Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Vid Stoschitzky

Izdelava spletnih aplikacij z ogrodjem Angular

diplomsko delo

visokošolski strokovni študijski program
prve stopnje
računalništvo in informatika

Mentor: viš. pred. dr. Aljaž Zrnec

Ljubljana, 2017

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Angular je zelo hitro po njegovem nastanku postal eden od najbolj uporabljenih ogrodij za izdelavo spletnih aplikacij. Prednosti izbire tega ogrodja so hitra in odzivna končna rešitev. Prav tako je izdelano spletno aplikacijo enostavno vzdrževati, saj je koda napisana v stilu komponent. V okviru diplomske naloge primerjajte ogrodje Angular z ogrodjem AngularJS in knjižnico React ter izdelajte spletno aplikacijo za iskanje igralcev za rekreativne športne dejavnosti v tem ogrodju.

V prvi vrsti se zahvaljujem mentorju viš. pred. dr. Aljažu Zrnecu, za vse nasvete in pomoč pri izdelavi celotnega diplomskega dela. Prav tako se zahvaljujem svoji družini, prijateljem in vsem, ki so mi tako ali drugače pomagali z nasveti.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
JS	JavaScript	JavaScript
HTML	Hyper Text Markup Language	jezik za označevanje hiper besedila
CSS	Cascading style sheets	Kaskadne stilske predloge
DOM	Document object model	Dokumentni objektni model
API	Application Programming Interface	Vmesnik za programiranje
ID	Identification	Identifikator
CLI	Command-line interface	Vmesnik ukazne vrstice
JSON	JavaScript Object Notation	objektna notacija JavaScript
HTTP	HyperText Transfer Protocol	protokol za prenos hiperteksta
XML	Extensible Markup Language	razširljiv označevalni jezik

Povzetek

Naslov: Izdelava spletnih aplikacij z ogrodjem Angular

Avtor: Vid Stoschitzky

Diplomska naloga je sestavljena iz predstavitve ogrodja Angular, njegovih ključnih funkcionalnosti in predstavitve ogrodja na konkretnem izdelku - spletni aplikaciji SportPlayer. V sklopu diplomske naloge bomo prav tako spoznali dve drugi orodji za razvoj spletnih aplikacij. Prvo je ogrodje AngularJS, drugo pa knjižnica React. Angular je ogrodje za razvoj spletnih aplikacij in ga uporabimo za razvoj prednjega dela (*ang. Front end*) neke spletne rešitve. Prvič ga je predstavilo podjetje Google v letu 2009. Obstaja sicer več različic in imen tega ogrodja, a vendar je zdaj ime poenoteno v Angular. Je eno najpogosteje uporabljenih ogrodij za izdelavo spletnih aplikacij, saj se ponaša s hitrim delovanjem izdelanih aplikacij, možnostjo delovanja na več platformah in z velikim številom uporabnikov. Vse te lastnosti so predstavljene na izdelani spletni aplikaciji SportPlayer.

Ključne besede: spletna aplikacija, Angular, AngularJS, React, SportPlayer .

Abstract

Title: Development of web applications with Angular framework

Author: Vid Stoschitzky

The thesis consists of describing the framework Angular, its key functionalities, and a demonstration of its usage on a concrete example - a web application SportPlayer. Furthermore the thesis also includes the description of two other options for web application development. The first option being the framework called AngularJS, and the other one a library called React. Angular is a framework for developing web applications, more specifically for the front end of web solutions. It was first announced in 2009 by Google. There are many different versions known by different names, however they all fall under the umbrella term Angular. This framework is one of the most used when it comes to web application development. Its main advantages are: Speed of the finished product, the possibility of deploying the final solution to multiple platforms and a large user community. All these features are showcased later on in the application SportPlayer.

Keywords: web application, Angualr, AngularJS, React, SportPlayer.

Poglavje 1

Uvod

Ko govorimo o novodobnih spletnih aplikacijah, imamo v mislih več karekteristik, ki jih mora po našem mnenju vsebovati tovrstna aplikacija. Le-ta mora biti zelo hitra, saj današnji življenjski slog praktično ne dopušča izgubljenega časa, zato počasne aplikacije niso konkurenčne. Ena od glavnih lastnosti pa je prav gotovo tudi odzivnost (*ang. Responsive*) spletne aplikacije. Novodobni uporabniki za dostop do spleta v veliki večini raje uporabljajo svoje mobilne naprave kot pa namizne ali prenosne računalnike. S stališč naročnika, razvijalca ali končnega uporabnika torej zahtevamo, da če želimo dostopati do neke spletne storitve iz mobilne naprave, se nam le-ta pokaže enako uporabno in pregledno, kot če bi do nje dostopali iz svojega prenosnega računalnika. Vse te navedene lastnosti najbolje zajamemo z ogrodjem Angular, katerega glavne prednosti so hitrost, odzivnost, pregledna koda, ponovna uporabnost kode in možnost razvoja za več platform naenkrat. Prav tako pa ima to ogrodje veliko uporabnikov in je razvoj še toliko lažji.

Seveda pa ogrodje Angular ni edino primerno za razvoj novodobnih spletnih aplikacij. V nadaljevanju diplomskega dela bomo spoznali še ogrodje React, ki ga je razvilo podjetje Facebook. Prav tako bo predstavljeno tudi ogrodje AngularJS, ki je sicer prvotna različica ogrodja Angular, vendar pa se od zdajšnje različice zelo razlikuje. Praktično je podjetje Google novejše

ogrodje razvilo na novo. Obstaja ogromno spletnih aplikacij, ki so razvite z ogrodjem AngularJS in tudi število uporabnikov je bistveno večje kot pri drugih omenjenih. Vendar, kot bomo lahko spoznali kasneje, tudi obe ostali ogrodji zelo hitro pridobivata na popularnosti ter številu novih uporabnikov.

Poglavje 2

Ogrodja za razvoj spletnih aplikacij

V poglavju, ki sledi, bom predstavil pogled naročnika in uporabnika na poljubno spletno aplikacijo. Bodisi že obstoječo, bodisi aplikacijo, ki je še v fazi načrtovanja. Spoznali bomo 2 ogrodji za izdelavo tovrstnih aplikacij ter eno knjižnico. Malce bolj podroben je opis ogrodja Angular, saj so njegovi sestavni deli zelo pomembni za kasnješo predstavitev naše spletne aplikacije SportPlayer.

2.1 Lastnosti spletnih aplikacij

Če se postavimo v vlogo naročnika neke spletne aplikacije, imamo v mislih kar nekaj različnih kriterijev, ki jih želimo implementirati v našo končno rešitev. Ena od njih je prav gotovo čas izdelave. Kot naročnik seveda želimo naš izdelek čimprej videti na trgu in za njega porabiti čim manj denarja. Za izpolnitev takšnih pogojev bomo potrebovali najboljšo možno tehnologijo, ki jo bomo implementirali v naš izdelek. Vsa napisana koda pa mora biti seveda lahko berljiva, kar pomeni, da ljudje, ki razvijajo ta produkt, z lahkoto dodajajo nove funkcionalnosti k naši spletni aplikaciji in da pri tem ne izgubljajo predragocenega časa. Želimo tudi, da je naša izbrana tehnologija

zasnovana na način, da je pripravljena na hitre spremembe, ki se dogajajo pri razvoju spletnih aplikacij. Pomembno pa je, da tehnologija, ki smo jo izbrali, ni zastarela že po nekaj letih. Kot velika prednost izbranega ogrodja ali tehnologije se lahko izkaže tudi velika baza že obstoječih uporabnikov. Seveda kot razvojniki velikokrat naletimo na težavo, ki na prvi pogled ni preprosta za odpravo ali celo sami ne najdemo rešitve. V takšnih primerih se izkaže veliko število uporabnikov zelo priročno, saj lahko hitro najdemo problem in rešitev na za to namenjenih portalih. Komentarjev na teh portalih se poslužujejo tudi razvojniki samega ogrodja, kar odgovor naredi še bolj kakovosten.

Seveda pa je pomemben vidik naše končne rešitve tudi pogled s strani samih uporabnikov. Novodobni uporabniki za dostopanje do spleta že dolgo več ne uporabljajo le svojih namiznih ali prenosnih računalnikov. Velika večina dostopanja do spleta se dogaja preko mobilnih telefonov ali tabličnih računalnikov, zato je zelo pomembno, da lahko uporabniku ponudimo našo rešitev na katerikoli platformi in pri tem ne izgubimo prepoznavnega izgleda ali delovanja rešitve. Če pa izberemo takšno tehnologijo, ki omogoča uporabo iste kode za razvoj aplikacije na različnih platformah, pa smo našim razvojnikom prihranili ogromno časa, sebi pa denarja. Naslednja ključna lastnost končne spletne aplikacije mora biti tudi hitrost. Uporabniki navadno nimajo časa čakati, da se stran odzove. Počasnost aplikacije tudi zelo poslabša uporabniško izkušnjo. Veliko študij [15, 16, 17] dokazuje, da se velika večina uporabnikov ne odloča za izbiro in kasnejše plačilo spletne aplikacije, ki ni performančno na zelo visokem nivoju, čeprav je primarni namen te aplikacije zelo dobro pokrit.

2.2 Angular

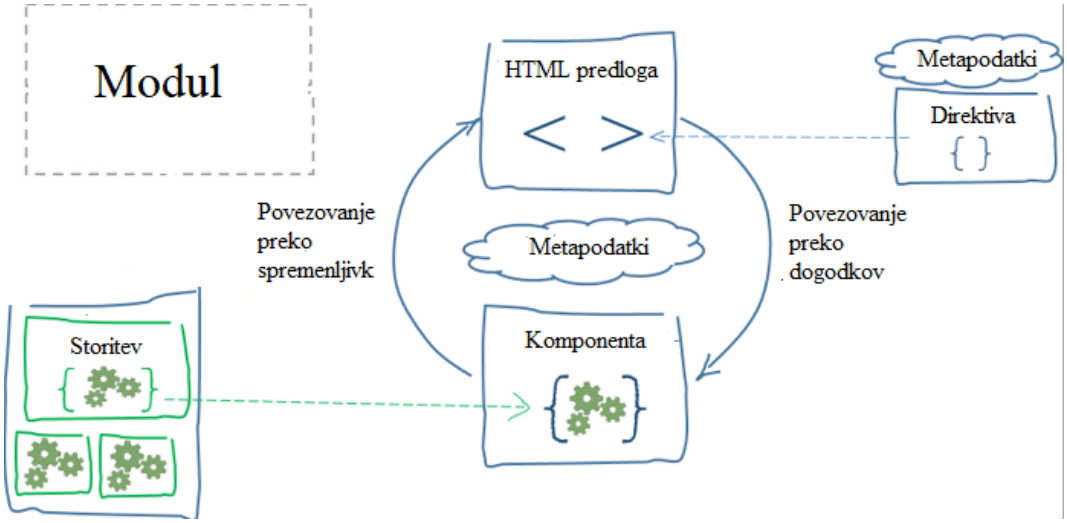
Angular je ogrodje, s katerim razvijamo prednji del (*ang. Front-end*) spletne apliakcije. Slika 2.1 prikazuje logotip tega ogrodja. Če razvijamo aplikacijo s

pomočjo ogrodja Angular, potem pri svojem delu uporabljamo skriptni jezik JavaScript. Ko govorimo o ogrodju Angular, za delo uporabljamo TypeScript, ki se na koncu prevede v JavaScript. TypeScript je odprtokodni programski jezik. Razvilo ga je podjetje Microsoft leta 2012. Pri programskem jeziku TypeScript gre za nadgradnjo programskega jezika JavaScript. Nekaj ključnih prednosti TypeScripta je uporaba razredov (*ang. Classes*), vmesnikov (*ang. Interface*), modulov (*ang. Modules*), enumov... Uporaba naštetih orodij nam je v skriptnem jeziku JavaScript onemogočena. Še ena pomembna prednost uporabe TypeScript-a je zagotovo preverjanje vrednosti statičnih tipov. Če imamo neko spremenljivko, za katero zahtevamo, da je tipa številka(*ang. Number*), le-tej pri uporabi TypeScript-a ne moremo dodeliti npr. niza (*ang. String*). V tem primeru bi nam TypeScript javil napako in se ne bi prevedel. Vse te lastnosti nam zelo pomagajo in olajšajo samo programiranje končne aplikacije.

Slika 2.1: Logotip ogrodja Angular

Ogrodje Angular je sestavljeno iz več knjižnic, veliko od njih je ključnih za delovanje, nekaj od njih pa je opcijskih. Angular aplikacija ima več sestavnih delov, ki jih imenujemo moduli. Modul kasneje implementira še nekaj pomembnih delov Angular aplikacije. Prvi je HTML datoteka, ki vsebuje značilne Angular označbe. Za "logiko"teh HTML datotek pa skrbi komponentni razred. Ta razred definiramo v modulu pod lastnostjo *declarati-*

ons. Posamezna komponenta iz pogleda pridobiva podatke in jih uporabi na zamišljen način. Komunikacija pa seveda deluje tudi v obratni smeri. Komponente ponavadi za klice na API - spletni vmesnik uporabljajo storitve (*Services*). Angular aplikacijo zaženemo tako, da zaženemo prvi modul(*ang. Root module*). V Datoteki *main.ts* nastavimo vse potrebno za opisan zagon Angular aplikacije.



Slika 2.2: Arhitektura ogrodja Angular

Slika 2.2 prikazuje arhitekturo ogrodja Angular. Glavna dela sta seveda komponenta in njena predloga, ki vsebuje HTML kodo. Komponento lahko s pomočjo metapodatkov nastavimo tako, da ustreza našim zahtevam. Večina komponent uporablja vsaj eno storitev. Storitev razširi funkcionalnost te komponente in skrbi za kompleksnejše dele kode. Vsaka HTML predloga lahko vključuje poljubno število direktiv. Tudi direktivo lahko nastavljamo po svoji želji s pomočjo metapodatkov. Vsaka komponenta pripada enemu modulu, ki jo deklarira, ko jo potrebuje. Podrobno razlago vsakega od omenjenih pojmov bomo spoznali v nadaljevanju.

2.2.1 Modul

Vsaka Angular aplikacija ima vsaj en modularni razred. Modularni razred predstavlja posamezen modul. Za manj zahtevne ali manjše aplikacije ponavadi zadostuje le en modul, ki naloži vse komponenete, ki jih uporablja ta aplikacija. Glavnemu modulu rečemo korenski modul (*Root module*). Če pa govorimo o večji bolj zahtevni aplikaciji, potem imamo navadno modulov več. Vsak modularni razred označimo z dekoratorjem *@NgModule*. NgModule je dekorator (*decorator*), ki kot parameter sprejme en objekt, ki ima več možnih lastnosti, s katerimi nakažemo, kako se bo modul obnašal. Glavni modul Angular aplikacije po privzetih nastavitvah vsebuje 4 lastnosti: *declarations, imports, providers, bootstrap.*

2.2.2 Komponenta

Komponenta je eden od ključnih delov Angular aplikacij. Komponenta komunicira s pogledom (HTML datoteka). Zadolžena je za pošiljanje podatkov na pogled, prav tako pa pridobi podatke s pogleda. Podatke lahko inicializira sama ali pa jih dobi iz raznovrstnih virov (največkrat jih pridobi iz storitve). Komponento prepoznamo po dekoratorju *@Component*. Ta objekt vsebuje kar nekaj lastnosti. Eni ključnih sta lastnosti *templateUrl* in *styleUrls*, ki vsebujeta pot do HTML datoteke in pot ali več poti do datotek CSS, ki so namenjena stiliranju pogleda komponente. Zanimiva je tudi lastnost *providers*, ki komponenti pove, katere storitve (*ang. Services*) oziroma ponudniki (*ang. Providers*) so ji na voljo. Na spodnji sliki (slika 2.3) lahko vidimo primer preproste komponente s privzetimi nastavitvami.

```
import { Router } from '@angular/router';
import { Component } from '@angular/core';

import { LoginService } from './login/login.service';

import * as moment from 'moment/moment';

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

Slika 2.3: Primer preproste komponente

2.2.3 Metapodatki

S pomočjo metapodatkov Angular pridobi informacijo o obnašanju določenega razreda. Služijo kot konfiguracijski podatki. Metapodatke lahko najdemo pri inicializaciji komponent, modulov, storitev...

2.2.4 Storitve

Storitve so najpogosteje razredi z dobro definiranim namenom. Uporabljamo jih kot razširitev delovanja komponent. Ogrodju Angular s tem povemo, da gre za storitev, ko le-tega označimo z dekoratorjem *@Injectable*. Ponavadi uporabimo storitev, ko želimo neki komponenti priskrbeti določene podatke iz strežnika ali pa ko preverjamo vnos uporabnika. Lahko ga uporabimo praktično za vsako kompleksnejšo funckionalnost in na tak način razbremenimo komponente. Omenimo še, da lahko eno storitev uporabi več kompo-

nent in njegova uporaba ni omejena le na eno. Na spodnji sliki (slika 2.4) lahko vidimo primer storitve z dvema javnima metodama.

```
1  import { Injectable } from '@angular/core';
2  import { Http, Response } from '@angular/http';
3  import { Observable } from 'rxjs';
4  import { Project } from './project';
5
6  @Injectable()
7  export class ProjectsService {
8      constructor(private http: Http) {}
9
10     extractData(res: Response) {
11         return res.json();
12     }
13
14     getProjects(): Observable<Project[]> {
15         return this.http.get('/api/projects').map(this.extractData);
16     }
17 }
```

Slika 2.4: Primer storitve

2.2.5 Predloga

Vsaka komponenta ima svoj pogled, ki ga upravlja. Pogled sestavlja HTML datoteka. To ni čisto navadna HTML koda, saj elementi vsebujejo sintaktične ukaze, ki so značilni le za Angular. Ti ukazi pomagajo komponenti, da se odzove na določene situacije, ki jih sproži uporabnik. Tu govorimo predvsem o dogodkih kot sta klik ali sprememba neke vrednosti. Če želimo na nek gumb dodati poslušalca klika in s tem kasneje klicati določeno metodo v komponenti, dodamo na element sintaktični ukaz *(click)*, ki mu sledi ime želene metode. Spodnja slika (slika 2.5) prikazuje del HTML kode, ki vsebuje angularjev značilen sintaktični ukaz.

```
<button
  type="submit"
  class="btn btn-lg btn-block"
  style="background-color: #c62828; color: white"
  (click)="onLogin(username, password)">Login
</button>
```

Slika 2.5: Primer Angular sintaktičnega ukaza (click)

2.3 AngularJS

AngularJS je prva različica že prej omenjenega ogrodja Angular. Na sliki spodaj (slika 2.6) lahko vidimo njegov prepoznavni logotip. Prvič je bil predstavljen leta 2010 in je do današnjega dne postal eden najbolj uporabljenih ogrodij za razvoj spletnih aplikacij. Če primerjamo prvotno različico AngularJS ogrodja in že prej omenjenega ogrodja Angular, so razlike zelo očitne. S kasnejšim razvojem ogrodja AngularJS pa so se razvijalci zelo približali strukturi trenutne različice Angularja in s tem omogočili morebitni lažji prehod na novejšo platformo. Tudi pri ogrodju AngularJS preko značilnih Angular oznak pošiljamo in prejemamo podatke iz pogleda, vendar načeloma ne govorimo o komponentah, temveč o kontrolerjih in direktivah. AngularJS je poznan po vpeljavi dvosmernega prenosa podatkov (*ang. Two-way data binding*). Če neko spremenljivko z določeno vrednostjo prikazujemo na pogledu in se med izvajanjem aplikacije ta vrednost spremeni s strani kontrolerja ali direktive, se bo ta vrednost samodejno osvežila in uporabnik bo lahko na zaslonu videl novo vrednost. Prav tako pa ta funkcionalnost deluje v obratni smeri, torej če bo to spremenljivko spremenil uporabnik, se bo vrednost spremenila v kontrolerju ali direktivi. Vse spremenljivke ali funkcije, ki jih uporablja pogled, so shranjene v spremenljivki scope(*$scope*). Do lastnosti v omenjeni spremenljivki imajo dostop vse metode, ki se uporabljaljo v določenem kontrolerju.

Slika 2.6: Logotip ogrodja AngularJS

2.4 React

React (velikokrat *ReactJS*) je odprtokodna JavaScript knjižnica za izdelavo prednjega dela (*ang. Front end*) spletne aplikacije. React je bil razvit s strani razvijalcev podjetja Facebook leta 2013. Zanj je značilno, da so končne rešitve hitre, do določene mere preproste in razširljive. React knjižnico lahko dodamo tudi k že obstoječemu projektu, ki je napisan v drugačni obliki ali z drugačnim orodjem. Tako kot smo že prej omenili za ogrodje Angular, tudi React uporablja ti. komponentni stil pisanja kode. React se v večini ukvarja le s samimi pogledi. Slika 2.7 nam prikazuje logotip omenjene knjižnice.

Slika 2.7: Logotip knjižnice React

2.4.1 JSX

JSX (JavaScript XML) je sintaktična razširitev JavaScripta. JSX-ov končni produkt so elementi, ki niso ne nizi ne deli HTML kode, ampak nekakšna kombinacija obojega. Primer omenjenega elementa lahko vidimo v sliki spodaj (slika 2.8). Torej neki poljubni spremenljivki lahko pripišemo vrednost, ki vsebuje HTML elemente. Potem pa React poskrbi, da se te spremenljivke ali deli kode preslikajo v DOM. Torej z uporabo jezika JSX dodamo jaziku JavaScript značilno XML sintakso. Za razvoj aplikacije s knjižnico React lahko uporabimo tudi druge jezike kot JSX, vendar uporaba slednjega končno kodo naredi lepšo in elegantnejšo.

```
var Hello = React.createClass({
    render: function() {
        return (
            <div>
                <h1>Hello from Kode</h1>
                <p>Maybe you prefer a hello from Adele</p>
            </div>
            );
    }
});

ReactDOM.render(<Hello/>, document.getElementById('react-target'));
```

Slika 2.8: Primer dekleracije spremenljivke v jeziku JSX

2.4.2 React Native

React Native je knjižnica, ki uporablja React arhitekturo, vendar služi za razvoj domorodnih Android, iOS in Windows Phone mobilnih aplikacij. Pogoj za razvoj mobilnih aplikacij s knjižnico React Native je dobro poznavanje React arhitekture, še posebaj tu pride do izraza poznavanje komponent.

Poglavje 3

Razvoj spletne aplikacije

Poglavje, ki sledi, opisuje izdelavo spletne aplikacije SportPlayer. Predstavitev izdelave aplikacije je razdeljena na opis vsakega pogleda. Pri tem bomo izpostavili dele kode, ki so za to stran ključne. Poleg opisa izdelave aplikacije bo poglavje vključevalo opis povezave z zaledjem (*ang. Back end*). Poglavje bomo zaključili s predstavitvijo ene od dodatnih knjižnic, ki ni prvotno del ogrodja Angular, ampak jo dodamo po potrebi.

3.1 Opis aplikacije

Spletna aplikacija SportPlayer je aplikacija, ki je v prvi vrsti namenjena rekreativnim športnikom. SportPlayer uporabnikom služi kot orodje za dogovor o lokaciji in času, kjer se bo odvijala določena športna aktivnost. Po vpisu v aplikacijo si uporabnik izbere želeni šport. Seveda so vsi športi, ki so na voljo, timski, saj drugače uporabnik ne bi potreboval aplikacije. Omenimo še, da je dodajanje novih športov omogočeno le razvijalcem, ki imajo dostop do podatkovne baze in lahko nad njo izvajajo SQL stavke. Po izbranem športu se glede na uporabnikovo trenutno lokacijo in izbrani radij (prvotna nastavitev je 40km) prikažejo vse športne površine, ki so na razpolago. Aplikacija išče športne površine, ki po zračni razdalji do uporabnika ne

presegajo izbranega radija. Uporabnik lahko radij iskanja spreminja in tako dobi želene športne površine znotraj nastavljene zračne razdalje. Prav tako si lahko pomaga z iskalnikom, ki filtrira športne površine glede na uporabnikov vnešen tekstovni niz. Poleg imena športne površine pa posamezen rezultat vsebuje še informacijo o času in razdalji od lokacije uporabnika do lokacije športne površine, če kot prevozno sredstvo izberemo avto. Vsak uporabnik lahko doda poljubno število športnih površin v radiju največ 90 kilometrov od svoje trenutne lokacije. Aplikacija bo zavrnila vse vnose, ki so preblizu druge športne površine, saj bi lahko v tem primeru šlo za duplikat. Ta razdalja je trenutno nastavljena na 1 kilometer. Ko uporabnik izbere želeno igrišče, se mu prikaže zemljevid z narisano potjo ter vsi urniki, ki so na voljo za izbran dan. Urnik vsebuje informacijo o začetnem in končnem času izvajanja, številu prijavljenih uporabnikov, moči igralcev ter o morebitnem dodatnem komentarju uporabnika, ki je urnik ustvaril. Moč igralcev se izračuna za vsak urnik sproti, in sicer je končna vrednost povprečje vseh ocen prisotnih uporabnikov. Vsak uporabnik dobi oceno 5/10 ob kreaciji svojega računa. Dodajanje urnika je omogočeno za poljuben dan v roku enega tedna, vendar se ta ne sme časovno prekrivati z urnikom, ki že obstaja.

Vsak prijavljen uporabnik lahko svoje nastavitve spreminja na pogledu profil, kjer lahko prav tako vidimo zadnjih 5 urnikov, na katerih smo bili prisotni. Tudi ti urniki vsebujejo imena in oceno vseh ostalih uporabnikov, ki so se udeležili tega urnika. Na tem pogledu nam je omogočeno ocenjevanje teh igralcev. Ocenimo jih lahko z oceno od 1 do 10. Ob kliku na izbrano oceno se shrani novo povprečje ocen. Ocenjevanje samega sebe je onemogočeno. Vsak uporabnik ima na voljo tri bližnjice do priljubljenega športa. Te bližnjice lahko modificira na profilni strani, vidimo pa jih lahko v navigacijski vrstici čez celotno aplikacijo.

3.2 Uporabniške zahteve

Kot večina aplikacij se tudi aplikacija SportPlayer začne na prijavnem oknu. Na začetku torej uporabnik navigira na stran, kjer se mu pojavita dve vnosni polji - eno za vnos uporabniškega imena, drugo pa za geslo. Aplikacija mora podpirati tudi kreiranje novega uporabniškega računa, ta funkcinalnost mora biti lahko dostopna in dobro vidna. Ko uporabnik svoj uporabniški račun ima, lahko vstopi na prvo stran glavnega dela aplikacije. Prva stran aplikacije je zelo pomemben del celotne spletne aplikacije, saj želimo ustvariti zelo dober prvi vtis na uporabnika, ker tako izboljšamo celotno uporabniško izkušnjo. Prav tako mora biti omenjena prva stran aplikacije na pogled preprosta in stilsko dovršena. Vsebinsko mora prva stran aplikacije vsebovati možnost izbire športa. Torej uporabnik si mora za nadaljevanje izbrati šport, s katerim se želi ukvarjati v tistem trenutku. Ko je izbira zaključena, se prikaže stran z izbiro športne površine. Želimo, da ima stran možnost iskanja po imenih športnih površin in da so le-te razvrščene po razdalji od najmanjše do največje. Prav tako mora ta stran vsebovati funkcionalnost, s katero uporabnik lahko nastavi radij, v katerem so prikazani vsi rezultati. Uporabniku želimo na tem pogledu prikazati čim več informacij, da bi bila izbira športne površine za njegovo trenutno lokacijo čim bolj optimalna. Lokacije vseh športnih površin mora aplikacija dobiti s strani uporabnikov. Torej aplikacija SportPlayer potrebuje stran, kjer uporabniki lahko na preprost način dodajo novo lokacijo športne površine. Ta stran bo dostopna s prej omenjene strani, kjer uporabniki lahko vidijo vse športne površine. Ko uporabnik izbere najbolj optimalno športno površino, se mu odpre nova stran, kjer lahko vidi vse urnike za časovni razpon enega tedna. Vsak uporabnik mora imeti pravice za dodajanje svojih urnikov, kjer si lahko nastavi maksimalno število prostih mest. Uporabniku moramo prav tako omogočiti vnos komentarja, ki bo viden vsem uporabnikom, ki bodo pregledovali urnik. Omogočiti moramo, da ko uporabnik dodaja nov urnik, le-tega v primeru časovnega prekrivanja ne more dodati. Ko pride do takšne situacije, uporabnika o tem obvestimo.

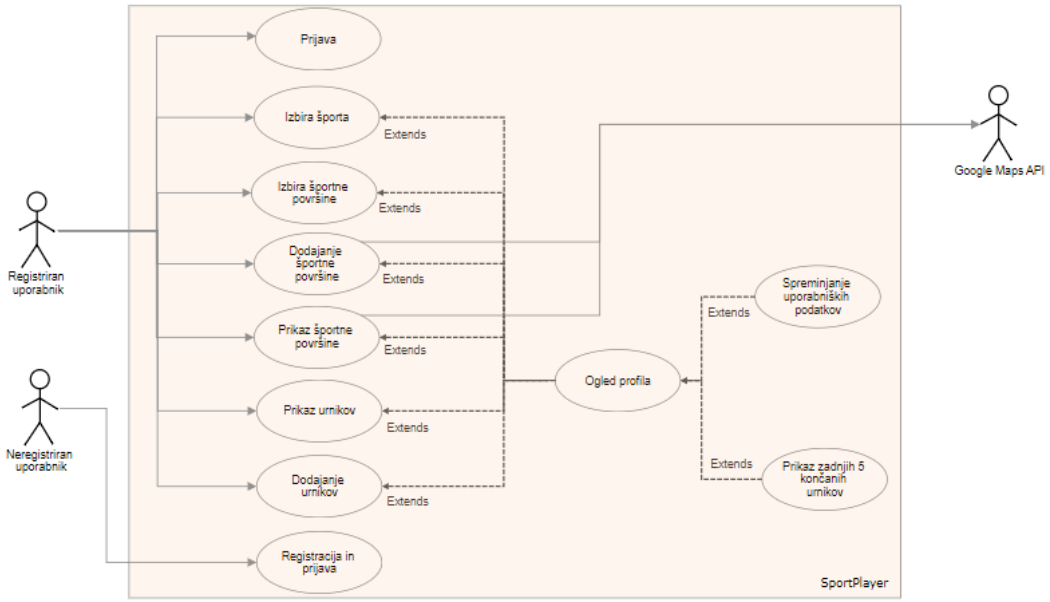
Na koncu moramo poskrbeti še za profilno stran, kjer lahko uporabnik spreminja svoje prvotne nastavitve in kjer mu je omogočena nastavitev bližnjic do priljubljenih športov. Ta pogled naj prav tako vključuje pregled urnikov, ki so že zaključeni.

3.3 Načrtovanje aplikacije

Zelo pomemben del načrtovanja je ocena končne skupine ljudi, ki bodo to aplikacijo dejansko uporabljali. Pri aplikaciji SportPlayer gre za vse ljudi, ki se rekreativno ukvarjajo s športi. To je skupina ljudi, stara med 10 in 50 let. Aplikacija je primerna tudi za ljudi, ki so se s športom šele začeli ukvarjati in se na določenem športnem področju želijo le izboljšati. Takšena oseba lahko za svojo igro izbira le takšne nasprotnike, ki so primerni njegovi ravni znanja. Prav tako lahko zelo dobri posamezniki najdejo sebi primerne nasprotnike, ker s slabšimi posamezniki svoje igre ne morejo izboljšati. Če malce pogledamo v prihodnost, lahko k omenjenemu krogu ljudi prištejemo še profesionalne športnike, ki imajo morda klubski premor ali počitnice. Tudi ti svoj nivo vzdržujejo s rekreativnimi dejavnostmi in bi jim uporaba aplikacije SportPlayer olajšala iskanje primernih posameznikov.

Torej, da bomo lahko v aplikaciji ustregli zgornjim zahtevam, bomo v prvi vrsti potrebovali sistem ocenjevanja igralcev. Ocenjevanje more biti razdelejno na posamezen šport, saj lahko en uporabnik aplikacije SportPlayer uporablja več športov. Ocenjevanje uporabnika je tako omogočeno na nivoju posameznega športa, vsak uporabnik pa lahko dobi podatek o skupni povprečni oceni za vsak šport, ki je na voljo. Posameznega uporabnika lahko ocenjujejo le tisti uporabniki, ki so z njim tekmovali. To bomo na dokaj preprost način lahko dosegli z urniki. Posamezen urnik nam točno opredeli prisotne igralce. In če predpostavljamo, da je bil urnik že zaključen, hitro dobimo primerno mesto, kjer bi se lahko izvedlo ocenjevanje. Uporabnika seveda ocenimo glede na njegovo spretnost pri izvajanju športa, ta informacija

pa je prisotna na posameznem urniku. Vsakemu uporabniku bo omogočeno dodajanje novih urnikov. Poleg prikaza vseh urnikov za določeno športno površino želimo na tem istem pogledu implementirati tudi prikaz uporabnikove trenutne lokacije in razdalje do izbrane lokacije športne površine. Za to bomo potrebovali nekakšen zemljevid. Zemljevid nam nudi spletni vmesnik Google Maps. Ta zemljevid lahko uporabimo na zelo veliko načinov. V omenjeni situaciji bomo uporabili način za prikaz razdalje. Zemljevid Google Maps spletnega vmesnika bomo uporabili še pri funkcionalnosti dodajanja nove športne površine. Tu bomo omenjeni zemljevid nastavili malce drugače. Dodali mu bomo krog z radijem 90 kilometrov, znotraj njega pa bo vsak klik dodal novo športno površino.



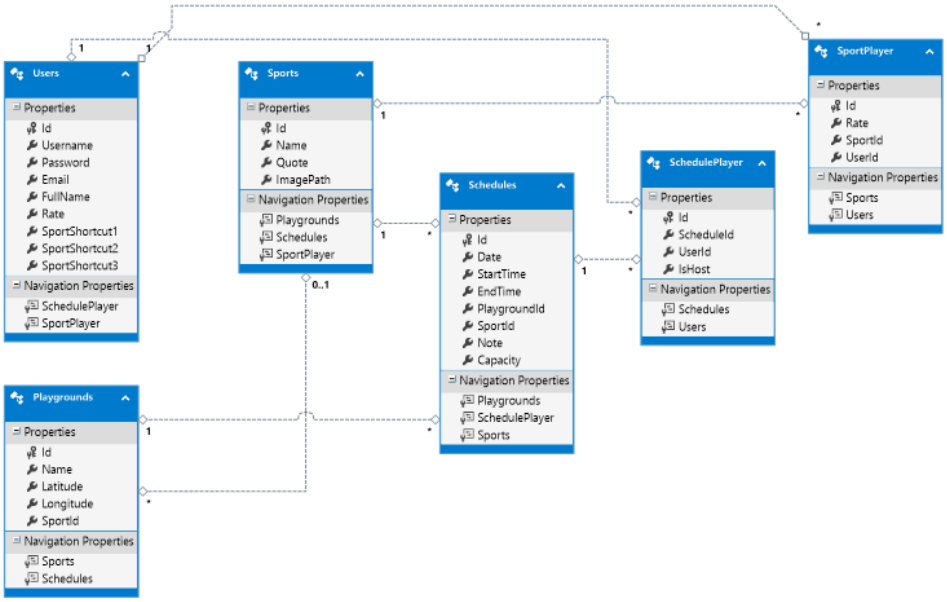
Slika 3.1: Diagram primera uporabe v aplikaciji SportPlayer

Slika 3.1 nam prikazuje posamezne funkcionalnosti v aplikaciji SportPlayer. Kot lahko razberemo, ločimo dve vrsti uporabnikov. Tiste, ki račun že imajo in tiste, ki ga še nimajo. Uporabnikom, ki račun že imajo, je

omogočena prijava v aplikacijo. Uporabniki, ki pa računa še nimajo to storijo z registracijo. Uporaba vseh nadaljnjih funkcionalnosti pa je obema vrstama uporabnikom skupna. V aplikaciji SportPlayer lahko definiramo en zunanji sistem, to je Google Maps API.

3.4 Podatkovni model

Tako kot vse druge spletne aplikacije, tudi aplikacija SportPlayer podatke shranjuje in prejema iz podatkovne baze. Podatkovna baza vsebuje več tabel, ki so med seboj v določeni relaciji. Dve tabeli med seboj povežemo s pomočjo tujih ključev (*ang. Foreign key*). V primeru podatkovne strukture aplikacije SportPlayer gre v vseh primerih za unikatni vrednosti iz obeh povezanih tabel. Podatkovna baza, ki jo uporablja aplikacija SportPlayer, vsebuje 6 tabel. Tabela, ki shranjuje vse podatke o uporabnikih, se imenuje *Users*. Ta ob registraciji novega uporabnika shrani vse podane parametre. Prav tako tabela *Users* vsebuje podatke o nastavljenih bližnjicah uporabnika. Tabela *Sports* vsebuje vse podatke o športih, ki so na voljo. Nove športe uporabniki zaenkrat ne morejo dodajati sami, zato za dodajanje novega športa nad to tabelo izvršimo ustrezen ukaz, ki vsebuje vse potrebne podatke o novem športu. Za potrebe ocenjevanja uporabnika na osnovi posameznega športa, smo dodali novo tabelo *SportPlayer*, ki vsebuje podatke o športu in posameznemu uporabniku. Ta tabela prav tako vsebuje podatke, o trenutni oceni uporabnika. Podatke o vseh športnih površinah vsebuje tabela *Playgrounds*. Kot smo že omenili, vsaki športni površini pripadajo urniki. Ti urniki so shranjeni v tabeli *Schedules*, ki med drugim vsebujejo tudi podatek o športni površini, ki ji urnik pripada. Zadnja tabela se imenuje *SchedulePlayer*. Ta tabela vsebuje vse potrebne podatke o posameznem igralcu, ki se je pridružil določenemu urniku. Slika v nadaljevanju prikazuje logični podatkovni model podatkovne baze (slika 3.2).

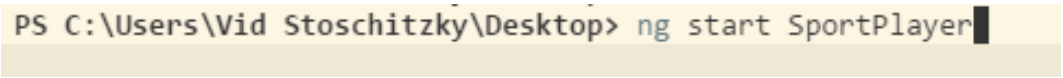


Slika 3.2: Logični podatkovni model v aplikaciji SportPlayer

3.5 Angular CLI

Preden začnemo z Angular projektom, moramo na svoj računalnik namestiti eno zadnjih različic orodij Node.js ter upravljalnika paketov - npm. Ko imamo omenjeni orodji nameščeni, lahko namestimo Angular CLI. Angular CLI je orodje, ki ima zelo pomembno vlogo pri razvoju Angular aplikacij. Angular CLI je vmesnik ukazne vrstice, ki preko preprostih ukazov skrbi za kreiranje novih projektov, gradnjo (*ang. build*) kode, konfiguracijo angular ogrodja, kreiranje komponent in ostalih sestavnih delov Angular ogrodja. Angular CLI namestimo z ukazom *npm install -g @angular/cli*. Opcija *-g* nam Angular CLI namesti globalno, kar pomeni, da lahko do njega dostopamo iz vsake lokacije našega računalnika. Ko s pomočjo orodja npm namestimo Angualr CLI, lahko pričnemo s kreiranjem projekta. Spodnja slika (slika 3.3) prikazuje, kako ustvarimo nov projekt in ga poimenujemo. Novo kreiran projekt vsebuje vse privzete nastavitve, prav tako pa lahko opazimo že

nameščene knjižnice, ki jih Angular potrebuje za svoje delo. Nastalo aplikacijo že lahko poženemo v brskalniku z ukazom *ng serve*.



Slika 3.3: Kreiranje novega angular projekta

3.6 Implementacija spletne aplikacije

Preden začnemo z implementacijo vseh zaslonskih mask in njihovih komponent, moramo poskrbeti za glavni modul ter za usmerjanje aplikacije (*angl. routing*). Želimo, da se strani nalagajo v skladu z URL naslovom. Tovrstno usmerjanje smo nastavili v modulu z imenom *app-routing.module.ts*. V tej datoteki torej za vsak URL naslov nastavimo komponento, ki se bo v tistem trenutku naložila. Prav tako pa lahko pokrijemo vse tiste situacije, kjer URL naslov ne ustreza nobenemu zapisu. V tem primeru se naloži poljubna komponenta. V glavnem modulu aplikacije potem ta usmerjevalni modul kličemo tako, da je le ta dostopen skozi celotno aplikacijo in je odziven na spremembe URL naslova.

```
@NgModule({
    declarations: [ AppComponent ],
    imports: [ routing, BrowserModule, HttpModule, BrowserAnimationsModule, MdNativeDateModule ],
    providers: [
        LoginService,
        SportService,
        PlaygroundService,
        MdSnackBar,
        OVERLAY_PROVIDERS,
        ScrollDispatcher,
        Platform,
        ScrollStrategyOptions,
        LiveAnnouncer,
        ScheduleService,
        AuthGuard
    ],
    bootstrap: [ AppComponent ],
    exports: [ MdDatepickerModule ]
})
export class AppModule {}
```

Slika 3.4: Glavni modul aplikacije

Slika 3.4 prikazuje glavni modul aplikacije SportPlayer. Kot vidimo pod lastnostjo bootstrap, se celotna aplikacija zažene z nalaganjem komponente AppComponent.

3.6.1 Počasno nalaganje

Kot smo že omenili, Angular aplikacijo sestavlja eden ali več modulov. Moduli kasneje deklarirajo eno ali več komponent. Pri razvoju lahko uporabimo dve vrsti nalaganja posameznih komponent. Prva možnost je, da za uporabo pripravimo vse komponenete ob zagonu aplikacije in jih potem poljubno uporabimo. V tem primeru naša aplikacija potrebuje le en modul. Druga možnost pa je zmožnost deklariranja posameznih komponent šele takrat, ko jih potrebujemo. Temu z angleško besedo rečemo *lazy-loading*. Če želimo uporabiti počasno nalaganje, moramo vsaki komponeneti priskrbeti svoj modul, ki naloži to komponento. Implementacija počasnega nalaganja zelo pospeši zagon same aplikacije, saj nalaganje vseh komponent ni potrebno. Lahko pa uporabimo tudi kombinacijo obeh možnosti.

```
const appRoutes: Routes = [
  { path: 'login', loadChildren: './login/login.module#LoginModule' },
  {
    path: 'sportselect',
    loadChildren: './select-sport/select-sport.module#SelectSportModule',
    canActivate: [ AuthGuard ]
  },
  { path: 'sport/:id', loadChildren: './sport/sport.module#SportModule', canActivate: [ AuthGuard ] },
  {
    path: 'sportDetail/:id',
    loadChildren: './sport-detail/sport-detail.module#SportDetailModule',
    canActivate: [ AuthGuard ]
  },
  {
    path: 'addPlayground/:sportId',
    loadChildren: './new-playground/new-playground.module#NewPlaygroundModule',
    canActivate: [ AuthGuard ]
  },
  { path: 'profile', loadChildren: './profile/profile.module#ProfileModule', canActivate: [ AuthGuard ] },
  { path: '', loadChildren: './login/login.module#LoginModule' },
  { path: '**', loadChildren: './login/login.module#LoginModule' }
];
```

Slika 3.5: Modul, ki je namenjen za usmerjanje aplikacije

Slika 3.5 prikazuje usmerjanje aplikacije. Implementiran je način počasnega nalaganja, to dosežemo z lastnostjo *loadChildren*, kjer podamo pot in na koncu poti ime želenega modula. Kot lahko vidimo pri enem od zadnjih elementov v seznamu *appRoutes*, lahko nastavimo komponento za URL naslov, ki se ne ujema z nobenim od zapisov ali pa za URL, ki je prazen.

3.6.2 Prijavna stran

Uporabniki najprej navigirajo na prijavno stran. Uporabniki navadno na isti strani pričakujejo še možnost nove registracije. Obe zahtevi sta realizirani v komponenti *login-componenet.ts*. HTML predloga vsebuje dve skupini, ki se izmenično prikazujeta glede na vrednost spremenljivke za prikaz. Prva skupina je HTML koda, ki vsebuje prijavna vnosa polja, druga pa vsebuje vnosa polja za registracijo novega uporabnika. Kot rečeno, uporabnik s klikom spremninja trenutni prikaz skupine. Ko se uporabnik prijavi ali registrira in prijavi, se izvrši klic na API spletni vmesnik, kjer v odgovoru dobimo potrditev, če je uporabnik vnesel pravilno uporabniško ime in geslo. Če je prijava uspešna, potem se uporabnikov ID shrani v lokalno shrambo, saj ga bomo

kasneje potrebovali za ostale klice na API spletni vmesnik.

```
setUser(user) {
    localStorage.setItem('userId', user.Id);
    this.user = user;
}
```

Slika 3.6: Del kode, kjer shranimo uporabnikov ID v lokalno shrambo

Slika zgoraj (slika 3.6) prikazuje del kode, kjer ob uspešni prijavi shranimo uporabnikovo unikatno številko - ID v lokalno shrambo. Uporabnikovo unikatno številko bomo pogosto potrebovali v nadaljevanju delovanja naše aplikacije. Lokalna shramba se nanaša na shrambo, ki jo ponuja internetni brskalnik. To vsebino lahko uporabnik določenega brskalnika zbriše v vsakem trenutku.

3.6.3 Izbira športne površine

Ko uporabnik izbere želeni šport, ga aplikacija preusmeri na stran, kjer so prikazane vse športne površine. Vso funkcionalnost te strani prevzame komponenta *sport.component.ts*. Ob inicializaciji te komponente se najprej izvede metoda pridobitve trenutne uporabnikove zemljepisne dolžine in širine. Ko dobimo te podatke, lahko izvedemo klic na API spletni vmesnik, kjer poleg zemljepisne širine in dolžine podamo še en parameter, in sicer izbrani radij v kilometrih. API potem vrne razultat z vsemi športnimi površinami znotraj podanega radija. Te podatke potem prikažemo na pogledu s pomočjo Angular sintaktičnega ukaza **ngFor* ,ki iterira čez podan seznam in ustvari za vsak element svoj del HTML kode. Uporabniku je omogočena tudi funkcionalnost iskanja športnih površin po imenu. Tu sem uporabil JavaScript metodo *.filter*, s katero vrnem vse razultate, ki v imenu vsebujejo dani niz. Ime in iskani niz pretvorim v male črke, saj s tem uporabniku ni potrebno paziti na morebitne velike začetnice v imenu določene športne površine.

```
private setPosition(position) {
    this.location = position.coords;
    const apiData = {
        currentUsersLatitude: position.coords.latitude,
        currentUsersLongitude: position.coords.longitude,
        distance: this.distance,
        sportId: this.sportId
    };
    this.playgroundService.getPlaygrounds(apiData).then(
        (response) => {
            this.playgrounds = response;
            this.getPlaygrounds(this.searchString);
        },
        (error) => {
            console.error('Could not recive playgrounds');
        }
    );
}
```

Slika 3.7: Del kode, s katero dobimo trenutni zemljepisni položaj uporabnika

Na sliki 3.7 lahko vidimo klic metode *getPlaygrounds*, ki jo implementira storitev *playgroundService*. Preden izvedemo klic omenjene metode, pa smo uspešno pridobili podatke o lokaciji uporabnika.

3.6.4 Dodajanje nove športne površine

Vse lokacije športnih površin dobimo z vnosom uporabnikov. Na strani, kjer izbiramo športne površine, smo dodali gumb, ki nas preusmeri na stran za dodajanje novih. Za ta pogled smo dodali novo komponento z imenom *NewPlaygroundComponent*. Ob inicializaciji te komponente zopet pridobimo uporabnikovo zemljepisno širino in dolžino. Ko dobimo te podatke, s pomočjo spletnega vmesnika Google Maps JavaScript API na pogledu izrišemo zemljevid, ki je centriran na prej pridobljeno zemljepisno lokacijo. Poleg izrisa in

centriranja dodamo še rdeči krog z radijem 90 kilometrov, ki služi kot grafično opozorilo uporabniku, do kam mu je omogočen klik in posledično dodajanje nove športne površine.

```
const cityCircle = new google.maps.Circle({
    strokeColor: '#c62828',
    strokeOpacity: 0.8,
    strokeWeight: 2,
    fillColor: '#c62828',
    fillOpacity: 0.15,
    map: map,
    center: { lat: this.location.latitude, lng: this.location.longitude },
    radius: 90 * 1000
});
cityCircle.addListener('click', (event) => {
    const playground: AddPlayground = {};
    playground.latitude = event.latLng.lat();
    playground.longitude = event.latLng.lng();
    playground.name = '';
    playground.disabled = false;
    this.playgroundList.push(playground);
    this._detector.detectChanges();
    addMarker(event, map, this.playgroundList.length);
});
```

Slika 3.8: Del kode, ki zazna klik uporabnika na zemljevidu

Ko uporabnik klikne na poljubno lokacijo v središču rdečega kroga, želimo, da se izvede metoda, ki kot parameter dobi zemlejepisno širino in dolžino izbrane lokacije. To storimo tako, da krogu z radijem 90 kilometrov dodamo poslušalca (*ang. event listener*), ki zazna vsak klik in izvede želeno metodo ali del kode (slika 3.8). Ko uporabnik klikne na poljubno površino znotraj kroga, dodamo pridobljene podatke v seznam, ki ga zopet prikažemo na pogledu, saj moramo od uporabnika dobiti še poljubno ime te športne površine. Ko uporabnik izbere še ime, se izvrši klic na API spletni vmesnik. Dodana športna površina pa je lahko s strani API-ja tudi zavrnjena, saj je lahko duplikat ali pa je preblizu že obstoječi športni površini. V tem primeru ali pa v primeru uspešne dodaje, pa uporabnika obvestimo s grafičnim prikazom

(*ang. Toast*).

3.6.5 Stran za prikaz športne površine

Ko uporabnik izbere želeno športno površino, ga aplikacija preusmeri na stran s HTML predlogo, ki jo upravlja komponenta *sport-detail.component.ts*. Ob inicializaciji te komponente zopet pridobimo trenutno uporabnikovo lokacijo na enak način, kot smo jo že v prejšnjih dveh komponenetah. Uporabnikovo lokacijo potrebujemo, saj želimo na zemljevidu prikazati pot od uporabnikove lokacije do lokacije izbrane površine. Zopet si pri tem pomagamo z Google Maps zemljevidom, natančneje s *directionServicom*, ki lahko kot parameter prejme zemljepisne podatke začetne točke izrisa ter podatke o lokaciji končne točke. Podamo pa mu lahko tudi način poti. V primeru aplikacije SportPlayer, je to način vožnje. Kot zanimivost lahko omenimo, da kot način poti lahko nastavimo tudi hojo, kolesarjenje in prevoz z javnimi prevoznimi sredstvi. Med pridobivanjem lokacije pa izvedemo še en klic na API spletni vmesnik, ki nam vrne vse urnike (slika 3.9), ki obstajajo za trenutno športno površino in za trenutno izbrani dan. Te podatke potem prikažemo na zaslonu. Uporabnik lahko spreminja izbran dan, omogočen mu je izbor do največ 7 dni vnaprej. Hkrati mu je dovoljeno dodajanje novega urnika, pri katerem pa lahko specificira, koliko je največje število dovoljenih igralcev, minimalno število pa je privzeto na vsaj 2 igralca. Pri dodajanju urnika od uporabnika zahtevamo začetni in končni čas. Za izbiro datuma in časa smo vzeli komponento iz knjižnice PrimeNG, kateri lahko podamo parameter, za katere dneve želimo izbor onemogočiti.

```
getSchedules() {
    this.scheduleService.getSchedules(this.scheduleDate, this.playground.Id).then(
        (result) => {
            this.schedules = result;
        },
        (error) => {
            console.log('Cannot receive schedules from server!');
        }
    );
}
```

Slika 3.9: Klic na API spletni vmesnik, ki vrne vse urnike

3.6.6 Profilna stran

Kot vsaka današnja spletna ali mobilna aplikacija, ima tudi apliakcija SportPlayer svojo profilno stran, do katere lahko pride vsak uporabnik, ki je uspešno vpisan. Uspešno vpisanemu uporabniku se nato pojavi gumb v navigacijski vrstici, ki ga lahko klikne na poljubnem pogledu čez celotno aplikacijo. Ob kliku na omenjeni gumb se izvršijo trije klici na strežnik. Prvi klic vrne vse podatke o vpisanem uporabniku, saj jih bomo potrebovali v primeru, če jih uporabnik želi spremeniti. Drugi klic vrne vse športe, ki jih bomo uporabili pri nastavitvah uporabnikovih bližnjic, ki so dostopne iz navigacijske vrstice. Zadnji klic pa vrne zadnjih 5 že zaključenih urnikov, na katerih je bil prisoten trenutno vpisan uporabnik. Ta funkcionalnost je namenjena predvsem ocenjevanju ostalih pristonih igralcev na prikazanih urnikih. Ko uporabnik klikne na številko, ki jo bo podal kot oceno za posamezenega igralca, se izvrši klic na strežnik, ta izračuna povprečje ocen in ga zaokroži na celo število. Dobljeno vrednost potem zapiše v podatkovno bazo.

3.6.7 Povezava z zaledjem

Da bi aplikacija SportPlayer delovala optimalno, mora prejemati podatke iz podatkovne baze in shranjevati podatke v podatkovno bazo. To smo

omogočili z implementacijo spletnega vmesnika - API. Spletni vmesnik vsebuje 4 končne točke, ki jih bomo potrebovali za klice iz spletne aplikacije in uporablja arhitekturni stil REST. Če želimo izvesti klic na REST spletni vmesnik(*lahko tudi RESTful spletni vmesnik*), moramo pri klicu uporabiti eno od HTTP metod GET, POST, PUT, DELETE, PATCH... Vsaka od omenjenih končnih točk v spletnem vmesniku ima večje število javnih metod, vsaka od njih uporablja eno od HTTP metod. Vsaka javna metoda določene končne točke spletnega vmesnika se povezuje s podatkovno bazo, v njo zapisuje ali bere, kot odgovor pa poda HTTP statusno kodo in morebitne dodatne parametre, ki se kasneje pretvorijo v JSON obliko.

```
@Injectable()
export class ScheduleService {
    private scheduleApiUrl = 'http://localhost:49243/api/schedules';

    constructor(private http: Http, private router: Router) {}

    private getOptions() {
        const headers = new Headers({ 'Content-Type': 'application/json' });
        const options = new RequestOptions({ headers: headers });
        return options;
    }

    public getSchedules(date: Date, playgroundId: Number): Promise<any> {
        const options = this.getOptions();
        return this.http
            .get(this.scheduleApiUrl + '?date=' + date.toDateString() + '&playgroundId=' + playgroundId, options)
            .toPromise()
            .then((response) => response.json())
            .catch();
    }
```

Slika 3.10: Storitev ki skrbi za urnike

V spletni aplikaciji smo implementirali storitve za prejem vseh podatkov. Vsaki končni točki spletnega vmesnika pripada ena storitev s svojo lastnostjo, katere vrednost je naslov do končne točke spletnega vmesnika. Če se želimo povezati z zaledjem, pa bomo potrebovali knjižnico HTTP, ki nam jo ponuja Angular. Ta knjižnica podpira vse prej omenjene HTTP metode, ki jih za komunikacijo uporablja REST.

V zgornjem primeru (slika 3.10) lahko vidimo del storitve *PlaygroundService*, ki skrbi za celotno funkcionalnost, ki se dotika športnih površin. Metoda

getPlaygrounds prejme en parameter, ki smo ga poimenovali *data*. Ta prejeti objekt želimo poslati spletnemu vmesniku, saj bomo glede na te vrednosti lahko vrnili ustrezen odgovor. Uporabili smo metodo POST, objekt z imenom *data* pa smo pretvorili v JSON obliko, saj ga drugače HTTP knjižnica ne bo poslala spletnemu vmesniku. Z metodo *.then* povemo, kaj naj se zgodi, ko dobimo odgovor s strežnika. V našem primeru želimo vrnjeno vrednost pretvoriti iz JSON oblike v objektno obliko. Pretvorjeni objekt potem vrnemo.

```
this.playgroundService.getPlaygrounds(apiData).then(
    (response) => {
        this.playgrounds = response;
        this.getPlaygrounds(this.searchString);
    },
    (error) => {
        console.error('Could not recive playgrounds');
    }
);
```

Slika 3.11: Klic metode getPlaygrounds v komponenti

Vse metode, ki so napisane v storitvah, potem kličemo iz posameznih komponent. Za dostop do njih moramo najprej uvoziti storitev, ki jo potrebujemo. Ko to storimo, lahko kličemo vse javne metode te storitve, kot je pokazano na primeru zgoraj (slika 3.9). Metoda *getPlaygrounds* (in vse ostale javne metode storitev) vrne obljubo (*ang. Promise*). Obljube služijo kot zelo dobro orodje za asinhrono delovanje spletne aplikacije. Potem, ko izvedemo klic na spletni vmesnik, ne vemo, koliko časa bo trajalo procesiranje te zahteve. Ko je odgovor poslan s strani strežnika, si pomagamo z metodo *.then*, ki vrne in izvrši naslednji del kode, ko je odgovor prejet. V primeru nastale napake pri vračanju odgovora ali pa če je HTTP statusna koda odgovora

takšna, da nakazuje napako, se izvrši del kode, ki v konzulo brskalnika izpiše poljubno besedilo. Morebitno napako v kodi storitve ujamemo z metodo *.catch*.

3.7 Uporaba zbirke PrimeNG

V sklopu izdelave spletne aplikacije smo uporabili nekaj dodatnih knjižnic. Ena od njih je zbirka PrimeNG, katere značilen logotip lahko vidimo na sliki spodaj (slika 3.12). PrimeNG je zbirka, ki vsebuje več kot 70 komponent, ki služijo za oblikovanje izgleda aplikacije. Za potrebe aplikacije SportPlayer smo uporabili komponento Rating, ki služi za vnos vrednosti, izbrane s klikom na poljubno število zvezdic. Komponento uporabljamo za prikaz ocene posamezenega uporabnika in prav tako tudi za vnos ocene, to funkcionalnost najdemo na profilni strani naše aplikacije.

Slika 3.12: Logotip zbrike PrimeNG

Poleg komponente za ocenjevanje pa smo v naši aplikaciji uporabili še komponento Tooltip (slika 3.13). Tooltip nam omogoča, da v primeru, če uporabnik miško pozicionira na želeni HTML element, prikažemo določeno tekstovno vsebino. To komponento smo zopet uporabili na profilni strani, in sicer, ko uporabnik z miško preleti HTML element za ocenjevanje posamezenega igralca.

```
e" (onRate)="ratePlayer(player)" pTooltip="Rate player" tooltipPosition="top"></p-rating>
```

Slika 3.13: Uporaba komponente Tooltip v HTML kodi

Še zadnja uporabljena stilistična komponenta pa je komponenta, ki služi kot vnosno polje za izbiro datuma. Ta komponenta nam ponuja ogromno možnosti, s katerimi lahko nastavimo delovanje te komponente. Če kot primer vzamemo vnosni polji na strani za prikaz športne površine, lahko omenimo opcije *minDate, maxDate* in *hourFormat*. Prvi dve sta namenjeni temu, da nastavimo najzgodnejši oziroma najkasnješi možni datum za izbiro. Obe kot paramter sprejmeta objekt tipa *Date*. Opcija *hourFormat* pa služi kot nastavitev časovnega formata. V aplikaciji je uporabljen 24-urni časovni format. Primer preproste nastavitve lahko vidimo na sliki spodaj (slika 3.14)

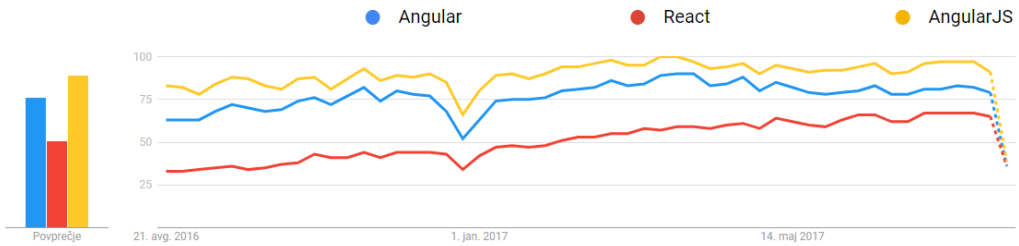
```
<p-calendar [(ngModel)]="newSchedule.To"
            showTime="showTime"
            hourFormat="24"
            [minDate]="minDate"
            [maxDate]="maxDate">
</p-calendar>
```

Slika 3.14: Uporaba komponente Calendar v HTML kodi

Poglavje 4

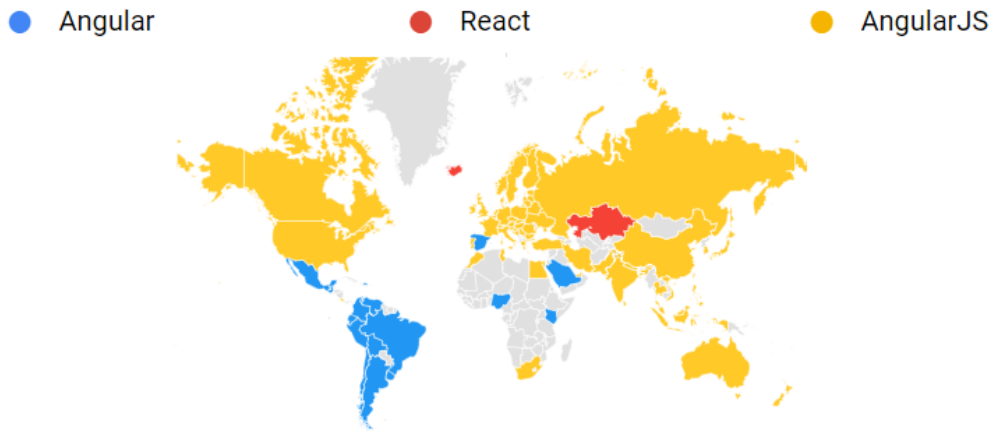
Primerjava Angular ogrodja z ostalimi

V tem poglavju bomo primerjali ogrodje Angular z ogrodjem AngularJS in knjižnico React. Obe ogrodji Angular (Angular in AngularJS) in knjižnica React so najbolj priljubljene za razvoj spletnih aplikacij. Da je povpraševanje po njih zelo veliko, lahko razberemo s spodnje slike (slika 4.1). Slika je zajeta s strani Google Trends, prikazuje pa nam število iskalnih nizov v Googlovih storitvah v časovnem obdobju enega leta. Kot lahko razberemo, je v prednosti ogrodje AngularJS, saj je na trgu najdlje in ima tudi največje število izdelanih spletnih aplikacij.



Slika 4.1: Priljubljenost izrazov React, Angular in AngularJS *Vir: Google Trends*

Spodnja slika prikazuje zanimanje za ogrodji AngularJS in Angular ter knjižnico React po območjih sveta (slika 4.2). Kot vidimo je še vedno najbolj priljubljen izraz AngularJS. Zanimivo je območje Južne Amerike, kjer v celoti prevladuje novejše ogrodje Angular.



Slika 4.2: Priljubljenost knjižnice React, ogrodja Angular in ogrodja AngularJS po omočjih sveta *Vir: Google Trends*

4.1 Primerjava ogrodja Angular z ogrodjem AngularJS

Ogrodje Angular je izšlo kot čisto prenovljeno ogrodje in ni nadaljevanje ogrodja AngularJS. Prva večja in morda največja sprememba je gotovo uporaba jezika Typescript. Razvijalci, ki poznajo ali obvladajo jezika, kot sta Java in .NET, praktično brez težav osvojijo vsa potrebna znanja za obvladovanje jezika Typescript.

Angular v celoti temelji na pisanju komponent, medtem ko pri ogrodju AngularJS večino časa uporabljamo kontrolerje (*ang. controllers*). Kontroler se preko direktive *ng-controller* poveže s pogledom in kreira nov primerek

spremenljivke *$scope*. Prav tako pri novejšem od ogrodij Angular ni več potrebe po omenjeni spremenljivki *$scope*, saj so jo nadomestile spremenljivke v komponentah ali direktivah. Spremenljivka *$scope* je v ogrodju AngularJS služila kot posrednik med pogledom in posameznim kontrolerjem.

Z novejšo različico ogrodja Angular smo dobili tudi bolj dodelano funkcionalnost vstavljanja odvisnosti (*ang. dependency injection*). Vstavljanje odvisnosti pri novejšem Angular ogrodju izvedemo tako, da kličemo želeno storitev ali knjižnico kar v konstruktorju komponente (slika 4.3). Pri ogrodju AngularJS pa je potrebno želeno odvisnost klicati v funkciji kontrolerja.

```
constructor(
    private loginService: LoginService,
    private sportService: SportService,
    private detector: ChangeDetectorRef,
    private snackBar: MdSnackBar,
    private appComp: AppComponent,
    private scheduleService: ScheduleService
) {
```

Slika 4.3: Primer vstavljanja odvisnosti - Angular

Spremenjeno je tudi začetno nalaganje aplikacije (*ang. bootstrapping*). Pri ogrodju AngularJS smo začetni modul dobili preko direktive *ng-app*, medtem ko pa pri ogrodju Angular v datoteki *main.ts* uporabimo ukaz *platformBrowserDynamic().bootstrapModule()*, ki naloži želeni modul. Naloženi modul kasneje deklarira komponento, katere HTML predlogo najprej prikažemo.

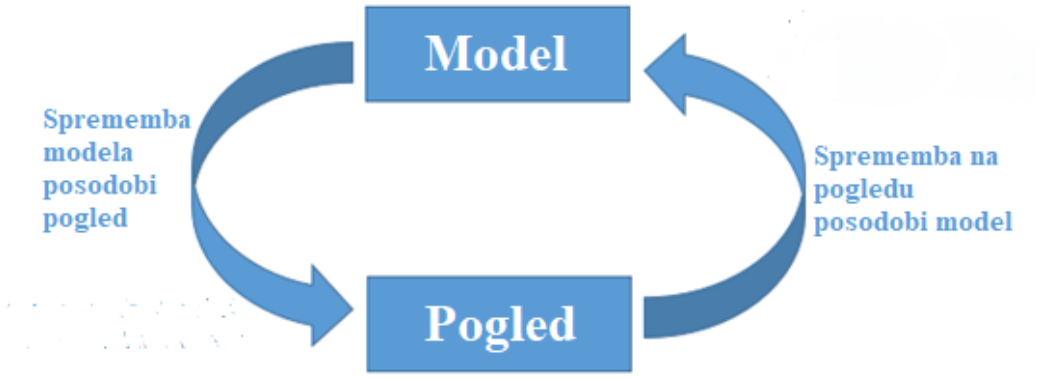
Če se odločimo za razvoj nove spletne aplikacije in izbiramo med ogrodjem AngularJS in Angular, bi seveda vzeli zadnjo stabilno različico ogrodja Angular. Vendar tudi izbira ogrodja AngularJS še zdaleč ni slaba. Dandanes obstaja veliko spletnih aplikacij, ki uporabljajo ogrodje AngularJS, kar kaže, da je to ogrodje še vedno zelo pogosta in dobra izbira. Število spletnih

aplikacij, ki uporabljajo AngularJS je celo večje od tistih, ki uporabljajo Angular, kar je odvisno tudi od leta izdaje. Če pa imamo že delujočo rešitev, ki uporablja ogrodje AngularJS, nimamo skrbi, da bi to ogrodje zastarelo, saj še vedno ustreza vsem zahtevam. Obstajajo tudi zelo dobri načini [20], kako napisano kodo spremenimo, da bo le-ta bolj podobna arhitekturi ogrodja Angular in s tem poenostavimo in pospešimo morebitno migracijo.

4.2 Primerjava ogrodja Angular s knjižnico React

Knjižnica React in ogrodje Angular sta trenutno dve najbolj popularni izbiri za razvoj spletnih aplikacij. Kot smo že omenili, je React razvilo podjetje Facebook, Angular pa podjetje Google. V obeh primerih gre seveda za zelo dobri izbiri za izgradnjo želene spletne aplikacije. Angular predstavlja celotno ogrodje za izgradnjo spletnih aplikacij, medtem ko je React knjižnica, ki je pretežno namenjena le pogledom.

Morda največjo razliko med ogrodjem Angular in knjižnico React opazimo pri delu s podatki med pogledom in komponento. Angular ima zmožnost dvosmernega dela s podatki (*ang. 2 way data binding*), torej ko se posodobi spremenljivka v komponenti, se tudi na pogledu in obratno (slika 4.4). Pri ogrodju Angular takšno namero nakažemo z direktivo *[(ngModel)]*. Na drugi strani pa React pozna le enosmerno delo s podatki. Torej ali spremenimo vrednost v komponenti, in to s pomočjo opazovalca prikažemo ali pa storimo obratno s pomočjo dogodka. V primeru aplikacije SportPlayer, je zato ogrodje Angular boljša izbira, saj praktično na vsakem pogledu uporabljamo dvosmerno komunikacijo. Na ta način je kode bistveno manj, kot pa če bi uporabili načine knjižnice React.



Slika 4.4: Dvosmerna podatkovna komunikacija - Angular

Veliko podobnosti med ogrodjem Angular in knjižnico React lahko najdemo pri zagonu končne aplikacije in navigaciji med posameznimi stranmi. Edina razlika je, da Angular prvotno naloži Modul, v primeru knjižnice React pa gre to za komponente (slika 4.5). Pristop ogrodja Angular je zelo priročen, posebno, če želimo v aplikaciji uporabiti razrede, ki jih definiramo le enkrat (*ang. singeltons*).

```
ReactDOM.render(
  <Provider {...rootStores} >
    <Router history={routerStore.history} >
      <App>
        <Switch>
          <Route exact path='/home' component={Home as any} />
          <Route exact path='/posts' component={Posts as any} />
          <Route exact path='/form' component={Form as any} />
          <Redirect from='/' to='/home' />
        </Switch>
      </App>
    </Router>
  </Provider >,
  document.getElementById('root')
)
```

Slika 4.5: Primer usmerjanja v aplikaciji, ki uporablja React

Pri razvoju določene aplikacije velikokrat ne pomislimo na kasnejšo razširljivost. Če se nam zgodi, da moramo kasneje aplikacijio razširiti, kar je pogost korak, potem lahko prednost pripišemo ogrodju Angular in njegovemu Angular CLI-ju. Prednost na področju dodajanja novih knjižnic in časa razvoja pa prav gotovo pripada knjižnici React. Omenimo še velikost končne produkcijske kode. Ta je v primeru ogrodja Angular lahko večja tudi do štirikrat, če govorimo o razvoju aplikacije, ki ima iste funkcionalnosti kot aplikacija, razvita s knjižnico React. Do tega pride zato, ker Angular uporablja bistveno več knjižnic.

Če pogledamo skupni imenovalec ugotovimo, da sta obe izbiri logični. Cikel razvoja aplikacije s knjižnico React je lahko zelo podoben ciklu razvoja s ogrodjem Angular, če pri tem uporabimo ustrezne knjižnice. Če se želimo od začetka naučiti uporabljati katerokoli od omenjenih izbir, potem nam bo največ časa prav gotovo vzelo učenje vseh funkcionalnosti ogrodja Angular, vendar bomo zraven osvojili še veliko drugih znanj. Ogrodje Angular je

primernejše za večja podjetja, saj si tovrstna podjetja lahko privoščijo nekaj več časa za učenje vseh funkcionalnosti. Če pa kot razvijalec iščemo nekaj bolj preprostega, pa je bolj smiselno uporabiti knjižnico React.

4.3 Primerjava orodij po kriterijih

To podpoglavje vsebuje primerjavo ogrodja AngularJS, ogrodja Angular in knjižnice React po določenih kriterijih. V nadaljevanju so predstavljeni vsi ti kriteriji.

4.3.1 Podprti programski jezik

Vsako od primerjanih ogrodij oz. knjižnice za programiranje uporablja drugačen programski jezik. AngularJS uporablja programski jezik JavaScript, Angular uporablja TypeScript, pri knjižnici React pa govorimo o jeziku JSX (JavaScript XML). Tako TypeScript kot JSX, sta osnovana na programskem jeziku JavaScript.

4.3.2 Težavnostna stopnja

Ogrodji AngularJS in Angular sta veliko obsežnejši kot React. Temu primeren je tudi podatek o stopnji težavnosti osvojitve vseh znanj, ki jih zajemata obe ogrodji Angular (AngularJS in Angular). Pri slednjih dveh nam učenje delovanja in vseh funkcionalnosti vzame bistveno več časa kot učenje celotne funkcionalnosti knjižnice React. To dejstvo dokazuje več študij [4, 8, 9].

4.3.3 Povezava s HTML predlogo

Povezava s HTML predlogo (*ang. Binding*) je področje, na katerem se knjižnica React najbolj razlikuje od obeh ogordij Angular in AngularJS. Angular in AngularJS se ponašata z dvosmerno komunikacijo, medtem ko gre pri knjižnici React za enosmerno komunikacijo.

4.3.4 Arhitekturni stil

Vsako od primerjanih orodij ima svoj malce drugačen arhitekturni stil. Pri ogrodju AngularJS govorimo o arhitekturi MVC (Model–view–controller). MVC vsebuje pojme, kot so: modeli, pogledi in kontrolerji. Vsi ti pojmi so med seboj v določeni povezavi. Ogrodje Angular je, kot smo že omenili, sestavljeno iz različnih komponent. Arhitekturo knjižnice React si lahko predstavljamo kot arhitekturo MVC, le da uporabi le del arhitekture, ki je namenjena pogledom (*ang. View*).

4.3.5 Mesto preslikave kode

Ogrodje Angular in knjižnica React svojo kodo preslikata (*ang. Rendering*) na strani svojega strežnika, medtem ko ogrodje AngularJS to stori na strani odjemalca. Prva možnost (preslikava na strani strežnika) se izkaže kot boljša, saj je posledično aplikacija hitrejša.

4.3.6 Vzdrževanje kode

Pri ogrodju Angular je vzdrževanje kode dokaj preprosto, če sledimo smernicam, ki nam jih predpisujejo razvijalci tega ogrodja. Smernice v večini primerov narekujejo korake, s katerimi razvijalec kodo stilira na tak način, da je kasnejše spreminjanje ali dopolnjevanje lažje. Te smernice pridejo še posebaj prav, ko naša aplikacija postane malce bolj obširna. Če omenimo še komponentni stil pisanja pri ogrodju Angular, lahko ugotovimo, da je vzdrežvanje dokaj preprosto. Podobna situacija je pri ogrodju AngularJS. Pomembno je, da novo napisano kodo v ogrodju AngularJS napišemo na tak način, da je ta podobna kodi, ki jo uporabljamo pri ogrodju Angular. Tak slog nenazadnje narekujejo tudi smernice, ki jih priporočajo pri ogrodju AngularJS. Če pišemo kodo na tak način, je tudi morebitna nadgraditev ogrodja na najnovejšo verzijo ogrodja Angular bistveno lažja. Koda knjižnice React je od naštetih najbolj čista in strukturirana, zato je preprosta za vzdrževanje. To zagotavlja arhitektura Flux.

4.3.7 Razvoj mobilnih aplikacij

Pri obeh ogrodjih Angular (AngularJS in Angular) in knjižnici React lahko napisane dele kode v veliki večini uporabimo pri morebitnem razvoju mobilne aplikacije. Pri ogrodjih AngularJS in Angular za to poskrbi ogrodje Ionic. Pri knjižnici React je temu namenjena knjižnica React Native.

4.3.8 Primerjalna tabela

Tabela (slika 4.6) vsebuje vse opisane kriterije. Zraven je podana še ocena za posamezni primerjalni objekt. Namen tabele je prikaz določenih prednosti posamezenga orodja na določenemu področju. Na tak način se lahko razvijalec odloči, katero ogrodje je bolj primerno za razvoj njegove spletne rešitve.

Kriteriji	**Angular**	**AngularJS**	**React**
Podprti programski jezik	TypeScript	JavaScript	JSX
Težavnostna stopnja	Srednja	Težka	Lahka
Povezava s HTML predlogo	Dvosmerna	Dvosmerna	Enosmerna
Arhitekturni stil	Komponentni stil	MVC	Samo pogledi
Mesto preslikave kode	Strežnik	Odjemalec	Strežnik
Vzdrževanje kode	Srednje preprosto	Srednje preprosto	Zelo preprosto
Razvoj mobilnih aplikacij	Ogrodje Ionic	Ogrodje Ionic	React Native

Slika 4.6: Primerjava po kriterijih

4.4 Ugotovitve

V tem poglavju smo spoznali ogrodje AngularJS, ogrodje Angular in knjižnico React še malce bolj podrobno. Predstavljeni so bili določeni kriteriji, po katerih lahko vsa tri našteta orodja razlikujemo med seboj. Rezultati so

bili tudi predstavljeni s tabelo, kjer smo za vsako od naštetih orodij podali oceno za določen kriterij. Ugotovili smo, da so ogrodje AngularJS, ogrodje Angular in knjižnica React zelo kakovostna izbira za razvoj spletnih aplikacij. V prednosti sta zagotovo ogrodje Angular in knjižnica React, saj sta novejša in se zelo hitro razvijata. Ogrodje AngularJS se sicer ne razvija več, vendar ga uporablja še vedno veliko število uporabnikov.

Poglavje 5

Zaključek

V diplomski nalogi smo spoznali ogrodje AngularJS in njegovo naslednjo različico, katere ime je poenoteno v Angular. Na koncu pa še knjižnico React, ki jo z dodatnimi knjižnicami lahko spremenimo v ogrodje, podobno Angularju.

Naslednje poglavje je bilo namenjeno predstavitvi poteka razvoja aplikacije SportPlayer. Pokrili smo vsa področja od opisa razvoja vsake strani oziroma komponente aplikacije posebej, do povezave spletne aplikacije z zaledjem, ki je zelo pomememben del. Omenili smo tudi, kako smo delo načrtovali in na kaj smo morali biti posebej pozorni. Za konec poglavja smo pogledali še dodatno uporabljeno knjižnico, ki nam omogoča raznovrstne komponente za stilistično podobo končne rešitve. Predvidevamo, da je oblikovni izgled zelo pomembnen, saj želimo, da ima uporabnik dober prvi vtis, s tem si tudi zelo povečamo možnost ponovne uporabe.

Zadnje poglavje je v prvi vrsti namenjeno razvijalcem, ki že poznajo ogrodje Angular in se sprašujejo, kako lahko izbira drugega ogrodja nadgradi boljši končni produk. Tu smo predstavljeno ogrodje in knjižnico iz prvega poglavja primerjali z ogrodjem, s katerim smo razvijali aplikacijo SportPlayer. Spoznali smo, da ima izbira vsakega izmed teh ogrodij ali knjižnice svoje prednosti ter področja, na katerem lahko drugo ogrodje bolje reši problem.

5.1 Izboljšave aplikacije SportPlayer

Vsaka aplikacija ima seveda veliko možnosti za nadgradnjo ali dodelavo. Aplikacijo bi lahko nadgradili s funkcionalnostmi, ki so predstavljene v nadaljevanju.

5.1.1 Pregled igralcev

Trenutno imamo v aplikaciji SportPlayer implementiran pregled prisotnih igralcev le za 5 preteklih urnikov. Seveda želimo uporabnikom ponuditi pregled vseh možnih igralcev in pri tem pokazati igralčevo oceno. Potrebovali bomo torej novo komponento, njen pogled pa bo moral vsebovati pameten prikaz vseh možnih igralcev, saj če bo le-teh veliko, lahko ta stran postane zelo počasna. Potrebna bi bila torej implementacija neskončnega nalaganja, s tem bi s API spletnega vmesnika dobivali omejeno število igralcev, seznam pa bi se polnil, ko bi se uporabnik premikal nižje po spletni strani.

5.1.2 Povabi igralca

Ko posamezen uporabnik doda nov urnik za določeno športno površino, se lahko mesta za ta urnik zelo hitro napolnijo. Uporabniku želimo omogočiti zmožnost povabila želenega igralca na ta urnik. Ta uporabnik bi potem prejel sporočilo o tem po elektronski pošti ali kakšno drugačno obliko obvestila.

5.1.3 Najmanjša možna ocena urnika

Ocene uporabnikov aplikacije se bodo za posamezen šport zelo razlikovale. Uporabnikom bi ob dodajanju novega urnika omogočili, da specificirajo najmanjšo možno oceno, s katero bi se uporabniki še lahko prijavili na novo dodan urnik. Ta funkcionalnost bi omogočila bolj izenačene urnike.

5.1.4 Povezava s socialnimi omrežji

V mislih imamo predvsem povezavo s Facebookom. Uporabnikom bi želeli omogočiti prijavo s Facebookom, prav tako pa bi lahko Facebook uporabili za posamezen urnik. Uporabniku, ki je kreiral nov urnik, bi bila omogočena kreacija novega skupnega pogovora, ki bi potekal na Facebooku. Na ta način bi se uporabniki dogovorili o določenih podrobnostih urnika.

Literatura

[1] Angular 2 vs React. What to chose in 2017? Dosegljivo: http://blog.techmagic.co/angular-2-vs-react-what-to-chose-in-2017/ . [Dostopano: 20. 08. 2017].

[2] Angular. Dosegljivo: https://angular.io/. [Dostopano: 20. 08. 2017].

[3] Learn How to Develop the Next Generation of Applications for the Web. Dosegljivo: https://developers.google.com/web/. [Dostopano: 21. 08. 2017].

[4] REACTJS VS ANGULAR COMPARISON: WHICH IS BETTER? Dosegljivo: https://da-14.com/blog/reactjs-vs-angular-comparison-which-better . [Dostopano: 21. 08. 2017].

[5] React vs Angular: An In-depth Comparison. Dosegljivo: https://www.sitepoint.com/react-vs-angular/ . [Dostopano: 22. 08. 2017].

[6] Lazy Loading a Module. Dosegljivo: https://angular-2-training-book.rangle.io/handout/modules/lazy-loading-module.html . [Dostopano: 22. 08. 2017].

[7] Angular vs. React: Which Is Better for Web Development? Dosegljivo: https://www.toptal.com/front-end/angular-vs-react-for-web-development . [Dostopano: 22. 08. 2017].

[8] Angular vs React — the DEAL BREAKER. Dosegljivo: https://hackernoon.com/angular-vs-react-the-deal-breaker-7d76c04496bc . [Dostopano: 22. 08. 2017].

[9] Angular vs React : A Side-By-Side Comparison. Dosegljivo: https://www.pluralsight.com/guides/front-end-javascript/angular-vs-react-a-side-by-side-comparison . [Dostopano: 22. 08. 2017].

[10] AngularJS. Dosegljivo: https://angularjs.org/ . [Dostopano: 23. 08. 2017].

[11] PrimeNG. Dosegljivo: https://www.primefaces.org/primeng// . [Dostopano: 23. 08. 2017].

[12] React. Dosegljivo: https://facebook.github.io/react/ . [Dostopano: 23. 08. 2017].

[13] React Native. Dosegljivo: https://facebook.github.io/react-native/ . [Dostopano: 23. 08. 2017].

[14] Angular CLI. Dosegljivo: https://cli.angular.io/ . [Dostopano: 23. 08. 2017].

[15] Website Response Times. Dosegljivo: https://www.nngroup.com/articles/website-response-times/ . [Dostopano: 29. 08. 2017].

[16] Response Times: The 3 Important Limits. Dosegljivo: https://www.nngroup.com/articles/response-times-3-important-limits/ . [Dostopano: 29. 08. 2017].

[17] Performance is User Experience. Dosegljivo: http://designingforperformance.com/performance-is-ux/ . [Dostopano: 29. 08. 2017].

[18] Angular vs. React: 7 Key Features Compared. Dosegljivo: https://code.tutsplus.com/articles/angular-vs-react-7-key-features-compared–cms-29044 . [Dostopano: 29. 08. 2017].

[19] Google Trends. Dosegljivo: https://trends.google.com/trends/explore?date=today%205-yq=Angular,%2Fm%2F012l1vxv,%2Fm%2F0j45p7w . [Dostopano: 29. 08. 2017].

[20] Refactoring Angular Apps to Component Style. Dosegljivo: https://teropa.info/blog/2015/10/18/refactoring-angular-apps-to-components.html . [Dostopano: 29. 08. 2017].