Izrada progresivnih web aplikacija u Angularu

Piskač, Maja

Master's thesis / Diplomski rad

2020

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Science / Sveučilište u Zagrebu, Prirodoslovno-matematički fakultet**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:217:084858

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-03-28**



Repository / Repozitorij:

Repository of the Faculty of Science - University of Zagreb







SVEUČILIŠTE U ZAGREBU

PRIRODOSLOVNO–MATEMATIČKI FAKULTET

MATEMATIČKI ODSJEK

Maja Piskač

IZRADA PROGRESIVNIH WEB APLIKACIJA U ANGULARU

Diplomski rad

Voditelj rada:
izv. prof. dr. sc. Ivica Nakić

Zagreb, rujan 2020.

Ovaj diplomski rad obranjen je dana _____ pred ispitnim povjerenstvom u sastavu:

1. _____, predsjednik
2. _____, član
3. _____, član

Povjerenstvo je rad ocijenilo ocjenom _____.

Potpisi članova povjerenstva:

1. _____
2. _____
3. _____

Hvala mojem dečku Joopu na neizmjernoj ljubavi i razumijevanju, roditeljima Mariji i Antunu na podršci tijekom cijelog školovanja, te prijateljicama i kolegicama Luciji, Emiliji i Ivi koje su bile uz mene uz najljepše i najteže trenutke studiranja.

Sadržaj

Sadržaj **iv**

Uvod **1**

1 Angular **3**
1.1 Uvod . . . 3
1.2 Priprema okuženja . . . 3
1.3 Angular Material . . . 5
1.4 Koncepti u Angularu . . . 7
1.5 Progresivna web aplikacija . . . 12

2 Parse Server **17**
2.1 Uvod . . . 17
2.2 Pokretanje . . . 17
2.3 Povezivanje s Angular aplikacijom . . . 20
2.4 MongoDB . . . 20
2.5 Parse objekti . . . 22
2.6 Parse upiti . . . 25
2.7 Uvođenje . . . 28
2.8 Parse Dashboard . . . 30

3 Aplikacija za rezervaciju predavaonica **33**
3.1 Opis aplikacije . . . 33
3.2 Full Calendar . . . 40
3.3 Struktura aplikacije . . . 41
3.4 Progresivna web aplikacija . . . 43

Bibliografija **47**

Uvod

Progresivna web aplikacija, skraćeno PWA, pojam je koji se koristi za aplikacije koje su brze, efikasne, pouzdane, ažurne, uvijek dostupne, te koje progresivno pokušavaju poboljšati iskustvo korisnika neovisno o pregledniku, platformi ili uređaju koji korisnik koristi. To su web aplikacije koje korisniku pružaju osjećaj korištenja nativne aplikacije, stoga je jedna od karakteristika, po kojoj se PWA može prepoznati, mogućnost dodavanja aplikacijske ikone na početni zaslon. Nastale su zbog sve veće upotrebe mobilnih uređaja, odnosno zbog potrebe korisnika da putem mobilnog uređaja, ali i svih računala, brzo i jednostavno, koristeći prilagođeno korisničko sučelje, obavi svakodnevne poslove, kao što je primjerice online kupnja.

Cilj ovog diplomskog rada je izraditi PWA za rezervaciju predavaonica na PMF - Matematičkom odsjeku koristeći Angular za izradu frontend dijela, te Parse Server za izradu backend dijela aplikacije. Mogućnosti aplikacije su odabir željene predavaonice te zatim rezervacija predavaonice na željeni datum i vrijeme, ili pretraga slobodne predavaonice u ovisnosti o odabranom datumu i terminu. U aplikaciji je prikazan interaktivni kalendar pomoću kojeg se može odabrati termin rezervacije, a koji je implementiran koristeći JavaScript biblioteku FullCalendar. Omogućen je i pristup administratoru koji može mijenjati razdoblje trajanja semestra te automatski učitati rezervacije predavaonica za predavanja i vježbe koje traju kroz cijeli semestar.

U radu će se dati pregled kroz sve korištene tehnologije, način njihova međusobnog povezivanja te će biti prikazana izrade same aplikacije.

Poglavlje 1

Angular

1.1 Uvod

Angular je okvir (engl. *framework*) za razvoj mobilnih i desktop web aplikacija kao i platforma za izgradnju efikasnih jednostraničnih (engl. *single-page*) aplikacija. Otvorenog je koda (engl. *open source*) i napisan u TypeScript-u. Razvijen je od strane Google-a 2010. godine kao AngularJS. Kao što i ime sugerira, prva verzija je bila razvijena u JavaScript-u, no već u sljedećoj verziji maknut je sufiks JS te se od tada Angular razvija u TypeScript-u, jeziku koji je razvijen i prevodi se u JavaScript te je njegov strogi sintaktički nadskup. Zadnja trenutno izdana verzija je Angular 10, izdana 24. lipnja 2020. Uz Angular okvir dolazi komandni alat Angular CLI koji vodi računa o stvaranju kostura same aplikacije, te o stvaranju osnovnih dijelova koji se nalaze unutar kostura, što ubrzava razvojni postupak. Angular 5, izdana 2017. godine, je bila prva verzija Angular-a koja je podržavala izradu PWA, što je omogućeno tako da je dodana Angular Service Worker skripta kao dio `@angular/service-worker` modula. Angular Service Worker zajedno s Angular CLI omogućuje kreiranje PWA aplikacije, ali i jednostavno unaprjeđenje Angular aplikacije u PWA aplikaciju.

1.2 Priprema okuženja

Da bi se mogao instalirati Angular, potrebno je imati instalirano sljedeće:

- *Node.js* - za pokretanje same Angular aplikacije.
  Da bi se instalirao Node.js, potrebno je prvo provjeriti da li već postoji instalirana neka verzija Node.js-a. To se može učiniti pokretanjem sljedeće naredbe u komandnoj liniji (engl. *command prompt*).

```
$ node -v
```

Ako se dobije ispis neke verzije, tada je već ranije instaliran Node.js. Inače, se Node.js može instalirati koristeći službenu stranicu čiji je link [1] dan u literaturi. Preporučno je instalirati posljednju LTS verziju, a potrebna je barem verzija 8.

- *npm package manager* - za instalaciju dodatnih paketa i funkcija.
  Također, prvo treba provjeriti da li već postoji instalirana verzija pokretanjem naredbe slične prethodnoj.

```
$ npm -v
```

  Kad se instalira Node.js, automatski bi se trebao instalirati i npm jer je npm distribuiran zajedno s Node.js. No, npm je zaseban projekt od Node.js, pa stoga ako je Node.js i tek sada instaliran, potrebno je ažurirati npm. To se može postići sljedećom naredbom.

```
$ npm install npm@latest -g
```

  Zastavica (engl. *flag*) `-g`, odnosno `--global` označava da će paket instalirati globalno, odnosno moći se koristiti u svim projektima, neovisno o trenutnom radnom direktoriju. Da bi se Angular mogao instalirati, potrebna je barem verzija 5.

Sada je moguće globalno instalirati Angular CLI koji se koristi za izradu Angular aplikacije sljedećom naredbom.

```
$ npm install @angular/cli -g
```

Nakon što je Angular CLI instaliran, dostupna je naredba ***ng*** pomoću koje se može kreirati, razvijati i graditi Angular aplikacija. Angular aplikacija kreira se koristeći sljedeću naredbu.

```
$ ng new ime-aplikacije
```

Tada Angular CLI pita da li se želi dodati *routing*, o kojem će biti više riječi u sekciji 1.4, te koji se format želi koristiti za stil aplikacije. Te postavke mogu se dati i odmah kod pokretanja naredbe za kreiranje. Primjerice, ovako je kreirana aplikacija u poglavlju 3.

```
$ ng new rezervacija-predavaonica --routing --style=scss
```

Angular CLI uključuje i server tako da se aplikacija može izgraditi (engl. *build*) i pokrenuti (engl. *serve*) lokalno. Za to je potrebno prvo postaviti se u radni direktorij projekta, te zatim pokrenuti sljedeću naredbu.

```
$ ng serve
```

Naredba `ng serve` pokreće server, prati datoteke unutar projekta te ponovno gradi aplikaciju ako uoči promjene u datotekama. Moguće je dodati zastavicu `-o`, odnosno `--open` za automatsko otvaranje preglednika na `http://localhost:4200` ili samostalno otvoriti preglednik i utipkati dani url. Ako su instalacija i izrada projekta bile uspješne, pojavit će se stranica naslova *„Welcome"* u zaglavlju (engl. *header*), te s tekstom *„ime-aplikacije is running!"*, linkovima na Angular dokumentaciju i prijedlozima za idući korak razvoja.

Iako se PWA gradi tako da je neovisna o web pregledniku, Google Chrome sadrži koristan alat *Lighthouse* unutar Developer Tools-a, o kojem će biti riječi prilikom evaluacije aplikacije u sekciji 3.4, stoga je za potrebe ovog rada korišten taj web preglednik.

1.3 Angular Material

Angular Material je biblioteka komponenata korisničkog sučelja (engl. *user interface*, *UI*) razvijena od strane Angulara. Služi konstruiranju atraktivnih, konzistentnih i funkcionalnih web stranica i web aplikacija, pritom pridržavajući se principa modernog web dizajna kao što su prenosivnost među preglednicima i neovisnost o uređaju na kojem se prikazuje, i pridonoseći kreiranju brzih i responzivnih web aplikacija, što su upravo i zahtjevi PWA.

Instalacija

Angular Material može se instalirati koristeći sljedeću naredbu.

```
$ ng add @angular/material
```

Pokretanjem te naredbe instalirat će se Angular Material, ali i *Component Dev Kit (CDK)* i *Angular Animations* koje omogućavaju kreiranje UI komponenata i mijenjanje stila HTML elemenata. Također, pojavit će se i nekoliko pitanja vezanih uz:

- odabir integracije predefinirane teme dizajna ili izrada vlastite *„custom"* teme - može se odabrati između četiri predefinirane teme, u ovom radu je korištena *deeppurple-amber.css*. Tema se sastoji od konfiguracije boja i tipografije. Konfiguracija boja sastoji se od pet paleta, među kojima su primarna paleta i paleta za naglaske (engl. *accent palette*), u ovom radu deeppurple i amber. Teme se generiraju statički u vrijeme izgradnje (engl. *build-time*) da bi se izbjeglo trošenje vremena na generiranje teme na početku svakog pokretanja.

- postavljanje animacija preglednika (engl. *browser animations*) - importiranje modula *BrowserAnimationsModule* koji omogućava Angular-ov sustav animacija. Odbijanjem ove opcije onemogućit će se većina animacija Anular Material-a.

Dodatno, izvršit će se sljedeće:

- dodat će se ovisnosti projekta (engl. *project dependecies*) u **package.json** datoteku projekta - package.json sadrži set svih paketa koji su potrebni aplikaciji za rad
- *Roboto* font i *Material Design icon* font u **index.html**
- globalni CSS stilovi za micanje margina u tijelu (engl. *body*) aplikacije, postavljanje visine na 100% u html i body, te postavljanje Roboto-a kao zadani font aplikacije.

Komponente

Angular Material nudi mnoštvo komponenata, a sve se mogu pronaći na sljedećem linku [2] u literaturi. U sekciji 3.3 navedene se sve komponente koje su korištene prilikom izrade aplikacije.

Pokažimo sada kako se dodaje neka Angular Material komponenta u Angular aplikaciju. Neka to bude Angular Material *Button* komponenta. Prvo treba importirati *MatButtonModule*, koji se želi koristiti, u **app.module.ts** datoteku.

```
1 import {MatButtonModule} from '@angular/material/button';
2 ...
3 @NgModule ({....
4   imports: [...,
5   MatButtonModule,
6 ...]
7 })
```

Listing 1.1: Import *MatButtonModule* u **app.module.ts**

Zatim se dodaje oznaka (engl. *tag*) <mat-button> u **app.component.html**.

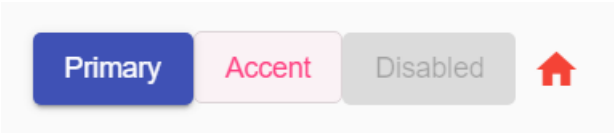
```
1 <button mat-button>Gumb</button>
```

Listing 1.2: Oznaka <mat-button> u **app.component.html**

Postoji više različitih vrsta dugmadi.

```
1 <button mat-raised-button color="primary">Primary</button>
2 <button mat-stroked-button color="accent">Accent</button>
3 <button mat-flat-button disabled>Disabled</button>
4 <button mat-icon-button color="warn">
5     <mat-icon>home</mat-icon>
6 </button>
```

Listing 1.3: Ostale oznake dugmadi u **app.component.html**

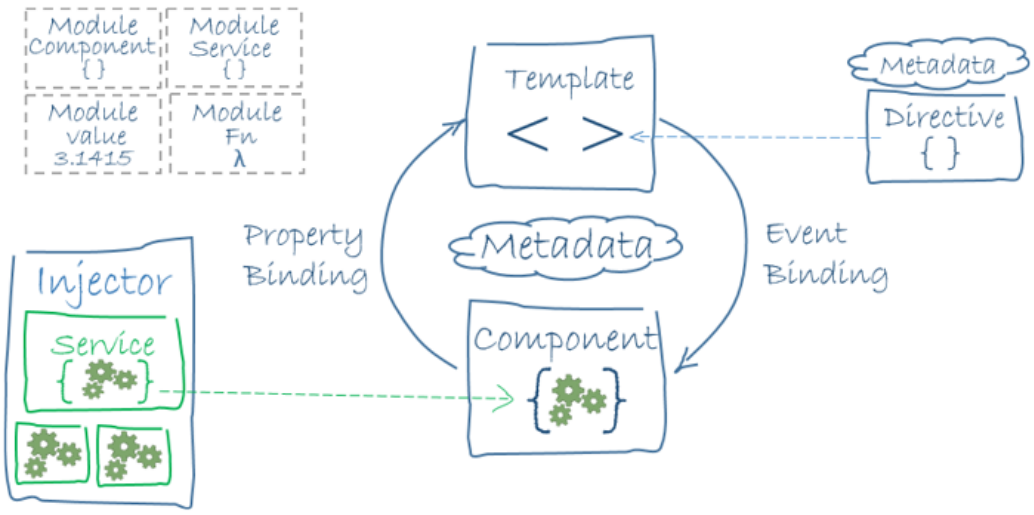


Slika 1.1: Angular Material, buttons

1.4 Koncepti u Angularu

Arhitektura Angular aplikacija zasniva se na određenim fundamentalnim konceptima. Temeljni blokovi koji grade aplikaciju su *NgModules*. Svaka aplikacija uvijek ima barem jedan korijenski modul (engl. *root module*) naziva *AppModule* koji omogućava *bootstrapping*, odnosno podizanje aplikacije, te obično sadrži mnoge druge module. Komponente definiraju pogled (engl. *view*), odnosno skup elemenata zaslona koje Angular mijenja u skladu s programskim kodom. Komponente koriste servise koji nude određene funkcionalnosti koje nisu direktno vezane uz pogled. Pružatelji servisa (engl. *service providers*) mogu biti ugrađeni (engl. *injected*) u komponentu kao ovisnosti (engl. *dependencies*), čime kod postaje modularan, ponovno upotrebljiv i efikasan.

Na sljedećem dijagramu, preuzetom sa službene Angular stranice [3], može se vidjeti veza između svih temeljnih koncepata.



Slika 1.2: Angular, dijagram koncepata

Moduli, komponente i servisi su klase koje koriste dekoratore koji označavaju nji-

hov tip i daju metapodatke koji govore Angular-u kako ih koristiti. Metapodaci povezuju klasu komponente s predloškom (engl. *template*) koji definira pogled. Predložak kombinira HTML s Angular direktivama i vezanjem oznaka, te time omogućava Angular-u modificiranje HTML-a prije njegovog prikazivanja.

Komponente aplikacije obično definiraju veći broj pogleda, stoga Angular pruža navigacijski servis (engl. *router service*) u kojem se definira navigacija puteva (engl. *paths*) između pogleda.

Moduli

NgModule je definiran klasom dekoriranom s `@NgModule`. Dekorator `@NgModule` je funkcija koja uzima pojedinačni objekt metapodatka čija svojstva opisuju modul. Najvažnija svojstva su:

- `declarations` - lista komponenata, direktiva i cijevi (engl. *pipes*) koje pripadaju ovom modulu
- `imports` - lista modula čije izvezene (engl. *exported*) klase su potrebne predlošcima komponenata u *ovom* modulu
- `providers` - lista usluga (engl. *services*) s kojima NgModule pridonosi globalnoj kolekciji usluga, odnosno koje su dostupne svim dijelovima aplikacije
- `exports` - podskup deklaracija koje će biti vidljive i moći se koristiti u predlošcima komponenata *drugih* modula
- `bootstrap` - glavni aplikacijski pogled, naziva se korijenska komponenta (engl. *root component*, koji sadrži sve ostale poglede aplikacije. Samo korijenski NgModule bi trebao postaviti to svojstvo.

Kod novostvorenog Angular projekta korijenski NgModule izgleda ovako.

```
1  import { BrowserModule } from '@angular/platform-browser';
2  import { NgModule } from '@angular/core';
3  import { AppRoutingModule } from './app-routing.module';
4  import { AppComponent } from './app.component';
5
6  @NgModule({
7    declarations: [AppComponent],
8    imports: [BrowserModule, AppRoutingModule],
9    providers: [],
10   bootstrap: [AppComponent]
11 })
```

```
12 export class AppModule { }
```

Listing 1.4: Početni **src/app/app.module.ts**

Komponente

Dekorator `@Component` identificira klasu kao komponentu i specificira njezine metapodatke. Metapodaci komponente govore Angular-u gdje se nalaze veći gradivni blokovi koji su potrebni za kreiranje i prikazivanje komponente i njezinog pogleda. Primjer osnovnih metapodataka za *AppComponent* prikazan je na sljedećem isječku koda.

```
1  import { Component } from '@angular/core';
2
3  @Component({
4    selector: 'app-root',
5    templateUrl: './app.component.html',
6    styleUrls: ['./app.component.scss']
7  })
8  export class AppComponent {
9    title = 'ime-aplikacije';
10 }
```

Listing 1.5: Početni **src/app/app.component.ts**

Ovaj primjer prikazuje neke od najvažnijih konfiguracijskih opcija komponente:

- `selector` - govori Angular-u da kreira i ubaci instancu komponente kad god naiđe na oznaku selektora u HTML predlošku. Na primjer, ako HTML sadrži oznake <app-root> i </app-root>, tada Angular ubacuje instancu AppComponent pogleda između tih oznaka.

- `templateUrl` - adresa HTML predloška komponente, može se i na to mjesto (engl. *inline*) upisati HTML predložak. Ovaj predložak definira pogled komponente.

- `styleUrls` - niz adresa stilova za HTML predložak.

Predlošci

Predložak izgleda kao uobičajeni HTML, uz dodatke koji omogućuju Angularu mijenjanje pojedinih dijelova ovisno o logici aplikacije. Neke od sintaksi za predložak mogu se vidjeti u tablici 1.1. Također, koriste se i ugrađene direktive. Neke od njih možemo vidjeti u tablici 1.2. Više sintaksa i direktiva može se naći u službenom podsjetniku na sljedećem linku [4].

Ime ove aplikacije je {{title}}.	Ako u komponenti imamo varijablu `title`, tada u predlošku možemo ispisati vrijednost te varijable.
<button (click) = ”onClick()” >	Ako u komponenti imamo funkciju `onClick()`, tada će se pritiskom na dani gumb pozvati ta funkcija.
<input [value] = ”title” >	Ako u komponenti imamo varijablu `title`, svojstvo *value* poprima vrijednost `title`-a.
<input [(value)] = ”title” >	Dvostruko vezivanje. Svaka izmjena varijable `title` mijenja svojstvo *value*, ali i svaka izmjena input-a mijenja vrijednost `title`-a.

Tablica 1.1: Sintakse za predložak

<div *ngIf = ”condition” >	Prikazuje se *div* element samo ako je vrijednost boolean-a `condition` *true*.
<li *ngFor = ”let item of list” >	Ispisuje se onoliko elemenata koliko ima stavaka (engl. *item*) u listi.

Tablica 1.2: Ugrađene direktive za predložak

Servisi

Komponenta delegira određene zadatke servisima, kao što su dohvat podataka sa servera i baze podataka, validacija korisnikovog unosa ili evidentiranje na konzolu (engl. *console log*). Da bi se klasa definirala kao servis u Angular-u, potrebno je dodati dekorator `@Injectable` kako bi se servis mogao ugraditi u komponentu kao ovisnost. Svakom servisu potrebno je registrirati barem jedan dobavljač usluge (engl. *provider*), a ako je on zadan kao *root*, tada Angular kreira jednu zajedničku instancu servisa i nju ugradi u svaku klasu koja zatraži taj servis. Servisi također mogu ovisiti o drugim servisima.

Dan je primjer servisa za dohvaćanje i ispisivanje podataka sa servera.

```
1  import { Injectable } from '@angular/core';
2  import { HttpClient } from '@angular/common/http';
3
4  @Injectable({
5    providedIn: 'root'
6  })
7  export class DataService {
8    private data;
9    constructor(private http: HttpClient) { }
10
```

```
11   logData(){
12     for(let object of data){
13         console.log(object);
14     }
15   }
16   getDataFromServer(){ ... }
17 }
```

Listing 1.6: **data.service.ts**

DataService servis zavisi o HttpClient servisu za dohvat podataka sa servera. U funkciji `getDataFromServer` se dohvaćaju podaci sa servera, o tome će biti više riječi u poglavlju 2, a u funkciji `logData` se ispisuju svi podaci na konzolu.

Navigacija unutar aplikacije

U jednostraničnoj aplikaciji ono što korisnik vidi mijenja se tako da se prikažu, odnosno prikriju, dijelovi prikaza koji pripadaju određenim komponentama. Kako korisnik prolazi kroz aplikaciju, tako je potrebno kretati se između različitih pogleda koji su definirani. Za to se koristi Angular *router*.

Na početku router treba izgledati kao kod ispod.

```
1  import { NgModule } from '@angular/core';
2  import { Routes, RouterModule } from '@angular/router';
3
4  const routes: Routes = [];
5
6  @NgModule({
7    imports: [RouterModule.forRoot(routes)],
8    exports: [RouterModule]
9  })
10 export class AppRoutingModule { }
```

Listing 1.7: Početni **src/app/app-routing.module.ts**

Ako se žele dodati putevi do komponenti FirstComponent i SecondComponent, to se treba dodati u *routes* niz.

```
1  const routes: Routes = [
2    { path: 'first-component', component: FirstComponent },
3    { path: 'second-component', component: SecondComponent },
4  ];
```

Listing 1.8: **routes**

Sada kada su putevi definirani, mogu se dodati u predložak aplikacije na sljedeći način.

```
1  <h1>Angular Router App</h1>
2  <nav>
```

```
3   <ul>
4     <li><a routerLink="/first-component" routerLinkActive="active">
          First Component</a></li>
5     <li><a routerLink="/second-component" routerLinkActive="active">
          Second Component</a></li>
6   </ul>
7 </nav>
8 <router-outlet></router-outlet>
```

Listing 1.9: Predložak koji koristi *router*

Element <nav> daje linkove na koje se može kliknuti, a svaki link govori router-u koji put koristiti. Element <router-outlet> govori Angular-u da treba ažurirati pogled aplikcije s pogledom odabranog puta, odnosno komponente.

Kreiranje pomoću Angular CLI

Koristeći Angular CLI, mogu se kreirati navedeni koncepti. Modul kreiramo sljedećom naredbom.

```
$ ng generate module ime-modula
```

Komponentu generiramo sličnom naredbom.

```
$ ng generate component ime-komponente
```

Analogno kreiramo i servis.

```
$ ng generate service ime-servisa
```

Koristeći ove naredbe, sve će biti kreirano u mapi `src/app`. Ako želimo drugačije, kod imena trebamo definirati mapu u koju želimo spremiti. Primjerice, ako module želimo spremiti u zasebnu mapu `modules`, onda treba utipkati `modules/ime-modula` umjesto samo `ime-modula`.

Početni AppRoutingModule može se postići tako da se projekt kreira sa zastavicom `--routing`, kako je objašnjeno u sekciji 1.2.

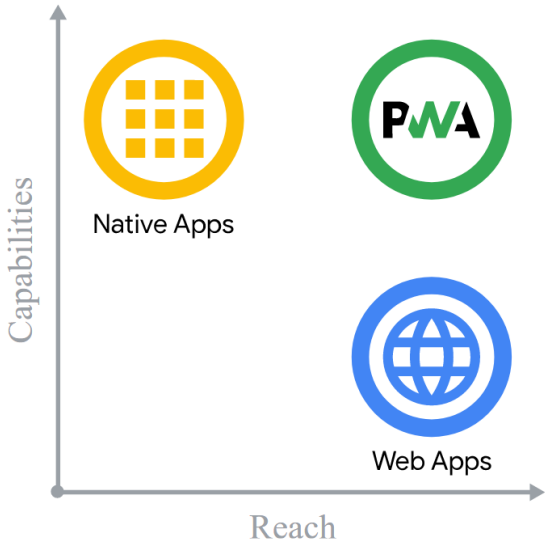
Na služenoj stranici Angular-a dostupan je tutorial koji pomaže u savladavanju temelja izrade Angular aplikacije, te je korišten tijekom izrade ovog diplomskog rada, a nalazi se na linku [5].

1.5 Progresivna web aplikacija

Kao što je rečeno u uvodu, progresivna web aplikacija, skraćeno PWA, hibrid je između nativne i web aplikacije. Sadrži prednost web aplikacije, a to je dostupnost svima, bilo gdje

i na bilo kojem uređaju jer je dijeljena putem web-a. S druge strane, nativne aplikacije su uvijek dostupne na računalu na kojem su instalirane, neovisno o dostupnoj ili nedostupnoj internetskoj vezi, mogu čitati i pisati u datoteke na lokalnom sistemu, pristupiti hardware-u povezanom preko USB-a ili bluetooth-a, odnosno one daju privid da su dio uređaja na kojem su instalirane.

Ako se razmišlja o nativnim i web aplikacijama u terminima mogućnosti i dosega, onda nativne aplikacije predstavljaju najbolje mogućnosti, dok web aplikacije predstavljaju najveći doseg. PWA su izgrađene i poboljšane modernim API-jima, odnosno aplikacijskim programskim sučeljima, kako bi imale mogućnosti, bile instalabilne i povjerljive kao nativne aplikacije, a pritom bile dostupne svima, bilo gdje i na bilo kojem uređaju. Grafički to možemo prikazati kao na sljedećem grafu preuzetom sa stranice [6].



Slika 1.3: Mogućnosti vs. doseg nativnih, web i progresivnih web aplikacija

Service worker

Service worker proširuje tradicionalni model razvoja web aplikacija i upotpunjuje aplikacije kako bi bile pouzdane i imale performanse kakve imaju nativne aplikacije. Dodavanje service worker-a u Angular aplikaciju je jedan od koraka prilikom transformiranja aplikacije u PWA. Service worker je skripta pokrenuta u web pregledniku koja upravlja međuspremanjem (engl. *caching*), odnosno privremenim spremanjem podataka kojima je nedavno pristupljeno u posebni memorijski podsistem kako bi bili brže dostupni.

Funkcionira kao mrežni posrednik (engl. *network proxy*). Presreće sve HTTP zahtjeve napravljene od strane aplikacije i može odlučiti kako će na njih reagirati. Primjerice, može pretražiti lokalni međuspremnik, odnosno cache, i proslijediti odgovor pronađen tamo. Nije ograničen na zahtjeve napravljene putem API-ja, uključuje i resurse navedene u HTML-u te inicijalni zahtjev prema `index.html`.

Sačuvan je i nakon što korisnik zatvori karticu (engl. *tab*). Sljedeći put kada preglednik učita aplikaciju, service worker će se prvi učitati i moći će presjeći svaki zahtjev prema resursima za učitavanje aplikacije. Ako je tako dizajniran, service worker može u potpunosti obaviti učitavanje aplikacije bez potrebe za internetskom vezom. Također, korištenje service worker-a ubrzava učitavanje i performanse aplikacije.

Angular-ov service worker ima sljedeća svojstva:

- aplikacija je međuspremljena kao cjelina te se sve datoteke ažuriraju zajedno
- kada korisnik osvježi aplikaciju, vidi zadnju potpuno međuspremljenu verziju
- ažuriranje se obavlja u pozadini, relativno brzo nakon što su promjene učinjene, a prethodna verzija se poslužuje do kad ažuriranje nije instalirano i gotovo
- service worker čuva propustnost (engl. *bandwidth*) kada je moguće, a resursi se ponovno spremaju samo ako su promjenjeni.

Da bi se omogućila ta svojstva, Angular service worker učitava takozvanu ***manifest*** datoteku. Ona opisuje resurse koje je potrebno međuspremiti i uključuje oznake (engl. *hashes*) svih datoteka. Kada je ažuriranje aplikacije završeno, sadržaj manifesta se mijenja, informirajući tako service worker da bi nova verzija aplikacije trebala biti preuzeta i međuspremljena. Manifest je generiran pomoću Angular CLI generirane datoteke naziva `ngsw-config.json`.

Kako bi se service worker mogao registrirati, potrebno je pristupiti aplikaciji preko HTTPS veze. Preglednici ignoriraju service worker koji je poslužen na nesigurnoj vezi. Nadalje, service worker zasad nije dostupan na svim web preglednicima. Zasad je dostupan je na posljednjim verzijama preglednika Chrome, Firefox, Edge, Safari, Opera, UC Browser (Android verzija) i Samsung Internet. Ako korisnik pristupa aplikaciji putem web preglednika koji ne pruža potporu za service worker, tada se service worker neće registrirati i radnje vezane uz njega se neće izvršiti, no aplikacija će raditi kao i ranije.

Svojstva PWA, uključujući i service worker, možemo dodati u postojeću aplikaciju koristeći sljedeću naredbu.

```
$ ng add @angular/pwa --project ime-aplikacije
```

Ta naredba će izvršiti sljedeće:

- dodati `@angular/service-worker` paket u projekt

- omogućiti podršku za service worker u Angular CLI
- unijeti i registrirati service worker u `app.module.ts`
- ažurirati `index.html` tako što će:
  - dodati link link za `manifest.webmanifest`
  - dodati meta tag `theme-color`
- instalirati ikone potrebne PWA
- kreirati service worker konfiguracijsku datoteku naziva `ngsw-config.json` koja specificira način međuspremanja i ostale postavke.

Kako bi se mogle vidjeti učinjene promjene, potrebno je izgraditi aplikaciju, odnosno pokrenuti sljedeću naredbu.

```
$ ng build --prod
```

Zastavica `--prod` govori da se aplikcija izgrađuje za produkciju. Kako `ng serve` ne radi sa service worker, potrebno je koristiti zaseban HTTP server kako bi se projekt mogao lokalno testirati. U ovom radu korišten je npm paket *http-server* čija je dokumentacija dostupna na linku [7]. Kako bi se aplikacija pokrenula na navedenom serveru, potrebno je pokrenuti sljedeću naredbu.

```
$ http-server -p 8080 -c-1 dist
```

Aplikaciji se tada može pristupiti na adresi `http://localhost:8080/`. Ako se ne koristi HTTPS veza, service worker će biti registriran samo ako se aplikaciji pristupa putem `localhost`. Više informacija može se pronaći na službenoj stranici navedenoj pod [8].

Poglavlje 2

Parse Server

2.1 Uvod

Parse Server je takozvani *Backend-as-a-Service*, skraćeno *BaaS*, okvir. Radi se o platformi koja automatizira backend stranu razvoja aplikacije i sadrži podršku za infrastrukturu u oblaku (engl. *cloud infrastructure*). Sadrži set alata koji pomažu upravljanju podacima te ubrzavaju razvoj backend dijela aplikacije. Inicijalno je razvijen kao Parse od strane kompanije Parse, Inc. u 2011. godini. Zatim 2013. godine dolazi pod vlasništvo Facebook-a, a u siječnju 2017. godine biva ugašen. U 2016. godini, kada je gašenje platforme najavljeno, nastaje Parse Server kao verzija Parse backend-a koja je otvorenog koda. Parse Server se može koristiti na svakoj infrastrukturi na kojoj se može pokrenuti Node.js. Svojstva Parse Server-a su sljedeća:

- Parse Server je neovisan o Parse-u
- Parse Server koristi MongoDB direktno, neovisan je o Parse bazi podataka
- postojeća aplikacija se može migrirati na vlastitu infrastrukturu
- aplikacija se može razvijati i testirati lokalno.

Službena dokumentacija može se pronaći na linku [9].

2.2 Pokretanje

Kako bi se Parse Server mogao pokrenuti, potrebno je imati instalirano sljedeće:

- *Node*, verzija 8 ili više

- *MongoDB*, verzija 3.6 ili više
- *Python*, verzija 2.x (za Windows korisnike potrebna je verzija 2.7.1).

Dodatno, u slučaju deployment-a aplikacije, potreban je pružatelj infrastrukture, primjerice *Heroku* ili *AWS*.

Prema uputama iz službene dokumentacije, dane su naredbe kojima se MongoDB i Parse Server mogu pokrenuti lokalno. Potrebno je pratiti sljedeće korake:

1. Prvo je potrebno napraviti instancu Parse Server-a.
   Na **Linux** operacijskom sustavu dovoljno je pokrenuti sljedeću naredbu u komandnoj liniji.

   ```
   $ sh <(curl -fsSL https://raw.githubusercontent.com/
     parse-community/parse-server/master/bootstrap.sh)
   ```

   Da bi se navedena naredba mogla izvšiti na **Windows** operacijskom sustavu, korišten je *Cygwin*, koji pruža funkcionalnosti slične Linux distribuciji na Windows-u, a dostupan je na linku [10]. Na **macOS** operacijskom sustavu potrebno je kopirati kod sa *https://raw.githubusercontent.com/parse-community/parse-server/master/bootstrap.sh* ili *https://github.com/parse-community/parse-server/blob/master/bootstrap.sh* stranice u lokalnu datoteku `bootstrap.sh`, te nakon toga pokrenuti sljedeću naredbu.

   ```
   $ sh bootstrap.sh
   ```

   Pokretanjem te naredbe pojavit će se pitanja vezana uz postavke Parse Server-a, a to su direktorij instalacije, ime aplikacije, id aplikacije, master key i mongodbURI. Njih se popunjava tako da se pogleda *parse-server-config.txt* datoteka priložena uz aplikaciju te popunjavaju podaci u skladu s tom datotekom. Posebno je važno da se odaberu isti id aplikacije i master key kao što je navedeno u toj datoteci, inače se Parse Server neće moći spojiti s Angular aplikacijom. MongodbURI koristi se za povezivanje Parse Server-a s MongoDB-om, odnosno za kreiranje baze koju će Parse Server koristiti te stoga nije moguće prenositi instancu Parse Server-a s jednog računala na drugo. Vrlo je bitno napomenti da ukoliko se stvara više instanci Parse Server-a na istom lokalnom računalu, svaka od njih mora imati drugačiji mongodbURI jer nije moguće spajati više Parse Server-a na istu bazu te ukoliko se to pokuša, doći će do pogreške kod autentifikacije te više ni stari ni novi Parse Server neće imati pristup bazi.

2. Zatim je potrebno pokrenuti MongoDB.
   Prvi način je koristeći `mongodb-runner`. On se instalira i pokrene koristeći sljedeće dvije naredbe.

```
$ npm install -g mongodb-runner
```

```
$ mongodb-runner start
```

Ukoliko dođe do problema kod pokretanja, MongoDB se može instalirati prema uputama sa službene stranice te zatim pokrenuti `mongod.exe` kako bi server počeo raditi. Ukoliko se ispiše *„Listening on 127.0.0.1"* i *„waiting for connections on port 27017"*, mongo je uspješno pokrenut. Također, može se pokrenuti `mongo.exe` kako bi se vidjeli podaci direktno u bazi. Koriste se sljedeće naredbe:

```
$ show dbs
```

za prikaz svih baza koje su spremljene na mongu. Za prikaz baze `parsebaza` koristimo sljedeću naredbu.

```
$ use parsebaza
```

Za prikaz kolekcija u toj bazi, koristimo

```
$ show collections
```

Tijekom korištenja aplikacije *Rezervacija predavaonica* stvorit će se kolekcije *Reservations*, *Rooms* i *SemesterDates* te kako punimo podatke kroz aplikaciju, možemo provjeriti da li su podaci kreirani u bazi. Primjerice, sve podatke u kolekciji *Rooms* možemo vidjeti pomoću sljedeće naredbe.

```
$ db.Rooms.find()
```

Bitno je da se MongoDB ostavi pokrenut tijekom rada Parse Server-a kako bi bio moguć pristup bazi podataka.

3. Potrebno je postaviti se u odabrani direktorij instalacije Parse Server-a te izvršiti naredbu za pokretanje.

```
$ npm start
```

Sada bi Parse Server trebao raditi, odnosno na komadnoj liniji bi se, između ostalog, trebalo ispisati *„parse-server running on http://localhost:1337/parse"*. Bitno je ostaviti Parse Server da radi u pozadini u otvorenoj komandnoj liniji, ako je instaliran samo lokalno, kako bi mu se moglo pristupiti u aplikaciji.

2.3 Povezivanje s Angular aplikacijom

Kako bismo povezali Parse Server s Angular aplikacijom potrebno je postaviti se u direktorij aplikacije i instalirati sljedeću biblioteku.

```
$ npm install --save @types/parse parse
```

Zatim, treba uključiti parse biblioteku i dodati sljedeće dvije linije koda u Angular aplikaciju.

```
1 import * as Parse from 'parse';
2
3 Parse.initialize(application_id, master_key);
4 (Parse as any).serverURL = 'http://localhost:1337/parse';
```

Listing 2.1: Povezivanje Parse Server-a s Angular aplikacijom

Umjesto `application_id` i `master_key` potrebno je staviti id aplikacije i master key koji su odabrani/generirani prilikom pokretanja Parse Server-a kako je objašnjeno na prethodnoj stranici. Kao serverURL potrebno je staviti onu adresu na kojoj je Parse Server pokrenut, ona se ispiše nakon pokretanja naredbe `npm start` i zadana je kao adresa u primjeru. Ove linije koda mogu se staviti u *data.service.ts* ili u *environment.ts* datoteku, ali tada treba uključiti *environment.ts* u *data.service.ts*.

2.4 MongoDB

Parse Server dopušta korištenje MongoDB ili Postgres baze podataka, ali preferira se MongoDB te je stoga ona u ovom radu korištena.

MongoDB je NoSQL dokumentska baza podataka opće namjene. NoSQL je generacija sustava za upravljanje bazama podataka koji uglavnom nisu relacijski, distribuirani su, otvorenog su koda i horizontalno su skalabilni. Originalno nastaju uglavnom zbog potreba modernih web aplikacija. Druge važne karakteristike koje mogu imati su: nemaju shemu podataka, jednostavno se repliciraju, jednostavno se koriste putem API-ja, nisu konzistntne, omogućuju rad s vrlo velikim količinama podataka.

Baza podataka je dokumentska ako se podaci pohranjuju u obliku kolekcija JSON dokumenata. Primjer JSON dokumenta možemo vidjeti ispod, a on je preuzet sa službene stranice MongoDB-a, čiji je link [11].

```
{
  "_id": "5cf0029caff5056591b0ce7d",
  "firstname": "Jane",
  "lastname": "Wu",
  "address": {
    "street": "1 Circle Rd",
```

```
    "city": "Los Angeles",
    "state": "CA",
    "zip": "90404"
  },
  "hobbies": ["surfing", "coding"]
}
```

Listing 2.2: Primjer JSON reprezentacije osobe

JSON dokument započinje s otvorenom vitičastom zagradom { i završava sa zatvorenom vitičastom zagradom }. S lijeve strane dvotočke nalazi se ključ. odnosno ono što bismo u relacijskoj bazi podataka nazvali atribut, a s desne strane nalazi se vrijednost koja može biti u obliku stringa, broja, JSON dokumenta, niza, boolean ili null. Upravo korištenje nizova i ugnježdenih JSON dokumenata kao vrijednosti omogućava fleksibilne i dinamične sheme u MongoDB.

Novi JSON objekt može se stvoriti jednostavno koristeći sljedeću liniju koda.

```
let _person = { "lastname": "Smith", "address": {"street": "Broadway
    72", "city": "New York"}, "hobbies": ["music", "running"]};
```

Listing 2.3: Novi objekt

Nije nužno da svi JSON dokumenti u kolekciji imaju iste ključeve, stoga su ovdje ispušteni neki ključevi radi jednostavnosti.

Ovime je dobiven Typescript objekt. On se pretvara u JSON dokument koristeći funkciju `JSON.stringify()`.

```
let person = JSON.stringify(_person);
```

Listing 2.4: Novi JSON objekt

Sada se može dohvatiti prezime osobe.

```
let prezime = person["lastname"];
```

Listing 2.5: Dohvat vrijednosti nekog ključa

Vrijednost nekog ključa može se modificirati na sljedeći način.

```
person["lastname"] = "Jones";
```

Listing 2.6: Modifikacija vrijednosti nekog ključa

Vrijednost unutar ugnježdenog JSON objekta može se dobiti kao u sljedećem primjeru.

```
let ulica = person.address["street"];
```

Listing 2.7: Dohvat vrijednosti unutar ugnježdenog JSON objekta

Prvom elementu u nizu unutar JSON dokumenta pristupa se na sljedeći način.

```
let hobi = person.hobbies[0];
```

Listing 2.8: Dohvat elementa u nizu unutar JSON dokumenta

2.5 Parse objekti

Spremanje podataka na Parse izgrađeno je oko `Parse.Object` koji je JSON oblika. Neka je definiran jednostavan `Parse.Object` u koji se sprema naziv igrača i postignut rezultat. Dan je jedan primjer takvog objekta, po uzoru na službenu dokumentaciju, dostupnu na linku [12].

```
{
  score: 1337,
  playerName: "Sean Plott"
}
```

Listing 2.9: Primjer Parse objeketa

Svaki `Parse.Object` je instanca specifične klase koja se može koristiti da se istaknu različiti tipovi objekata. Primjerice, navedeni objekt neka pripada klasi *GameScore*. Nova klasa se može kreirati koristeći `Parse.Object.extend` metodu.

```
1 // kreiranje nove klase Parse objekta
2 let GameScore = Parse.Object.extend("GameScore");
3
4 // kreiranje nove instance te klase
5 let gameScore = new GameScore();
```

Listing 2.10: Primjer kreiranja nove klase i njezine instance za Parse Server u Angular-u

Nakon što je pokrenut Parse Server na računalu i znamo kako izgledaju objekti koje spremamo u bazu, potrebno je vidjeti kako kreirati, dohvatiti, ažurirati i obrisati te objekte unutar Angular aplikacije.

Kreiranje objekata

Kreiranje objekta, kakav je definiran na vrhu stranice, u Angular aplikaciji za Parse Server, može se učiniti kao u sljedećem kodu. Za kreiranje objekata koristi se metoda `set`.

```
1  const GameScore = Parse.Object.extend("GameScore");
2  const gameScore = new GameScore();
3
4  gameScore.set("score", 1337);
5  gameScore.set("playerName", "Sean Plott");
6
7  gameScore.save()
8  .then((gameScore) => {
9    // spremanje uspjelo
10   console.log('New object created with objectId: ' + gameScore.id);
11 }, (error) => {
12   // spremanje nije uspjelo
```

```
13    console.log('Failed to create new object, with error code: ' +
          error.message);
14  });
```

Listing 2.11: Primjer kreiranja objekta za Parse Server u Angular-u

Važno je naglasiti da nije bilo potrebno kreirati klasu *GameScore* prije pokretanja ovog koda. Parse kreira tu klasu prvi put kada na nju naiđe. Također, neka polja popunjava Parse, stoga ona ne postoje na `Parse.Object` do kad operacija spremanja nije završena. To su:

- `objectId` - jednistveni identifikator za svaki spremljeni objekt
- `createdAt` - datum i vrijeme kreiranja objekta
- `updatedAt` - datum i vrijeme zadnje modifikacije objekta.

Ugnježđeni objekti, dakle objekti gdje je vrijednost nekog atributa neki drugi Parse objekt, spremaju se na sljedeći način.

```
1  let Child = Parse.Object.extend("Child");
2  let child = new Child();
3
4  let Parent = Parse.Object.extend("Parent");
5  let parent = new Parent();
6
7  parent.save({child: child});
```

Listing 2.12: Primjer kreiranja ugnježđenih objekata za Parse Server u Angular-u

Dohvat objekata

Nakon što je objekt spremljen na Parse Server, on se najjednostavnije može dohvatiti pomoću `objectId` koristeći `Parse.Query` metodu. Kako bi se dobile vrijednosti atributa objekata koristi se `get` metoda. Za polja `objectId`, `createdAt` i `updatedAt` vrijednosti su dane kao svojstva, one se ne mogu dohvatiti koristeći `get` metodu, ni modificirati koristeći `set` metodu.

```
1  let GameScore = Parse.Object.extend("GameScore");
2  let query = new Parse.Query(GameScore);
3  query.get("xWMyZ4YEGZ") // ovdje se upisuje objectId
4  .then((gameScore) => { // dohvat uspio
5    let score = gameScore.get("score");
6    let playerName = gameScore.get("playerName");
7    let objectId = gameScore.objectId;
8    let updatedAt = gameScore.updatedAt;
9    let createdAt = gameScore.createdAt;
```

```
10 }, (error) => { // dohvat nije uspio
11   console.log(error.message);
12 });
```

Listing 2.13: Primjer dohvaćanja objekta iz Parse Server-a u Angular-u koristeći objectId

Ažuriranje objekata

Ažuriranje postojećih objekata izvršava se koristeći `set` metodu. Prikazan je primjer ažuriranja rezultata objekta dohvaćenog koristeći njegov `objectId`.

```
1  let GameScore = Parse.Object.extend("GameScore");
2  let query = new Parse.Query(GameScore);
3  query.get("xWMyZ4YEGZ") // ovdje se upisuje objectId
4  .then((gameScore) => {
5    // dohvat uspio
6    gameScore.set("score", 1557);
7  }, (error) => {
8    // dohvat nije uspio
9    console.log(error.message);
10 });
```

Listing 2.14: Primjer mijenjanja objekta iz Parse Server-a u Angular-u koristeći objectId

Parse prepoznaje koja polja se mijenjaju, odnosno ponovno postavljaju, iako se koristi ista metoda za kreiranje i ažuriranje objekata, stoga nije potrebno upisivati vrijednosti svih atributa, nego je dovoljno upisati samo onaj koji se želi promijeniti.

Ukoliko postoje numeričke vrijednosti atributa objekata i one se žele povećati ili smanjiti za određen konstantan iznos, onda se može koristiti metoda `increment`.

```
1  let GameScore = Parse.Object.extend("GameScore");
2  let query = new Parse.Query(GameScore);
3  query.get("xWMyZ4YEGZ") // ovdje se upisuje objectId
4  .then((gameScore) => {
5    // dohvat uspio
6    gameScore.increment("score", 4);
7  }, (error) => {
8    // dohvat nije uspio
9    console.log(error.message);
10 });
```

Listing 2.15: Primjer korištenja inkrement metode za mijenjenje objekta iz Parse Server-a u Angular-u koristeći objectId

Kao drugi argument može se zadati proizvoljni broj, pozitivan u slučaju povećanja ili negativan u slučaju smanjenja. Ukoliko se ispusti drugi argument, on će biti zadan kao 1, odnosno izvršit će se povećanje rezultata za 1.

Za mijenjanje nizova, odnosno dodavanje i brisanje elemenata u nizu, koriste se tri sljedeće metode:

- `add` - dodavanje danog elementa na kraj niza
- `addUnique` - dodavanje danog elementa samo ako već nije u nizu
- `remove` - makni sve instance danog elementa iz niza.

Za primjer, neka objekt klase *GameScore* sadrži i niz `skills` u kojem se spremaju vještine igrača. Tada možemo upotrijebiti `addUnique` metodu na sljedeći način.

```
1 gameScore.addUnique("skills", "flying");
2 gameScore.addUnique("skills", "kungfu");
3 gameScore.save();
```

Listing 2.16: Primjer ažuriranja niza objekta iz Parse Server-a u Angular-u

Brisanje objekata

Kako bi se obrisao objekt, koristi se metoda `delete`. Primjer korištenja metode.

```
1 gameScore.destroy().then((gameScore) => {
2   // objekt je obrisan
3 }, (error) => {
4   // brisanje nije uspjelo
5 });
```

Listing 2.17: Primjer brisanja cijelog objekta iz Parse Server-a u Angular-u

Može se obrisati i samo jedno polje iz objekta koristeći `unset` metodu.

```
1 myObject.unset("playerName");
2 myObject.save();
```

Listing 2.18: Primjer brisanja jednog polja objekta iz Parse Server-a u Angular-u

Nakon toga polje koje sadrži vrijednost *„playerName”* će biti prazno.

2.6 Parse upiti

U prethodnoj sekciji vidjeli smo kako se objekt može dohvatiti koristeći `Parse.Query` i `objectId`, no uglavnom ne znamo id objekta te objekt želimo dohvatiti koristeći neka druga njegova svojstva, ili želimo dohvatiti niz objekata koji zadovoljavaju neki uvjet.

Osnovni upit

Generalni uzorak za osnovni upit je koristiti `Parse.Query`, postaviti uvjete koristeći neku od metoda navedenih u sljedećem odlomku, primjerice metodu `greaterThan`, i dohvatiti niz objekata koji zadovoljavaju uvjete koristeći metodu `find`.

Primjerice, želimo li dohvatiti imena svih igrača koji imaju rezultat veći od 1000 i ispisati ih na konzolu, koristit ćemo sljedeći upit.

```
1  const GameScore = Parse.Object.extend("GameScore");
2  const query = new Parse.Query(GameScore);
3  query.greaterThan("score", 1000);
4  const results = await query.find();
5  console.log("Postoji " + results.length + "takvih objekata");
6
7  // ispis na konzolu
8  for (let i = 0; i < results.length; i++) {
9    var object = results[i];
10   console.log(object.id + ' - ' + object.get('playerName'));
11 }
```

Listing 2.19: Primjer osnovnog upita za Parse Server-a u Angular-u

Uvjetne metode

Postoji više načina kako se mogu postaviti uvjeti na metodu `Parse.Query`, a to su:

- `equalTo` - koristimo kad želimo da vrijednost atributa bude jednaka nekoj određenoj vrijednosti. Primjerice, želimo samo objekt gdje je ime igrača „Sean Plott". Tada u upitu 2.19 trebamo promijeniti treći redak u sljedeće.

```
1      query.equalTo("playerName", "Sean Plott");
```

Listing 2.20: Primjer equalTo uvjeta za Parse Server u Angular-u

- `notEqualTo` - koristimo kada želimo da vrijednost atributa bude različita od neke određene vrijednosti. Primjerice, želimo objekte gdje je ime igrača različito od „John Smith". Tada u upitu 2.19 trebamo promijeniti treći redak u sljedeće.

```
1      query.notEqualTo("playerName", "John Smith");
```

Listing 2.21: Primjer notEqualTo uvjeta za Parse Server u Angular-u

- `greaterThan` - koristimo kada želimo da vrijednosti atributa bude strogo veća od neke određene vrijednosti. Primjer je dan u upitu 2.19. Analogno imamo i sljedeće tri metode.

- `greaterThanOrEqual` - koristimo kada želimo da vrijednosti atributa bude veća ili jednaka od određene vrijednosti.
- `lessThan` - koristimo kada želimo da vrijednost atributa bude strogo manja od određene vrijednosti.
- `lessThanOrEqual` - koristimo kada želimo da vrijednost atributa bude manja ili jednaka od određene vrijednosti.
- `limit` - koristimo kada želimo dohvatiti najviše određeni broj objekata. Na primjer, želimo dohvatiti najviše 3 objekta koji imaju vrijednost rezultata veću od 1000. Tada u upit 2.19 između trećeg i četvrtog retka trebamo ubaciti sljedeće.

```
1     query.limit(3);
```

Listing 2.22: Primjer limit uvjeta za Parse Server u Angular-u

Bitno je naglasiti da je zadano da limit bude 100, stoga, želimo li dohvatiti više objekata od 100, također trebamo koristiti limit i postaviti veću granicu.

- `first` - koristimo kada želimo dohvatiti samo jedan objekt koji zadovoljava zadane uvjete, navodi se kao alternativa korištenju `Parse.Query` i `limit`-a s granicom 1. Na primjer, želimo dohvatiti ime samo jednog igrača čiji je rezultat jednak 1000.

```
1     const GameScore = Parse.Object.extend("GameScore");
2     const query = new Parse.Query(GameScore);
3     query.equalTo("score", 1000);
4     const object = await query.first();
```

Listing 2.23: Primjer first upita za Parse Server u Angular-u

- `skip` - koristimo kada želimo preskočiti prvih nekoliko dohvaćenih objekata. Na primjer, želimo preskočiti prvih 10 objekata.

```
1     query.skip(10);
```

Listing 2.24: Primjer skip uvjeta za Parse Server u Angular-u

- `ascending` - koristimo kada želimo objekte sortirati uzlazno po određenom atributu. Primjerice, želimo sortirati uzlazno sve igrače po rezultatu.

```
1     query.ascending("score");
```

Listing 2.25: Primjer uzlaznog sortiranja za Parse Server u Angular-u

- `descending` - koristimo kada želimo objekte sortirati silazno po određenom atributu. Primjer je analogan prethodnom.

- `containedIn` - koristimo kada želimo dohvatiti objekte čija je vrijednost atributa jednaka nekoj od vrijednosti u danoj listi. Primjerice, želimo dohvatiti sve igrače čije je ime „Harry Walsh” ili „Emma Potter”.

```
1    query.containedIn("playerName",
2                  ["Harry Walsh", "Emma Potter"]);
```

Listing 2.26: Primjer containedIn uvjeta za Parse Server u Angular-u

- `notContainedIn` - koristimo kada želimo dohvatiti sve objekte čija je vrijednost atributa različita od svih vrijednosti danih u listi.
- `exists` - koristimo kada želimo dohvatiti sve objekte koji imaju postavljenu vrijednost određenog atributa.
- `doesNotExist` - koristimo kada želimo dohvatiti sve objekte koji nemaju postavljenu vrijednost određenog atributa.
- `select` - koristimo kada želimo dohvatiti samo vrijednosti određenih atributa objekata. Specijalni, odnosno ugrađeni atributi `objectId`, `createdAt`, `updatedAt` se uvijek dohvaćaju, neovisno o `select` metodi. Primjerice, želimo li dohvatiti samo atribute „score” i „playerName” (pretpostavimo da ima više atributa), onda koristimo sljedeći upit.

```
1    var GameScore = Parse.Object.extend("GameScore");
2    var query = new Parse.Query(GameScore);
3    query.select("score", "playerName");
4    query.find().then(function(results) {
5      // svaki od rezultata ima samo odabrana polja
6    });
```

Listing 2.27: Primjer select upita za Parse Server u Angular-u

- `exclude` - koristimo kada ne želimo dohvatiti vrijednosti određenih atributa, odnosno kada želimo isključiti neka polja.

Više informacija o Parse objektima, upitima i drugim dodatnim mogućnostima dostupno je na linku Parse dokumentacije [12], ovdje su opisane uglavnom samo mogućnosti koje su korištene tijekom izrade aplikacije.

2.7 Uvođenje

Parse Server moguće je razviti (engl. *deploy*). Neki od najvećih i najčešćih poslužitelja (engl. *provider*) na koje se razvija su Heroku i Back4App.

Heroku

Kako bi se Parse Server implementirao na Heroku, potrebno je pratiti sljedeće korake:

1. Posjetiti službenu stranicu [13] te stvoriti vlastiti račun.
2. Otvoriti *Heroku Dashboard* iz padajućeg izbornika u toolbar-u i odabrati *„Create new app"* iz padajućeg izbornika kako bi se kreirala nova aplikacija.
3. Kliknuti na *Resources* i instalirati *MongoLab* u *Add-ons*.
4. Kliknuti na *Deploy*, te zatim *GitHub* i integrirati vlastiti repozitorij Parse Server-a na GitHub-u s Heroku aplikacijom. Prije toga je potrebno stvoriti navedeni repozitorij na GitHub-u.
5. Kliknuti na *Deploy branch*. Time bi implementacija trebala završiti. Može se kliknuti na *View* za otvaranje aplikacije.

Back4App

Back4App je backend baziran na Parse Platformi koji je jednostavan za korištenje, fleksibilan i skalabilan. Vrlo je koristan kod upoznavanja s Parse Serverom jer ima integriran *Parse Dashoboard*, o kojem će biti više riječi u sljedećoj sekciji, a koji omogućuje vizualno predočenje podataka u bazi i cijelog Parse Server-a.

Za kreiranje backend-a na Back4App potrebno je pratiti sljedeće korake:

1. Kreirati besplatni korisnički račun na Back4App dostupan na službenoj stranici [14].
2. Nakon registracije, Back4App automatski generira *Sample Blog App* za upoznavanje s tehnologijom. Može joj se pristupiti ako se na toolbar-u klikne na *My Apps* ili *Dashboard* te odabere aplikacija. Njezin Dashboard izgleda kao na slici 2.1.
3. Nova aplikacija može se jednostavno kreirati tako da se klikne na *NEW APP* u gornjem desnom kutu. Back4App obavlja cijelu konfiguraciju.
4. Aplikaciji se može pristupiti kako je navedeno u koraku 2.

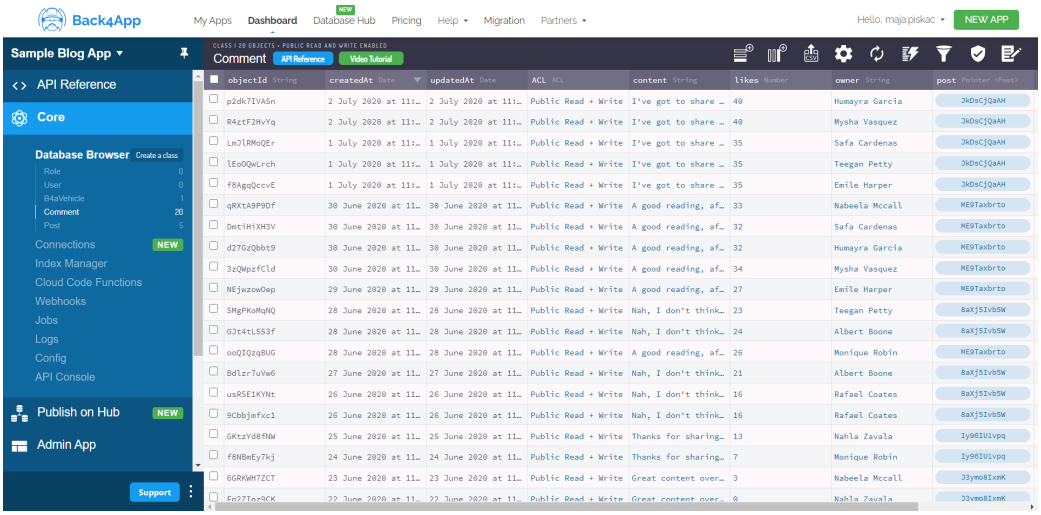
Za integraciju Back4App-a s Angular aplikacijom, potrebno je kao u sekciji 2.3 dodati sljedeće linije koda.

```
1     Parse.initialize(appId, javascriptKey);
2     (Parse as any).serverURL = 'https://parseapi.back4app.com/';
```

Listing 2.28: Povezivanje Back4App-a s Angular-om

Potrebne `appId` i `javascriptKey` može se pronaći tako da se klikne na *My Apps*, zatim kod željene aplikacije klikne na *SERVER SETTINGS*, te zatim kod *Core settings* na *SETTINGS*.

Moguće je i migrirati lokalni Parse Server na Back4App koristeći instrukcije na linku `https://www.back4app.com/docs/app-migration/introduction`.



Slika 2.1: Back4App, Dashboard, Sample Blog App

2.8 Parse Dashboard

Parse Dashboard je samostalna kontrolna ploča za upravljanje Parse Server-om. Ukoliko se želi koristiti, ona se mora naknadno instalirati nakon instaliranja Parse Server-a. Preduvjeti za instaliranje su:

- *Node.js*, verzija 8.9 ili viša
- *Parse Server*, verzija 2.1.4 ili viša.

Instalira se koristeći sljedeću naredbu.

```
$ npm install -g parse-dashboard
```

Može se pokrenuti za određenu aplikaciju koristeći sljedeću naredbu i appId te masterKey koji su odabrani/generirani prilikom pokretanja Parse Server-a. Za serverURL ne može se odabrati `localhost` adresa jer dođe do sukoba s CORS mehanizmom, stoga je potrebno

koristiti url s neke druge domene, primjerice ako se Parse Server razvije na Heroku infrastrukturi. Ime aplikacije je opcionalno.

```
$ parse-dashboard --appId yourAppId --masterKey yourMasterKey
    --serverURL "https://example.com" --appName optionalName
```

Nakon što se Parse Dashboard pokrene, pristupi joj se na `http://localhost:4040`. Izgleda slično kao i Back4App Dashboard. Više informacije može se pronaći na službenom linku [15].

Poglavlje 3

Aplikacija za rezervaciju predavaonica

3.1 Opis aplikacije

Izrađena je aplikacija naziva *Rezervacija predavaonica*, osmišljena kao aplikacija u kojoj se mogu rezervirati predavaonice na PMF - Matematičkom odsjeku. Izgled i funkcionalnosti su sljedeće.



Slika 3.1: Rezervacija predavaonica, početni prikaz

Na početnom zaslonu prikazuje se naziv aplikacije te mogućnost odabira između opcija *Pregled po svim predavaonicama* i *Potraži slobodnu predavaonicu po datumu i vremenu*. Ukoliko se odabere prva opcija prikazuje se lista svih predavaonica koje se nalaze na PMF - Matematičkom odsjeku, kao na slici 3.2. Moguće je kliknuti na bilo koju od njih te pregledati rezervacije koje postoje u toj predavaonici.



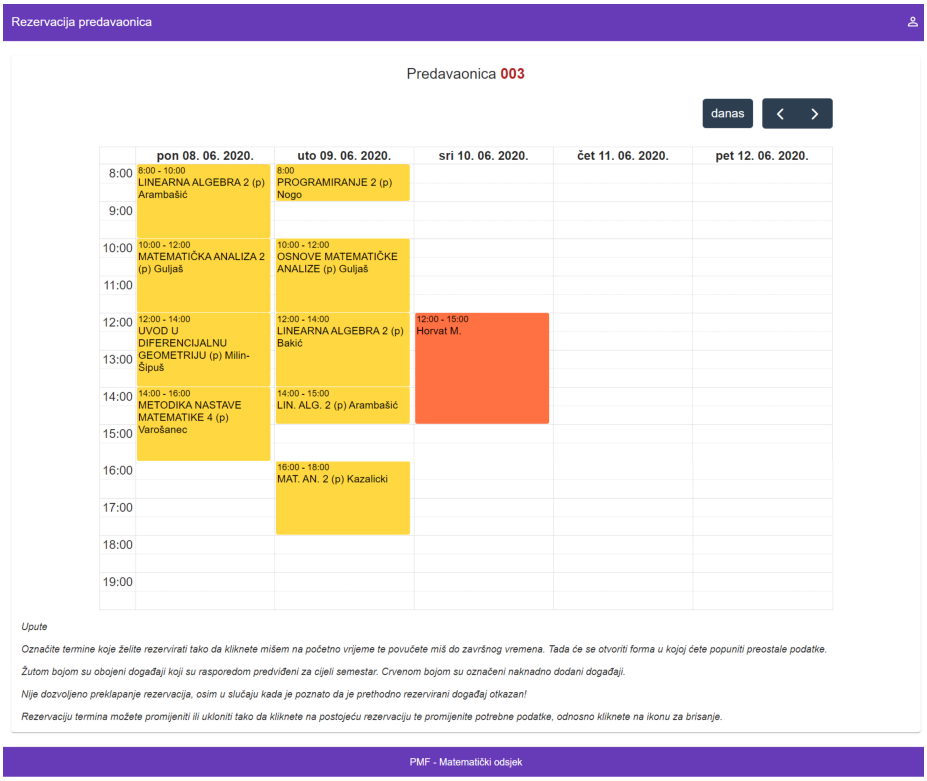
Slika 3.2: Rezervacija predavaonica, pregled svih predavaonica

Primjerice, odabere li se *003*, prikaz će izgledati kao na slici 3.3. Prikazat će se interaktivan kalendar na kojem je moguće koristiti strelice u gornjem desnom kutu kako bi se prikazao prethodni, odnosno sljedeći tjedan. Kliknući na dugme *danas* kalendar se vraća na prikaz trenutnog tjedna. Na gornjoj traci prikazuju se dani u tjednu od ponedjeljka do petka uz pripadni datum, a na lijevoj traci termini od 8:00 do 20:00. Moguće je kliknuti na početak željenog termina na željeni datum te povući miš do završetka, time će se otvoriti forma za spremanje nove rezervacije kao na slici 3.4.

Kako bi se pohranila nova rezervacija potrebno je upisati ime osobe koja rezervira termin, upisati datum i vrijeme u traženom formatu i označiti *Da* ako se rezervacija želi pohraniti za cijeli semestar (radi se o predavanjima, vježbama ili redovitim demonstraturama iz nekog kolegija) ili *Ne* ako se radi o rezervaciji samo za taj dan. Zadano je da se ljetni semestar održava od prvog ponedjeljka u ožujku kroz 7 tjedana nastave, zatim 2 kolokvijska tjedna i 6 tjedana nastave, a zimski semestar od prvog ponedjeljka u listopadu kroz 7 tjedana nastave, 2 kolokvijska tjedna, 3 tjedna nastave, 2 tjedna zimskih praznika i 3 tjedna nastave.

Datumi i vremena će se automatski popuniti u ovisnosti o tome na što se klikne, ali se to može i promijeniti u formi. Da bi se promijenili datumi, potrebno je kliknuti na ikonice kalendara te odabrati datum na kalendaru. Vrijeme se može promijeniti tako da se upiše novo vrijeme na mjesto starog, jedino treba paziti da bude u formatu 'HH:MM'. Nakon što se popune svi obavezni podaci, dugme *Spremi* će poplaviti te će se na njega moći kliknuti i time spremiti rezervacija. Ukoliko se pritisne dugme *Ispočetka*, tada se forma resetira, tj. prikazuju se isti podaci kao i prilikom njenog otvaranja.

Tijekom procesa spremanja rezervacije pojavit će se traka učitavanja, a nakon što je

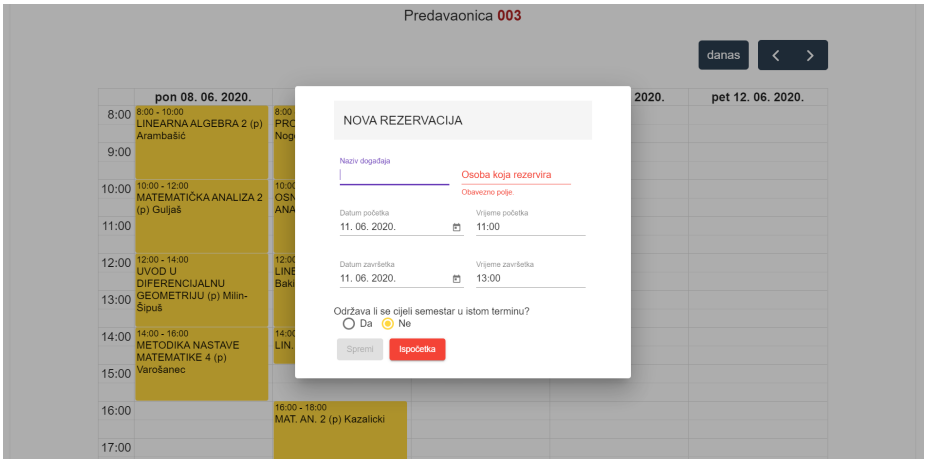


Slika 3.3: Rezervacija predavaonica, pregled rezervacija u predavaonici 003

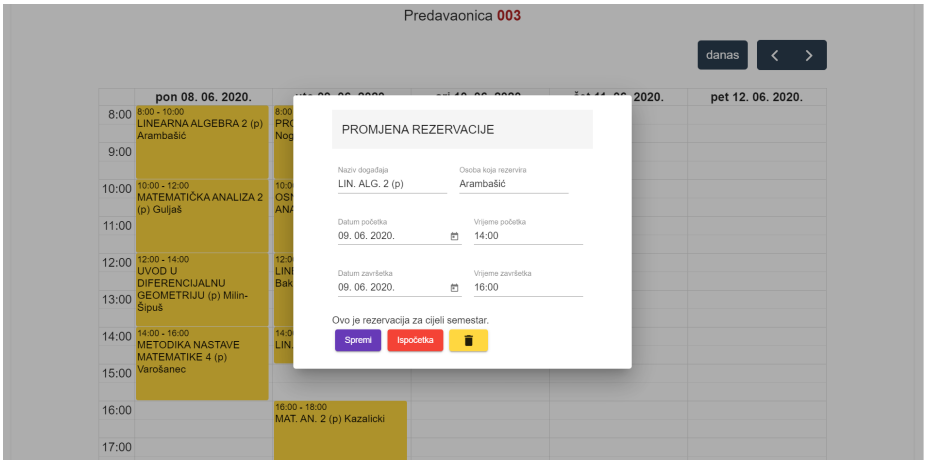
spremanje završeno pojavit će se rezervacija na kalendaru. Ukoliko je spremljena rezervacija za cijeli semestar, ona će biti obojena žuto, a inače ako je rezervacija samo za taj datum, crveno. Iako je moguće spremiti novu rezervaciju u terminu kada postoji već neka rezervacija, korisnika se u uputama ispod kalendara savjetuje da to učini samo ako je siguran da je prvobitna rezervacija za taj termin otkazana, ali nije ažurirana u kalendaru.

Na postojeću rezervaciju može se kliknuti kako bi se ažurirale informacije o toj aplikaciji. Moguće je promijeniti naziv događaja, ime osobe koja rezervira, datume i vremena početka i završetka rezervacije, jedino nije moguće promijeniti rezervaciju da bude za cijeli semestar ako je prvotno spremljena da ne bude za cijeli semestar i obrnuto.

Svi podaci koji se mijenjaju za rezervaciju koja je predviđena za cijeli semestar, mijenjaju se za svaki termin u tom semestru, dakle dovoljno je kliknuti na jedan od datuma u kojem je ta rezervacija spremljena i sve ostale će se zajedno s njome promijeniti. Na ovoj formi dostupno je i dugme na kojem je ikona kante za smeće koja se koristi za brisanje rezervacije. Ukoliko kliknemo na to dugme, pojavit će se skočni prozor na kojem se traži potvrda brisanja. I u ovom slučaju, ukoliko se želi obrisati rezervacija koja je spremljena



Slika 3.4: Rezervacija predavaonica, forma za novu rezervaciju

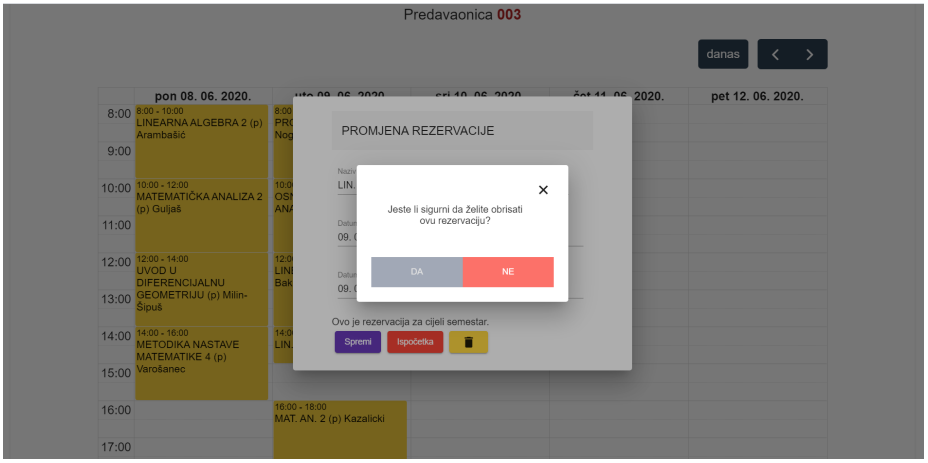


Slika 3.5: Rezervacija predavaonica, forma za promjenu rezervacije

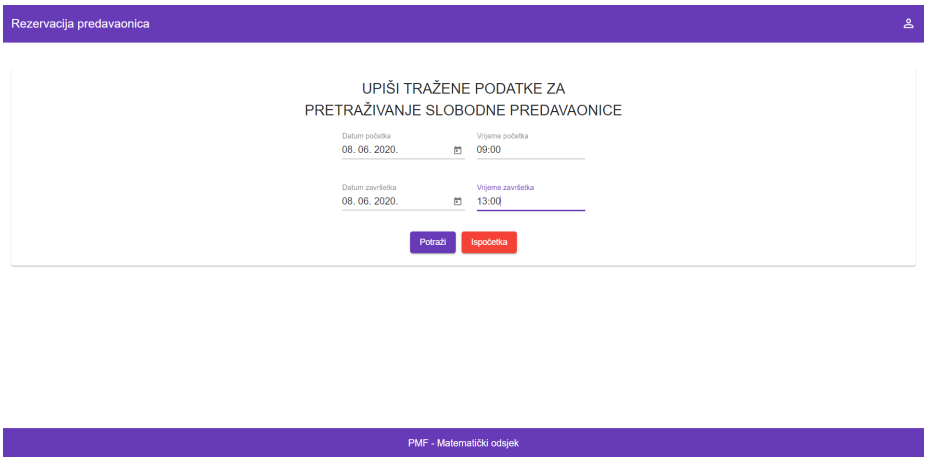
za cijeli semestar, obrisat će se sve rezervacije za cijeli semestar zajedno s njome. Izgled skočnog prozora može se vidjeti na slici 3.6.

Ukoliko se klikne na naziv aplikacije, dakle *Rezervacija predavaonica*, vratit će se prikaz početne stranice gdje je moguće odabrati *Pregled po svim predavaonicama* ili *Potraži slobodnu predavaonicu po datumu i vremenu*. Klikom na drugu opciju, prikazat će se forma sa slike 3.7. Na toj formi mogu se odabrati datumi početka i završetka željene rezervacije putem ikonice kalendara, te vremena početka i završetka rezervacije. Klikom na dugme *Potraži*, aplikacija će proći kroz sve postojeće rezervacije u svim predavaonicama

te ispisati samo one predavaonice u kojima na željene datume i vremena nema spremljenih rezervacija, odnosno koje su slobodne u željenom terminu.



Slika 3.6: Rezervacija predavaonica, skočni prozor za potvrdu brisanja rezervacije



Slika 3.7: Rezervacija predavaonica, pretraga slobodne predavaonice prema željenom terminu

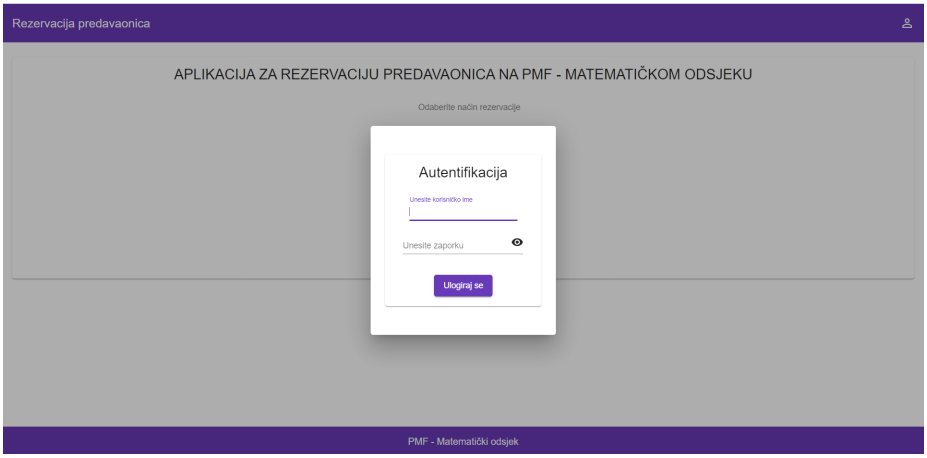
Primjerice, odaberemo li za datum početka i završetka *08. 06. 2020.* te za vrijeme početka *09:00* i vrijeme završetka *13:00*, ispisat će se sve predavaonice osim *003* i *004* jer smo u njima imali predavanja iz nekih kolegija te one nisu bile dostupne. Ispis možemo

vidjeti na sljedećoj slici. Tada se ponovno može kliknuti na neku od predavaonica te izvršiti rezervaciju. Time je završen pregled funkcionalnosti koje su omogućene svakom korisniku.

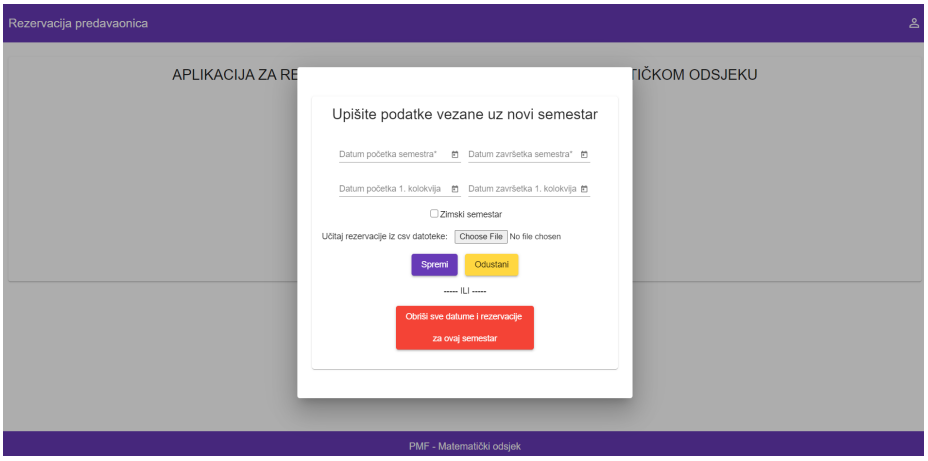


Slika 3.8: Rezervacija predavaonica, ispis slobodnih predavaonica za željeni termin

Nadalje, postoje i funkcionalnosti koje su omogućene samo korisniku višeg pristupa, odnosno administratoru. Njima se može pristupiti ako se klikne na ikonicu čovječuljka u gornjem desnom kutu zaglavlja aplikacije. Otvara se prozor u koji je potrebno unijeti korisničko ime i zaporku administratora. One se mogu promijeniti u datoteci *rezervacija-predavaonica/src/app/modules/admin/admin.component.ts* u loginForm-i.



Slika 3.9: Rezervacija predavaonica, autentifikacija administratora



Slika 3.10: Rezervacija predavaonica, forma za administratora

Ukoliko se administrator uspješno ulogirao, nude mu se sljedeće mogućnosti. Može promijeniti datum početka i završetka semestra (kao završetak semestra smatra se zadnji dan održavanja nastave), datum početka i završetka prvih kolokvija (koja se u trenu stvaranja aplikacije održavaju unutar semestra te u njihovo vrijeme nema održavanja nastave, stoga se u vrijeme održavanja prvih kolokvija neće popuniti rezervacije za cijeli semestar), te datum početka i završetka zimskih praznika ukoliko se radi o zimskom semestru. Spremanjem novih datuma, brišu se prethodno odabrani datumi za trajanje semestra te sve rezervacije za prethodni semestar (smatra se da će se rezervacije za novi semestar popunjavati tek nakon što završi prethodni semestar, a brisanjem starih rezervacija želi se rasteretiti sustav od spremanja previše rezervacija).

Moguće je učitati csv datoteku sa svim rezervacijama za semestar kako bi se moglo lakše i brže učitati sve te rezervacije. Jedan redak u csv-u odgovara jednoj rezervaciji za semestar. Redak treba imati sljedeći format:
`OZNAKA_PREDAVAONICE,NAZIV_DOGAĐAJA,ime_osobe,DAN_U_TJEDNU,HH:MM,HH:MM`
pri čemu se prvo upisuje vrijeme početka, a zatim vrijeme završetka. Kao `DAN_U_TJEDNU` treba upisati `PON`, `UTO`, `SRI`, `ČET` ili `PET`. Primjerice, jedan redak može sadržavati sljedeće podatke: `003,PROGRAMIRANJE 2 (p),Nogo,UTO,08:00,10:00`.
Time će se spremiti rezervacija za utorak od početka do kraja semestra u terminu 08:00 - 10:00. Važno je napomenuti da se kod csv formata ne smiju ostavljati razmaci pored zareza, posebice kod polja za predavaonicu, dan i vremena.

Kao posljednje, administratoru je omogućeno da obriše sve rezervacije unutar trenutnog semestra, uključujući i rezervacije koje nisu za cijeli semestar, ukoliko primjerice vidi da je unesena pogreška koju je najlakše ispraviti brisanjem svih podataka.

3.2 Full Calendar

Za prikaz interaktivnog kalendara korištena je JavaScript-ina biblioteka *FullCalendar* koja također pruža komponentu za spajanje s Angular-om. Kako bi se mogla koristiti, potrebno je izvršiti sljedeću naredbu, gdje je *@fullcalendar/timegrid* potreban za prikaz tjednog kalendara po satima, a *@fullcalendar/interaction* kako bi se omogućila interakcija.

```
$ npm install --save @fullcalendar/angular @fullcalendar/
    timegrid @fullcalendar/interaction
```

Zatim je u korijen aplikacije potrebno dodati sljedeće linije koda.

```
1 import { FullCalendarModule} from '@fullcalendar/angular';
2 import timeGridPlugin from '@fullcalendar/timegrid';
3 import interactionPlugin from '@fullcalendar/interaction';
4
5 FullCalendarModule.registerPlugins([
6   timeGridPlugin,
7   interactionPlugin
8 ]);
```

Listing 3.1: Povezivanje FullCalendar-a s Angular-om

U predložak komponente, u kojoj se želi koristiti, upisuje se sljedeće.

```
1 <full-calendar #calendar [options]="calendarOptions">
2 </full-calendar>
```

Listing 3.2: Prikazivanje FullCalendar-a u Angular-u

U komponenti se određuju opcije kalendara. Za ovu aplikciju korištene su sljedeće opcije.

```
1  import { CalendarOptions } from '@fullcalendar/angular';
2
3  calendarOptions : CalendarOptions = {
4      initialView: 'timeGridWeek',
5      initialDate: new Date(localStorage.getItem('lastDateChecked')),
6      editable: true,
7      selectable: true,
8      eventClick: this.handleDateClick.bind(this),
9      select: this.handleDateSelect.bind(this),
10     weekends: false,
11     firstDay: 1,
12     locale: 'hr',
13     slotLabelFormat:{
14       hour: 'numeric',
15       minute: '2-digit'
16     },
17     slotDuration: '00:30:00',
18     slotMinTime: '08:00:00',
```

```
    slotMaxTime: '20:00:00',
    height: "auto",
    buttonText:{
      today: 'danas'
    },
    eventColor: "#FFD740",
    eventTextColor: "black",
    headerToolbar: {
      start: '',
      center: '',
      end: 'today prev,next'
    },
    dayHeaderFormat: { weekday: 'short', month: 'numeric', day: '
       numeric', year: 'numeric', omitCommas: true},
    allDaySlot: false
};
```

Listing 3.3: Rezervacija predavaonica - opcije FullCalendar-a

Da bi se moglo pristupiti komponenti kalendara u Angular-u i primjerice saznati posljednji datum na koji je na kalendaru kliknuto, potrebne su sljedeće dvije linije koda.

```
@ViewChild('calendar') calendarComponent: FullCalendarComponent;
this.calendarComponent.getApi().getDate();
```

Listing 3.4: Pristup komponenti kalendara u Angular-u

Dodatne mogućnosti FullCalendar-a mogu se pronaći na službenoj stranici na linku [16].

3.3 Struktura aplikacije

Aplikacija je građena po uzoru na literaturu [19]. Stoga su izgrađeni moduli *SharedModule* i *CoreModule*. U *SharedModule* uvoze se svi ostali moduli i izlažu sve zajedničke komponente i moduli koji će biti korišteni kroz cijelu aplikaciju. U ovoj aplikaciji zajedničke module čine moduli iz Angular Material, FullCalendarModule i moduli za izradu formi, a zajedničke komponente čine sve komponente aplikacije osim korijena.

```
const SHARED_MODULES = [ CommonModule, MatToolbarModule,
    MatCardModule, MatIconModule, MatButtonModule, MatDividerModule,
    MatFormFieldModule, MatInputModule, MatProgressBarModule,
    MatDialogModule, MatRadioModule, MatCheckboxModule,
    MatDatepickerModule, MatNativeDateModule, ReactiveFormsModule,
    FormsModule, RouterModule, FullCalendarModule ];

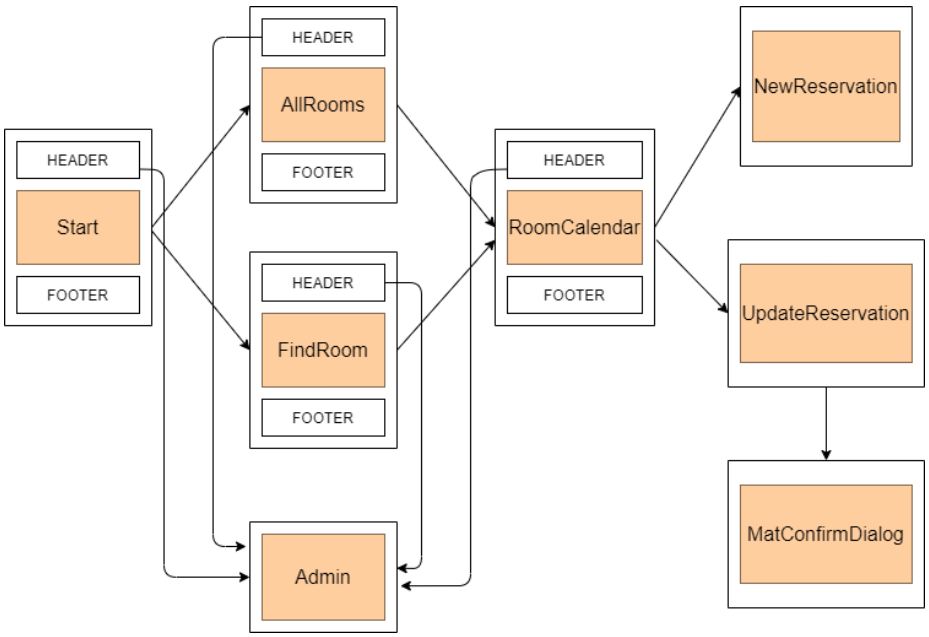
const SHARED_COMPONENTS = [ StartComponent, AllRoomsComponent,
    FindRoomsComponent, RoomCalendarComponent,
    NewReservationComponent, UpdateReservationComponent,
    MatConfirmDialogComponent, AdminComponent ];
```

```
@NgModule({
  imports: [...SHARED_MODULES],
  declarations: [...SHARED_COMPONENTS],
  exports: [...SHARED_MODULES, ...SHARED_COMPONENTS]
})
export class SharedModule {}
```

Listing 3.5: Rezervacija predavaonica - SharedModule

CoreModule sadrži sve pružatelje usluga koji su zasebni od komponenata i koji se inicijaliziraju čim aplikacija započne rad, on se uvozi samo jednom u korijen aplikacije *AppModule*. U ovoj aplikaciji on sadrži `data.service.ts` i `CSVRecord`. U `data.service.ts` obrađuju se svi upiti na Parse Server, a `CSVRecord` je klasa koja sadrži varijable koje odgovoraju pojedinim poljima u jednom retku csv datoteke, dakle jedna instance te klase odgovara jednom retku csv-a.

Aplikacija ima plan (engl. *layout*) koji se sastoji od zaglavlja (engl. *header*), tijela (engl. *body*) i podnožja (engl. *footer*). Zaglavlje i podnožje ostaju isti kroz različite poglede, a u tijelu se izmjenjuje pogled određene komponente.



Slika 3.11: Rezervacija predavaonica, skica aplikacije

Admin, *NewReservation*, *UpdateReservation* i *MatConfirmDialog* nemaju zaglavlje i podnožje jer se otvaraju kao skočni prozori povrh otvorenog zaslona. Prelazak iz jednog u

drugi pogled kontrolira se u *Router*-u. Za svaku komponentu treba definirati put kojim se dođe do nje.

```
const routes: Routes = [
  { path: '',
    component: StartComponent },
  { path: 'all-rooms',
    component: AllRoomsComponent },
  { path: 'find-rooms',
    component: FindRoomsComponent },
  { path: 'all-rooms/room-calendar/:roomLabel',
    component: RoomCalendarComponent },
  { path: 'find-rooms/room-calendar/:roomLabel',
    component: RoomCalendarComponent }
];
@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }
```

Listing 3.6: Rezervacija predavaonica - AppRoutingModule

Primjerice, ako želimo iz komponente *AllRooms* preći u komponentu *RoomCalendar* kada kliknemo na gumb željene sobe, trebamo dodati sljedeći *routerLink* u <button> element u predlošku komponente *AllRooms*.

```
<button mat-raised-button color="accent" class="button"
*ngFor="let room_capacity of rooms_capacities | keyvalue; index as
    roomLabel" [routerLink]="['room-calendar', room_capacity.key]">
    <b>{{room_capacity.key}}</b> <br> ({{room_capacity.value}})
</button>
```

Listing 3.7: Rezervacija predavaonica - AllRooms button

Gdje *room_capacity.key* sadrži podatak o oznaci sobe, a *room_capacity.value* broj dostupnih sjedala u predavaonici.

3.4 Progresivna web aplikacija

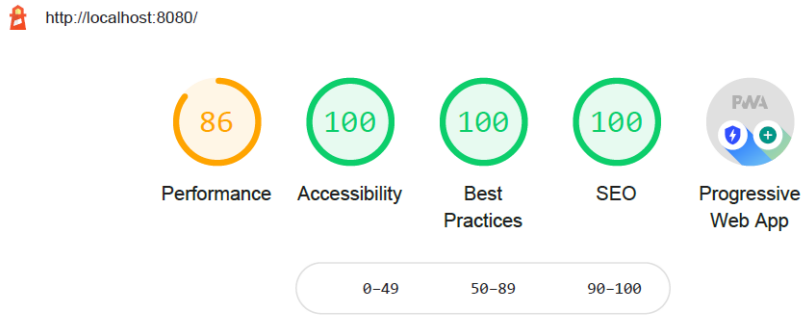
Izrađenoj aplikaciji dodana su svojstva PWA kako je opisano u sekciji 1.5. Nakon toga je generiran izvještaj u Google Lighthouse-u kojem se može pristupiti u Google Chrome pregledniku tako da se klikne na postavke u desnom kutu alatne trake, zatim na *More Tools*, nakon toga na *Developer Tools* te na alatnoj traci pronađe *Lighthouse*. Prvi izvještaj imao je loše performase. Razlog tomu je bilo nekoliko instaliranih paketa koji nisu bili korišteni, no bilo je zaboravljeno maknuti ih. Popis tih paketa može se pronaći pokretanjem sljedeće naredbe.

```
$ depcheck
```

Naredba vraća popis paketa u `package.json` koji nisu potrebni. Oni se tada mogu obrisati iz te datoteke, a iz *node_modules* mape mogu se maknuti koristeći sljedeću naredbu.

```
$ npm prune
```

Također, aplikaciju je preporučeno graditi sa zastavicom `--prod`. Nakon toga izvještaj je bio sljedeći. Aplikacija je ocjenjena kao brza i povjerljiva, te instalabilna, no nije zadovoljila svojstvo PWA optimiziranosti jer ne preusmjerava HTTP promet na HTTPS vezu. Zbog istog problema performasa aplikacije nije ocjenjena zelenom bojom.



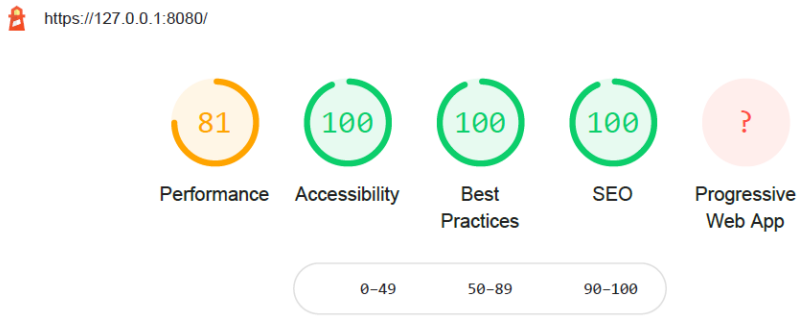
Slika 3.12: Rezervacija predavaonica, Lighthouse report, http-server

Kako bi se riješio taj problem, pokušano je više metoda. Prvo je dodan samododjeljen ključ i certifikat pomoću openssl-a kako bi se omogućio HTTPS na http-serveru. Međutim, to nije uspjelo popraviti problem jer je tada Lighthouse javljao grešku kod pokušaja utvrđivanja preumjeravanja HTTP prometa na HTTPS. Sažetak izvještaja može se vidjeti na slici 3.13.

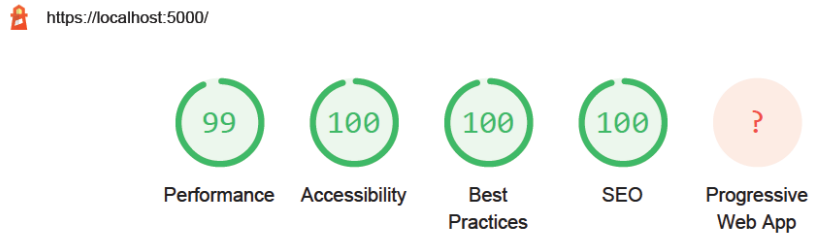
Nakon toga pokušano je pokretanje aplikacije na *simplehttp2server*-u čija je dokumentacija dostupna na [17]. On poslužuje trenutni direktorij na HTTP/2.0 serveru. Time je popravljen navedeni problem, no sada nije uspjelo registriranje service worker-a, stoga ponovno aplikacija nije prepoznata kao PWA. Sažetak izvještaja može se vidjeti na slici 3.14.

Konačno, kako bi se provjerilo da li aplikacija uistinu zadovoljava svojstva PWA, ona je postavljena na Firebase čija je dokumentacija dostupna na [18]. Izvještaj koji se tada dobio zadovoljavao je sva svojstva PWA. Sažetak izvještaja dan je na slici 3.15.

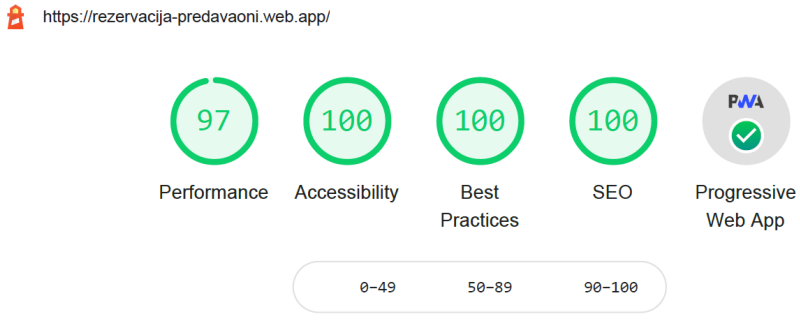
Time je razvoj aplikacije završen. Cjelokupni kod aplikacije može se pronaći na linku `https://github.com/pimaja/Angular-PWA`.



Slika 3.13: Rezervacija predavaonica, Lighthouse report, https-server



Slika 3.14: Rezervacija predavaonica, Lighthouse report, simplehttp2server



Slika 3.15: Rezervacija predavaonica, Lighthouse report, Firebase

Bibliografija

[1] *Node.js*, https://nodejs.org/en/, posjećena 27.7.2020.

[2] *Angular Material komponente*, https://material.angular.io/components/categories, posjećena 27.7.2020.

[3] *Angular, Intro to Basic Concepts*, https://angular.io/guide/architecture, posjećena 28.7.2020.

[4] *Angular Cheatsheet*, https://angular.io/guide/cheatsheet, posjećena 29.7.2020.

[5] *Angular, A Sample App*, https://angular.io/start, posjećena 30.7.2020.

[6] *What are Progressive Web Apps?*, https://web.dev/what-are-pwas/, posjećena 31.7.2020.

[7] *http-server*, https://www.npmjs.com/package/http-server, posjećena 5.8.2020.

[8] *Angular, Service Worker*, https://angular.io/guide/service-worker-getting-started, posjećena 5.8.2020.

[9] *Parse Server Guide*, https://docs.parseplatform.org/parse-server/guide/, posjećena 6.8.2020.

[10] *Cygwin*, https://cygwin.com/index.html, posjećena 6.8.2020.

[11] *MongoDB*, https://www.mongodb.com/, posjećena 6.8.2020.

[12] *Parse Server, Javascript Guide*, https://docs.parseplatform.org/js/guide/#objects, posjećena 7.8.2020.

[13] *Heroku*, https://devcenter.heroku.com/, posjećena 8.8.2020.

[14] *Back4App*, https://www.back4app.com/, posjećena 8.8.2020.

[15] *Parse Dashboard*, https://github.com/parse-community/parse-dashboard, posjećena 8.8.2020.

[16] *FullCalendar*, https://fullcalendar.io/docs/angular, posjećena 9.8.2020.

[17] *SimpleHttp2Server*, https://github.com/GoogleChromeLabs/simplehttp2server, posjećena 10.8.2020.

[18] *Firebase Hosting*, https://firebase.google.com/docs/hosting/, posjećena 10.8.2020.

[19] M. Hajian, *Progressive Web Apps with Angular*, Apress, 2019.

Sažetak

U ovom diplomskom radu opisana je izrada progresivne web aplikacije u Angular-u koristeći Parse Server kao backend te JavaScript-inu biblioteku FullCalendar za izradu interaktivnog kalendara. Namjena aplikacije je rezervacija predavaonica na PMF - Matematičkom odsjeku. Također, opisane su osnove Angular-a u prvom poglavlju i Parse Server-a u drugom poglavlju.

Summary

This master thesis describes the construction of a progressive web application in Angular using Parse Server as a back-end technology and the JavaScript library FullCalendar for making an interactive calendar. The purpose of the application is to reserve lecture halls in the Mathematics department of the Faculty of Science at the University of Zagreb. Furthermore, given are the basics of Angular and Parse Server.

Životopis

Maja Piskač rođena je 18. 01. 1997. u Varaždinu. Odrasla je u Ivancu, Varaždinskoj županiji, gdje je i pohađala Osnovnu školu Ivana Kukuljevića Sakcinskog i Srednju školu Ivanec, smjer opća gimnazija. Nakon završetka srednjoškolskog obrazovanja 2015. godine upisuje preddiplomski studij Matematika na Matematičkom odsjeku Prirodoslovno-matematičkog fakulteta Sveučilišta u Zagrebu. Isti završava 2018. godine te tada upisuje diplomski studij Računarstvo i matematika na istom fakultetu.