Angular web aplikacija za fizikalnu terapiju i rehabilitaciju

Platužić, Josip

Master's thesis / Diplomski rad

2024

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:630905

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-09**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

Angular web aplikacija za fizikalnu terapiju i rehabilitaciju

Diplomski rad

Josip Platužić

Osijek, 2024.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac D1: Obrazac za imenovanje Povjerenstva za diplomski ispit

Osijek, 30.04.2024.

Odboru za završne i diplomske ispite

Imenovanje Povjerenstva za diplomski ispit

Ime i prezime Pristupnika:	Josip Platužić
Studij, smjer:	Diplomski sveučilišni studij Računarstvo
Mat. br. Pristupnika, godina upisa:	D-1241R, 07.10.2021.
OIB studenta:	10674426797
Mentor:	izv. prof. dr. sc. Ivica Lukić
Sumentor:	,
Sumentor iz tvrtke:	
Predsjednik Povjerenstva:	izv. prof. dr. sc. Mirko Köhler
Član Povjerenstva 1:	izv. prof. dr. sc. Ivica Lukić
Član Povjerenstva 2:	Miljenko Švarcmajer, mag. ing. comp.
Naslov diplomskog rada:	Angular web aplikacija za fizikalnu terapiju i rehabilitaciju
Znanstvena grana diplomskog rada:	**Informacijski sustavi (zn. polje računarstvo)**
Zadatak diplomskog rada:	Web aplikacija sa svrhom pružanja usluga i korisnog sadržaja u području fizioterapije. Aplikacija će biti napravljena pomoću Angular-a, te će imati mogućnost prijave različitih tipova korisnika. Omogućit će termine za prijavu i praćenje procesa rehabilitacije.
Prijedlog ocjene pismenog dijela ispita (diplomskog rada):	Izvrstan (5)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 3 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 3 bod/boda Jasnoća pismenog izražavanja: 3 bod/boda Razina samostalnosti: 3 razina
Datum prijedloga ocjene od strane mentora:	30.04.2024.

Potvrda mentora o predaji konačne verzije rada:	*Mentor elektronički potpisao predaju konačne verzije.*
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA
I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 21.05.2024.

Ime i prezime studenta:	Josip Platužić
Studij:	Diplomski sveučilišni studij Računarstvo
Mat. br. studenta, godina upisa:	D-1241R, 07.10.2021.
Turnitin podudaranje [%]:	12

Ovom izjavom izjavljujem da je rad pod nazivom**: Angular web aplikacija za fizikalnu terapiju i rehabilitaciju**

izrađen pod vodstvom mentora izv. prof. dr. sc. Ivica Lukić

i sumentora ,

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD

Tema diplomskog rada je izrada web aplikacije za fizikalnu terapiju i rehabilitaciju. U današnje vrijeme sve više osoba provodi duge sate sjedeći. Bilo to na radnim mjestima, automobilima ili u najčešćem slučaju, pred ekranima. Nedostatak tjelesne aktivnosti ima niz negativnih posljedica na naše zdravlje, uključujući slabljenje mišića i kostiju uzrokujući smanjenju otpornosti na ozljede.

Tijekom povijesti, rehabilitacija i terapija je uglavnom bila dostupna samo bogatoj populaciji i onima s pristupom medicinskim stručnjacima. Tek u 20. stoljeću, zahvaljujući napretku medicine i fizioterapije, rehabilitacija je postala dostupnija širem stanovništvu.

Trenutno se medicina i znanje o rehabilitaciji neprestano unaprjeđuju. Dostupni su nam moderniji načini liječenja i oporavka od ozljeda. No, ironično je da mnoge osobe, baš zbog navedenog načina života se sve češće ozljeđuju i reagiraju neispravno, ne prateći potrebne korake pri rehabilitaciji. Rezultirajući time nove ozljede ili povratak istih ozljeda, ne iskorištavaju u potpunosti korake rehabilitacije.

Cilj ovog diplomskog rada je izrada web aplikacije za fizikalnu rehabilitaciju koja će biti lako dostupna svima. Aplikacija će pružati korisnicima jednostavan i jasan pristup specifičnim programima, znanju, vježbama i savjetima za ne samo oporavak od različitih ozljeda nego i prevenciju samih, te potaknuti ljude da žive aktivnije i zdravije, time poboljšati zdravlje i kvalitetu života

1.1. Zadatak završnog rada

Zadatak diplomskog rada je izraditi web aplikaciju koja će omogućiti korisnicima pretragu rehabilitacijskog programa i pretplatu na iste. Moći će pratiti napredak, i imati pregled na vježbe. Sa druge strane fizijatri će moći sastavljati i uređivati svoje rehabilitacijske programe te ih objaviti da budu dostupni svim korisnicima.

2. PREGLED PODRUČJA TEME

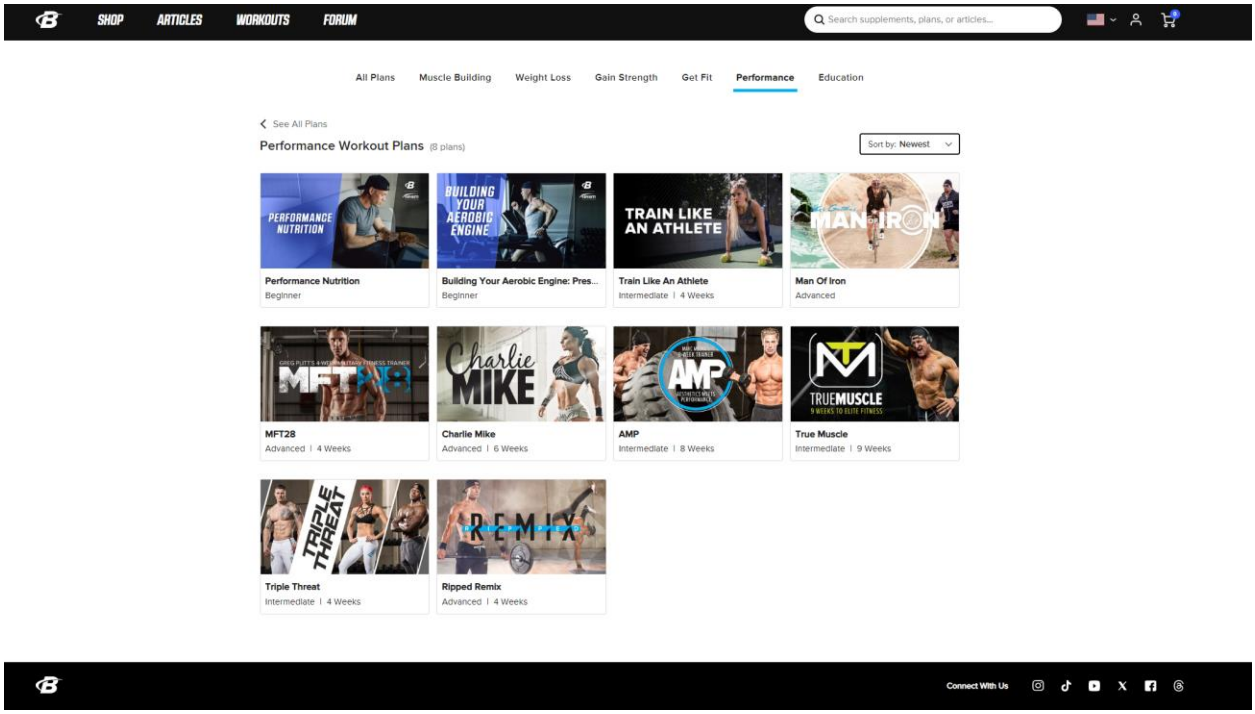
U ovom poglavlju su predstavljena i opisana postojeća rješenja sa sličnom tematikom i funkcionalnostima kao i zadatak diplomskog rada.

2.1. Bodybuilding.com

Iako na prvi pogled *Bodybuilding.com* stranica se čini kao internet trgovina za prodaju sportske prehrane i fitnes opreme, no zapravo je velika zajednica gdje se potiče zdravi način života. Sadrži veliku bazu vježbi s detaljnim uputama, slikama, video zapisima, te može biti odličan resurs za pronalaženje specifičnih vježbi za vlastite potrebe. [1]

Također se na web stranici može pronaći velika količina članaka o različitim fitnes temama koje su većinom usmjerene na izgradnju tijela, ali se može pronaći i puno informacija o prevenciji ozljeda, oporavku i rehabilitacijskim vježbama za različite dijelove tijela.

Sadrži i forum gdje se ljudi mogu povezati i podijeliti iskustva, pronaći ljude sa sličnim ili istim ozljedama te razviti svoje znanje.

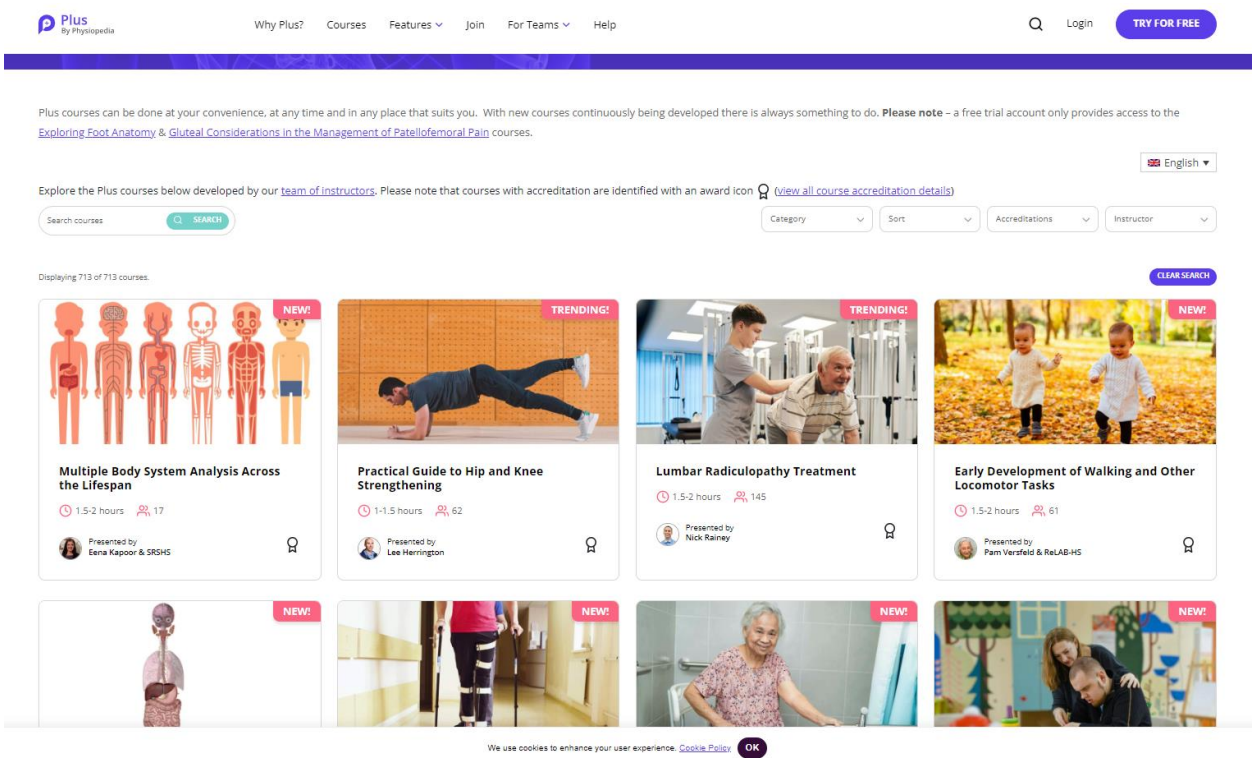


Slika 2.1. *Stranica 'Workouts' Bodybuilding-a*

2.2. Physiopedia

Physiopediaa je online enciklopedija o fizikalnoj rehabilitaciji i terapiji dostupna ne više jezika. Cilj stranice je pružiti znanje i pristup kvalitetnim informacijama o nizu tema od stručnjaka širokoj javnosti. *Physiopedia* je volonterski projekt kojeg održavaju fizioterapeuti i drugi stručnjaci cijelog svijeta. Sadrži informacije o širokom rasponu tema, od anatomije, fiziologije do dijagnostike i liječenja. [2]

Također, u sklopu stranice je platforma *Physiopedia Plus* koja nudi online edukaciju i profesionalni razvoj koji se temelji na *Physiopedia* sadržaju. Nudi online tečajeve, forume za raspravu, certificiranje, i mnogo toga. Usmjerena je za fizioterapeute, studente, i druge stručnjake koji žele poboljšati svoje znanje i vještine. Usluga Physiopedia Plus se naplaćuje za pojedine resurse ili za godišnju pretplatu. [3]

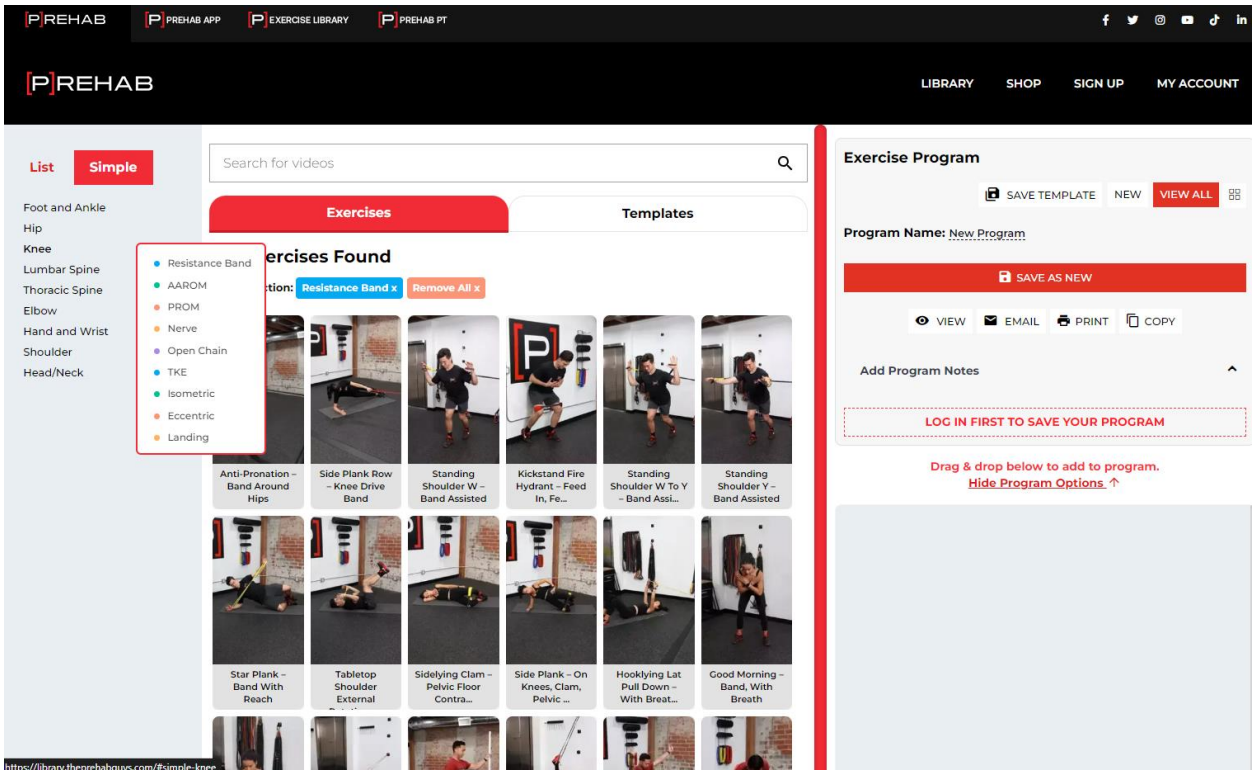


Slika 2.2 *Stranica 'Courses' Physipedia Plus-a*

2.3. [P]Rehab

[P]Rehab web stranica je osmišljena za pružanje korisnih i informativnog sadržaja za fokusiranje na vježbe za smanjenje fizičke boli, prevenciju ozljeda, kao i vježbe za rehabilitaciju nakon ozljeda.

Stranica sa sastoji od mnoštvo vježba sa video instrukcijama i omogućava izgradnju osobnog programa koristeći već postojeće vježbe. Također su već dodani gotovi programi koje je moguće naknadno urediti po osobnim preferencijama. [4]



Slika 2.3 *Stranica 'Exercise library' [P]Rehab stranice*

2.4. Prijedlog rješenja

Primjer web aplikacija za rehabilitaciju je da se napravi aplikacija sa dva korisnička sučelja, fizijatar i korisnik.

2.4.1. Uloga fizijatar

Fizijatar će imati mogućnost kreiranja rehabilitacijskog programa, moći će napraviti plan izvođenja vježbi, kao i detaljne instrukcije u izvođenju programa i pojedinih vježbi. Moći će

dodavati proizvoljan broj vježbi po treningu, a i treninga po programu. Nakon kreiranja programa moći će objaviti svoj program da bude vidljiv drugim korisnicima.

2.4.2. Uloga korisnik

Korisnik će imati mogućnost pregleda i filtriranja programa. Pronalaskom određenog programa korisnik će se moći pretplatiti na program što će mu omogućiti izvođenje treninga i vježbi. Moći će pratiti napredak i dane instrukcije od autora programa.

3. KORIŠTENE TEHNOLOGIJE

U ovom poglavlju su opisane korištene tehnologije potrebne za razvoj poslužiteljskog i klijentskog dijela web aplikacije.

3.1. Angular

Angular je platforma i okvir za izgradnju SPA (engl. *Single Page Applications*). Angular je napisan u TypeScript-u. Arhitektura se sastoji od nekoliko osnovnih koncepata, a glavni od njih su komponente [5]. Komponente definiraju prikaze (engl. *view*) što su skupovi elemenata na ekranu koje Angular bira i modificira u skladu prema logici programa. Komponenta se sastoji se od tri dijela:

- Typescript klase – upravlja ponašanjem i logikom komponente
- HTML (engl. *HyperText Markup Language*) predložak koji definira sadržaj koji se prikazuje korisniku.
- CSS datoteka stilova – Definira klase i stilove za izgled prikaza.

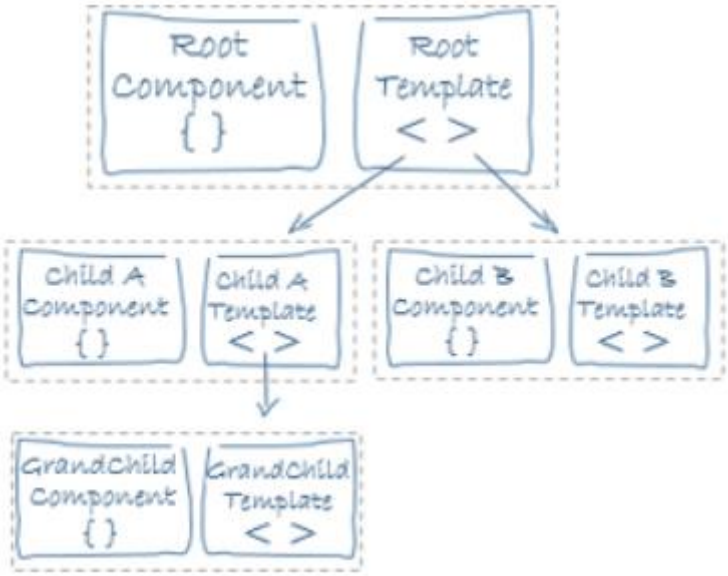
Angular komponente predstavljaju zasebnu jedinicu funkcionalnosti unutar aplikacije. Sadrže dodatne informacije vezane uz kod koje daju upute Angular-u kako koristiti komponentu i kako ju prikazati [5]. Primjer sa slike 3.1 prikazuje strukturu komponente sa dodatnim informacijama unutar oznake *@Component* , gdje se navode informacije:

- s kojim selektorom se govori Angular-u da pozove instancu komponente ako naiđe na taj selektor u HTML predlošcima
- Putanja do HTML predloška komponente.
- Niz komponenata, direktiva, i paketa koje se referenciraju unutar komponente.
- Određuje se hoće li komponenta biti samostalna ili ne.

```
@Component({
  standalone: true,
  selector:    'app-hero-list',
  templateUrl: './hero-list.component.html',
  imports:     [ NgFor, NgIf, HeroDetailComponent ],
  providers:  [ HeroService ]
})
export class HeroListComponent implements OnInit {
  /* . . . */
}
```

Slika 3.1 *Primjer strukture koda Angular komponente.*

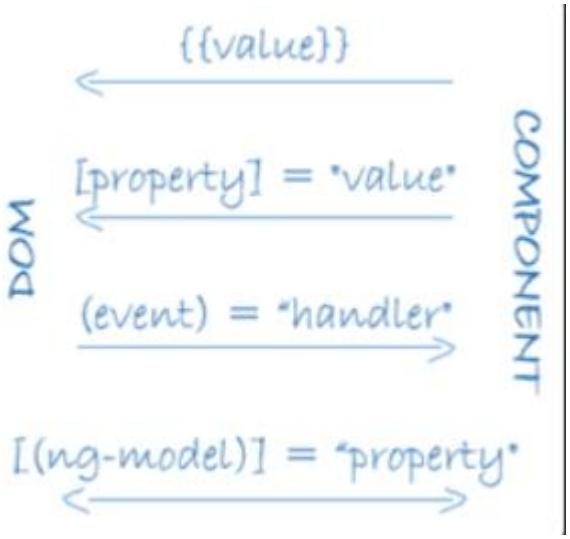
Prikazi se u Angular-u strukturiraju hijerarhijski, time dopuštajući lakše upravljanje korisničkim sučeljen. Svaka komponenta može sadržavati više drugih komponenata ili više instanca iste kao na slici 3.2, a svaka od njih ima svoj glavni prikaz.



Slika 3.2 *Hijerarhija angular komponenata*

Angular sadrži dvostruko povezivanje podataka (engl. *Two way databinding*). Mehanizam za povezivanje dijelova HTML predloška komponente sa Typescript kodom. Te govori Angular-u

kako povezati obje strane, primjer korištenja na slici 3.3. Povezivanje podataka također ima bitnu ulogu u povezivanju roditeljskih i dječjih komponenata.



Slika 3.3 *Povezivanje podataka*

Također bitan dio Angular-a su direktive, koje transformiraju DOM (engl. *Document Object Model*), strukturu prema danim instrukcijama. Djelu se na strukturalne i atributne direktive. Strukturalne utječu na strukturu DOM-a manipulirajući elementima, dok atributne direktive modificiraju ponašanje postojećih elemenata.

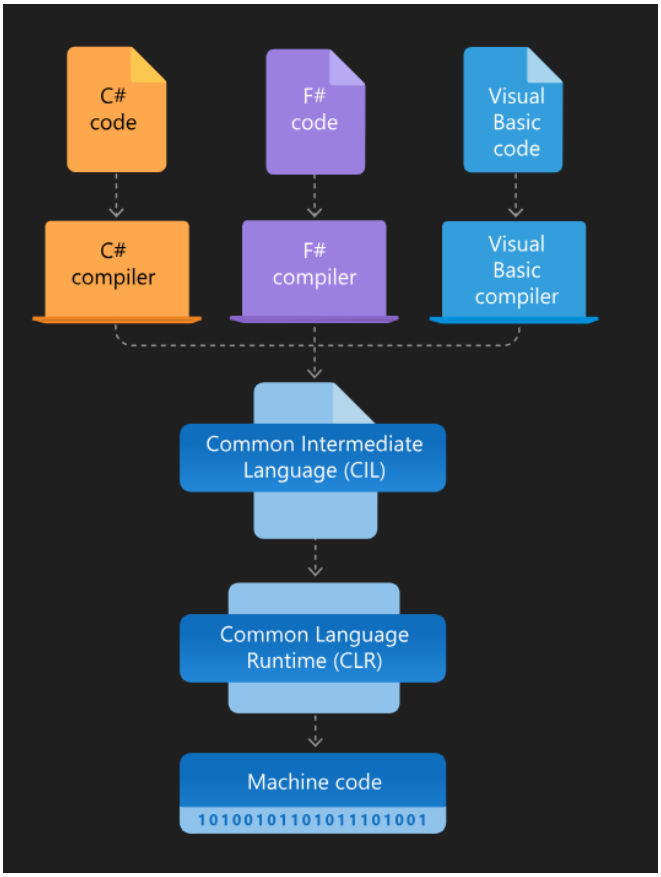
3.2. .NET

.NET je razvojni okvir (engl. *framework*) razvijen od strane Microsofta koja nudi programerima gotova rješenja i funkcionalnosti da bi se ubrzao razvoj aplikacija i servisa. Najvažnija stavka .NET-a je *Common Language Runtime* ili skraćeno CLR. To je sustav koji osigurava upravljanje memorijom, gdje automatski upravlja alokacijom i oslobađanjem memorije za objekte koje koristi kod. Također osigurava upravljanje nitima, udaljeno izvršavanje, sigurnost i preciznost.

Razlikujemo .NET Framework i .NET Core, gdje je .NET Framework originalna verzija .NET-a, koja se većinom koristi za izgradnju Windows aplikacija i web aplikacija, dok je .NET Core dostupan na više platformi i omogućava izgradnju aplikacija na više platformi, te se trenutno preporučuje. [6]

.NET aplikacije se mogu pisati u mnogo jezika, ali CLR ne prepoznaje niti jedan jezik osim MSIL-a (engl. *Microsoft Intermediate Language*), s toga postoje brojni kompajleri koji će programski

jezik prevesti u MSIL kako bi ga CLR razumio. Te po tome JIT kompajler (engl. *Just In Time*) će prevesti kod u izvorni strojni kod računala, primjer na slici 3.3. [7]



Slika 3.3 *CLR .NET-a*

.NET Framework biblioteka klasa je kolekcija tipova za ponovnu upotrebu koja je blisko integrirana sa CLR-om. Biblioteka je objektno orijentirana i definira osnovne tipove podataka, tipove često korištenih struktura podataka poput nizova, listi i sl. Također nudi klase za čitanje i pisanje podataka u datoteke i druge izvore, omogućuje pristup informacijama u učitanim tipovima i mnogo drugih funkcionalnosti. [8]

.NET Core je novija verzija .NET Framework-a. Koristi se za razvoj raznolikih aplikacija, uključujući mobilne, desktop, web, mikro servise itd. .NET Core je platforma otvorenog koda, omogućuje rad na Windowsu, MAC OS-u i Linuxu. Modularan je, što znači da se mogu instalirati samo one komponente koje su potrebne za aplikaciju. [8]

3.2.1. Entity Framework Core

Entity Framework Core ili skraćeno EF Core je ne zahtjevan, proširiv alat otvorenog koda, koji služi kao ORM (engl. *Object relational mapper*) koji omogućava .NET programerima da upravljaju bazom podataka koristeći .NET objekte, eliminira potrebu za viškom koda za pristup

podatcima koji se obično mora ručno pisati, i omogućava *code-first* pristup gradnje aplikacije. Kod *code-first* pristupa programer se fokusira na definiranju modela aplikacije, a baza podataka se generira na temelju tog modela. Dok pri *database-first* pristupu, kod se piše direktno u SQL i na kraju se definiraju modeli na temelju baze podataka. Obje verzije imaju svoje prednosti i nedostatke. U *code-first* pristupu prilikom kreiranja modela, koriste se EF migracije za izgradnju baze podataka iz napisanog modela. Migracije omogućuju razvoj baze paralelno sa promjenama na modelu, te praćenjem tih promjena. [9]

3.3. MySql

MySQL je jedan od najpopularnijih SQL (engl. *Structured Query Language*) sustava otvorenog koda za upravljanjem bazom podataka, razvijen od strane *Oracle Corporation*. Baza podataka je strukturirana kolekcija bilo kakvih podataka. Za upravljanje bazom podataka, što podrazumijeva dodavanje, pristup i obradu podataka pohranjenih u bazi, potreban je sustav poput MySQL-a. [10]

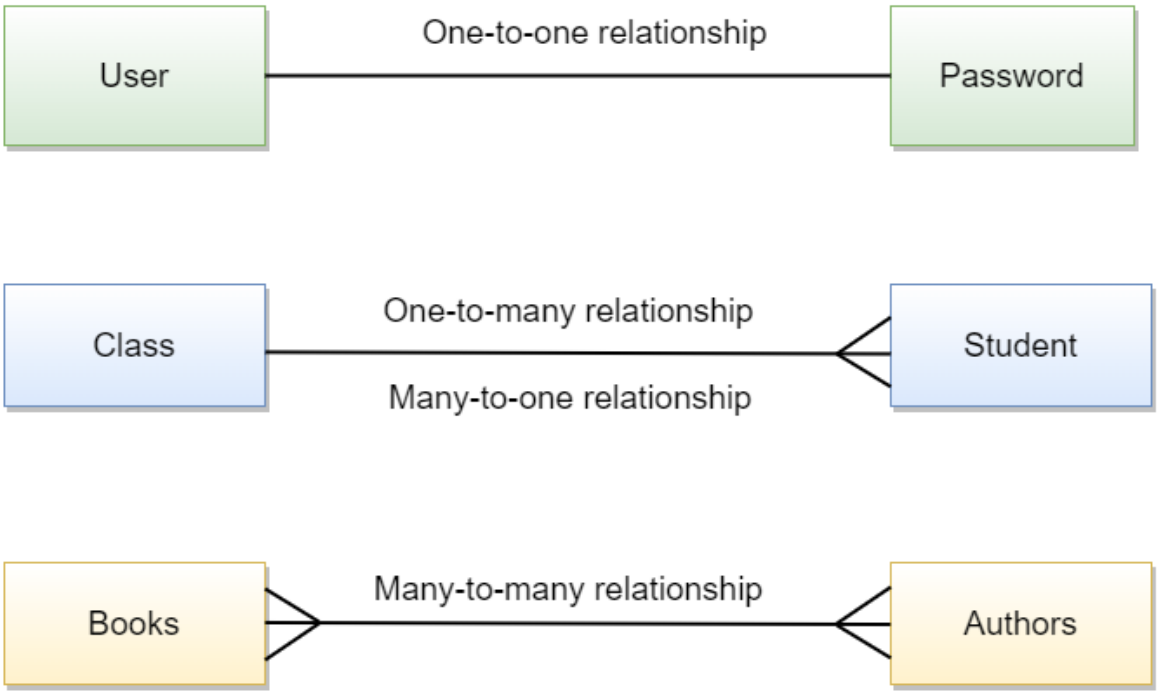
MySQL baza podataka je relacijska, što znači da pohranjuje podatke u odvojenim tablicama koje sadrže retke i stupce. Stupci definiraju atribute ili karakteristike podataka u tablici, dok redovi predstavljanju pojedinačne zapise ili podatke u tablici.

Relacije se koriste za opisivanje odnosa koji postoje između tablica u relacijskoj bazi podataka. Odnos između dvije tablice baze podataka se podrazumijeva da jedna tablica referencira primarni ključ druge tablice. Primarni ključ predstavlja stupac koji je jedinstven i predstavlja vrijednost koju se može referencirati za sve zapise, jer svaki zapis će imati jedinstveni primarni ključ. Svaka tablica može imati samo jedan primarni ključ [11].

Tipovi relacija u bazi:

- Jedan prema više (engl. *One to many*) – Kada zapis iz tablice A je povezan sa više zapisa iz tablice B, ali zapisi u tablici B mogu imati vezu samo sa jednim zapisom iz tablice A.
- Više prema više (engl. *Many to many*) – Kada su zapisi iz tablice A povezani sa više zapisa iz tablice B i obratno.
- One to one (engl. *One to one*) - Kada je zapis iz tablice A povezan samo sa jednim zapisom iz tablice B i obratno. [12]

Više na slici 3.4.



Slika 3.4 *Tipovi relacije u relacijskoj bazi podataka*

4. PROGRAMSKO RJEŠENJE

U ovom poglavlju se prolaze koraci programskog rješenja, od zahtjeva i arhitekture rješenja, do same implementacije funkcionalnosti.

4.1. Zahtjevi

Web aplikacija za rehabilitaciju mora sadržavati dvije korisničke uloge, fizijatar i korisnik. Što znači da se mora biti moguća registracija i prijava oba korisnika, te ograničiti pristup određenim podatcima i dijelovima aplikacije određenom korisniku. Pri registraciji se mora ponuditi korisniku odabir uloge.

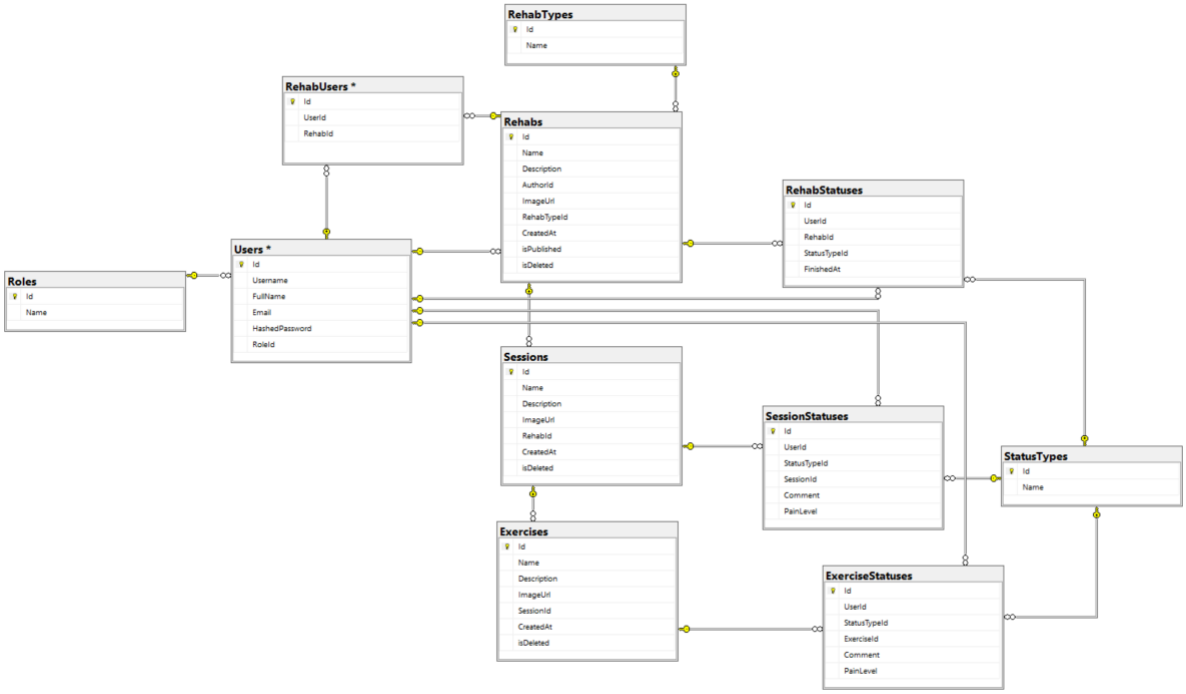
Korisnik sa ulogom fizijatar mora moći kreirati vlastiti rehabilitacijski program, što znači da može ostaviti detaljne i po potrebi opširne upute, instrukcije i savjete o izvođenju programa, treninga i pojedinih vježbi. Po svakom rehabilitacijskom programu, mora biti moguće kreirati proizvoljan broj treninga sa proizvoljnim imenom, kao i proizvoljan broj vježbi po pojedinom treningu. Osim kreiranja rehabilitacijskih programa, fizijatar mora imati mogućnost uklanjanja i uređivanja programa u cijelosti zajedno sa treninzima i vježbama. Pristup već kreiranim vježbama će biti na odvojenoj stranici, gdje će osim pregleda i pristupu uređivanju, moći odabrati hoće li pojedini program biti objavljen javno ili biti skriven.

Običan korisnik će imati mogućnost pregleda svih kreiranih rehabilitacijskih programa objavljeni javno od svih fizijatara. Rehabilitacijski programi će se moći filtrirati po imenu programa i po tipu programa, tj. na koje dijelove tijela se odnosi program. Osim svih programa korisnik će imati i prikaz njegovih započetih ili završenih programa. Na program se mora moći pretplatiti na sučelju prikaza svih programa. Pri pokretanju ili nastavku rehabilitacijskog programa, prikazivaju se pojedini treninzi, sa popisom vježba koje sadrži. Mora biti vidljivo koji su treninzi završeni, a koji slijede. Započinjanjem treninga prikazuju se pojedine vježbe treninga kao koraci. Svaki korisnik će moći ostavljati rezultate treninga kao komentar i razinu boli nakon svakog treninga. Završetkom svake vježbe treninga, trening se označava kao završen, te završetkom svakog treninga, rehabilitacijski program se označava kao završen.

4.2. Arhitektura rješenja

Aplikacija dijeli arhitekturu rješenja s obzirom na dva tipa korisnika. Za samu logiku postojanja više tipova korisnika moramo imati tablicu *Users* zajedno sa tablicom *Roles.* Gdje je veza između tablica jedan naprema više (1-N), što znači da jednu rolu može koristiti više korisnika, dok korisnik

može imati samo jednu rolu. Osnovne tablice za logiku aplikacije su *Rehabs*, *Sessions*, *Exercises*. Koje predstavljanju redom rehabilitacijske programe, treninge, i vježbe. Svaka od tih tablica sadrži svoju sukladnu tablicu. Kao primjer ako se uzme tablica *Rehab*, ona je vezana sa tablicom *RehabStatus*, koja predstavlja trenutni status rehabilitacijskog programa pojedinog korisnika. Pomoću te tablice će se moći pratiti da li je rehabilitacija aktivna ili neaktivna s obzirom na korisnika. Zbog toga je tablica vezana i sa tablicom *User*.



Slika 4.1 *Struktura baze podataka*

4.3. Postavljanje razvojnog okruženja

Nakon ideje rješenja i arhitekture, poterbno je kreirati programsko rješenje funkcionalnosti iz zahtjeva. Postavljaju se radne okoline i struktura organizacije koda.

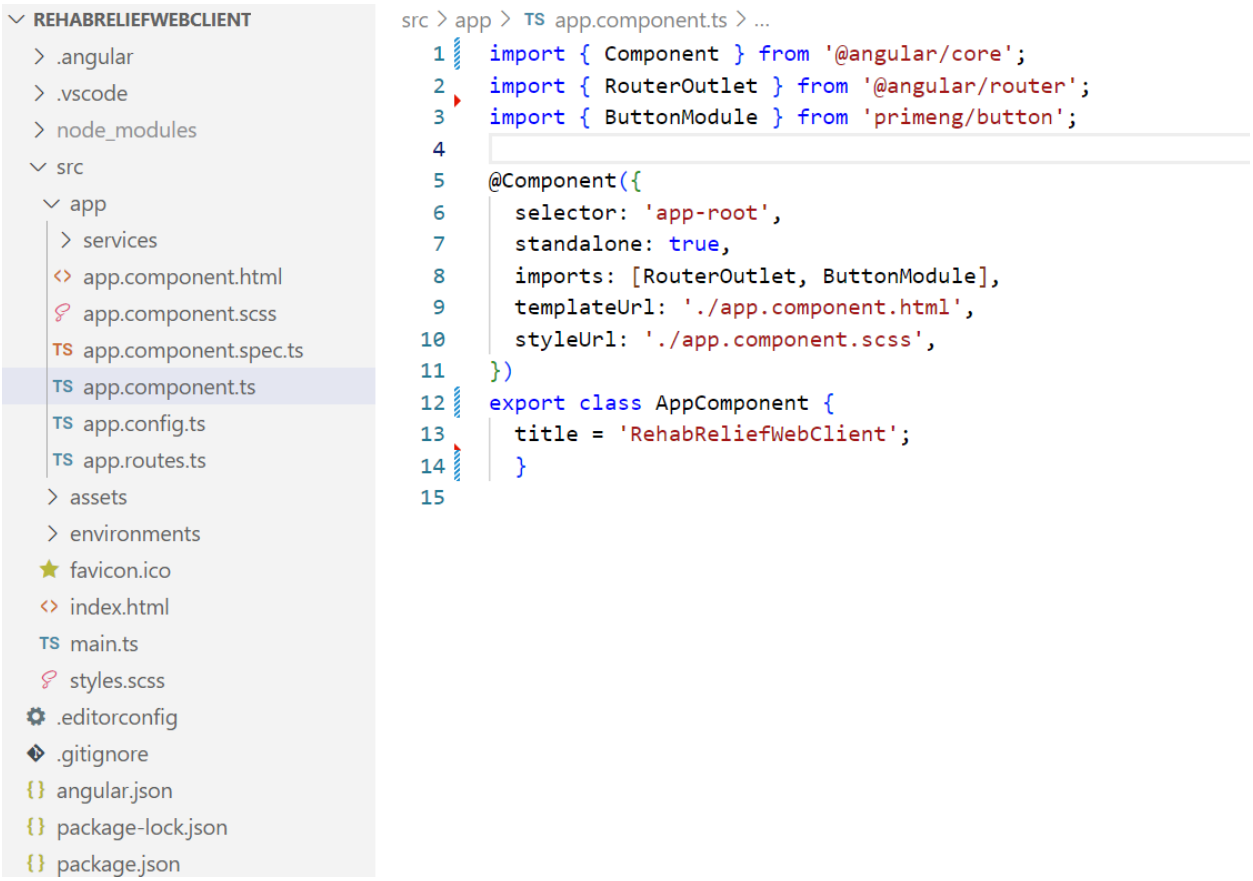
4.3.1. Angular okruženje

Za kreiranje Angular projekta potreban je *Node.js*. *Node.js* je okruženje za izvršavanje Javascripta na više platformi. Koristi se za pisanje aplikacija na serverskoj strani [13]. Uz Node.js, potreban je i npm (engl. *Node Package Manager*) koji služi za instaliranje paketa koji pomažu u razvoju aplikacije [13]. Naposljetku instaliramo Angular CLI-i (engl. *Command Line Interface*) te možemo kreirati Angular projekt komandom sa slike. 4.1

```
ng new RehabRelief
```

Programski kod 4.1 *Kreiranje angular projekta*

Kreiranjem Angular projekta dobiva se početna struktura projekta, slika 4.2. Korištena verzija Angulara je 17, te su korištene samostalne (engl. *standalone*) komponente. Što znači da komponente neće biti ovisne o modulima, nego su same odgovorne za kontroliranje ovisnosti. Početna kreirana komponenta je *app.component* komponenta, koja će služiti kao osnovna roditeljska komponenta i početna točka naše aplikacije, u kojoj će naknadno biti dodane rute koje će biti objašnjene naknadno.



Slika 4.2 *Angular struktura*

Kao biblioteku komponenti za izgradnju korisničkog sučelja koristit ćemo PrimeNg biblioteku, koja sadrži već gotove komponente poput gumbova, polja za unos, kartica, dijaloga, ikona i sl. olakšavajući kreiranje responzivne i intuitivne web stranice. Razvijen je od strane *PrimeTekInfomatics.* Instalira se programskim kodom 4.2

```
npm install primeng
```

Programski kod 4.2 *Instalacija PrimeNG biblioteke*

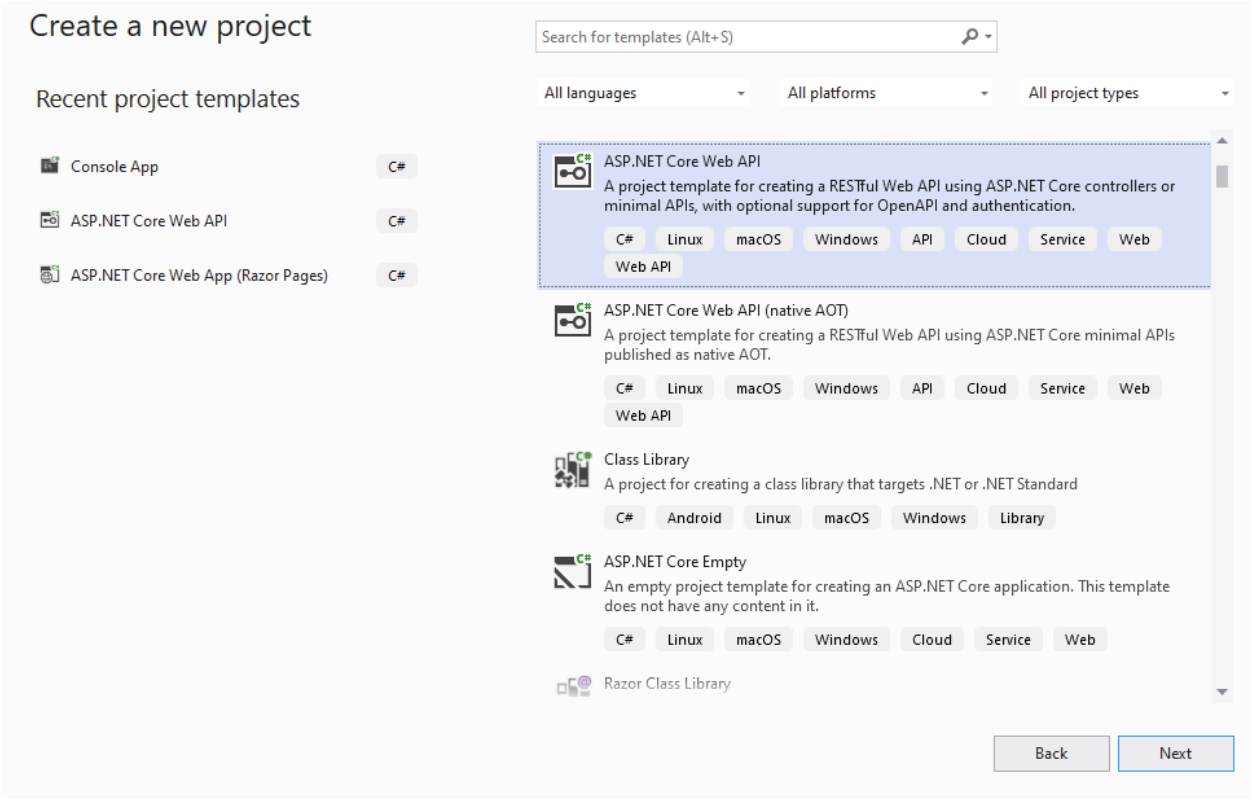
Kao biblioteku za CSS (engl. *Cascading Style Sheets*) koristit će se Tailwind CSS. Sadrži gotove klase za pomoć pri izgradnji web aplikacija omogućujući dizajniranje stranica direktno u HTML kodu pisanjem predefinirani klasa, instalira se programskim kodom 4.3.

```
npm install -D tailwindcss
```

Programski kod 4.3 *Instalacija Tailwind CSS-a*

4.3.2. .NET okolina

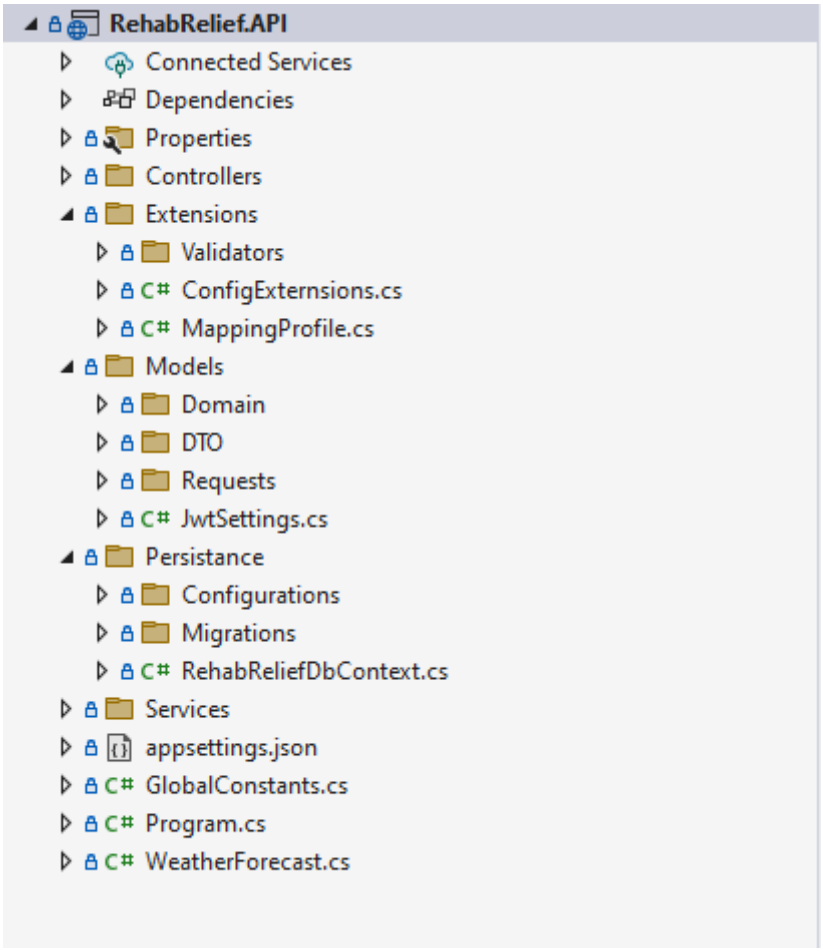
Za kreiranje .NET okoline, kreirano novi ASP.NET Core Web API projekt. ASP.NET je okvir koji se koristi za olakšavanje kreiranja web aplikacija, servisa i stranica.



Slika 4.1 *Kreiranje .NET projekta*

Struktura datoteka su organizirane prema slici 4.4. Razlikujemo mapu *Controller* gdje će biti kontroleri koji su zaduženi za primanje HTTP (engl. *Hypertext Transfer Protocol*) zahtjeva od korisnika te vraćaju podatke, obično u JSON (engl. *Javascript Object Notation*) ili XML (engl. *Extensible Markup Language*) formatu. Kontroleri će pozivati validatore koji se nalaze u *Validators* mapi. Uloga validatora je provjeriti ispravnost zahtjeva poslanih od korisnika. Modeli korišteni u projektu se nalaze u *Models* mapi, a konfiguracije entiteta baze podataka su smješteni u *Persistance* mapi zajedno sa migracijama, dok su servisi koji imaju definiraju logiku

funkcionalnosti svih metoda, u *Services* mapi. Za validaciju se koristi *FluentValidation* biblioteka, a za mapiranje objekata *AutoMapper* biblioteka.



Slika 4.2 *Struktura API projekta*

4.3.3. Povezivanje klijenta i poslužitelja

Nakon kreiranja klijentske aplikacije i poslužiteljske aplikacije, mora se omogućiti komunikaciju između njih. Specifično za *Visual studio,* datoteka *launchSettings.json* definira konfiguraciju za pokretanje aplikacije. Profil *https sa slike 4.4* sadrži *applicationUrl* stavku koja definira URL na kojoj će aplikacija biti dostupna.

```
"https": {
  "commandName": "Project",
  "dotnetRunMessages": true,
  "launchBrowser": true,
  "launchUrl": "swagger",
  "applicationUrl": "https://localhost:7273;http://localhost:5187",
  "environmentVariables": {
    "ASPNETCORE_ENVIRONMENT": "Development"
  }
},
```

Slika 4.3 *Profil https u launchSettings.json-u*

U *program.cs* datoteci se dodaje CORS (engl. Cross-Origin Resource Sharing) prema slici 4.4 da bi Angular aplikacija koja se pokreće na drugom portu mogla pristupiti API-ju. Zapravo se radi o dodavanju komponente međusloju (engl. *middleware*) cjevovoda (engl. *pipeline*) obrade zahtjeva. Provjeravat će se svaki nadolazeći HTTP zahtjev i provjeravati jesu li iz dozvoljenog izvora.

```
builder.Services.AddCors(options =>
{
    options.AddPolicy(name: "CorsPolicy",
        builder =>
        {
            builder.WithOrigins("http://localhost:4200")
                                .AllowAnyHeader()
                                .AllowAnyMethod()
                                .AllowCredentials();
        });
});
```

Slika 4.4 *Cors u program.cs-u*

U Angular aplikaciji moramo dodati temeljni URL za naš ASP.NET Core API, na koji će se slati svi HTTP zahtjevi.

```
1  export const environment = {
2    production: true,
3    baseUrl: 'https://localhost:7273/api',
4  };
5
```

Slika 4.5 *enviorment.ts*

4.4. Rješenja funkcionalnosti

Prvi korak jest kreiranje baze podataka. Koristi se *Entity Framework Core* za *code-first* pristup. Definiraju se modeli koji će biti temelj za kreiranje entiteta u bazi. Kao primjer pogledati *User* model u programskom kodu 4.4. Svaki entitet mora imati primarni ključ nazvan *Id* koji predstavlja integer tip podatka.

```
public class User
 {
     public int Id { get; set; }
     public string Username { get; set; }
     public string FullName { get; set; }
     public string Email { get; set; }
     public string HashedPassword { get; set; }

     public int RoleId { get; set; }
     public Role? Role { get; set; }

     public ICollection<Rehab>? OwnedRehabs { get; set; } = null;
     public ICollection<RehabUser>? RehabUsers { get; set; } = null;
     public ICollection<RehabStatus>? RehabsStatuses { get; set; } = null;
     public ICollection<SessionStatus>? SessionStatuses { get; set; } = null;
     public ICollection<ExerciseStatus>? ExerciseStatuses { get; set; } = null;

 }
```

Programski kod 4.4 *Model User-a*

Za fleksibilniju i odvojenu konfiguraciju veza između entiteta u *Entity Framework-u* koristi se *IEntityTypeConfiguration* sučelje, vidi programski kod 4.5. Ovakav pristup omogućava definiranje konfiguracije pojedinih entiteta u odvojenoj klasi, poboljšavajući preglednost i modularnost koda.

```
public class UserConfiguration: IEntityTypeConfiguration<User>
{
    public void Configure(EntityTypeBuilder<User> builder)
    {
        builder.HasOne(x => x.Role)
            .WithMany(x => x.Users)
            .HasForeignKey(x => x.RoleId)
            .OnDelete(DeleteBehavior.Restrict);

        (..)

    }
}
```

Programski kod 4.5 *Konfiguracija User-a*

DbContext klasa definira interakciju s bazom podataka u *Entity framework*-u (programski kod 4.6). Predstavlja sesiju s bazom podataka i omogućava rad sa entitetima, kao što je kreiranje, čitanje, uređivanje i brisanje podataka nad bazom. Također *DbContext* prati promjene koje su napravljene na entitetima te omogućava da se promjene sačuvaju u bazi podataka, te omogućuje obradu transakcija gdje se izvrše sve ili niti jedna operacija ovisno o uspjehu.

```
public class RehabReliefDbContext : DbContext
 {
     public RehabReliefDbContext(DbContextOptions options) : base(options)
     {
     }

     public DbSet<Session> Sessions { get; set; }
     public DbSet<User> Users{ get; set; }
     public DbSet<Role> Roles{ get; set; }
     public DbSet<Rehab> Rehabs { get; set; }
     public DbSet<RehabStatus> RehabStatuses{ get; set; }

     (..)

     protected override void OnModelCreating(ModelBuilder modelBuilder)
     {
         modelBuilder.ApplyConfigurationsFromAssembly(
         Assembly.GetExecutingAssembly());
     }
 }
```

Programski kod 4.6 *Konfiguracija DbContext-a*

4.4.1. Autorizacija

Za autorizaciju se koristi JSON Web Token (skraćeno JWT) koji definira sažet i kompaktan način za sigurno prenošenje informacije između klijenta i poslužitelja. Informacije u tokenu se mogu provjeriti i smatrati pouzdanima jer su digitalno potpisane. Nakon što se korisnik prijavi, dobiva JWT, koji će omogućiti korisniku pristup rutama, servisima i resursima koji su dopušteni tim tokenom.

Sami token se sastoji od tri dijela: zaglavlja, sadržaja i potpisa. Zaglavlje se obično sastoji od dva dijela, tip tokena i algoritma heširanja. Sadržaj predstavlja skup podataka koje želimo prenijeti s tokenom, dok potpis sadrži tri komponente: zaglavlje, sadržaj i tajnu. Potpis se kreira heširanjem kodiranog zaglavlja i sadržaja korištenjem tajne. [14]

U *program.cs* datoteci prema programskom kodu 4.7 definira se tip autentikacije koji će biti *Json Web Token*. Definira se izdavatelja tokena, publika, trajanje, ključ za potpisivanje tokena, te tip sadržaja koji se koristi za dohvat korisničkih uloga iz tokena.

```
builder.Services.AddAuthentication(x =>
{
    x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
    x.DefaultScheme = JwtBearerDefaults.AuthenticationScheme;
}).AddJwtBearer(x =>
{
    x.TokenValidationParameters = new TokenValidationParameters
    {
        ValidIssuer = jwtSettings.Issuer,
        ValidAudience = jwtSettings.Audience,
        IssuerSigningKey = new
SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Key)),
        ValidateIssuer = true,
        ValidateAudience = true,
        ValidateLifetime = true,
        ValidateIssuerSigningKey = true,
        RoleClaimType = ClaimTypes.Role,
    };
});
```

Programski kod 4.7 *Konfiguracija JWT-a*

Prilikom autentikacije, sa klijentske aplikacije se šalje zahtjev za prijavu pomoću email-a i lozinke, nakon uspješne validacije podataka iz zahtjeva što uključuje provjeru postoji li korisnik sa odgovarajućim e-mailom, te da li je lozinka odgovarajuća, korisniku se vraća generirani token prema programskom kodu 4.8. Sadržaj (engl. *claims*) tokena se definira i prosljeđuju se informacije koje bitne da token sadrži, te kreira se token sa odgovarajućim parametrima uključujući sadržaj tokena.

```
public TokenDTO GetToken(User user)
{
    var jwtSettings =
_config.GetSection<JwtSettings>(GlobalConstants.JWT_SETTINGS_KEY);

    // Create claims
    var claims = new List<Claim>
    {
        new (JwtRegisteredClaimNames.Sub, user.Id.ToString()),
        new (JwtRegisteredClaimNames.Name, user.Username),
        new (JwtRegisteredClaimNames.Email, user.Email),
        new Claim(ClaimTypes.Role, user.Role!.Name),
    };

    // Create token credentials
```

```
    var key = new SymmetricSecurityKey(Encoding.UTF8.GetBytes(jwtSettings.Key));
    var creds = new SigningCredentials(key, SecurityAlgorithms.HmacSha256);

    // Create token
    var token = new JwtSecurityToken(
        issuer: jwtSettings.Issuer,
        audience: jwtSettings.Audience,
        claims: claims,
        expires: DateTime.UtcNow.AddMinutes(30),
        signingCredentials: creds);


    // Generate token
    var tokenHandler = new JwtSecurityTokenHandler();
    var generatedToken = tokenHandler.WriteToken(token);

    return new TokenDTO { Token = generatedToken };
}
```

Programski kod 4.8 *Generiranje JWT*

Na klijentskoj aplikaciji nalazi se servis *AuthService* koji je zadužen za funkcionalnosti autentikacije. Za rješenje se koriste Angular Signali.

Signali predstavljaju reaktivni način za upravljanje stanjima u komponentama. Omogućuju laki pristup i promjenu stanja. Ako se vrijednost signala promjeni, automatski se obavještavaju sve komponente koje su pretplaćene na taj signal i ne uključuje ručno provjeravanje promjene. Vrijednost signala se može mijenjati samo kroz funkciju *update* što sprječava nepredviđene promjene. [15]

Stanje prijavljenog korisnika se inkapsulira u signal. Signal može sadržavati bilo koju vrstu ili tip vrijednost. Signal *user* će sadržavati vrijednost tipa *TokenModel* koji predstavlja model korisnika. Signalu *user*, daje se početna vrijednost, te prilikom poziva login metode, odgovor koji se očekuje da bude JWT koji se generirao na poslužiteljskoj aplikaciji, sprema se u lokalnu pohranu preglednika. Te vrijednost iz lokalne pohrane dohvaća se sa *getDecodedToken* metodom gdje dekodiramo token i mapiramo ga u *TokenModel* klasu, te postavljamo novu vrijednost signala *user*. Koristeći *user* signal možemo pristupiti i koristiti informacije o trenutnom korisniku bilo gdje u aplikaciji, te pratiti moguće promjene.

```
  isAuthenticated = signal(false);
  user: WritableSignal<TokenModel> = signal({
    userId: 0,
    role: RoleType.Unknown,
    name: '',
  });
```

```
  login(model: LoginRequest) {
    return this.httpClient.post<any>(`${this.API_URL}/users/login`, model).pipe(
      tap((response) => {
        localStorage.setItem('jwt_token', response.token);
        this.isAuthenticated.set(true);
        this.user.set(this.getDecodedToken())
      })
    );
  }
```

Programski kod 4.9 *Metoda za prijavu korisnika*

4.4.2. Upravljanje rehabilitacijskim programima

Kao model po kojemu se kreira entitet Rehabilitacija u bazi podataka, koristimo klasu *Rehab, (*programski kod 4.10*)*, koja od bitnih atributa sadrži ime i opis programa koji su tipa tekst (engl. *string)*, strani ključ na autora programa tipa broj (engl. *integer)*, zatim tip rehabilitacije koji daje informaciju na rehabilitaciju kojeg dijela tijela se program odnosi. Model sadrži i kolekcije, kao primjer, uzmimo *Session kolekciju.* Session model sadrži strani ključ na tablicu *Rehab*, a *Rehab* tablica sadrži kolekciju *Session-a*, što predstavlja više naprema jedan (N-1) vezu.

```
public class Rehab
{
    public int Id { get; set; }
    public string Name { get; set; }
    public string Description { get; set; }
    public int AuthorId { get; set; }
    public User? Author { get; set; }
    public int RehabTypeId { get; set; }
    public bool isPublished { get; set; }
    public bool isDeleted { get; set; }
    public RehabType? RehabType { get; set; }
    public DateTime CreatedAt { get; set; }
    public ICollection<RehabUser>? RehabUsers { get; set; } = null;
    public ICollection<Session>? Session { get; set; } = null;
    public ICollection<RehabStatus>? RehabStatuses { get; set; } = null;

    public string? ImageUrl { get; set; } = null;

    (..)
}
```

Programski kod 4.10 *Rehab model*

Sa klijentske aplikacije se šalje zahtjev za kreiranje rehabilitacijskog programa, šaljući *AdministrationCreateRehabRequest*, model kojega API očekuje. Uz podatke o programu šaljemo i *Id* od trenutnog korisnika, to jest autora programa.

```
createRehab(model: AdministrationCreateRehabRequest) {
    return this.httpClient.post<AdministrationRehab>(
      `${this.API_URL}/rehab/administration/`,
      {
        name: model.name,
        description: model.description,
        rehabTypeId: model.rehabTypeId,
        authorId: model.authorId,
      }
    );
  }
```

Programski kod 4.11 *Metoda za kreiranje nove rehabilitacije*

Metoda kontrolera *AdministrationCreateRehab* rukuje sa dolazećim HTTP zahtjevima, (programski kod 4.12), provjerava validnost podataka pozivajući *validationService* koji je ubrizgan u kontroler klasu *RehabController*. Ako je validacija prošla uspješno, dalje se poziva servis *rehabService* u kojemu je napisana logika kreiranja rehabilitacijskog programa (programski kod 4.12).

```
[HttpPost("administration")]
 public async Task<IActionResult> AdministrationCreateRehab([FromBody]
AdministrationCreateRehabRequest request)
 {
     var validationResult =
await_validationService.ValidateCreateRehabRequest(request);
     if (!validationResult.IsValid)
     {
         return NotFound(validationResult.Errors);
     }

     var response = await _rehabService.AdministrationCreateRehab(request);
     return Ok(response);
 }
```

Programski kod 4.12 *Kontroler za dodavanje rehabilitacije*

U pogramskom kodu 4.13 je metoda *rehabService-a* koja prima *AdminstrationCreateRehabRequest* zahtjev, kreira novi rehabilitacijski program pomoću konstuktora, te koristeći *dbContext* dodaje kreirani program, te sprema promjene. Kao odgovor servis vraća kreirani rehabilitacijski program koji se mapira u *RehabDto* model. DTO je skraćeno

od *Data transfer object* i koristi se za prenošenje podataka. DTO prenosi samo potrebne podatke, time štiti sigurnost podataka od klijenta.

```
public async Task<RehabDto>
AdministrationCreateRehab(AdministrationCreateRehabRequest request)
{
    var rehab = new Rehab(request.Name, request.Description, request.AuthorId,
request.RehabTypeId);

    await dbContext.AddAsync(rehab);
    await dbContext.SaveChangesAsync();
    var response = mapper.Map<RehabDto>(rehab);
    return response;
}
```

Programski kod 4.13 *funkcija za dodavanje rehabilitacije*

Kod uređivanja rehabilitacijskog programa (programski kod 4.14), u zahtjevu se šalju isti podatci kao kod kreiranja novog rehabilitacijskog programa, ali uz *Id* programa kojega se uređuje. Prvo se iz baze podataka pomoću *DbContext-a* dohvaća program sa odgovarajućim Id-em, te mu se ažuriraju podatci sa onima u zahtjevu.

```
public async Task<RehabDto>
AdministrationUpdateRehab(AdministrationUpdateRehabRequest request)
{
    var rehab = dbContext.Rehabs
         .Include(x => x.Author)
        .Include(x => x.RehabType)
        .Where(x => !x.isDeleted && x.Id == request.Id).FirstOrDefault();

    rehab.Name = request.Name;
    rehab.Description = request.Description;
    rehab.RehabTypeId = request.RehabTypeId;
    rehab.isPublished = request.IsPublished;

    dbContext.Update(rehab);
    await dbContext.SaveChangesAsync();
    var response = mapper.Map<RehabDto>(rehab);
    return response;
}
```

Programski kod 4.14 *Funkcija za uređivanje rehabilitacije*

Dok kod brisanja programa nam je potreban samo *Id*. Pa kao zahtjev primamo samo jedan parametar tipa integer. Umjesto potpunog brisanja iz baze podataka. Koristit ćemo logičko brisanje, što predstavlja samo označavanja određenog atributa koji predstavlja stanje entiteta da li

je aktivan ili ne. Korištenjem logičkog brisanja omogućavamo vraćanje obrisanih podataka i sprječavamo greške.

```
public async Task AdministrationDeleteRehab(int id)
{
    var rehab = dbContext.Rehabs.Where(x => x.Id == id).FirstOrDefault();
    rehab!.isDeleted = true;
    dbContext.Update(rehab);
    await dbContext.SaveChangesAsync();
}
```

Programski kod 4.15 *Funkcija za brisanje rehabilitacije*

4.4.3. Pregled kreiranih rehabilitacijskih programa

Pri dohvatu svih rehabilitacijskih programa kao zahtjev šalju se parametri pomoću kojih se filtriraju svi programi. Filteri su tip programa i traženi pojam, tj. ime programa. Ti filteri će se spremati u *getRehabRequest* koji će predstavljati signal iz razloga što će se ta vrijednost mijenjati pri korisnikovoj promjeni filtera, te pri svakoj promjeni želimo dohvaćani novu listu rehabilitacijskih programa, tj. želimo slati novi HTTP zahtjev. Programskom kodu 4.16 prikazuje korištenje *rehabs$* varijable kojoj je sufiks znak „$“, što je sintaksa za označavanje Observable-a. Observable predstavlja funkcionalnost iz RxJs biblioteke koja omogućava način za pretplatu i reagiranje na promjene. Koristi se *toObservable* metoda kojoj se kao parametar daje *getRehabRequest* signal. Te kada se promjeni stanje *getRehabRequest* signala, ažuriraju se HTTP parametri pomoću kojih šaljemo dodatne informacije, u našem slučaju podatke za filtriranje.

```
  rehabs = signal<AdministrationRehab[]>([]);
  loading = signal(false);

  getRehabRequest = signal<AdministrationGetRehabsRequest>({
    userId: this.user().userId,
  });

  private rehabs$ = toObservable(this.getRehabRequest).pipe(
    switchMap((request) => {
      this.loading.set(true);
      let params = new HttpParams().append('userId', request.userId);

      if (request.nameSearchTerm) {
        params = params.append('nameSearchTerm', request.nameSearchTerm);
      }

      if (request.rehabTypeId) {
        params = params.append('rehabTypeId', request.rehabTypeId);
      }

      return this.httpClient.get<AdministrationRehab[]>(
        `${this.API_URL}/rehab/administration/author`,
```

```
        {
          params,
        }
      );
    }),
    tap((list) => {
      this.rehabs.set(list);
      this.loading.set(false);
    })
  );

  readOnlyFacts = toSignal<AdministrationRehab[], AdministrationRehab[]>(
    this.rehabs$,
    {
      initialValue: [],
    }
  );
```

Programski kod 4.16 *Dohvat liste rehabilitacija*

Na API-ju dohvaćamo sve rehabilitacijske programe, te provjeravamo iz primljenog zahtjeva, da li su parametri za filtriranje tipa i imena programa poslani sa klijentske strane, te ovisno o tome filtriramo dohvaćene programe i vraćamo listu kao odgovor.

```
public async Task<IEnumerable<RehabDto>>
AdministrationGetRehabListForAuthor(AdministrationGetRehabListRequest request)
{
    var query = dbContext.Rehabs
        .Include(x => x.Author)
        .Include(x => x.RehabType)
        .Where(x => x.AuthorId == request.UserId && !x.isDeleted);

    if (!string.IsNullOrWhiteSpace(request.NameSearchTerm))
    {
        query = query.Where(x =>
x.Name.ToLower().Contains(request.NameSearchTerm.ToLower()));
    }

    if (request.RehabTypeId != null)
    {
        query = query.Where(x => x.RehabTypeId == request.RehabTypeId);
    }

    var result = await query.ToListAsync();
    var response = mapper.Map<IEnumerable<RehabDto>>(result);

    return response;
}
```

Programski kod 4.17 *Funkcija za dohvat liste rehabilitacija*

4.4.4. Pretplata na program

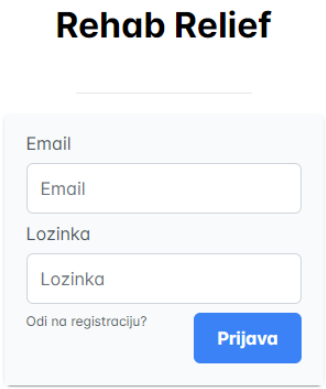
Pri pretplati na program, šalje se zahtjev prema API-u koji sadrži *Id* trenutnog korisnika i *Id* rehabilitacijskog programa na koji se korisnik želi pretplatiti. Prvo je potrebno kreirati vezu između *Rehabs* i *Users* tablice. Ta veza je više naprema više (M-N) koja se implementira pomoću dodatne relacijske tablice *RehabUser*, dodavajući novi zapis u tablicu sa *Id*-em korisnika i programa. Nakon toga se dohvaća program s odgovarajućim *Id*-em, te se dohvaća pojedini trening i vježba svakog treninga. Kreiraju se *RehabStatus*, *SessionStatus*, i *ExerciseStatus* gdje svaki zapis sadrži status i trenutno stanje pojedinog treninga i vježbe programa prema programskom kodu 4.18.

```
public async Task SubscribeToRehab(SubscribeRequest request)
 {
     var rehabUser = new RehabUser(request.UserId, request.RehabId);
     var sessions = dbContext.Sessions.Include(x => x.Exercises).Where(x =>
x.RehabId == request.RehabId).ToList();
     var exercises = sessions.SelectMany(s =>  s.Exercises!);

     var rehabStatus = new RehabStatus(request.UserId, request.RehabId);
     var sessionSatuses = sessions.Select(x => new SessionStatus(request.UserId,
x.Id));
     var exerciseStatuses = exercises.Select(x => new ExerciseStatus(request.UserId,
x.Id));

     await dbContext.AddAsync(rehabStatus);
     await dbContext.AddAsync(rehabUser);
     await dbContext.AddRangeAsync(sessionSatuses);
     await dbContext.AddRangeAsync(exerciseStatuses);
     await dbContext.SaveChangesAsync();

 }
```

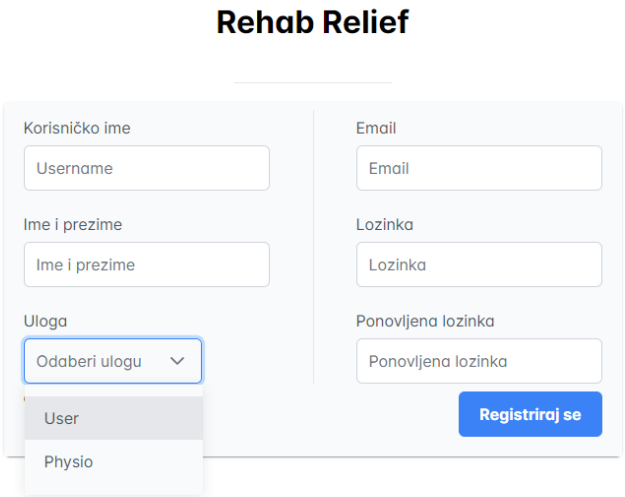
Programski kod 4.18 *Pretplata na rehabilitaciju*

5. KORIŠTENJE PROGRAMSKOG RJEŠENJA

Pristupom na aplikaciju se prikazuje sučelje za prijavu priakzano na slici 5.1, klikom na tekst „Odi na registraciju“, korisniku se prikazuje sučelje za registraciju prikazano na slici 5.2, gdje ima mogućnost odabira uloge, fizijatra ili običnog korisnika.



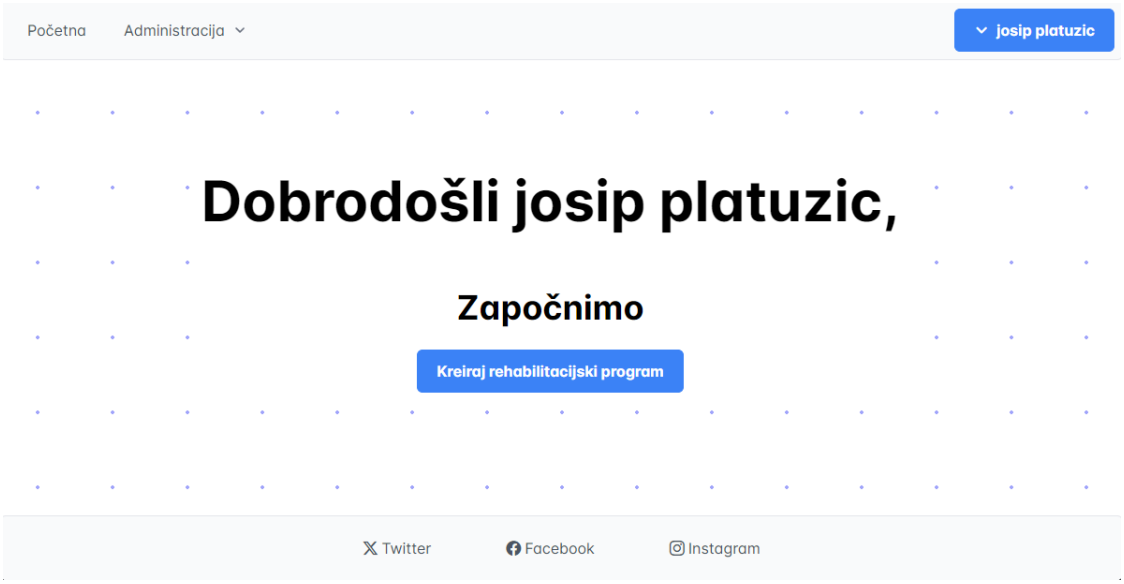
Slika 5.1 *Prijava korisnika*



Slika 5.2 *Registracija korisnika*

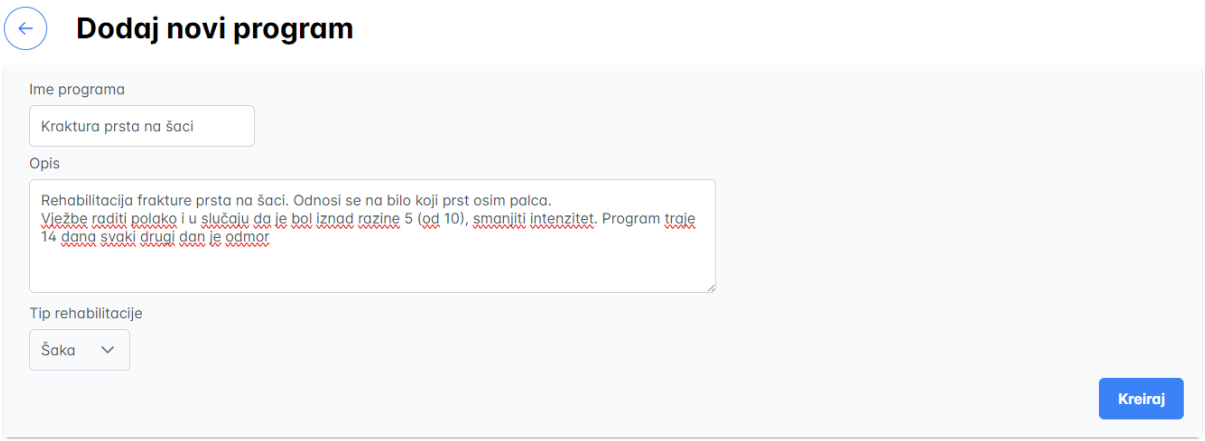
5.1. Korisničko iskustvo uloge fizijatar

Prijavom korisnika prikazuje mu se početna stranica prikazana na slici 5.3.



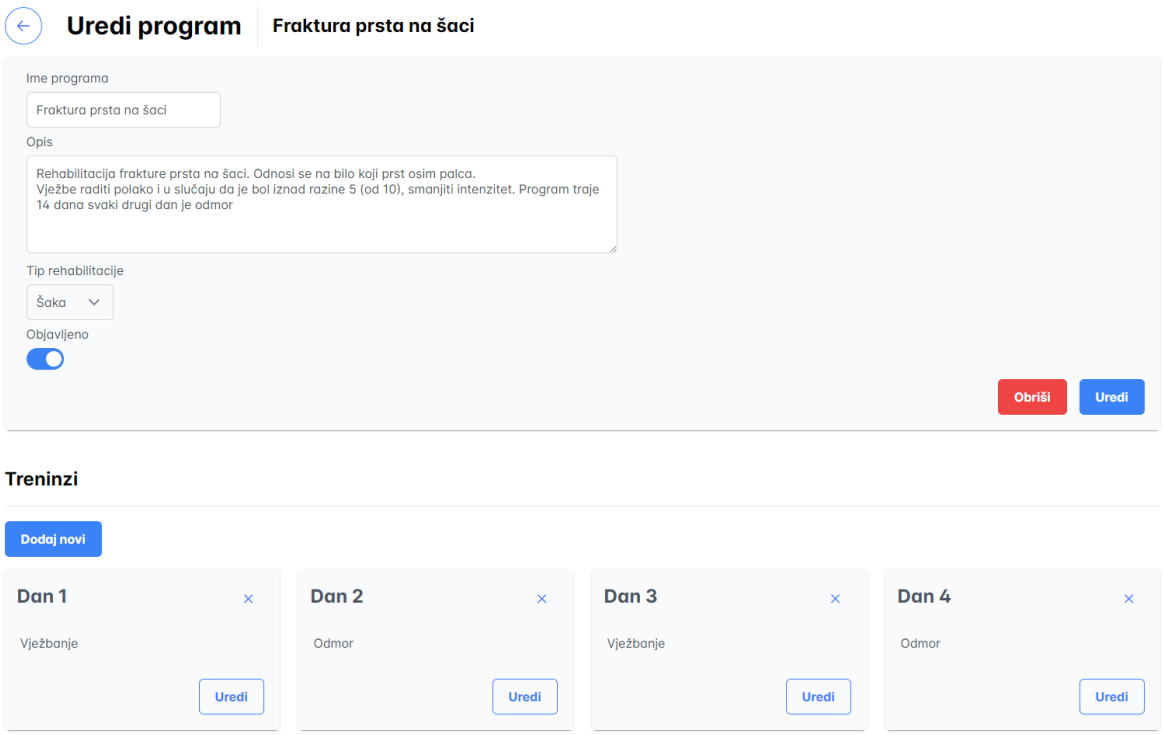
Slika 5.3 *Početna stranica*

Ako je prijavljeni korisnik sa ulogom fizijatar, omogućeno mu je dodavanje novih programa. Na slici 5.4 vidimo formu za dodavanje, gdje se definira ime i opis programa kao i tip rehabilitacije.



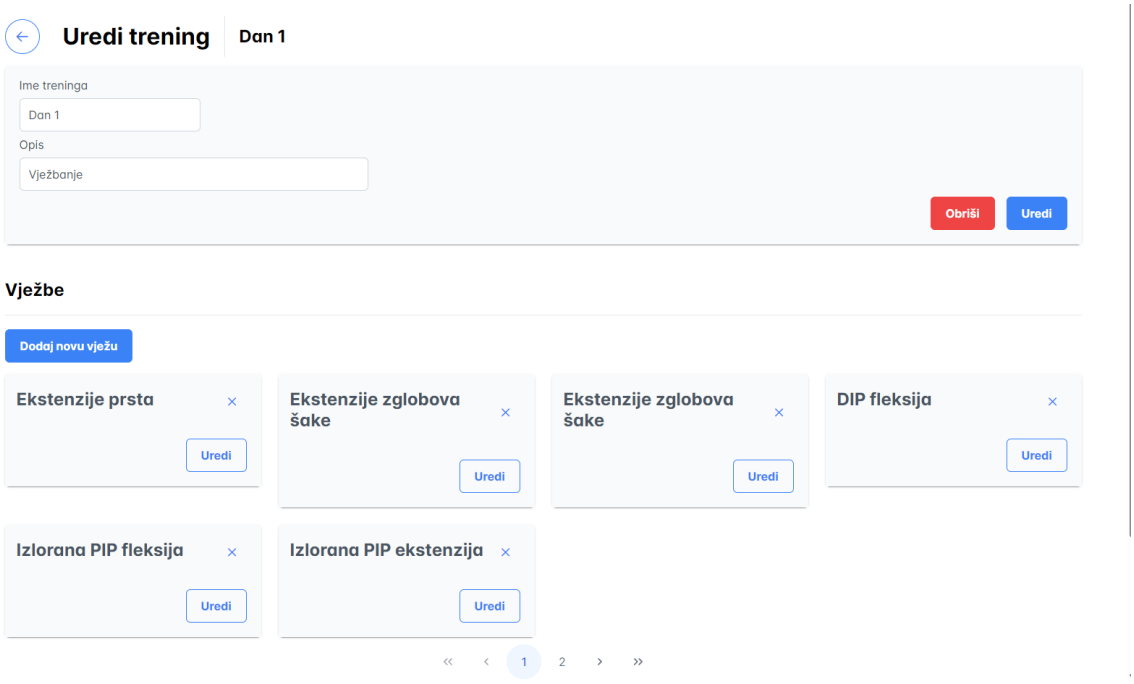
Slika 5.4 *Dodavanje novog programa*

Kreiranjem programa, korisniku se odma prikazuje sučelje za uređivanje kreiranog programa kao i prikaz treninga koje je moguće dodati ili obrisati po želji, kao i sami rehabilitacijski program, prikazano na slici 5.5.

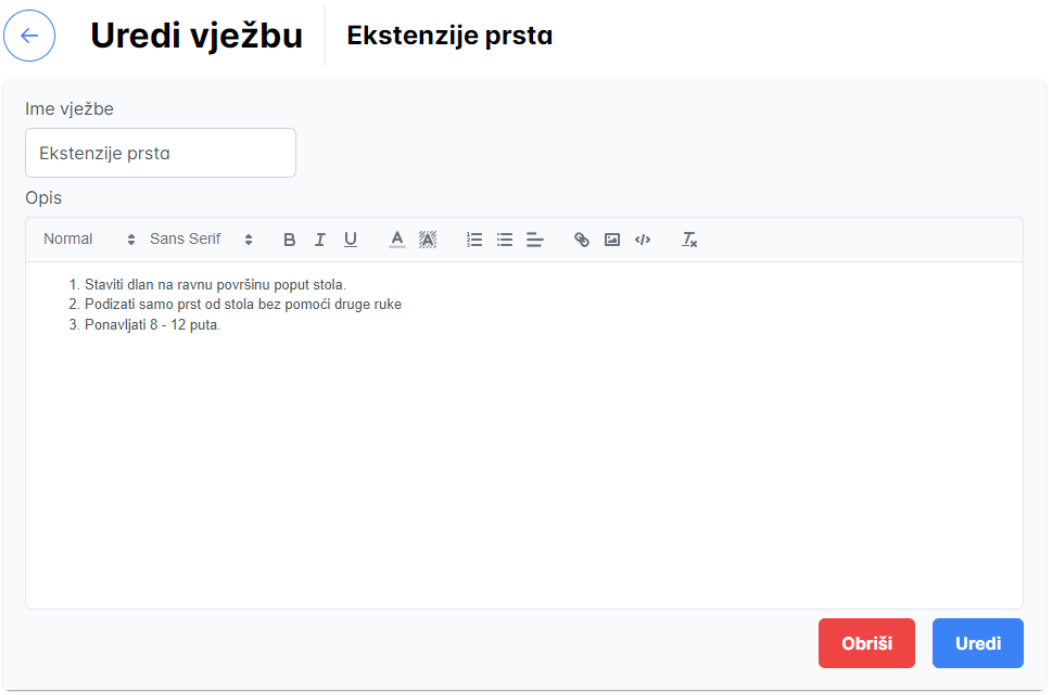


Slika 5.5 *Uređivanje programa*

Nastavkom na sučelje za uređivanje treninga prikazanom na slici 5.6, imamo mogućnost dodavanja novih i uređivanja postojećih vježbi za trenutno odabrani trening, prikazano na slici 5.7.

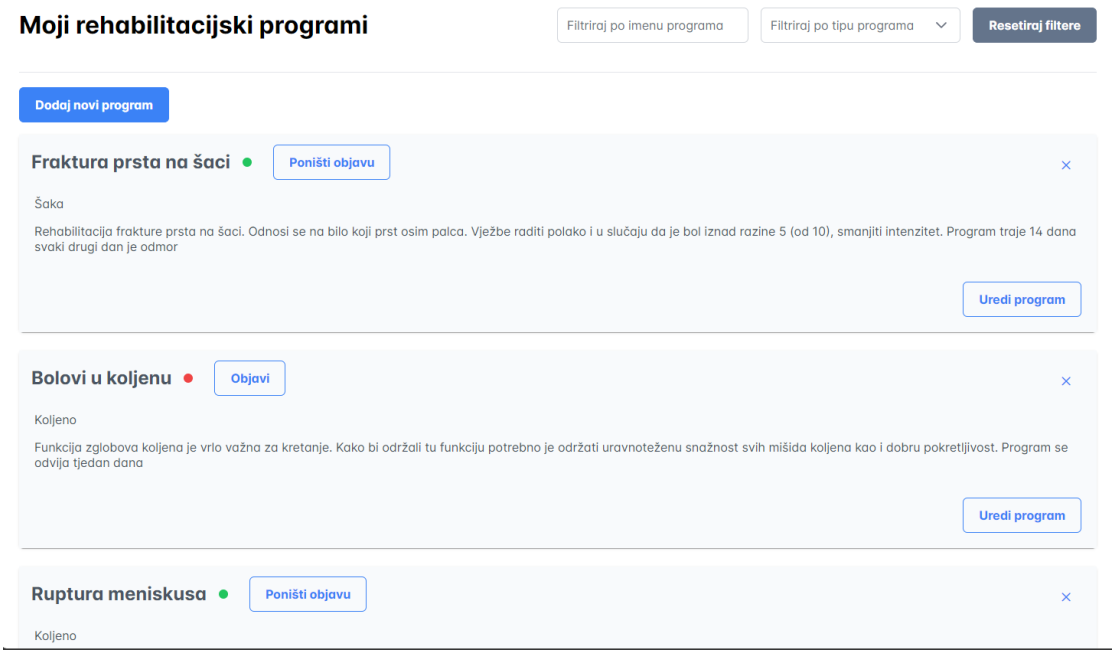


Slika 5.6 *Uređivanje treninga*



Slika 5.7 *Uređivanje vježbe*

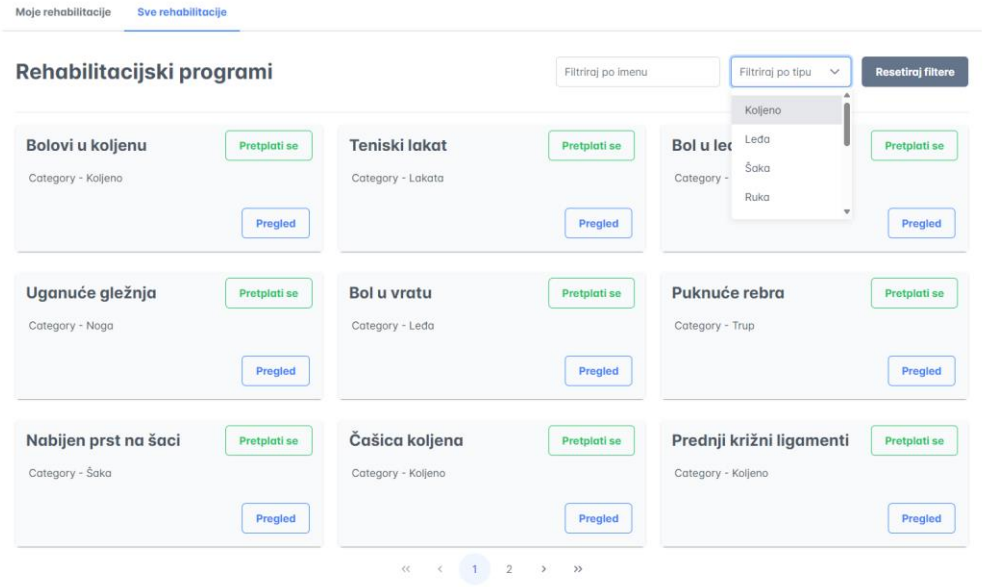
Fizijatar također ima pregled svih svojih kreiranih programa i mogućnost filtriranja po imenu i tipu programa. Svaki rehabilitacijski program može objaviti javno ili poništiti objavu , kao što je prikazano na slici 5.8.



Slika 5.8 *Pregled kreiranih programa*

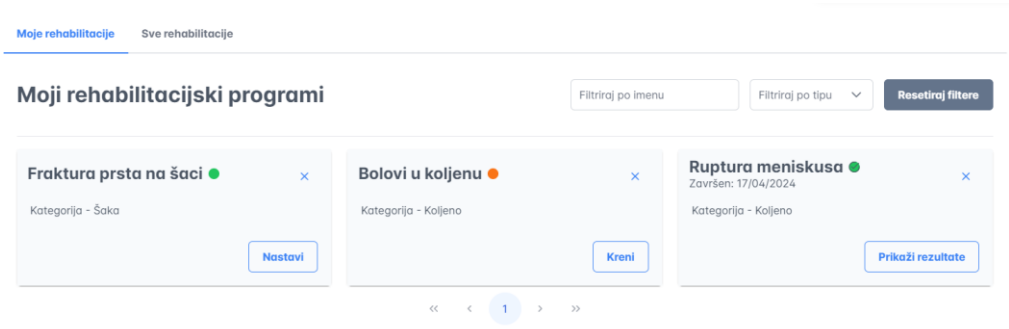
5.2. Korisničko iskustvo uloge korisnik

Prijavom običnog korisnika, prikazuju mu se svi rehabilitacijski programi prema slici 5.9. Korisnik ima mogućnost filtriranja po imenu i tipu programa. Može dobiti uvid u program gdje mu se prikazuje opis programa. Pretplatom na program, program se dodaje u „Moje rehabilitacije“, i omogućava korisniku praćenje napretka i obavljanje programa.



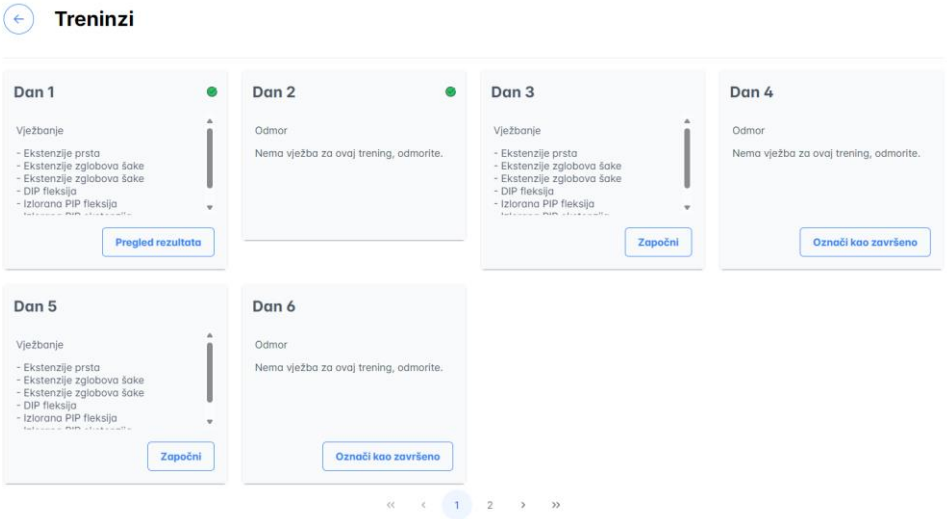
Slika 5.9 *Pregled svih rehabilitacijskih programa*

Na slici 5.10 vidimo programe na koje je korisnik trenutno pretplaćen. Označeno ikonom razlikujemo aktivne, završene i programe u statusu čekanja. Započinjanjem programa u statusu čekanja on se prebacuje u status aktivan.



Slika 5.10 *Pregled pretplaćenih programa*

Pristupom programu, prikazuju se svi treninzi programa sa pregledom vježba pojedinog treninga prema slici 5.11.



Slika 5.11 *Pregled treninga programa*

Započinjanjem vježbe prikazuje se sučelje treninga, gdje se prikazuju vježbe jedna po jedna gdje je na svakoj vježbi moguće ostaviti osobno izvješće, završetkom svih vježbi, stanje odgovarajućeg trening se prebacuje u završeno, te završavanjem svih treninga stanje cijelog programa se prebacuje u završeno.

Slika 5.12 *Izvršavanje vježbi*

6. ZAKLJUČAK

Predstavljena je aplikacija za fizikalnu rehabilitaciju koja nudi niz funkcionalnosti za olakšavanje procesa rehabilitacije. Kreirani su zahtjevi za aplikaciju gdje su navedene glavne funkcionalnosti koje aplikacija mora sadržavati da bi ispunjavala ideju i učinila iskustvo korisnika pozitivno. Predstavljena je arhitektura projekta pomoću koje se mogu implementirati zahtjevi koji uključuju detaljno kreiranje programa i plana programa, uključujući vježbe i upute od strane fizijatra kojemu je omogućeno javno dijeliti svoje kreirane programe, dok će korisnik imati mogućnost pregleda i pronalaska određenog programa za rehabilitaciju koji odgovara korisničkim upitima. Aplikacija omogućuje praćenje napretka, što može pomoći korisnicima da ostanu motivirani i da završe program u potpunosti. Kreirali smo i objasnili proces razvoja aplikacije pomoću tehnologija .NET -a za poslužiteljsku aplikaciju, dok smo kao klijentsku aplikaciju koristili Angular okruženje za izradu dinamičkog i interaktivnog korisničkog sučelja. Počelo se od kreiranja oba okruženja, pa kreiranje baze podataka pomoću *Entity Framework* biblioteke te se komunikacija između dvije okoline odvija pomoću REST API-a, pružajući dobar temelj za moderno web rješenje.

LITERATURA

[1] Bodybuilding.com, https://www.bodybuilding.com/ (posjećeno 19.4.2024)

[2] Physiopedia, https://members.physio-pedia.com/why-physioplus/ (posjećeno 19.4.2024)

[3] About Phisiopedia, https://www.physio-pedia.com/Physiopedia:About (posjećeno 19.4.2024)

[4] [P]Rehab, https://theprehabguys.com/about/ (posjećeno 19.4.2024)

[5] Angular arhitecture, https://angular.io/guide/architecture (posjećeno 22.4.2024)

[6] NET Core intruduction, https://learn.microsoft.com/en-us/dotnet/core/introduction (posjećeno 22.4.2024)

[7] NET CLR, https://learn.microsoft.com/en-us/dotnet/standard/clr (posjećeno 22.4.2024)

[8] Microsoft, What is .NET, https://dotnet.microsoft.com/en-us/learn/dotnet/what-is-dotnet-framework (posjećeno 23.4.2024)

[9] Microsoft, EF Core,https://learn.microsoft.com/en-us/ef/core/ (posjećeno 23.4.2024)

[10] What is mysql, https://www.oracle.com/mysql/what-is-mysql/ (posjećeno 23.4.2024)

[11] Mysql doc, https://dev.mysql.com/doc/refman/8.0/en/partitioning-limitations-partitioning-keys-unique-keys.html (posjećeno 24.4.2024)

[12] Devart, https://blog.devart.com/types-of-relationships-in-sql-server-database.html (posjećeno 24.4.2024)

[13] About NodeJs, https://nodejs.org/en/about (posjećeno 25.4.2024)

[14] JWT Token introtuction, https://jwt.io/introduction (posjećeno 25.4.2024)

[15] Angular signals, https://angular.io/guide/signals (posjećeno 25.4.2024)

SAŽETAK

Cilj diplomskog rada je razvoj aplikacije za fizikalnu rehabilitaciju, koristeći .NET i Angular tehnologije. Tema aplikacije je olakšati proces fizikalne rehabilitacije, osiguravajući time fizijatarima i klijenatima platformu za korištenje. Fizijatri imaju mogućnost kreiranja i sastavljanja rehabilitacijskog programa te mogućnost prilaganja informacija i uputa, te ga objaviti javno korisnicima na raspolaganje. Korisnici imaju uvid u sve rehabilitacijske programe objavljeno javno od strane fizijatara, te mogućnosti pretplate na program, čime omogućuju praćenja napretka programa. Web aplikacije je razvijena u tehnologijama Angular i .NET-u.

Ključne riječi: Angular, C#, Entity Framework, rehabilitacija, Typescript, .NET Framework.

ABSTRACT

Angular web application for physical therapy and rehabilitation

The goal of this thesis is to develop a physical rehabilitation web aplication using .NET and Angular technologies. The application's purpose is to facilitate the physical rehabilitation process by providing a platform for both physiotherapists and clients. Physioterapists have the ability to create and deseign rehabilitation programs, attach information and instructions, and publish them publicly for users. Users can view all publicly available rehabilitation programs created by physiotherapists, subscribe to programs, and track their progress. The web application is developed using Angular and .NET technologies.

Keywords: Angular, C#, Entity Framework, rehabilitation, Typescript, .NET Framework.

PRILOZI

Prilog 1. Programsko rješenje aplikacije: Jokas1999/Diplomski at dev (github.com).