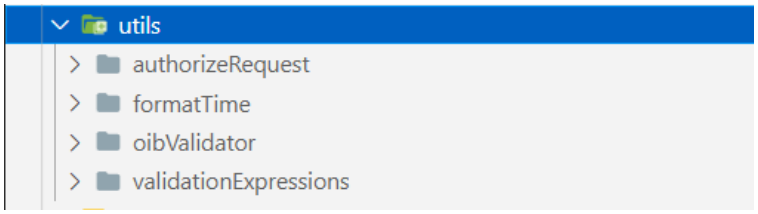
Programski kod 4. Primjer ruti za ugostiteljske objekte

```
1.  import express from 'express';
2.  import { establishmentController } from '../controllers';
3.
4.  const establishmentRouter = express.Router();
5.
6.  establishmentRouter.get('/', establishmentController.Get());
7.  establishmentRouter.get('/search', establishmentController.GetByKeyword());
8.  establishmentRouter.get('/owner', establishmentController.GetByOwner());
9.  establishmentRouter.post('/', establishmentController.Post());
10. establishmentRouter.delete('/', establishmentController.Delete());
11.
12. export default establishmentRouter;
13.
```



Slika 10. Prikaz strukture ruta

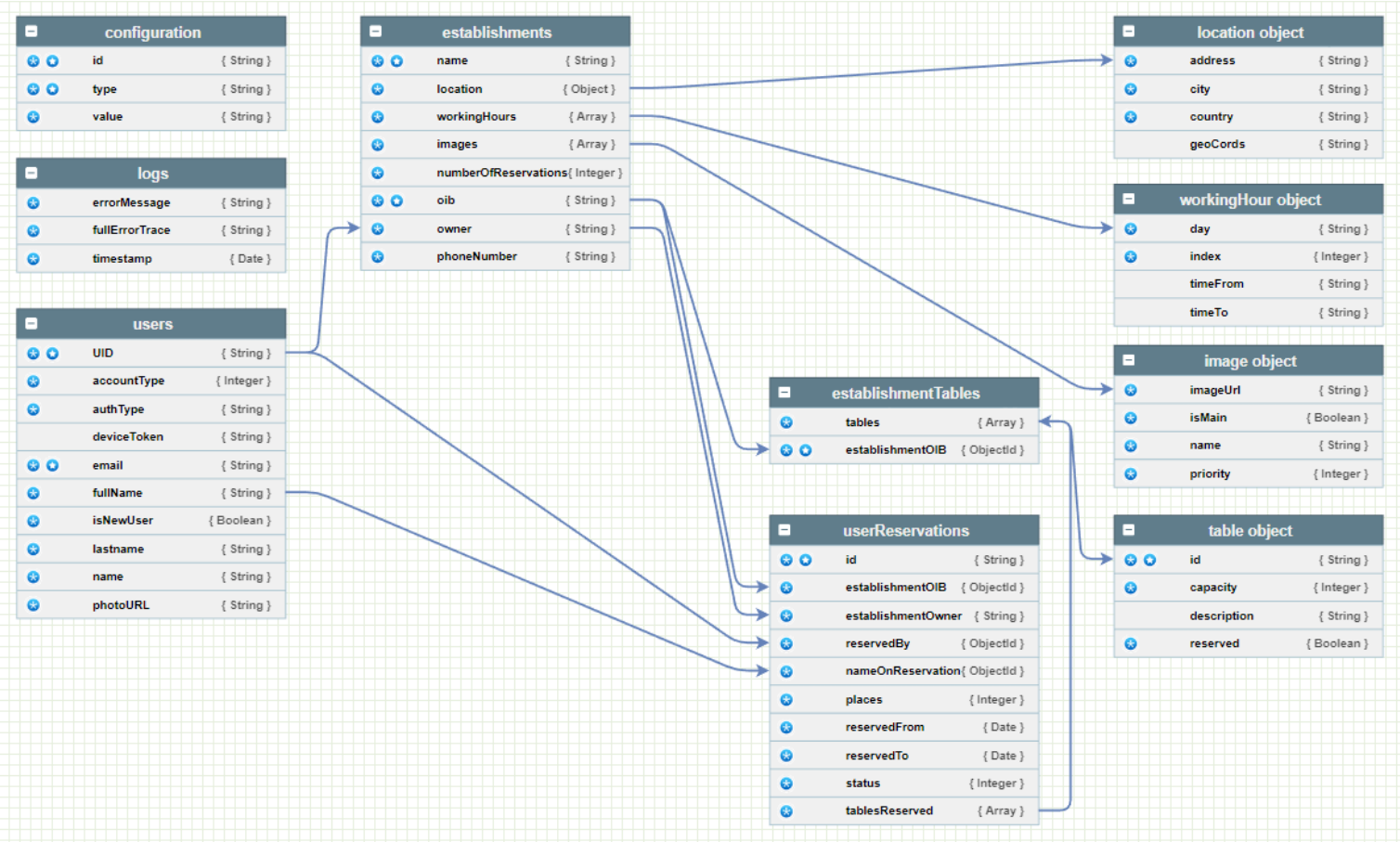
Razni alati sadrže kako i samo ime kaže razne alate koji se koriste kao što su validator osobnog identifikacijskog broja, pomoćne validacijske funkcije, te middleware koji se koristi za autorizaciju poziva na API (Prilog 6.).



Slika 11. Prikaz strukture raznih alata

3. Struktura baze podataka (Firestore)

Ovaj dio rada će objasniti strukturu baze podataka i međuovisnosti u njoj. Baza podataka sadrži 6 kolekcija svaka od njih sadrži svoje dokumente te njihove podatke, podatci se povezuju pomoću unikatnih identifikatora određenih objekata unutar baze. (Slika 12.).



Slika 12. Struktura baze

Kolekcija ugostiteljskih objekata (eng. establishments) predstavlja kolekciju dodanih ugostiteljskih objekata od strane bilo kojeg ugostitelja. Ugostiteljski objekti i ugostitelji su vezani pomoću parametara „owner“ od ugostiteljskog objekta i „UID“ parametra od ugostitelja koji ovise jedno o drugomu. Pomoću ove poveznice imamo mogućnost dohvaćanja ugostiteljskih objekata koji pripadaju samo određenom ugostitelju.

Stolovi ugostiteljskih objekata se nalaze u posebnoj kolekciji (eng. establishemntTables) koja sadrži međuovisnost „oib“ parametra ugostiteljskog objekta i „establishmentOIB“ parametra stolova. Preko te međuovisnosti možemo doći do stolova pojedinačnog ugostiteljskog objekta.

Kolekcija logova sadrži sve zapise greški do kojih je došlo pri radu API-a. Konfiguracijska kolekcija sadrži bitne podatke koje ne bi željeli spremiti negdje u aplikaciju da im netko može pristupiti.

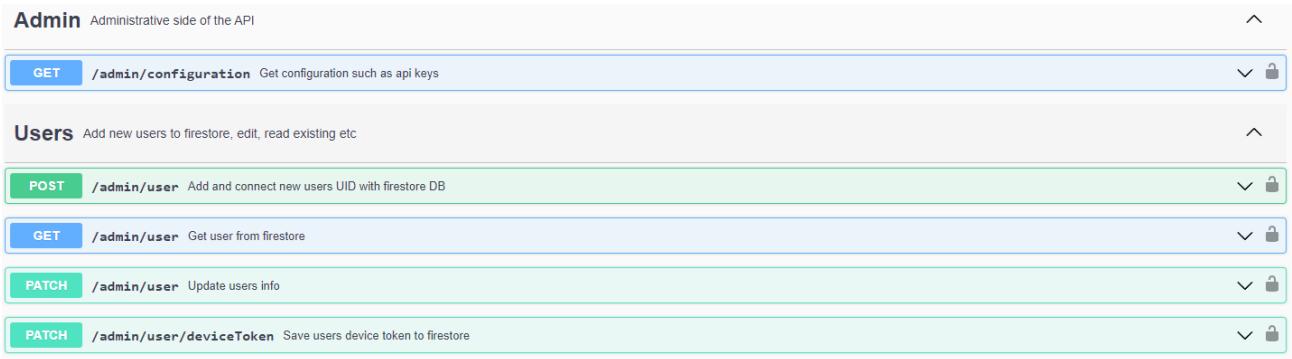
Korisnici (eng. users) i njihove rezervacije (eng. userReservations) su razdvojeni u dvije kolekcije. Rezervacije korisnika imaju 4 međuovisnosti od kojih su dvije od njih na tablicu ugostiteljskih objekta a dvije na tablicu korisnika. Kod ugostiteljskih objekata ovise o „oib“ parametru ugostiteljskog objekta koji se veže na „establishmentOIB“ parametar rezervacije radi ove ovisnosti dobivamo mogućnost dohvaćanja rezervacija bilo kojeg ugostiteljskog objekta. Ovisnost o „owner“ parametru ugostiteljskog objekta koji se veže na „establishmentOwner“ parametar rezervacije, pomoću koje dobivamo mogućnost dohvaćanja svih rezervacija u svakom od ugostiteljskih objekata određenog ugostitelja. Međuovisnosti prema korisnicima su ovisnost o „UID“ parametru korisnika koji se veže na „reservedBy“ parametar rezervacije, pomoću koje možemo dohvatiti sve rezervacije koje je određeni korisnik zatražio. Ovisnost o „fullName“ parametru korisnika koji se veže na „nameOnReservation“ parametar rezervacije koji nam omogućuje da znamo korisnikovo puno ime tijekom zahtjeva za rezervaciju.

4. Funkcionalnosti aplikacijskog programskog sučelja

Ovaj dio rada će objasniti funkcionalnosti aplikacijskog programskog sučelja te sve krajnje točke koje su definirane. Krajnje točke su podijeljene u 4 skupine te će svaka od njih u nastavku rada biti objašnjena.

4.1. Administracijske krajnje točke

Administracijske krajnje točke se sastoje od dvije skupine a to su krajnja točka za konfiguraciju te krajnje točke za korisnike. One se pozivaju na drugačiji način od ostalih krajnjih točaka API-a za njih je potrebno umjesto korisnikovog JWT-a poslati generirani administracijski JWT koji zapravo ima pristup na te krajnje točke.



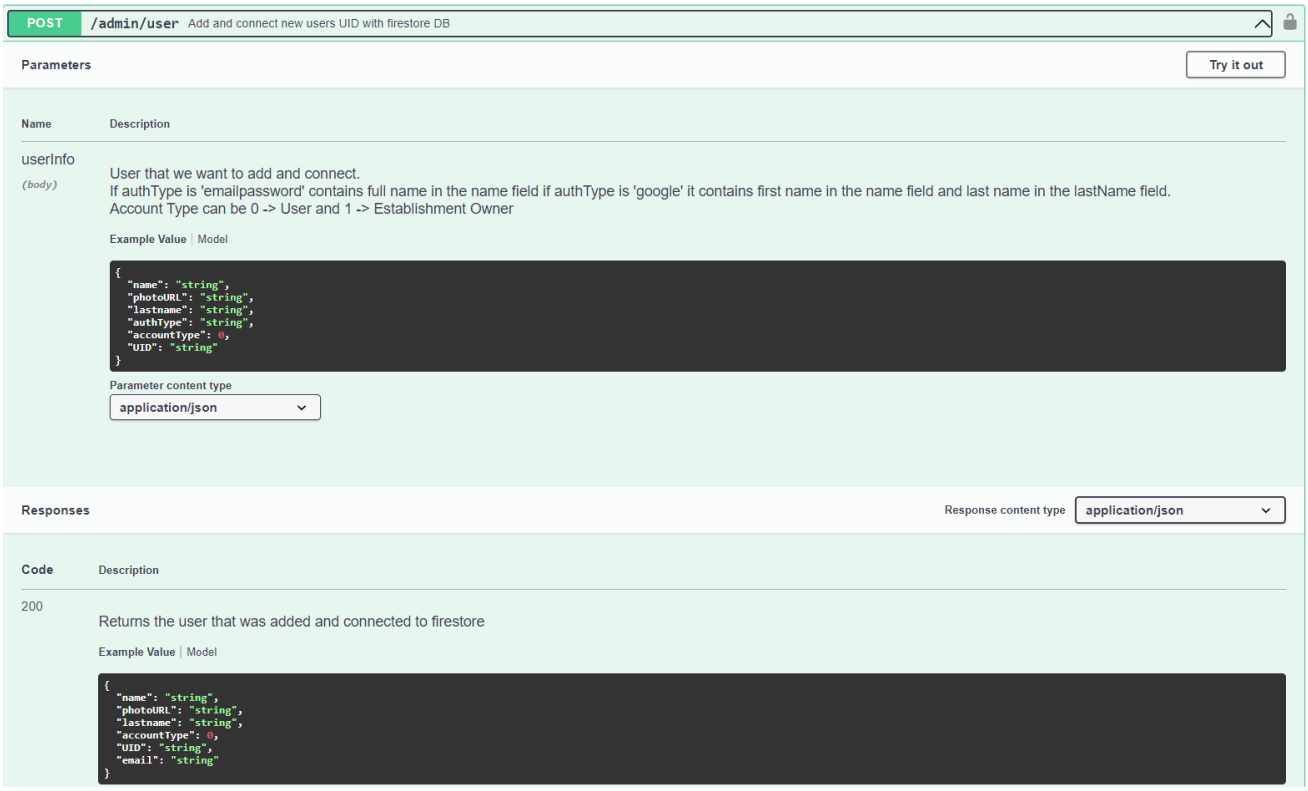
Slika 13. Izgled administracijskih krajnjih točaka u Swagger dokumentaciji

4.1.1. Konfiguracijska krajnja točka

Konfiguracijska krajnja točka će klijentskoj aplikaciji vratiti bitne podatke koji su osjetljivi te ih ne bi trebalo spremati unutar aplikacijskog koda ili negdje gdje se mogu lako pronaći.

4.1.2. Krajnje točke za korisnike

Ove krajnje točke sadrže sve akcije koje je moguće izvršiti nad korisnicima. Postoje 4 krajnje točke. Krajnja točka za spajanje korisnika koji se ulogira preko klijentske aplikacije s njegovim dodatnim podatcima koji se spremaju u bazu podataka za daljnje korištenje. Neki od tih podataka su tip računa koji može biti korisnik ili ugostitelj, korisnikova slika, tip prijave itd.

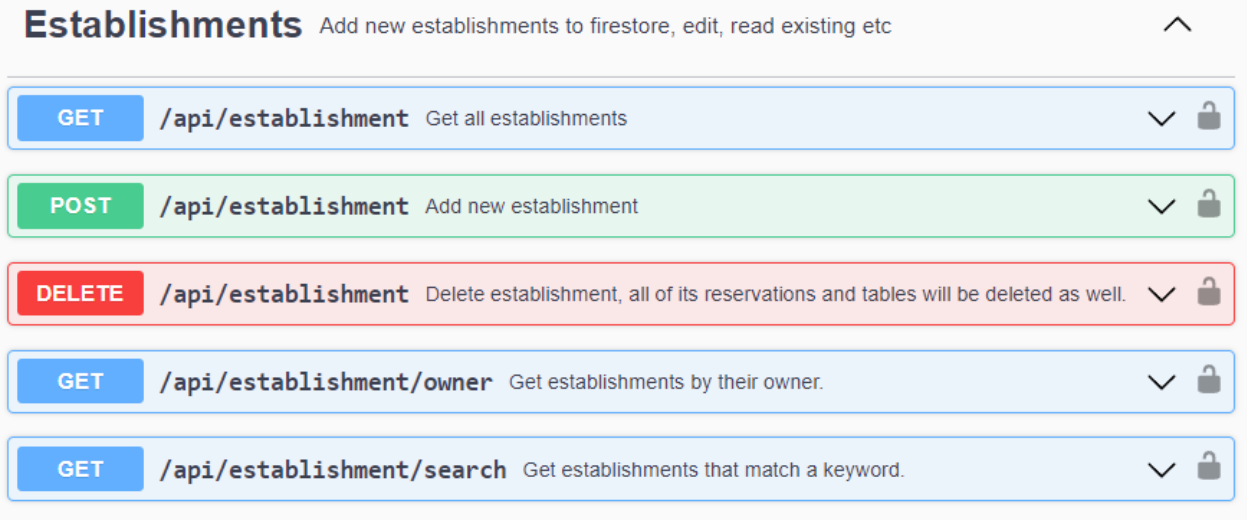


Slika 14. Izgled zahtjeva te odgovora krajnje točke za kreiranje korisnika

Krajnja točka za dohvaćanje korisnika prima identifikator korisnika tj. UID u upitu (eng. query) te vraća samo korisnika kojemu pri pada taj identifikator. Zadnje dvije krajnje točke služe za ažuriranje podataka korisnika. Jedna od njih je dinamička i može ažurirati jedno od navedenog e-poštu, lozinku, ili ime i prezime korisnika. Druga sprema identifikator mobilnog uređaja koji korisnik koristi. Identifikator korisnikovog uređaja je važan pri slanju obavijesti (eng. notification). Krajnje točke za ažuriranje će vratiti samo jedan parametar a to je uspjeh tj. jesu li podatci uspješno ažurirani ili ne.

4.2. Krajnje točke za ugostiteljske objekte

Krajnje točke za ugostiteljske objekte sadrže sve moguće akcije nad ugostiteljskim objektima. Kako bi im se pristupilo tijekom poziva krajnje točke se uz poziv šalje korisnikov JWT radi autorizacije.



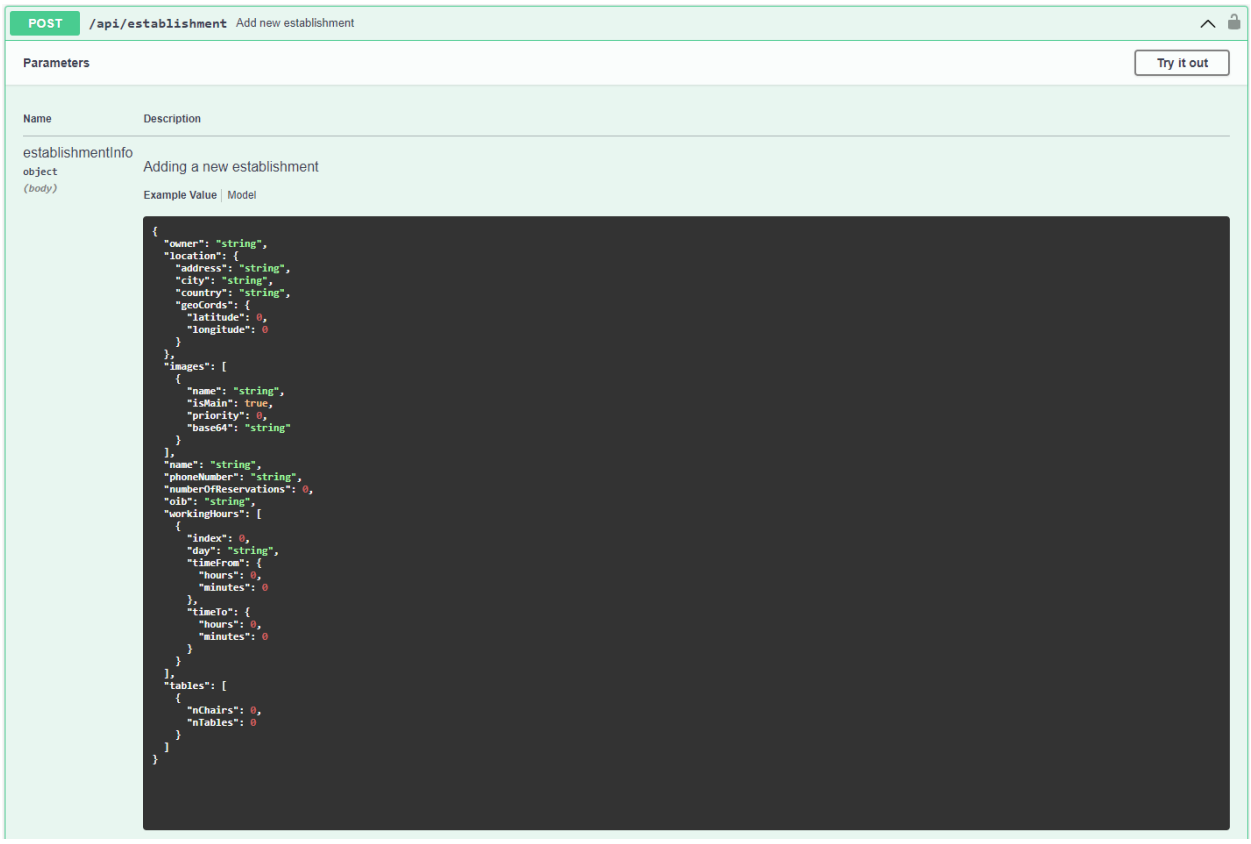
Slika 15. Izgled krajnjih točaka za ugostiteljske objekte

4.2.1. Krajnje točke za dohvaćanje ugostiteljskih objekata (GET akcije)

Postoje 3 krajnje točke koje dohvaćaju podatke o ugostiteljskim objektima. Jedna od njih će vratiti polje sa svim ugostiteljskim objektima koji postoje u bazi podataka. Druga prima u upitu (eng. query) identifikator ugostitelja te će vratiti samo objekte kojima je taj ugostitelj vlasnik. Zadnja krajnja točka prima bilo koju riječ u upitu (eng. query) te će vratiti sve ugostiteljske objekte koji sadrže tu riječ u svom imenu.

4.2.2. Krajnja točka za dodavanje ugostiteljskog objekta (POST akcija)

Krajnja točka za dodavanje prima sve podatke o ugostiteljskom objektu (Slika 16.) te ih onda formatira i provjerava kako bi kreirala format koji se sprema i dodaje u bazu podataka.



Slika 16. Izgled poziva na krajnju točku za dodavanje novog objekta

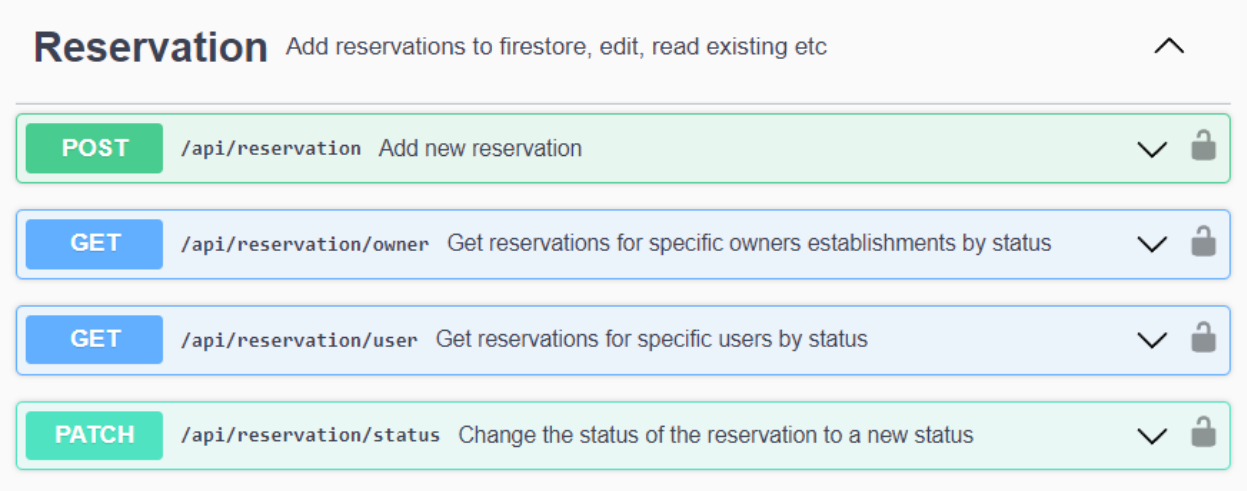
4.2.3. Krajnja točka za brisanje ugostiteljskog objekta (DELETE akcija)

Kao što i samo ime akcije kaže ova krajnja točka je namijenjena za brisanje ugostiteljskog objekta u upitu (eng. query) prima osobni identifikacijski broj ugostiteljskog objekta koji želimo obrisati. Brisanje ugostiteljskog objekta je malo kompliciraniji proces pošto se dosta toga veže na ugostiteljski objekt kao što su njegove rezervacije, stolovi, slike unutar Firebase Storage-a, generirani QR kodovi rezervacija koje su odobrene. Brisanjem ugostiteljskog objekta moramo pobrisati i sve te povezane stvari s njim.

4.3. Krajnje točke za rezervacije

Krajnje točke za rezervacije su također osigurane pomoću korisnikovog JWT-a te sadrže sve akcije koje mogu biti izvršene nad rezervacijama. Rezervacije mogu biti u 4 statusa. Prilikom kreiranja rezervacija će biti u statusu 0 koji znači da rezervacija čeka odgovor vlasnika ugostiteljskog objekta. Pri odobrenju od strane vlasnika ugostiteljskog objekta rezervacija će poprimiti status 1, a ako vlasnik odbije rezervaciju ona će poprimiti status 2. Prilikom

otkazivanja rezervacije od strane korisnika ona poprima status 4.Rezervacija poprima status 3 ako je trenutni datum veći od datuma kraja rezervacije što znači da je završena tj. prošla je.



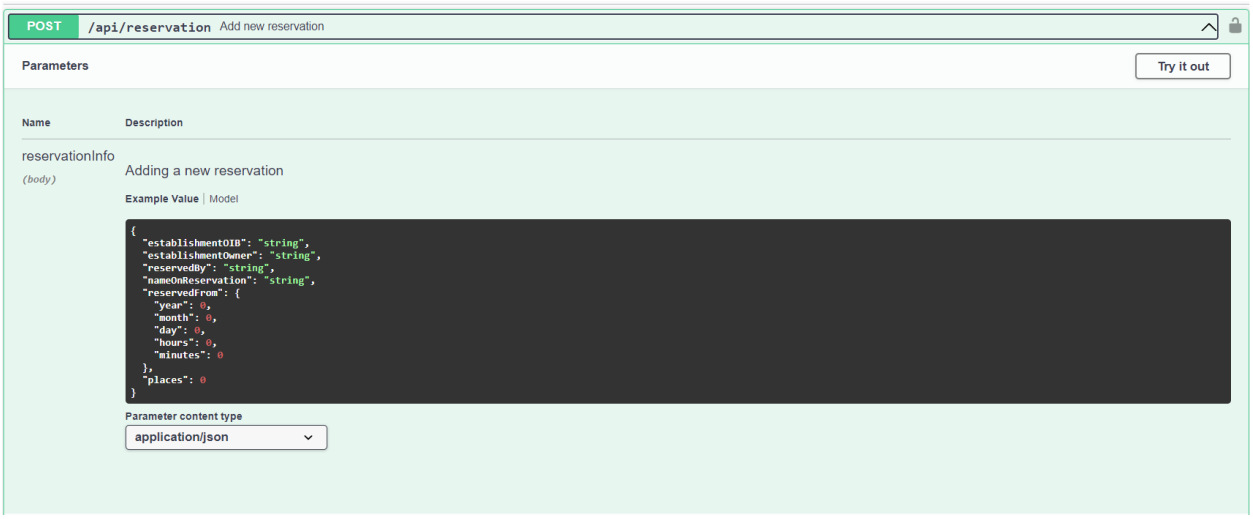
Slika 17. Izgled krajnjih točaka za rezervacije

4.3.1. Krajnje točke za dohvaćanje rezervacija (GET akcija)

Rezervacije se mogu dohvatiti na dva različita načina pomoću dvije krajnje točke. Jedan od njih je da se u upitu (eng. query) pošalje identifikator vlasnika ugostiteljskog objekta. Pri tom slučaju vrača se polje koje sadrži sve rezervacije od svakog ugostiteljskog objekta kojemu je ugostitelj s tim identifikatorom vlasnik. Drugi način je da se pošalje identifikator korisnika koji je zatražio rezervaciju te pomoću njega se mogu dohvatiti sve rezervacije tog korisnika.

4.3.2. Krajnja točka za dodavanje rezervacije (POST akcija)

Kod dodavanja nove rezervacije API će pomoću podataka koji su poslani na krajnju točku te algoritma odlučiti ima li zatraženi objekt dovoljno stolova koji su prikladni za potrebe zatražene rezervacije (Prilog 7.). Algoritam će prvo pokušati naći stol s točnim brojem mjesta koliko je i zatraženo. Ako to ne uspije naći onda će tražiti bilo koji stol koji ima maksimalno 2. mjesta više od zatraženih. U slučaju da i takvog stola nema pokušat će spojiti više stolova kako bi dobio dovoljno mjesta za zatraženu rezervaciju također u ograničenju toga da više od 2 mjesta ne smiju ostati prazna. Pri dodavanju rezervacija će poprimiti status 0 koji znači da je rezervacija u obradi i čeka odgovor vlasnika ugostiteljskog objekta. Obavijest vlasniku da ima novu rezervaciju koja čeka njegov odgovor će također biti poslana.



Slika 18. Izgled zahtjeva za dodavanje nove rezervacije

4.3.3. Krajnje točke za ažuriranje rezervacije (PATCH akcija)

Postoji jedna krajnja točka za ažuriranje rezervacije ona ima mogućnost promjene statusa rezervacije. Funkcionira tako što u upit (eng. query) prima identifikator rezervacije čiji status treba promijeniti, osobni identifikacijski broj ugostiteljskog objekta koji pripada toj rezervaciji, i novi status na koji da se rezervacija ažurira. Ako je novi status 1 onda API generira QR kod s podatcima rezervacije i sprema ga u Firebase Storage uz to još povećava brojač na ugostiteljskom objektu koji sadrži broj koliko puta je rezerviran taj ugostiteljski objekt. Za kraj će korisniku koji je zatražio rezervaciju poslati obavijest da je njegova rezervacija odobrena. Ako

je novi status 2 onda API samo šalje korisniku obavijest da je njegova rezervacija odbijena. Pri mijenjanju statusa na status 4 API će poslati obavijest vlasniku ugostiteljskog objekta da je korisnik otkazao rezervaciju.

Programski kod 5. Primjer slanja obavijesti (eng. notification)

```
1.  if (newStatus === 2) {
2.        await cloudMessagingActions.sendNotificationToSentUsers({
3.          UIDs: [reservation.reservedBy],
4.          messageTitle: establishment.name,
5.          messageBody: 'Vaša rezervacija je odbijena!',
6.          messageSubtitle: 'Rezervacije',
7.          largeTextToShow: `Vlasnik objekta ${establishment.name} je odbio vašu rezervaciju!`,
8.        });
9.      }
10.
```

5. Zaključak

Prije početka razvoja aplikacijskog programskog sučelja bilo je potrebno pobliže razjasniti problematiku koja će se rješavati njime te proučiti detaljno arhitekturu koja će se koristiti za izradu. Najveći fokus je stavljen na implementaciju arhitekture. Aplikacijsko programsko sučelje nije ostalo onako kako je na početku bilo osmišljeno ono se tijekom razvoja mijenjalo kako je dolazilo do promjena u načinu rada i tehnologija koje su se koristile. Radi toga je arhitektura sustava strukturirana tako da bude prilagođena promjenama i nadogradnji ako ikada dođe do izmjene nekog dijela ili neke tehnologije. Imat će mogućnost da se nove tehnologije ili promijene bez puno rada integriraju u trenutnu arhitekturu. Iako je arhitektura ono nešto što korisnici ne vide ona je uistinu jako bitan dio koji se bi trebao ignorirati pošto ona utječe na sigurnost, fleksibilnost, skalabilnost i na još mnoštvo toga.

Ovo programsko rješenje je provedeno u djelo pomoću JavaScript programskog jezika te nekoliko alata koji su izgrađeni na njemu kao što su Node.js, Express i još mnoštvo drugih. Baza podataka je kreirana kako bi aplikacijsko programsko sučelje moglo komunicirati s njom i upravljati podatcima koji su potrebni klijentskoj strani aplikacije. Radi Babel-a ovo aplikacijsko programsko sučelje radi na svim preglednicima bezobzira o njihovoj starosti.

Razvoj ovakvog aplikacijskog programskog sučelja donio je dobar doprinos i povećanje razumijevanja JavaScript programskog jezika, Node.js ekosustava te pisanja skalabilnog i fleksibilnog koda uz pomoć dobro proučene i uspostavljenje arhitekture.

Popis literature

[1] MDN contributors, JavaScript, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript

[2] Wikipedia, First-class function, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://en.wikipedia.org/wiki/First-class_function

[3] Wikipedia, Prototype-based programming, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://en.wikipedia.org/wiki/Prototype-based_programming

[4] MDN contributors, About JavaScript, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://developer.mozilla.org/en-US/docs/Web/JavaScript/About_JavaScript

[5] Red Hat, What is a REST API?, 2020, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://www.redhat.com/en/topics/api/what-is-a-rest-api

[6] JSON.org, Introducing JSON, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://www.json.org/json-en.html

[7] Babel, What is Babel?, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://babeljs.io/docs/en/index.html

[8] Webpack contributors, Concepts, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://webpack.js.org/concepts

[9] Webpack contributors, Getting Started, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://webpack.js.org/guides/getting-started/

[10] Swagger.io, What is OpenAPI?, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://swagger.io/docs/specification/about/

[11] Auth0, Introduction to JSON Web Tokens, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://jwt.io/introduction

[12] OpenJS Foundation, About Node.js, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://nodejs.org/en/about/

[13] W3Schools, Node.js Introduction, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://www.w3schools.com/nodejs/nodejs_intro.asp

[14] Wikipedia, Node.js, 2022, Pristupljeno 5. Rujna 2022. [Online], Dostupno na: https://en.wikipedia.org/wiki/Node.js

[15] OpenJS Foundation, Express, 2021, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://expressjs.com

[16] Wikipedia, Express.js, 2022, Pristupljeno: 5. Rujna 2022. [Online], Dostupno na: https://en.wikipedia.org/wiki/Express.js

[17] Google, Cloud Firestore, 2022, Pristupljeno: 6. Rujna 2022. [Online], Dostupno na: https://firebase.google.com/docs/firestore

[18] Google, Cloud Storage for Firebase, 2022, Pristupljeno: 6. Rujna 2022. [Online], Dostupno na: https://firebase.google.com/docs/storage

[19] Google, Firebase Authentication, 2022, Pristupljeno: 6. Rujna 2022. [Online], Dostupno na: https://firebase.google.com/docs/auth

[20] Google, Add the Firebase Admin SDK to your server, 2022, Pristupljeno: 6. Rujna 2022. [Online], Dostupno na: https://firebase.google.com/docs/admin/setup

[21] Robert C. Martin (Uncle Bob), The Clean Architecture, 2012, Pristupljeno: 6. Rujna 2022. [Online], Dostupno na: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Popis slika

Popis programskih kodova

Prilog 1: Primjer subjekta

```
1.  import isOibValid from '../../utils/oibValidator';
2.  import { checkIfValidPhoneNumber } from '../../utils/validationExpressions';
3.
4.  export default function buildMakeEstablishment({
5.    makeImages,
6.    makeWorkingHours,
7.    makeLocation,
8.    makeTables,
9.  }) {
10.   return function makeEstablishment(
11.     {
12.       location,
13.       name,
14.       numberOfReservations = 0,
15.       oib,
16.       images,
17.       phoneNumber,
18.       workingHours,
19.       tables,
20.       owner,
21.     },
22.     usersReservations,
23.     action
24.   ) {
25.     let numberOfReservationsInStatusPending = 0;
26.     usersReservations.forEach((uR) => {
27.       if (uR.status === 0) {
28.         numberOfReservationsInStatusPending += 1;
29.       }
30.     });
31.     const validatedImages = action === 'post' ? makeImages(images) : null;
32.     const validatedWorkingHours =
33.       action === 'post' ? makeWorkingHours(workingHours) : null;
34.     tables = makeTables(action, tables, usersReservations);
35.     location = makeLocation(location);
36.     if (!owner || owner.trim().length < 1) {
37.       throw new Error('Vlasnik objekta mora biti poslan.');
38.     }
39.     if (!name || name.trim().length < 1) {
40.       throw new Error('Ime objekta mora biti poslano.');
41.     }
42.     if (!oib || !isOibValid(oib)) {
43.       throw new Error('OIB objekta mora biti pravilan.');
44.     }
45.     if (!phoneNumber || !checkIfValidPhoneNumber(phoneNumber)) {
46.       throw new Error('Broj objekta mora biti poslan i pravilan.');
47.     }
48.     return Object.freeze({
49.       getName: () => name,
50.       // eslint-disable-next-line arrow-body-style
51.       getImages: () => {
52.         return action === 'post' ? validatedImages.getImages() : images;
53.       },
54.       getOwner: () => owner,
55.       getImagesForUpload: () => validatedImages.getImagesForUpload(),
56.       getOIB: () => oib,
57.       getPhoneNumber: () => phoneNumber,
58.       getLocation: () => ({
59.         address: location.getAddress(),
60.         city: location.getCity(),
```

```
61.         country: location.getCountry(),
62.         geoCords: location.getGeoCords(),
63.       }),
64.       // eslint-disable-next-line arrow-body-style
65.       getWorkingHours: () => {
66.         return action === 'post'
67.           ? validatedWorkingHours.getWorkingHours()
68.           : workingHours.sort((a, b) => (a.index > b.index ? 1 : -1));
69.       },
70.       getTables: () => tables.getTables(),
71.       getNumberOfReservations: () => Number(numberOfReservations),
72.       getNumberOfReservationsInStatusPending: () =>
73.         numberOfReservationsInStatusPending,
74.     });
75.   };
76. }
77.
```

Prilog 2: Primjer slučaja korištenja

```
1.  import { makeUser } from '../../entities';
2.
3.  export default function makeCreateUser({
4.    usersCollection,
5.    CRUDDb,
6.    authenticationActions,
7.  }) {
8.    return async function createUser({ userInfo }) {
9.      if (
10.       !userInfo ||
11.       typeof userInfo !== 'object' ||
12.       Object.keys(userInfo).length === 0
13.     ) {
14.       throw new Error('Podatci o novom korisniku moraju biti poslani.');
15.     }
16.     if (
17.       await CRUDDb.checkIfDocWithIdExistsInCollection({
18.         collection: usersCollection,
19.         id: userInfo.UID,
20.       })
21.     ) {
22.       throw Error('Korisnik sa tim identifikacijskim brojem vec postoji.');
23.     }
24.     const email = await authenticationActions.getUsersEmail({
25.       UID: userInfo.UID,
26.     });
27.     userInfo.email = email;
28.     const user = await makeUser('post', userInfo);
29.     const data = {
30.       fullName: user.getFullName(),
31.       lastname: user.getLastname(),
32.       name: user.getName(),
33.       accountType: user.getAccountType(),
34.       UID: user.getUID(),
35.       email: user.getEmail(),
36.       photoURL: user.getPhotoURL(),
37.       isNewUser: user.getIsNewUser(),
38.       authType: user.getAuthType(),
39.     };
40.     const insertedUser = await CRUDDb.insertIntoCollectionById({
41.       collection: usersCollection,
42.       data,
43.       id: user.getUID(),
44.     });
45.     return insertedUser;
46.   };
47. }
48.
```

Prilog 3: Primjer kontrolera

```
1.  import { userUseCases } from '../../use-cases';
2.  import makeExpressCallback from '../../adapters/expressCallback';
3.  import makeGetUser from './getUser';
4.  import makePostUser from './postUser';
5.  import makePatchUserDeviceToken from './patchUserDeviceToken';
6.  import makePatchUserInfo from './patchUserInfo';
7.
8.  export default function makeUserController() {
9.    function Get() {
10.     const getUser = makeGetUser({ fetchUserInfo: userUseCases.fetchUserInfo });
11.     return makeExpressCallback(getUser);
12.   }
13.   function Post() {
14.     const postUser = makePostUser({ createUser: userUseCases.createUser });
15.     return makeExpressCallback(postUser);
16.   }
17.   function PatchUserDeviceToken() {
18.     const patchUserDeviceToken = makePatchUserDeviceToken({
19.       saveUserDeviceToken: userUseCases.saveUserDeviceToken,
20.     });
21.     return makeExpressCallback(patchUserDeviceToken);
22.   }
23.   function PatchUserInfo() {
24.     const patchUserInfo = makePatchUserInfo({
25.       updateUserInfo: userUseCases.updateUserInfo,
26.     });
27.     return makeExpressCallback(patchUserInfo);
28.   }
29.   return Object.freeze({
30.     Get,
31.     Post,
32.     PatchUserDeviceToken,
33.     PatchUserInfo,
34.   });
35. }
36.
```

Prilog 4: Primjer akcije

```
1.  import logger from '../../logger';
2.
3.  export default function makeGetUser({ fetchUserInfo }) {
4.    return async function getUser(httpRequest) {
5.      const headers = {
6.        'Content-Type': 'application/json',
7.      };
8.      try {
9.        const { UID } = httpRequest.query;
10.       const user = await fetchUserInfo({ UID });
11.       return {
12.         headers,
13.         statusCode: 201,
14.         body: { user },
15.       };
16.     } catch (e) {
17.       logger({ errorMessage: e.message, fullErrorTrace: e.stack });
18.       return {
19.         headers,
20.         statusCode: 400,
21.         body: {
22.           errorMessage: e.message,
23.           fullErrorTrace: e.stack,
24.         },
25.       };
26.     }
27.   };
28. }
29.
```

Prilog 5: Adaptera za express callback

```
1.  import logger from '../../logger';
2.
3.  export default function makeExpressCallback(controller) {
4.    return (req, res) => {
5.      const httpRequest = {
6.        body: req.body,
7.        query: req.query,
8.        params: req.params,
9.        ip: req.ip,
10.       method: req.method,
11.       path: req.path,
12.       headers: {
13.         'Content-Type': req.get('Content-Type'),
14.         Referer: req.get('referer'),
15.         'User-Agent': req.get('User-Agent'),
16.       },
17.     };
18.     controller(httpRequest)
19.       .then((httpResponse) => {
20.         if (httpResponse.headers) {
21.           res.set(httpResponse.headers);
22.         }
23.         res.type('json');
24.         res.status(httpResponse.statusCode).send(httpResponse.body);
25.       })
26.       .catch((e) => {
27.         logger({ errorMessage: e.message, fullErrorTrace: e.stack });
28.         res
29.           .status(500)
30.           .send({ errorMessage: e.message, fullErrorTrace: e.stack });
31.       });
32.   };
33. }
34.
```

Prilog 6: Middleware koji autorizira svaki poziv prema API-u

```
1.  import dotenv from 'dotenv';
2.  import { getAuthentication } from '../../data-access/database';
3.
4.  dotenv.config();
5.
6.  const auth = getAuthentication();
7.
8.  const authorizeRequest = (req, res, next) => {
9.    const token =
10.     req.headers.authorization && req.headers.authorization.split(' ')[1];
11.   if (!token) {
12.     res
13.       .status(401)
14.       .send({ errorMessage: 'Autorizacijski header nije postavljen' });
15.     return;
16.   }
17.   if (req.originalUrl.includes('admin')) {
18.     if (process.env.ADMIN_TOKEN === token) next();
19.     else {
20.       res.status(403).send({
21.         errorMessage: 'Pristup administracijskom dijelu api-a nije dozvoljen',
22.       });
23.     }
24.   } else {
25.     auth
26.       .verifyIdToken(token)
27.       .then(() => next())
28.       .catch(() => {
29.           res.status(401).send({
30.             errorMessage: 'Autorizacijski token je neispravan',
31.           });
32.       });
33.   }
34. };
35.
36. export default authorizeRequest;
37.
```

Prilog 7: Algoritam za dodavanje nove rezervacije

```
1.  // If establishment has no empty tables notify user.
2.      if (tablesAvailableForReservation.length <= 0) {
3.        throw Error('Odabrani objekt nema slobodnih stolova.');
4.      }
5.      // If establishment has one table check its capacity and if it does not have enough
    places
6.      // notify user.
7.      if (
8.        tablesAvailableForReservation.length === 1 &&
9.        tablesAvailableForReservation[0].capacity < reservation.getPlaces()
10.     ) {
11.       throw Error('Odabrani objekt nema stol sa dovoljno mjesta.');
12.     }
13.     // Check if table has enough places and if it fulfils the owners restriction on empty
    seats
14.     if (
15.       tablesAvailableForReservation.length === 1 &&
16.       Math.abs(
17.         tablesAvailableForReservation[0].capacity - reservation.getPlaces()
18.       ) > allowedEmptySeats
19.     ) {
20.       throw Error(
21.         `Odabrani objekt ima jedan preostali stol sa dovoljno mjesta ali je vlasnik objekta
    specificirao da ne želi omogućiti rezervaciju stola ako više od ${allowedEmptySeats}. mjesta
    bude prazno.`
22.       );
23.     }
24.     // If establishment has one table and it passed all of above checks reserve it.
25.     if (tablesAvailableForReservation.length === 1) {
26.       reservationToInsert = await getReservationForInsert(reservationInfo, [
27.         tablesAvailableForReservation[0].id,
28.       ]);
29.     }
30.     // Find tables that match the asked number of places on the reservation.
31.     const tablesThatMatchRequiredPlaces = tablesAvailableForReservation.filter(
32.       (t) => t.capacity === reservation.getPlaces()
33.     );
34.     // If exists reserve table that has as many places as the user asked for
35.     // if a table was not reserved before.
36.     if (
37.       tablesThatMatchRequiredPlaces.length >= 1 &&
38.       reservationToInsert === null
39.     ) {
40.       reservationToInsert = await getReservationForInsert(reservationInfo, [
41.         tablesThatMatchRequiredPlaces[0].id,
42.       ]);
43.     }
44.     // Find tables that have enough places as specified on the reservation and
45.     // that fullfil the owners request for empty seats.
46.     const tableThatMatchEmptySeatsAndHaveEnoughPlaces =
47.       tablesAvailableForReservation.filter(
48.         (t) =>
49.           t.capacity > reservation.getPlaces() &&
50.           Math.abs(t.capacity - reservation.getPlaces()) <= allowedEmptySeats
51.       );
52.     // If exists reserve table that has enough places as specified on the reservation and
53.     // that fulfils the owners request for empty seats.
54.     if (
55.       tableThatMatchEmptySeatsAndHaveEnoughPlaces.length >= 1 &&
56.       reservationToInsert === null
```

```
57.     ) {
58.       reservationToInsert = await getReservationForInsert(reservationInfo, [
59.         tableThatMatchEmptySeatsAndHaveEnoughPlaces[0].id,
60.       ]);
61.     }
62.     if (reservationToInsert === null) {
63.       const tables = [];
64.       // Find table that has the closest available places as required by the reservation
65.       let table = findTableThatHasTheClosestNumberOfPlacesAskedForInReservation(
66.         tablesAvailableForReservation,
67.         reservation.getPlaces()
68.       );
69.       tables.push(table);
70.       tablesAvailableForReservation = tablesAvailableForReservation.filter(
71.         (t) => t.id !== table.id
72.       );
73.       // Init initial number of available places found
74.       let availablePlaces = tables.reduce(
75.         (accumulator, t) => accumulator + t.capacity,
76.         0
77.       );
78.       // Init initial number of places left to find
79.       let placesLeftToFind = Math.abs(
80.         availablePlaces - reservation.getPlaces()
81.       );
82.       while (availablePlaces < reservation.getPlaces()) {
83.         if (placesLeftToFind >= 1) {
84.           table = findTableThatHasTheClosestNumberOfPlacesAskedForInReservation(
85.             tablesAvailableForReservation,
86.             placesLeftToFind
87.           );
88.           tables.push(table);
89.           tablesAvailableForReservation = tablesAvailableForReservation.filter(
90.             // eslint-disable-next-line no-loop-func
91.             (t) => t.id !== table.id
92.           );
93.         }
94.         // Update the current number of available places in found tables
95.         availablePlaces = tables.reduce(
96.           (accumulator, t) => accumulator + t.capacity,
97.           0
98.         );
99.         // Update places left to find
100.          placesLeftToFind = Math.abs(availablePlaces - reservation.getPlaces());
101.        }
102.        if (
103.          Math.abs(availablePlaces - reservation.getPlaces()) <= allowedEmptySeats
104.        ) {
105.          reservationToInsert = await getReservationForInsert(
106.            reservationInfo,
107.            tables
108.            // tables.map((t) => t.id)
109.          );
110.        } else {
111.          throw Error(
112.            `Odabrani objekt ima dovoljno slobodnih stolova ali je vlasnik objekta
    specificirao da ne želi omogućiti rezervaciju stolova ako više od ${allowedEmptySeats}.
    mjesta bude prazno.`
113.          );
114.        }
115.      }
116.
117.      if (reservationToInsert !== null) {
118.        await CRUDDb.insertIntoCollectionById({
119.          collection: usersReservationsCollection,
```

```
120.           data: reservationToInsert,
121.           id: reservationToInsert.id,
122.         });
123.         await cloudMessagingActions.sendNotificationToSentUsers({
124.           UIDs: [establishment.owner],
125.           messageTitle: establishment.name,
126.           messageBody: 'Imate novu rezervaciju!',
127.           messageSubtitle: 'Rezervacije',
128.           largeTextToShow: `Korisnik ${reservationToInsert.nameOnReservation} je zatražio
    rezervaciju ${reservationToInsert.tablesReserved.length}. stola!`,
129.         });
130.       } else {
131.         throw Error(
132.           'Odabrani objekt nema dovoljno slobodnih stolova koji ispunjavaju vaše kriterije.'
133.         );
134.       }
135.
```

Veleučilište u Virovitici

OBRAZAC 5

IZJAVA O AUTORSTVU

Ja, Mario Šomođi

izjavljujem da sam autor/ica završnog/diplomskog rada pod nazivom

Izrada poslužiteljskog dijela aplikacije koristeći Node ekosustav

Svojim vlastoručnim potpisom jamčim sljedeće:

- da je predani završni/diplomski rad isključivo rezultat mog vlastitog rada koji se temelji na mojim istraživanjima i oslanja se na objavljenu literaturu, a što pokazuju korištene bilješke i bibliografija,
- da su radovi i mišljenja drugih autora/ica, koje sam u svom radu koristio/la, jasno navedeni i označeni u tekstu te u popisu literature,
- da sam u radu poštivao/la pravila znanstvenog i akademskog rada.

Potpis studenta/ice

Mario Šomođi

Veleučilište u Virovitici

OBRAZAC 6

ODOBRENJE ZA POHRANU I OBJAVU ZAVRŠNOG/DIPLOMSKOG RADA

Ja Mario Šomodi

dajem odobrenje za objavljivanje mog autorskog završnog/diplomskog rada u javno dostupnom digitalnom repozitoriju Veleučilišta u Virovitici te u javnoj internetskoj bazi završnih radova Nacionalne i sveučilišne knjižnice bez vremenskog ograničenja i novčane nadoknade, a u skladu s odredbama članka 83. stavka 11. Zakona o znanstvenoj djelatnosti i visokom obrazovanju (NN 123/03, 198/03, 105/04, 174/04, 02/07, 46/07, 45/09, 63/11, 94/13, 139/13, 101/14, 60/15, 131/17).

Potvrđujem da je za pohranu dostavljena završna verzija obranjenog i dovršenog završnog/diplomskog rada. Ovom izjavom, kao autor navedenog rada dajem odobrenje i da se moj rad, bez naknade, trajno javno objavi i besplatno učini dostupnim:

a) široj javnosti
b) studentima i djelatnicima ustanove
c) široj javnosti, ali nakon proteka 6 / 12 / 24 mjeseci (zaokružite odgovarajući broj mjeseci).

Potpis studenta/ice

Mario Šomodi

U Virovitici, 5. 9. 2022

**U slučaju potrebe dodatnog ograničavanja pristupa Vašem završnom/diplomskom radu, podnosi se pisani obrazloženi zahtjev.*