Razvoj web aplikacije u programskom jeziku TypeScript primjenom NextJS okvira

Fric, Matija

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Fric

Razvoj Web aplikacije u programskom jeziku TypeScript primjenom NextJS okvira

ZAVRŠNI RAD

Varaždin, 2023.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Matija Fric

JMBAG: 0016146624

Studij: Informacijski i poslovni sustavi

Razvoj Web aplikacije u programskom jeziku TypeScript primjenom NextJS okvira

ZAVRŠNI/DIPLOMSKI RAD

Mentor:

doc. dr. sc. Matija Novak

Varaždin, srpanj 2023.

Matija Fric

Izjava o izvornosti

Izjavljujem da je moj završni/diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor/Autorica potvrdio/potvrdila prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Tema ovog rada je prikazati *NextJS* kao okvir paketa tehnologija (eng. *full-stack*) za razvoj Web aplikacija. Rad se bavi opisom te načinom korištenja *NextJS* okvira u programskom jeziku *TypeScript* te upućuje na potencijalne prednosti i nedostatke samog okvira kao i prednosti i nedostatke programskog jezika *TypeScript*. Kompletan okvir bit će detaljno opisan u radu i popraćen s primjerima programskog koda. U sklopu rada bit će razvijena i opisana kompletna Web aplikacija na temu podrške poslovanja teretana. Za izradu aplikacije bit će korišten *NextJS* okvir za razvoj pomoću paketa tehnologija (eng. *full-stack)* te *SQLite* baza podataka.

Ključne riječi: NextJS; TypeScript; Web aplikacija; okvir; SQLite

Sadržaj

1. Uvod

Razvoj Web aplikacija vrlo je popularan u suvremenom svijetu te primjena *TypeScript-a* također raste u popularnosti. *TypeScript* nadograđuje *JavaScript* te pruža programerima lakše razumijevanje programskog koda zbog mogućnosti tipizacije. Naravno nije to jedina prednost *TypeScript-a*, no može se reći kako je tipizacija jedna od njegovih glavnih prednosti.

Izrada Web aplikacija pomoću paketa tehnologija (eng. *full-stack*) pridonosi smanjenju potrebe za uporabom više različitih okvira i biblioteka za razvoj Web aplikacija na klijentskoj (*front-end*) i serverskoj (*back-end*) strani što smanjuje kompleksnost izrade same Web aplikacije te pridonosi povećanju produktivnosti programera. Svi okviri za razvoj Web aplikacija pomoću paketa tehnologija (eng. *full-stack*) temelje se na postojećim bibliotekama za razvoj na strani klijenta pa se tako *NextJS* temelji na *ReactJS* biblioteci, dok se primjerice *NuxtJS* temelji na *VueJS* biblioteci, a *NestJS* na *Angular* biblioteci.

U ovom radu opisan je kompletan *NextJS* okvir te je prikazan u procesu razvoja kompletne Web aplikacije te će biti prikazani koraci potrebni za uspostavljanje *NextJS* projekta. Osim toga, razmotrit će se najbolje prakse za razvoj Web aplikacija pomoću *TypeScript-a* i *NextJS-a* kako bi se osigurala visoka kvaliteta same Web aplikacije. Također će se razmotriti osnovni koncepti *TypeScript-a* koji će se primijeniti u razvoju Web aplikacije.

Rad ne uključuje opis osnovnih Web tehnologija (HTML, CSS i JavaScript) te biblioteka za razvoj na strani klijenta (ReactJS), stoga je za razumijevanje potrebno imati predznanje u području osnovnih Web tehnologija te biblioteke ReactJS.

Za izradu Web aplikacije koja služi kao podrška poslovanja teretane korišteno je razvojno okruženje *Visual Studio Code*, *NextJS* kao okvir paketa tehnologija (eng. *full-stack*) u programskom jeziku *TypeScript* te *SQLite* baza podataka uz nekoliko manjih biblioteka. Kroz praktičan primjer, ovaj završni rad će prikazati glavne prednosti razvoja modernih Web aplikacije pomoću *TypeScript-a* i *NextJS-a.*

Ostatak ovog rada strukturiran je na sljedeći način: Poglavlje 2 opisuje Web aplikacije te njihovu povijest kao i razne principe njihovog razvoja, poglavlje 3 bavi se tehnologijama i jezicima koji se koristi za razvijanje Web aplikacija, poglavlje 4 opisuje *TypeScript* i njegove glavne značajke, tema poglavlja 5 je upoznavanje sa glavnim prednostima i mogućnostima *NextJS-a,* dok je u poglavlju 6 prikazan primjer razvijene Web aplikacije u *NextJS-u*.

2. Web aplikacije

Razvoj Web aplikacija započeo je 1990-tih godina u CERN-u (znanstveni laboratorij u Švicarskoj) kada je Tim Berners-Lee prezentirao prijedlog sustava za razmjenu informacija koji bi omogućio dijeljenje resursa i znanja putem interneta. Od samog početka Interneta, glavna zamisao korištenja bila je korištenje Interneta kao univerzalnog medija za razmjenu informacija između računala i računalnih mreža. Sam sustav kasnije se proširio u ono što se danas može nazvati *World Wide Web* [1].

Tim Berners-Lee odlučio je implementirati *hypertext* sistem kako bi povezao dokumente te omogućio praćenje referenca do njihovih izvora. Ideja *hypertext* sistema nije bila nova, no Berners-Lee imao je viziju kreiranja grafičkog sučelja i preglednika za *hypertext* dokumente što je pokrenulo revoluciju u načinu kojim se upravljalo informacijama na Internetu. Iako je razvoj Web-a započeo početkom 1990-tih godina, njegovo široko korištenje počelo je nekoliko godina kasnije što je potaknuto naglim kreiranjem Web aplikacija sredinom 1990-tih godina [2].

Glavna ideja *World Wide Web-a* bila je kreirati virtualnu knjižnicu dokumenata za razmjenu informacija i resursa između znanstvenika, svaki od dokumenata se dohvaćao pomoću jedinstvenog lokatora resursa, URL-a(*Universal Resource Locator)*. Važno je napomenuti kako *Web* nije došao niotkud te je on izgrađen na temelju Internet protokola koji su postojali i prije početka *Web*-a. Razumijevanje povezanosti između *Web-a* i Internet protokola je temelj za dizajniranje i implementaciju Web aplikacija [1].

Eksponencijalni rast *Web-a* može se djelomično pripisati njegovom širenju kao alat za osobno objavljivanje. Temeljna tehnologija koja se krije iza *Web-a* je relativno jednostavna, računalo spojeno na Internet na kojem je pokrenut *Web* server koji poslužuje dokumente. CERN i NCSA (*National Center for Supercomputer Applications*) imali su javno dostupan *Web* server na kojem se uz malo poznavanja HTML-a mogla kreirati vlastita Web stranica [1].

2.1. Web stranice i Web aplikacije

U početku ljudi su koristili Internet u svrhu razmjene statičnih informacija te je bilo vrlo malo pravih dinamičnih Web stranica. Pojavom više dinamičkih Web stranica bilo je potrebno proširiti funkcionalnosti same Web stranice te tako nastaju Web aplikacije. Web aplikacije su korak više od same Web stranice, to su aplikacije koje koriste Web preglednik kao klijentski program za povezivanje s poslužiteljima putem Interneta. Web stranice isporučuju sadržaj statičnih datoteka, dok Web aplikacije predstavljaju dinamički sadržaj baziran na zahtjevima korisnika [1].

Web aplikacija nije obična Web stranica niti obična aplikacija na stolnom računalu. Web aplikacija je negdje između dvojeg koristeći elemente obje strane. Dok sama Web stranica sadrži stranice podataka, Web aplikacija sastoji se od podataka se posebnih mehanizama za njihovu isporuku. Kod Web aplikacija stranice nisu isto kao kod Web stranica, iako se može činiti da Web aplikacija ima 10 stranica, dodavanjem više podataka u bazu podataka povećava se broj stranica bez potrebe za dodavanjem dodatnog izvornog koda na samoj aplikaciji. S raznim mogućnostima kao što je pretraživanje, koje je vođeno od strane samog korisnika, Web aplikacija može imate skoro beskonačan broj stranica [3].

2.2. Principi razvoja Web aplikacija

Dva glavna principa razvoja Web aplikacija su jednostranične (eng. *single-page web applications*) i višestranične (eng. *multipage web applications*) Web aplikacije. Razvoj jednostraničnih Web aplikacije jest relativno novija ideja te radi na principu preuzimanja cijele stranice (ili dobre veličine stranice) u preglednik korisnika što smanjuje potrebe za slanje zahtjeva svakog puta kada korisnik želi pristupiti drugom dijelu Web stranice. Većina današnjih biblioteka i okvira omogućuje jednostavan razvoj jednostraničnih Web aplikacija [4].

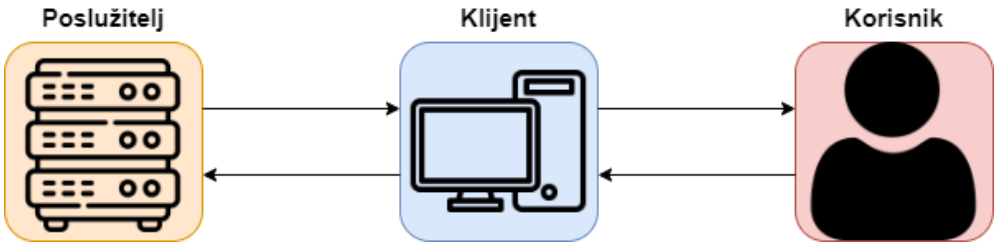
S druge strane, razvoj višestranične Web aplikacije je više tradicionalan pristup kod kojeg se svaka Web stranica poslužuje slanjem zahtjeva na server. To što je ovaj pristup više tradicionalan ne mora značiti da je lošiji od razvoja jednostraničnih Web aplikacija. Danas postoje razni zahtjevi te svaki pristup ima svoje prednosti i nedostatke. Također postoji i mogućnost kombinacije jednostraničnog i višestraničnog razvoja Web aplikacije koji se naziva hibrdni (eng. *hybrid*). Hibridni razvoj Web aplikacije može dio sadržaja posluživati jednostranično dok drugi dio može posluživati višestranično [4].

Osim jednostraničnih i višestraničnih Web aplikacija, postoje još mnogi principi razvoja Web aplikacija. Neki od principa su:

- Prvo mobilni dizajn (eng. *Mobile first*) – Web aplikacije su produkti koji su u početku bili namijenjeni samo za računala. Pojavom pametnih mobilnih uređaja, pretraživanje *Web-a* putem mobilnih preglednika bilo je loše korisničko iskustvo. Ovaj pristup stavlja fokus na razvoj Web aplikacija s poboljšanjem korisničkih iskustva za mobilne uređaje [5].
- Prvo sadržaj princip (eng. *Content first*) – Ovaj princip stavlja fokus na sadržaj same Web aplikacije prije razmišljanja o stilu, tehnologiji i arhitekturi. U idealnom slučaju, sve što se smatra sadržajem mora se prikupiti u početnoj fazi planiranja [6].
- Prvo pristupačnost (eng. *Accessibility first*) – Princip koji stavlja pristupačnost na prvo mjesto promiče dizajniranje Web aplikacija koje su od samog početka pristupačne osobama s invaliditetom što uključuje: pružanje alternativnog teksta za slike, mogućnost navigacije tipkovnicom, kontrast boja, čitljivost fonta i slično [6].
- Razvoj vođen testiranjem (eng. *Test driven development*) – Razvoj vođen testiranjem stavlja u prvi plan pisanje kratkih i brzih jediničnih testova prije pisanja programske logike što osigurava kreiranje skalabilih i održivih Web aplikacija [7].

3. Tehnologije i jezici za razvoj Web aplikacija

Trenutno postoji puno različitih tehnologija i biblioteka za razvoj Web aplikacija. Kada govorimo o tehnologijama valja napomenuti kako postoje dvije strane svake Web aplikacije, to su klijentska strana(*front-end*) i poslužiteljska strana(*back-end*). Svaka strana Web aplikacije se razvija zasebnim tehnologijama i bibliotekama, no postoje i okviri kojima se razvijaju obje strane (*full-stack* okviri).



Slika 1: Struktura Web aplikacije [autorski rad]

3.1. Razvoj na strani klijenta

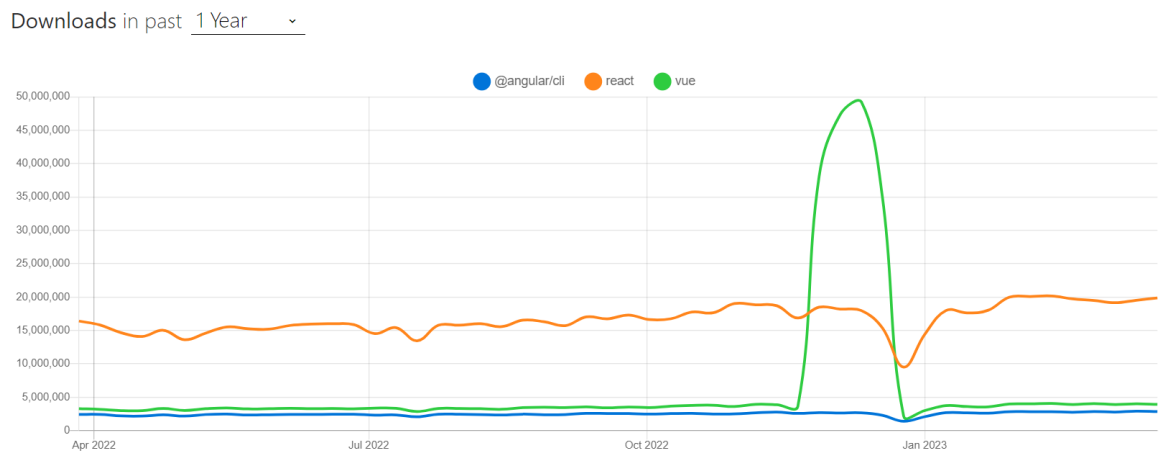
Razvoj na strani klijenta ili popularnije *front-end* razvoj vidljivi je dio Web aplikacije koji se prikazuje krajnjim korisnicima. Glavni fokus razvoja na strani klijent jest interakcija sa korisnikom kao i kreiranje privlačnog, jednostavnog, interaktivnog te responzivnog korisničkog sučelja. Kao što je već napomenuto postoji puno različitih biblioteka koje služe za razvoj na strani klijenta, no na kraju se svaka od njih svodi na tri osnovna jezika:

- HTML (*HyperText Markup Language*) jest osnovni jezik koji služi za izgradnju same strukture Web aplikacija [8].
- CSS (*Cascading Style Sheets*) jest stilski koji služi za opisivanje izgleda dokumenata napisanih u HTML-u, CSS opisuje kako elementi trebaju izgledati na ekranu [9].
- *JavaScript* je programski jezik koji služi za implementiranje funkcionalnosti Web aplikacija te pretvara statične Web stranice u prave dinamične web aplikacije [10].

Web aplikacije se danas razvijaju vrlo rijetko pomoću čistog HTML-a, CSS-a i *JavaScript-a* već postoje mnoge biblioteke i okviri za razvoj na klijentskoj strani koje nude jednostavniji te ponovno iskoristivi razvoj od kojih su tri najpopularnije:

- *React – JavaScript* biblioteka kreirana od inženjera Facebooka kako bi riješili probleme i prepreke vezane uz izgradnju korisničkih sučelja. Temelji se na komponentnom pristupu, gdje se korisničko sučelje igrađuje od ponovno iskoristivih komponenata [11].
- *Angular – JavaScript* okvir otvorenog koda pisan u *TypeScript-u* kreiran od strane Google-a kojem je glavna svrha jednostavan razvoj jednostraničnih Web aplikacija [12].
- *Vue – JavaScript* okvir kreiran od Even You-a koji služi za izgradnju korisničkih sučelja te pruža deklarativni model programiranja temeljen na komponentama. Ističe se po svojoj jednostavnosti te ga to čini popularnim odabirom za programere [13].

Sve biblioteke i okviri imaju određene prednosti i nedostatke te će odabir varirati ovisno o vrsti projekta te osobnim preferencijama dok je na slici 2 prikazan graf godišnjeg preuzimanja prethodno navedenih tehnologija.



Slika 2: Usporedba tehnologija na strani klijenta [14]

3.2. Razvoj na strani poslužitelja

Razvoj na strani poslužitelja ili popularnije *back-end* uključuje kreiranje i upravljanje procesima koji se izvode na Web aplikacijama. Za razliku od razvoja na strani klijenta, razvoj na strani poslužitelja čini ne vidljivi dio Web aplikacije što uključuje: pohranu podataka, sigurnost i razne funkcionalnosti koje se obavljaju na poslužitelju. Programeri na strani poslužitelja moraju imati puno tehničkih znanja, analitičko razmišljanje te izvrsne vještine suradnje [15].

Neki od procesa koje se provode na poslužiteljskoj strani su [15]:

- Razvoj i održavanje Web aplikacije.
- Pisanje kvalitetnog i čitljivog koda.
- Provođenje testiranja.
- Analiza brzine i efikasnosti Web aplikacija.

Za razvoj na strani poslužitelja također je potreban programski jezik i okvir, odabir programskog jezika ovisi o potrebama Web aplikacije. Neki od najpopularnijih jezika te njihovi okviri su [15]:

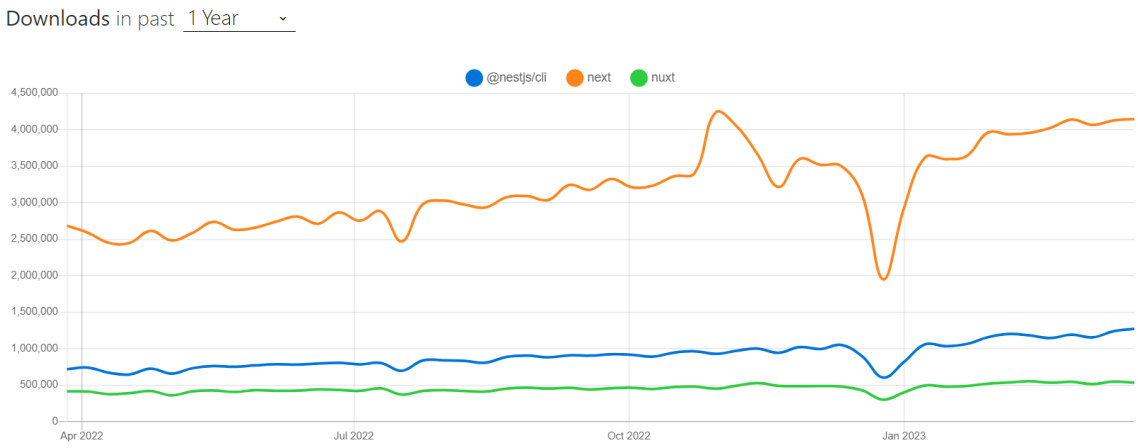
- *Python* (*Django*) – *Django* je okvir koji omogućuje izradu Web aplikacija na strani poslužitelja s relativno malo linija koda. Neke od prednosti *Python* programskog jezika su: robusnost, fleksibilnost te jednostavnost [16].

- PHP (*Laravel*) – *Laravel* je okvir koji pridonosi brzini razvoja te razini sreće programera. Korištenjem *PHP-a* pridonosi čist, jednostavan i održiv programski kod [17].

- *JavaScript* (*Node.js*) – *Node.js* je program koji omogućuje izvršavanje *JavaScript* programskoj jezika na serverskoj strani. Glavne prednosti su brzina i skalabilnost [4].

S ubrzanim razvojem Web aplikacija došla je i potrebna za objedinjenjem razvoja na strani klijenta i poslužitelja u svrhu pojednostavnjivanja razvoja. Kako bi se riješio problem kompleksnosti i smanjila krivulja učenja razvoja na obje strane razvijeni su mnogi takozvani okviri paketa tehnologija (eng. *full-stack*). Većina takvih okvira bazirana je na već postojećim bibliotekama i okvirima na klijentskoj strani kako bi se olakšalo razumijevanje i učenje te integriraju razvoj na strani poslužitelja. Najpopularniji okviri za razvoj pomoću paketa tehnologija (eng. *full-stack*) s pripadajućim bibliotekama i okvirima na klijentskoj strani su:

- NextJS (React)
- NestJS (Angular)
- NuxtJs (Vue)



Slika 3: Usporedba tehnologija na strani poslužitelja [18]

3.3. Usporedba okvira za razvoj Web aplikacija

Danas postoji puno različitih okvira i kombinacija okvira za razvoj Web aplikacija. Ovaj rad će se fokusirati na usporedbu okvira za razvoj Web aplikacija pomoću paketa tehnologija (eng. *full-stack*). Tri najpopularnija takva okvira su: *NuxtJS*, *NextJS*, *NestJS*. Svaki od navedenih okvira ima svoje prednosti i nedostatke koji će biti opisani u nastavku. Odabir okvira također ovisi i o samim preferencijama programera i prethodnom iskustvu.

Prvi takav okvir jest *NuxtJS.* Ovaj okvir je otvorenog koda i nadograđuje se na *VueJS.* Glavni cilj izrade aplikacija pomoću *NuxtJS-a* jest fleksibilnost pri kojem okvir pomaže programerima u kreiranju složenih, brzih i univerzalnih Web aplikacija. U nastavku je navedeno nekoliko stvari u kojima se *NuxtJS* ističe [19].

- Izrada univerzalnih Web aplikacije bez poteškoća – Univerzalne aplikacije su aplikacije u kojima se *JavaScript* kod može izvršiti na klijentskoj kao i na serverskoj strani. *NuxtJS* omogućuje programerima metode kojima mogu dohvatiti podatke na strani poslužitelja u samim komponentama te doprinosi stvaranju univerzalnih aplikacija [19].
- Automatsko dijeljenje koda – *NuxtJS* generira statičnu verziju same Web aplikacije te za svaku stranicu automatski generira program koji kreira vlastitu *JavaScript* datoteku. Također pridonosi povećanju brzine te održavanju veličina *JavaScript* datoteka u odnosu na veličinu same aplikacije [19].
- Jednostavnost pisanja komponenata – Omogućuje jednostavno kreiranje i pisanje komponenata za određene elementa stranica [19].

Prednosti *NuxtJS-a* [20]:

- Optimizacija za tražilice kod jednostraničnih Web aplikacija.
- Mobilne performanse.
- Jednostavna mogućnost kreiranje statičkih Web stranica.
- Automatsko razdvajanje programskog koda.

Nedostaci *NuxtJS-a* [20]:

- Nedostatak uobičajenih dodataka (eng. *plugins*).
- Mala zajednica.
- Otklanjanje pogrešaka je teže.

Drugi popularan okvir koji se temelji na *Angularu* jest *NestJS*. Ovaj okvir se koristi za kreiranje efikasnih, skalabilnih *NodeJS* aplikacija. Kombinira različite elemente od objektno orijentiranog programiranja do funkcionalno orijentiranog programiranja. Kao i kod *Angulara, NestJS* koristi *TypeScript*. U nastavku su navedene neke od značajka *NestJS-a* [19].

- *NestJS* je otvorenog koda.
- Ima dobar naredbeni redak koji povećava produktivnost programera.
- Koristi *TypeScript.*

NestJS pruža programerima gotovu strukturu aplikacije koja im omogućuje da kreiraju skalabilne aplikacije koje se lako testiraju i održavaju [19].

Prednosti *NestJS-a* [21]:

- Koristi *TypeScript.*
- Olakšava testiranje.
- Dovoljno dokumentacije.

Nedostac**i** *NestJS-a* [21]:

- Cirkularne ovisnosti.
- Strmija krivulja učenja.

Treći popularan okvir jest *NextJS* koji koristi *React* za kreiranje Web stranica na klijentskoj strani. Prema mišljenju programera, *NextJS* je najbolji okvir za izradu Web stranica jer ima puno prednosti i značajka koje ga čini prvim izborom. Jedna od stvari po kojoj se razlikuje od prethodno navedenih okvira jest da ne treba nikakvu pred konfiguraciju. *NextJS* dolazi sa svojom konfiguracijom te pomoću osnovnog znanja *React-a* i *JavaScript-a* omogućuje izgradnju kompletnih Web aplikacija [19]. U nastavku ovog rada bit će objašnjen kompletan *NextJS* okvir.

Prednosti *NextJS-a* [22]:

- Jako dobre performanse.
- Puno dokumentacije i podrške programerima.
- Jako dobra iskustva korisnika.
- Dobro optimiziran za tražilice.

Nedostaci *NextJS-a* [22]:

- Neki programi žale se na tvrdoglavost.
- Programeri se žale na način rada preusmjeravanja (eng. *routing*).

4. TypeScript

TypeScript je programski jezik razvijen od strane Microsofta koji nadograđuje i poboljšava *JavaScript. TypeScript* će programeru pomoći učiniti program sigurnijim i razumljivijim zbog mogućnosti provjere uobičajenih pogrešaka prije izvršavanja koda. *Typescript* također pridonosi povećanju produktivnosti programera kao i smanjenju potrebe za testiranjem [23].

Nekoliko čestih pogrešaka koje *TypeScript* može otkriti [23]:

- Zbrajanje brojeva i liste.
- Pozivanje funkcije sa parametrom krivog tipa.
- Pozivanje metoda nekog objekta koja ne postoji.

Za razliku od samog *JavaScript-a* koji će navedene pogreške pokušati izbjeći, *TypeScript* će baciti pogrešku te opisati problem kako bi pomogao programerima u hvatanju grešaka. Sve pogreške koje *JavaScript* izbjegava će se naknadno pojaviti nakon pokretanja samog programa ili u najgorem slučaju tek nakon dolaska pravih korisnika [23].

Kompletan *TypeScript* kod se ne izvršava nego se kompajlira u *JavaScript* kod koji se onda normalno izvršava u pregledniku.

4.1. Postavljanje okruženja

Kako bi krenuli sa korištenjem *TypeScript-a*, potrebno ga je instalirati korištenjem nekog od upravitelja paketa kao što je *npm.* Komandom `npm install -g typescript` pokreće se instalacija *TypeScript-a*.

Svaki *TypeScript* projekt mora sadržavati datoteku *tsconfig.json* u svojem korijenskom direktoriju. Sljedeća dio izvornog koda prikazuje kako izgleda sadržaj *tsconfig.json* datoteke.

```
{ „compilerOptions“: {
   „lib“: [„es2015“],
   „module“: „commonjs“,
   „outDir“: „dist“,
   „sourceMap“: true,
   „strict“: true,
   „target“: „es2015“ },
 „include“: [ „src“ ]}
```

Kratki opis sadržaja datoteke:

- *include* – Mapa u kojoj se nalaze *TypeScript* datoteke koje je potrebno kompajlirati.
- *lib* – Upućuje na API koji se koristi u okruženju izvršavanja koda.
- *module* – Koji modul će se koristiti za kompajliranje koda.
- *outDir* – Direktorij u koji će se smjestiti kompajlirani *JavaScript* kod.
- *strict* – Opcija kojom se definira strogost provjere koda.
- *target* – Koju *JavaScript* verziju će kompajler koristiti za kompajliranje koda.

Važno je napomenuti da su ovo samo neke od opcija koje se mogu definirati u datoteci *tsconfig.json* te da sa novijim verzija postoji mogućnost novih opcija.

4.2. Tipovi podataka

Glavna prednost *TypeScript-a* su tipovi podataka koji se mogu dodijeliti na više načina. Sljedeći programski kod prikazuje deklaraciju varijabla sa tipovima podataka:

```
let a: number = 1;
let b: string = „Pozdrav!“;
let c: boolean = true;
```

Drugi način jest prepuštanje dodjele tipova *TypeScript-u* kod inicijalizacije:

```
let a = 1;
let b = „Pozdrav!“;
let c = true;
```

TypeScript kod inicijalizacije sam dodjeljuje tip ovisno o pridruženoj vrijednosti, stroga su tipovi u oba primjera jednaki. Neki od najčešće korištenih tipova podataka su: any, unknown, boolean, number, string…

Postoji i mogućnost kombiniranja više tipova pomoću operatora unije (|), kod kojeg varijabla može poprimiti bilo koji od navedenih tipova:

```
let a: number | string;
a = 5;
a = „Dobar dan!“;
```

4.2.1. Any i unknown

Tip *any* jest tip koji može zamijeti bilo koji tip podataka ukoliko programer ne može otkriti koji tip podataka mu treba. Ovaj tip trebao bi biti zadnja opcija i bi se trebao izbjegavati kada je to moguće [23]. Ukoliko se neka varijabla deklarira sa tipom *any*, onda se toj varijabli može dodati vrijednost bilo kojeg drugog tipa te se može pozvati nepostojeća metoda bez poruke o pogrešci.

```
let a: any = 1;
a += „Dobar dan“;
a.banana();
```

Tip *unknown* sličan je tipu *any* te se koristi u slučajevima kada se tip ne zna prije vremena. Prednost jest što *TypeScript* neće dopustiti programeru rad sa varijablom tipa *unknown* prije provjere njenog tipa, stoga ga je bolje koristiti nego *any* [23]. Provjera tipa vrši se operatorom *typeof*.

```
let a: unknown = 30;
if(typeof a === „number“)
  let a += 2;
```

4.2.2. Polja

Kao i kod *JavaScript-a,* polja su specijalni tip objekata koji podržavaju operacije poput spajanja, dodavanja i slično [23]. Polja se mogu tipizirati na dva način, prvi način jest korištenjem znaka „[]“:

```
let a: number[] = [1, 2, 3];
a.push(4);
console.log(a); // [1, 2, 3, 4]
```

Drugi način je korištenjem ključne riječi *Array*<>:

```
let a: Array<number> = [1, 2, 3];
a.push(4);
console.log(a); // [1, 2, 3, 4]
```

4.2.3. Objekti

TypeScript također nudi mogućnost tipiziranja objekata. Tipiziranje objekata omogućuje programera saznanje o vrstama podataka koji će biti prisutni u samim objektima. Osim toga *TypeScript* omogućava automatsko nadopunjavanje koda (eng. *autocompletion*) što može biti vrlo korisno kod velikih objekata. U nastavku je prikazan primjer definiranja tipa objekta osobe.

```
let osoba: {ime: string, prezime: string, godine: number} = {ime: „Ivo“, prezime: „Ivić“, godine: 21};
```

Ukoliko postoji razlog da neko svojstvo objekta bude opcionalno, koristi se operator **?** [23]:

```
let a: {b: number, c?: number};
a = {b: 10};
a = {b: 10, c: 20};
```

Ukoliko se opcionalnom svojstvu ne pridruži vrijednost, poprimit će vrijednost *undefined* [23].

4.2.4. Kreiranje tipova

TypeScript omoguće kreiranje vlastitog tipa podataka korištenjem ključne riječi *type, v*lastiti tip može biti jednostavniji ili složeniji. U nastavku je prikazan jednostavni tip koji može primiti vrijednosti *M* ili *Ž*.

```
type Spol = {'M' | 'Ž'};
type Osoba = {ime: string, prezime: string, spol: Spol};
let a: Osoba = {ime: „Ana“, prezime: „Anić“, spol: „Ž“};
let b: Osoba = {ime: „Ana“, prezime: „Anić“, spol: „B“}; //greška
```

4.3. Funkcije

U *JavaScript-u,* funkcije su objekti prve klase, što znači da se mogu koristiti kao i sve ostale varijable ili objekti što uključuje: pridruživanje varijablama, prosljeđivanje drugim funkcijama i slično. Postoje razne mogućnosti korištenja funkcije, *TypeScript* poboljšava te mogućnosti uvođenje tipova [23].

Ovako izgleda primjer jedne funkcije koja zbraja dva broja:

```
function Zbroji(a: number, b:number): number{
  return a + b;
}
Zbroji(1, 2);
```

Ukoliko je neki parametar funkcije opcionalan, koristi se operator *?*:

```
function Ispis(ime: string, prezime?: string): string{
  return ime + „ „ + prezime;
}
console.log(Ispis(„Ivo“));  //Ispisuje „Ivo undefined“
console.log(Ispis(„Ivo“, „Ivić“));  //Ispisuje „Ivo Ivić“
```

Često se kod funkcija može nalaziti neki parametar koji ima zadanu vrijednost. *TypeScript* je dovoljno pametan da može zaključiti o kojem se tipu zadane vrijednosti radi te ukoliko se kasnije pokuša pridružiti drugi tip vrijednosti javlja pogrešku. U nastavku je prikazana funkcija koja ispisuje datum ovisno o danim parametrima:

```
function Datum(dan: number, mjesec: number, godina = 2023){
  return dan + „.“ + mjesec + „.“ + godina + „.“;
}
console.log(Datum(31,5));  //Ispisuje „31.5.2023.“
console.log(Datum(31,5,2023));  //Ispisuje „31.5.2023.“
console.log(Datum(31,5,“dvadeset“));  //Greška
```

5. NextJS

5.1. Opis NextJS okvira

Kao što je već ranije spomenuto, *NextJS* je okvir za razvoj Web aplikacija korištenjem *React-a*. *NextJS* nudi dodatnu strukturu, značajke i optimizacije za aplikacije. Sam okvir vrlo je fleksibilan i prilagodljiv pojedincu ili grupi programera te omogućuje izgradnju interaktivnih, dinamičnih i brzih Web aplikacija [24].

Glavne značajke *NextJS* okvira su [24]:

- Usmjeravanje (eng. *routing*) – Ruter baziran na datotečnom sustavu izgrađen nad serverskim komponentama.
- Generiranje (eng. *rendering*) – Generiranje na strani poslužitelja i statičko generiranje stranice.
- Dohvaćanje podataka (eng. *data fetching*) – Pojednostavljeno dohvaćanje podataka sa *async/await* podrškom korištenjem *fetch()* API-a.
- Oblikovanje (eng. *styling*) – Podrška preferiran metoda oblikovanja uključujući razne CSS module.
- Optimizacija (eng. *optimizations*) – Optimizacija slika, fonta u svrhu poboljšanja korisničkog iskustva.
- *TypeScript* – Poboljšana podrška *TypeScript*-a.

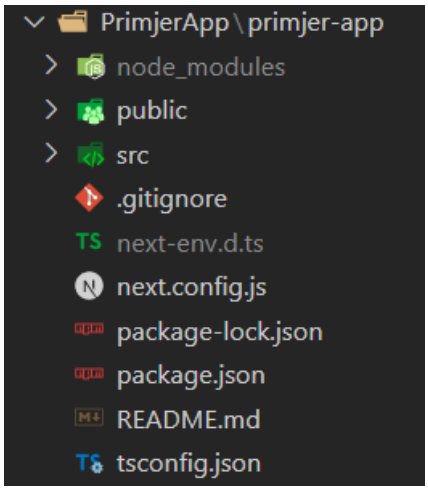
Kako je *NextJS* baziran na *React-u*, za daljnje razumijevanje ovog rada potrebno je osnovno znanje HTML-a, CSS-a, *JavaScipt-a* te *React-a.*

5.2. Instalacija

Kako bi krenuli sa razvojem aplikacije u *NextJS-u,* potrebno je preuzeti okvir. Najbolji način za preuzimanje okvira jest komanda `npx create-next-app@latest` pomoću koje će se nakon preuzimanja svih potrebnih paketa kreirati novi projekt. Prilikom instalacije postoji nekoliko upita od strane terminala koje se ispunjavaju prema preferencijama projekta (npr. Korištenje *TypeScript-a,* ime projekta…) [24].

5.3. Struktura projekta

Nakon kreiranja novog projekta, *NextJS* kreira svoju strukturu datoteka (slika 5). Ovisno o odabranim opcijama u prethodnom koraku, struktura se može malo razlikovati.



Slika 4: Struktura projekta [autorski rad]

Datoteke koje se nalaze u korijenskom direktoriju su [24]:

- *node_modules* – Preuzeti *NodeJS* moduli.
- *public* – Statička sredstva (slike, ikone…).
- *src* – Korijenski direktorij aplikacije.
- *.gitignore* – Git datoteke i mape koje će se ignorirati.
- *next*-env.d.ts – *TypeScript* deklaracijska datoteka za *NextJS.*
- *next.config.js* – Konfiguracijska datoteka za *NextJS.*
- *package-lock.json* i *package.json* – Ovisnosti projekta i skripte.
- *tsconfig.json* – Konfiguracijska datoteka za *TypeScript.*

U datoteci *package.json* nalaze se osnovne skripte za pokretanje projekta [24]:

- *dev* – Pokreće projekt u razvojnom modu
- *build* – Pokreće izgradnju aplikacije za produkciju
- *start* – Pokreće produkcijski server

Postoji mogućnost dodavanja varijabla okoline, *NextJS* ima četiri vrste varijabla okoline [24]:

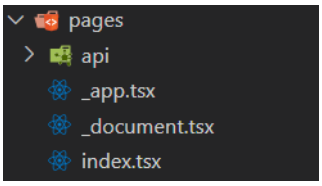
- *.env* – Obične varijable okoline.
- *.env.local* – Lokalne varijable okoline.
- *.env.production* – Produkcijske varijable okoline.
- *.env.development* – Razvojne varijable okoline.

Datoteka *src* sadrži dva glavna direktorija [24]:

- *pages* – sadrži stranice same aplikacije.
- *styles* – sadrži CSS datoteke.

Osim glavnog direktorija, moguće je kreirati i vlastite direktorije (npr. direktorij *components* za komponente i slično).

U *pages* korijenskom direktoriju smještaju se sve datoteke Web aplikacije namijenjene za rad na klijentskoj strani, dok se u *api* direktoriju nalaze datoteke za rad na poslužitelju (slika 6) [24].



Slika 5: Struktura *pages* direktorija [autorski rad]

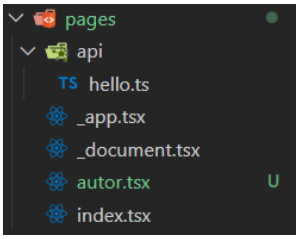
Nekoliko stranica koje se već nalaze u predlošku nakon kreiranja projekta su [24]:

- *_app.tsx* – koristi se za inicijalizaciju stranica (npr. zadržavanje izgleda ili stanja između stranica, globalne CSS datoteke…).
- *_document*.tsx – dokument pomoću koje se mogu uređivati *html* i *body* oznake koje se koriste za prikaz stranica (npr. jezik i slično).
- *Index*.tsx – predložak početne stranice.

Nakon unosa komande `npm run dev`, pokreće se razvojni server sa predloškom stranice na portu 3000.

5.4. Usmjeravanje

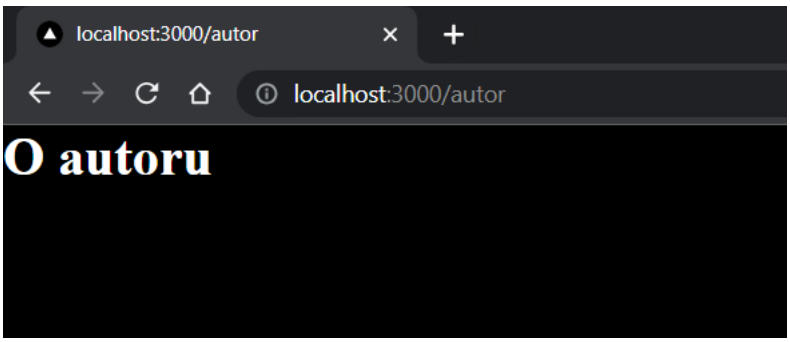
Usmjeravanje je bazirano na stranicama, tako bilo koja datoteka koja se dodaje u *pages* direktorij bit će dostupna kao ruta. U *NextJS-u* stranica je *React* komponenta izvezena iz *.js, .jsx, .ts* ili *.tsx* datoteke. Svaka stranica je povezana sa rutama na temelju svog naziva [24]. U nastavku je primjer kreiranje stranice o autoru na putanji *pages/autor.tsx* (slika 7).



Slika 6: Kreiranje datoteke o autoru [autorski rad]

U datoteci *autor.tsx* nalazi se kod koji kreira *React* komponentu koja ispisuje u naslovu tekst „O autoru“ kao što je prikazano u nastavku te je dostupna na ruti */autor* (slika 8).

```
export default function Autor(){
  return <h1>O Autoru </h1>
}
```



Slika 7: Ruta autora [autorski rad]

Indeksirane rute su također jedna od mogućnosti *NextJS-a*. Ukoliko u datoteci postoji stranica koja se zove *index*, ta se stranica smatra korijenskom putanjom direktorija te će biti dostupna na ruti / [24]. Nekoliko primjera indeksiranih ruta:

- *pages/index.tsx* – Će biti dosupna na /.
- *pages/korisnik/index.tsx* – Će biti dostupna na */korisnik*.

5.4.1. Dinamičko usmjeravanje

Ukoliko unaprijed nisu poznati točni nazivi segmenata kada se žele kreirati rute iz dinamičkih podataka, postoji mogućnost korištenja dinamičkih segmenata koji se popunjavanja u vremenu zahtjeva. Dinamički segmenti kreiraju se omotavanjem imena segmenta u uglate zagrade [24].

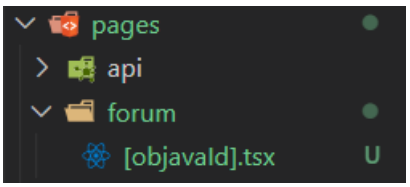
Kako bi se pristupilo dinamičkom segmentu, potrebno je koristiti *useRouter* kuku (eng. *hook*). Ova kuka u svom upitu (eng. *query)* sadrži podatke o dinamičnom segmentu ovisno o nazivu koji je omotan u uglatim zagradama kao što je prikazano u nastavku. Sama struktura datoteka je prikaza na slici broj 9.

Najprije je potrebno uvesti *useRouter* iz biblioteke *next/router.*

```
import {useRouter} from „next/router“;
```

Zatim se kreira nova *React* komponenta u kojoj se inicijalizira usmjernik te vraća paragraf koji sadrži Id objave:

```
export default function Objava(){
  const router = useRouter();
  return <p> Objava {router.query.objavaId} </p>
}
```



Slika 8: Struktura foruma [autorski rad]

5.4.2. Poveznice i navigacija

Usmjernik *NextJS-a* omogućuje izvođenje usmjeravanja između stranica na klijentskoj strani. Postoje dva načina usmjeravanja, jedan koristi *Link* komponentu dok drugi koristi *useRouter* kuku [24].

Kako bi se uspostavila navigacija pomoću *Link* komponente, potrebno je uvesti komponentu iz *next/link* biblioteke.

```
import Link from „next/link“;
```

Sljedeći korak je kreiranje *React* komponente koja sadrži linkove na različite stranice. Primjer u nastavku sadrži linkove na početnu stranicu, stranicu autora te stranicu foruma.

```
export default function Početna() {
  return (
    <div>
      <Link href="/">Početna</Link>
      <Link href="/autor">O autoru</Link>
      <Link href="/forum/vijesti">Forum vijesti</Link>
    </div>
  );
}
```

Drugi način uspostave navigacije je korištenjem *useRouter* kuke. Ovaj način je preporučen od samih autora *NextJS-a* jer pokriva sve mogućnosti i scenarije usmjeravanja [24].

Kako bi se uspostavila navigacija, potrebno je na događaj pritiska miša (eng. *onClick*), pozvati metodu *.push()* usmjernika te joj proslijediti rutu. U nastavku je prikaz identičan primjer kao i kod korištenja *Link* komponente.

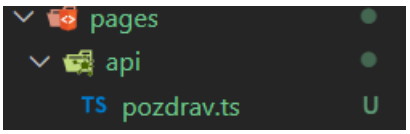
```
import {useRouter} from „next/router“;


export default function Pocetna() {
  const router = useRouter();
  return (
    <div>
      <button onClick={()=>router.push(„/“)}> Početna </button>
      <button onClick={()=>router.push(„/autor“)}> Autor </button>
      <button onClick={()=>router.push(„/forum/vijesti“)}> Forum
vijesti </button>
    </div>
  );
};
```

5.5. API usmjeravanje

API usmjeravanje u *NextJS-u* omogućuje kreiranje vlastitih API-a. Bilo koja datoteka unutar *pages/api* mape smatrat će se kao API krajnja točka (eng. *API endpoint*). API usmjeravanje također podržava indeksirane rute kao i dinamičko usmjeravanje [24]. U nastavku je prikazan primjer API rute koja je smještena u direktoriju *pages/api/pozdav.ts* (slika 10).

Za normalan rad najprije je potrebno uvesti tipove zahtjeva (eng. *request*) i odgovora (eng. *response)* iz *next* biblioteke te kreirati tip odgovora. Nakon toga potrebno je definirati funkciju koja će upravljati zahtjevima i vratiti HTML statusni kod 200 sa porukom odgovora u *json* obliku kao što je prikazano u nastavku.

```
import {NextApiRequest, NextApiResponse) from „next";

type Odgovor = {
  ime: string;
};

export default function handler(req: NextApiRequest, res:
  NextApiResponse<Odgovor>) {
    res.status(200).json({ime: „Ivo Ivić"});
}
```



Slika 9: API struktura [autorski rad]

Ukoliko postoji potreba za rukovanjem različitim HTTP metodama, u zahtjevu odgovora nalazi se svojstvo *.method* u kojem je zapisan tip HTTP metode [24]. Ovisno o tipu metode moguće je odraditi zaseban dio programske logike kao što je prikazano u nastavku.

```
…
export default function handler(req: NextApiRequest, res:
  NextApiResponse<Odgovor>) {
    if(req.method === „GET“){
    //Programska logika GET metode
    }
    else if(req.method === „POST“){
    //Programska logika POST metode
    }
}
```

Metoda zahtjeva (eng. *request*) osim svojstva *.method* ima i druga korisna svojstva [24]:

- *.cookies* – Objekt koji sadrži kolačiće koji su pristigli sa zahtjevom.
- *.query* – Objekt koji sadrži tekst upita.
- *.body* – Objekt koji sadrži tijelo zahtjeva.

Metoda odgovora (eng. *response*) također sadrži nekoliko metoda [24]:

- *.status(code)* – Funkcija koja postavlja HTTP statusni kod odgovora.
- *.json(body)* – Funkcija koja šalje odgovor u JSON obliku.
- *.send(body)* – Funkcija koja šalje odgovor u HTTP obliku, *body* element mora biti tekstualnog tipa ili u obliku objekta.
- *.redirect([status], path)* – Funkcija za preusmjeravanje na specifičan URL, status mora biti validan HTTP kod.

5.6. Generiranje (eng. *rendering*)

Prema zadanim postavkama, *NextJS* unaprijed generira svaku stranicu, što znači da generira HTML kod unaprijed umjesto da taj posao odrađuje klijentski *JavaScript.* Generiranje unaprijed može rezultirati boljim performansama te boljem optimizacijom pretraživanja. Svaki generirani HTML kod je povezan sa minimalnom količinom potrebnog *JavaScript* koda. U trenutku učitavanja stranice od strane preglednika, pokreće se *JavaScript* kod te čini stranicu interaktivnom. Ovaj proces naziva se hidratacija (eng. *hydration*) [24].

U *NextJS-u* postoje dvije vrste generiranja unaprijed koje se razlikuju po vremenu generiranje HTML koda [24]:

- Statično generiranje - HTML kod je generiran u procesu izgradnje (eng. *build*) te je ponovno iskoristiv na svakom zahtjevu.
- Generiranje na poslužitelju – HTML kod se generira na svakom zahtjevu.

5.6.1. Generiranje na poslužitelju

Ukoliko stranica koristi generiranje na poslužiteljskoj strani (eng. *server-side rendering*), HTML stranica jest generirana unaprijed na svakom zahtjevu. Za korištenje generiranja na poslužiteljskoj strani potrebna je asinkrona funkcija koja ima predefinirani naziv *getServerSideProps.* Ova funkcija se poziva od stane poslužitelja na svakom od zahtjeva [24].

U nastavku je prikaz primjer kako iskoristiti generiranje na poslužitelju kada stranica treba unaprijed prikazati podatke sa nekog vanjskog API-a.

```
export default function Stranica({data}){
  //rad sa pristiglim podacima
}
//Funkcija koje se poziva na svakom zahtjevu
export async function getServerSideProps(){
  //Dohvat podataka sa vanjskog API-a
  const odgovor = await fetch(„https://...“);
  const podaci = await odgovor.json();
  return {props: {data}};
}
```

5.6.2. Statičko generiranje stranice

Ukoliko stranica koristi statičko generiranje (eng. *static site generation*), HTML stranica je generirana u procesu izgradnje (eng. *build*). Generirana HTML stranica će se ponovno iskoristiti na svakom od zahtjeva. Za statičko generiranje stranice koristi se asinkrona funkcija koja ima predefinirani naziv *getStaticProps* [24]. U nastavku je prikazan primjer stranice koja dohvaća podatke sa vanjskog API-a.

```
export default function Stranica({data}){
  //Rad sa pristiglim podacima
}
//Funkcija koja se poziva samo u procesu izgradnje
export async function getStaticProps(){
  //Dohvat podataka sa vanjskog API-a
  const odgovor = await fetch(„https://...“);
  const podaci = await odgovor.json();
  return {props: {data}};
}
```

Postoje mnogi slučajevi kada je korisno koristiti statičko generiranje te je preporuka da se ono koristi što je više moguće zbog boljih performansi. Ovo su neki od primjera kada bi bilo dobro koristiti statičko generiranje stranice [24]:

- Stranice marketinga.
- Blog postovi i portfelj stranice.
- Stranice dokumentacije.

Ukoliko stranica mora konstantno prikazivati ažurirajuće podatke, statičko generiranje nije dobra ideja te je u takvim situacijama bolje koristiti generiranje na poslužitelju [24].

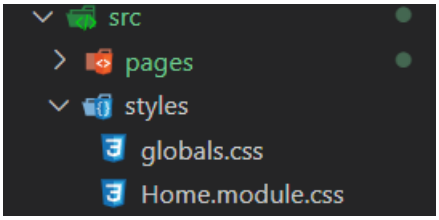
5.7. Stilovi i optimizacija

5.7.1. Stilovi

NextJS nudi mogućnost korištenja različitih načina korištenja CSS-a, neki od načina su [24]:

- *Global CSS* – Jednostavan, običan CSS.
- *CSS modules* – Omogućuje kreiranje lokalnih CSS klasa kako bi se izbjegli konflikti u nazivima.
- *Tailwind CSS* – CSS okvir koji omogućuje brzi dizajn korištenjem gotovih klasa.
- *Sass* – Popularni CSS predprocesor koji proširuje mogućnosti samog CSS-a.
- *CSS-in-JS* – Omogućuje kreiranje CSS-a direktno u komponentama.

Sve CSS datoteke nalaze se u svojem direktoriju unutar *style* mape (slika 11).



Slika 10: CSS *styles* mapa [autorski rad]

5.7.2. Optimizacija

NextJS dolazi sa više različitih optimiziranih dizajna i komponenata kako bi se poboljšala brzina same aplikacije [24].

Prva od takvih komponenta jest *Image* komponenta koja optimizira običan <*img*> element na način da dodaje lijeno učitavanje (eng. *lazy loading*) te automatsku promjenu veličine ovisno o veličini uređaja [24]. Kako bi se koristila *Image* komponenta, potrebno je istu uvesti iz *next/image* biblioteke. Sama komponenta sadrži sva svojstva koja sadrži običan <*img*> element uz neka dodatna svojstva.

```
import Image from „next/image“;
…
<Image
  src={//izvor}
  alt=“Tekst koji zamjenjuje sliku“ />
…
```

Sljedeća mogućnost optimizacije jest optimizacija fonta. Biblioteka *next/font* će automatski optimizirati font te ukloniti vanjske mrežne zahtjeve kako bi se poboljšale performanse te povećala privatnost [24]. U nastavku je prikazan primjer korištenja *Inter* fonta.

```
import {Inter} from „next/font/google“;
//Ukoliko postoji više podskupa fonta
const inter = Inter({ subsets: [„latin“]});

export default function Pocetna(){
  return (
    <p className={inter.className}> Početna! </p>
  );
}
```

Također jedna od korisnih komponenata jest komponenta *Script,* koja je nadogradnja samog <*script*> elementa. Ova komponenta omogućuje više kontrole u učitavanju i pokretanju skripata treće strane [24]. U nastavku je prikaz uvoz komponente iz *next/script* biblioteke te njezino korištenje.

```
import Script from „next/script“;

export default function Pocetna(){
  return (
    <div>
      <Script src=“https://...“ />
    </div>
  );
}
```

6. Primjer Web aplikacije korištenjem NextJS okvira

6.1. Opis ideje za izradu praktičnog djela

Web aplikacija kojoj je cilj pomoći fitnes centrima u smanjenju gužve. Aplikacija ima jednu vrstu korisnika koji se prijavljuje, no ovisno o tome što radi na stranici imat će drugačiju ulogu. Svaki korisnik može biti član neke teretane u kojoj će moći odabrati neki od ponuđenih termina, može biti djelatnik određene teretane te dodavati termine za istu teretanu kako bi se korisnici mogli prijaviti u dodani termin te može biti vlasnik teretane ako želi dodati novu teretanu na stranicu.

6.2. Popis funkcionalnost po ulogama

6.2.1. Korisnik

- Svaki korisnik ima pristup početnoj stranici gdje vidi sve opće informacije o Web stranici.
- Korisnik se može registrirati na Web stranici. Registracija se vrši pomoću upisivanja email adrese, imena, prezimena, te lozinke. Nakon uspješnog validiranja podataka korisnik je prebačen na stranicu za unos aktivacijskog koda kojeg je dobio u poruci na svojoj email adresi. Ako korisnik unese ispravan aktivacijski kod, može pristupiti prijavi na Web stranici.
- Svaki registrirani korisnik ima mogućnost prijave pomoću email adrese te odgovarajuće lozinke.
- Svaki prijavljeni korisnik ima mogućnost pregleda svog korisničkog profila, kao i promjenu korisničkih podataka.
- Svaki prijavljeni korisnik ima mogućnost pregleda teretana koje su dodane na Web stranici što uključuje i pretraživanje teretana po nazivu teretane.
- Svaki prijavljeni korisnik ima mogućnost odabira jedne ili više teretana te poslati zahtjev za učlanjivanje.
- Svaki prijavljeni korisnik ima mogućnost kreiranja vlastite teretane, kada korisnik kreira teretanu, postaje vlasnik iste.
- Ako je korisnik član određene teretane, može odabrati jedan od ponuđenih termina vježbanja.
- Ako je korisnik član neke teretane, može poslati zahtjev za djelatnika teretane.

6.2.2. Djelatnik

- Svaki djelatnik teretane ima mogućnost pregleda prijava korisnika u teretanu. Prijava se može prihvatiti ili odbiti.
- Svaki djelatnik teretane ima mogućnost pregleda članova u teretani. Djelatnik ima mogućnost uklanjanja članova.
- Svaki djelatnik teretane ima mogućnost dodavanja termina na koje će korisnici imati mogućnost prijave.
- Svaki djelatnik teretane ima mogućnost uređivanja i brisanja kreiranih termina.
- Svaki djelatnik teretane ima mogućnost pregleda svih prijavljenih korisnika u terminu, kao i mogućnosti brisanja odabira korisnika.
- Svaki djelatnik teretane ima mogućnost pregleda statistike odabranih termina u nekom vremenskom razdoblju za teretanu u kojoj je djelatnik.

6.2.3. Vlasnik

- Svaki vlasnik teretane ima mogućnost uređivanja općih informacija o dodanoj teretani.
- Svaki vlasnik teretane ima pregled prijava za djelatnika teretane. Prijava se može prihvatiti ili odbiti.
- Svaki vlasnik teretane ima mogućnost pregleda svih termina od svih djelatnika kao i mogućnost uređivanja i brisanja svih termina.
- Svaki vlasnik teretane ima mogućnost pregleda djelatnika u teretani. Vlasnik ima mogućnost uklanjanja djelatnika.
- Svaki vlasnik ima mogućnost brisanja vlastite teretane.
- Svaki vlasnik ima mogućnost pregleda statistike posjeta po svim teretanama u kojima je vlasnik.

6.3. Korištene biblioteke i moduli

U nastavku su opisane sve biblioteke i moduli koji su korišteni prilikom izrade same aplikacije. Većina biblioteka i modula korištena je zbog jednostavnije i sigurnije implementacije funkcionalnosti.

- *Tailwind CSS* – Biblioteka koja sadrži gotove CSS klase te omogućuje lakše dizajniranje aplikacije bez potrebe za pisanjem vlastitog CSS-a.

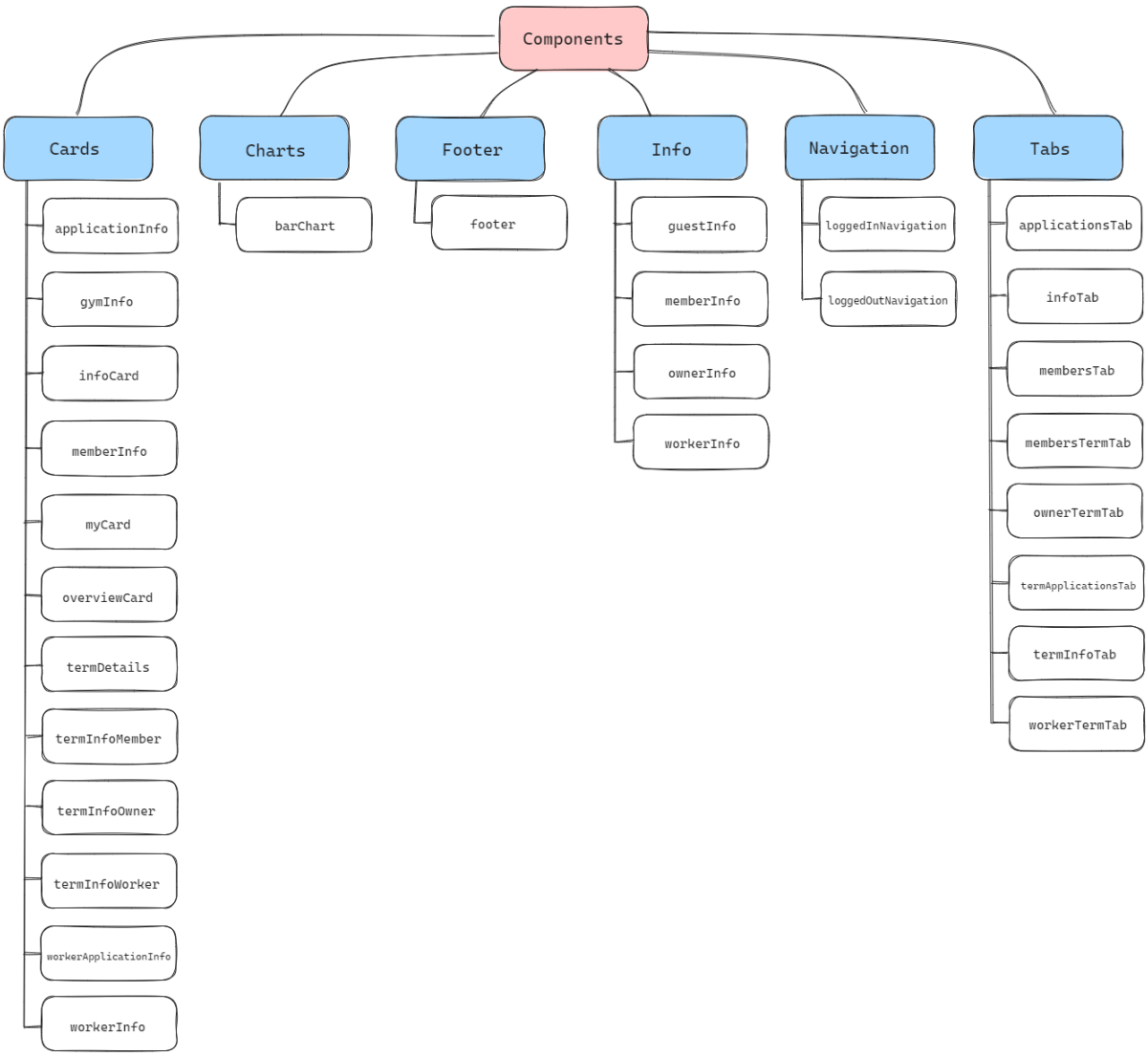
- *Prisma* – ORM (eng. *Object relation mapping*) alat koji omogućuje usklađivanje programskog koda sa bazom podataka.

- *Sqlite* – Lokalna relacijska baza podataka.
- *Axios* – Biblioteka koja služi za kreiranje HTTP zahtjeva.
- *Heroicons* – Biblioteka koja sadrži ikone.
- *Bcrypt* – Biblioteka koja služi za kriptiranje podataka.
- *NextAuth* – Omogućuje lakšu i sigurniju implementaciju autentifikacije.
- *Nodemailer* – Biblioteka za slanje email poruke.
- *DaisyUI* – Dodatak za Tailwind CSS koji sadrži teme i gotove komponente.
- *Chakraui* – Biblioteka koja sadrži gotove komponente poput skočnih prozora.
- *Recharts* – Biblioteka za kreiranje grafova.

6.4. Struktura Web aplikacije

Programiranje na klijentskoj strani je u *React-u* koji se bazira na programiranju sa komponentama. Ovaj završni rad napravljen je na način da bude što pregledniji te da sam programski kod stranice nema previše programske logike, stoga ima puno komponenata koje su ponovno iskoristive. Same komponente podijeljene su u 6 različitih datoteka zbog lakšeg razumijevanja, te datoteke su:

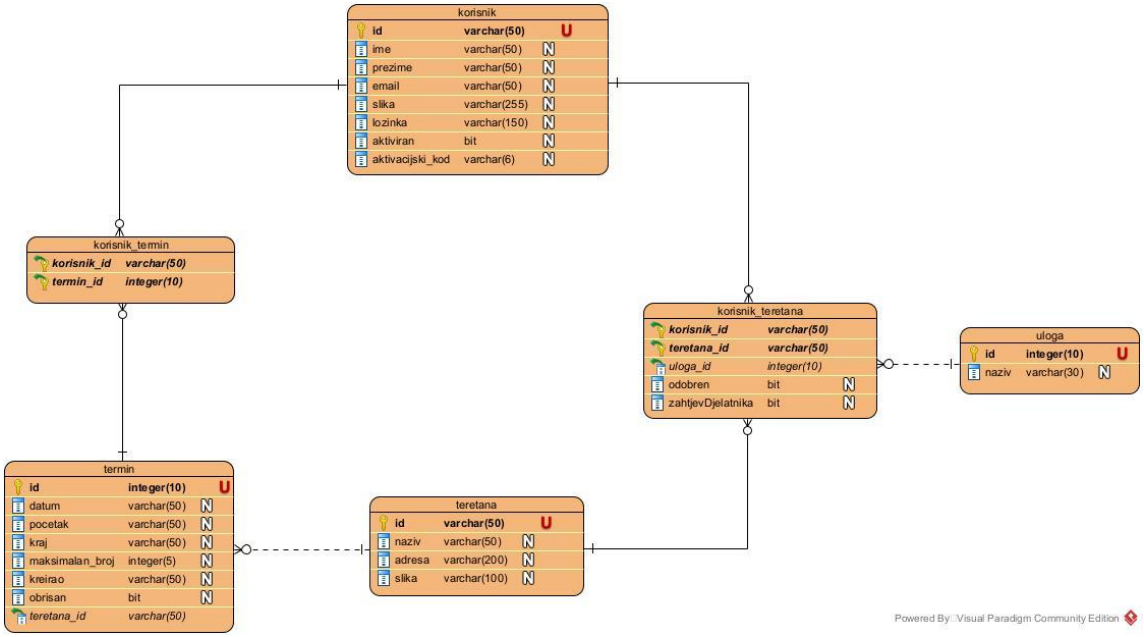
- *Cards* - Sadrži ponovno iskoristive kartice.
- *Charts* - Sadrži komponentu grafa.
- *Footer* - Sadrži komponentu zaglavlja.
- *Info* - Sadrži komponente koje služe za prikaz informacije ovisno o ulozi.
- *Navigation* - Sadrži komponente za navigaciju.
- *Tabs* - Sadrži komponente za prikaz kartica u detaljima teretane.



Slika 11: Dijagram komponenata [autorski rad]

6.5. ERA dijagram

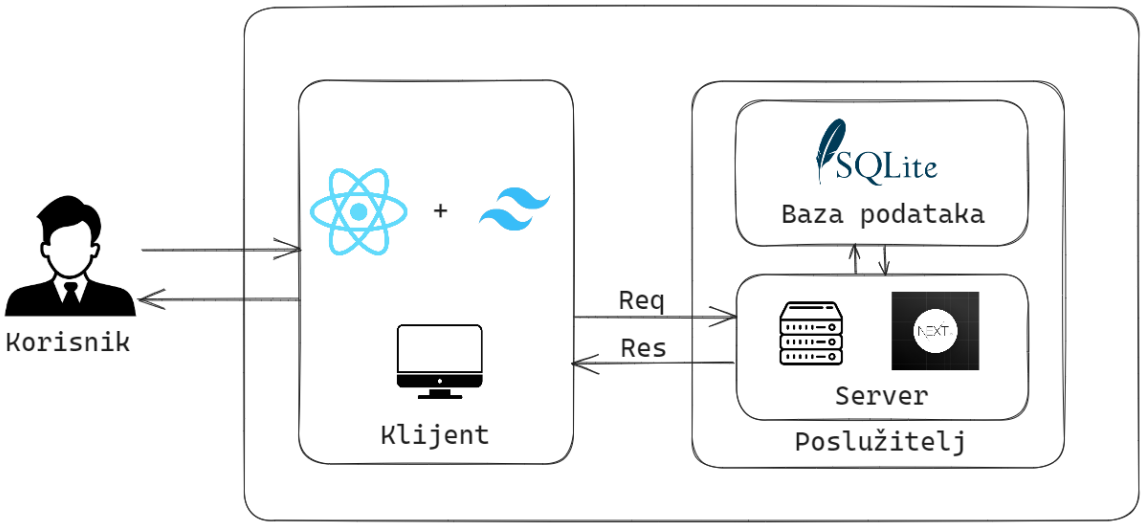
Relacijska baza podataka sastoji se od 6 tablica, to su: *korisnik*, *korisnik_teretana*, *uloga*, *teretana*, *termin* i *korisnik_termin*.



Slika 12: ERA dijagram [autoriski rad]

6.6. Dijagram arhitekture

Arhitekture Web aplikacije sastoji se od klijentske strane (eng. *frontend*) i poslužiteljske strane (eng. *backend*) te korisnika. Korisnik komunicira sa klijentskom stranom koja je napravljena pomoću *React-a* i dizajnirana *Tailwind CSS-om.* Klijentska strana pomoću API (eng. *Application programming interface*) zahtjeva komunicira sa poslužiteljem koji je napravljen u *NextJS-u* te ima svoju lokalnu *SQLite* bazu podataka.



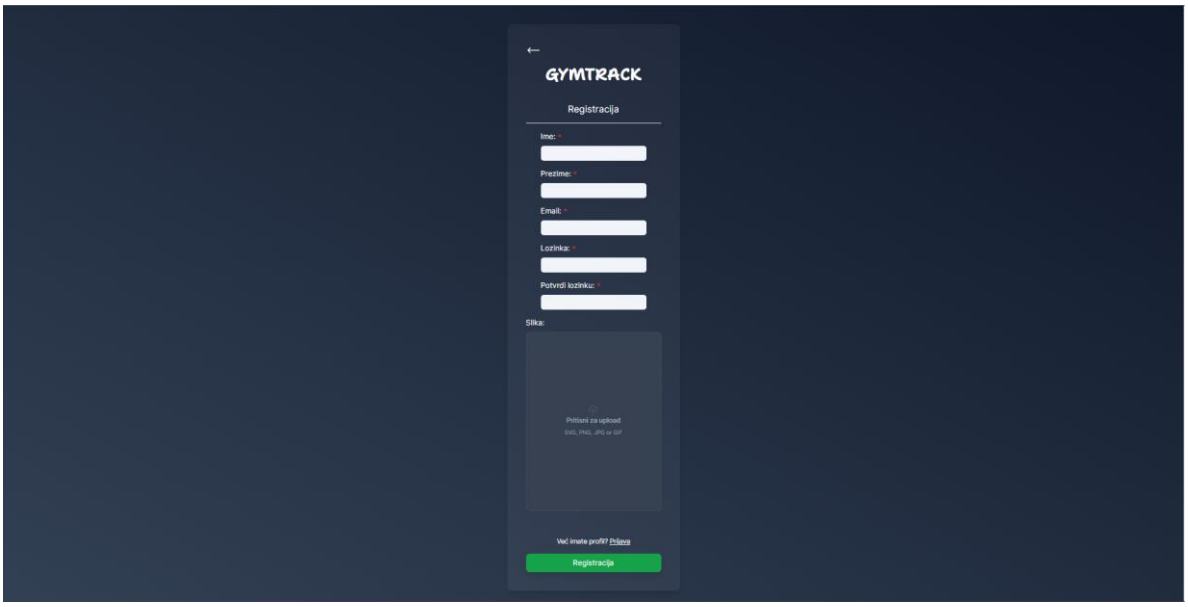
Slika 13: Dijagram arhitekture [autorski rad]

6.7. Najvažnije funkcionalnosti

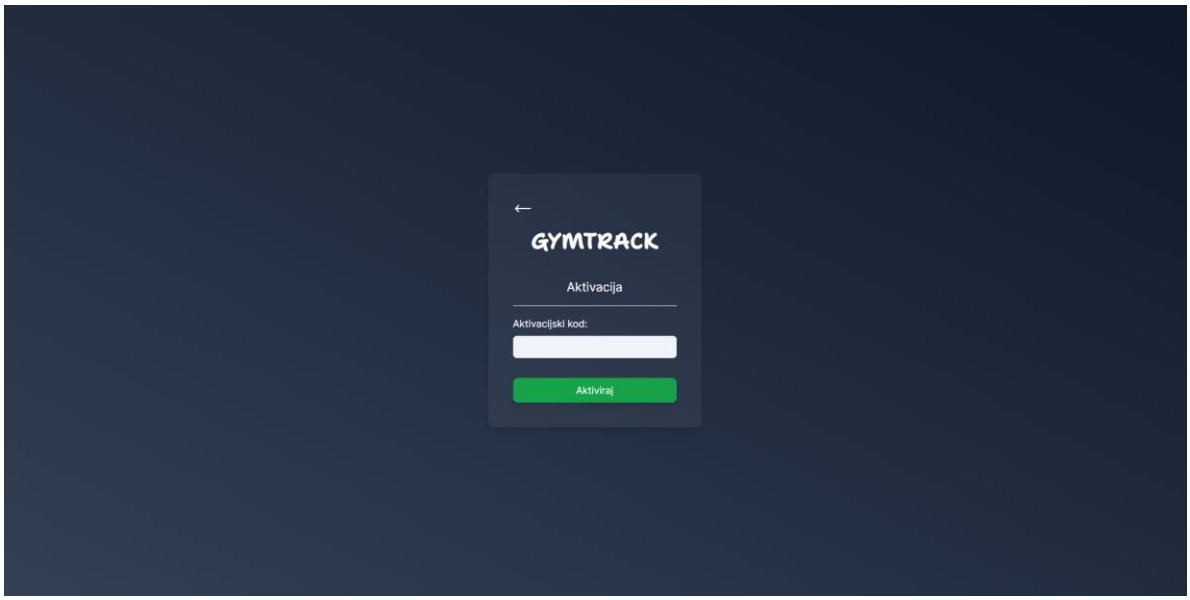
U nastavku su prikazane najvažnije funkcionalnosti izrađene Web aplikacija zajedno sa ekranima te najvažnijim dijelovima programskog koda.

6.7.1. Registracija

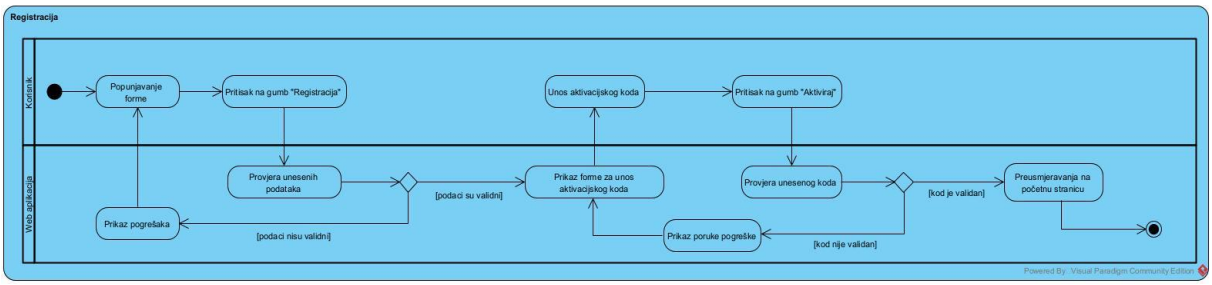
Registracija u sustav sastoji se od dva koraka: popunjavanje forme te unos aktivacijskog koda sa email adrese. Nakon popunjavanja forme vrši se validacija unesenih podataka te provjera ako je uneseno sve što je bilo potrebno. U nastavku su prikazane slike ekrana za registraciju te dijagram aktivnosti registracije.



Slika 14: Ekran registracija [autorski rad]



Slika 15: Ekran za unos aktivacijskog koda [autorski rad]



Slika 16: Dijagram aktivnosti registracije [autorski rad]

U nastavku je prikazana funkcija te njezin programski kod na klijentskoj strani nakon što je korisnik pritisnuo na gumb registracije. Ukoliko je sve u redu, usmjernik preusmjeruje korisnika na stranicu aktivacije.

```
const handleOnSubmit = async () => {
 if (!provijeriMail.test(email)) setEmailError(true);
 else setEmailError(false);
 if (lozinka.length < 6) setLozinkaError(true);
 else setLozinkaError(false);
 if (pLozinka != lozinka) setPLozinkaError(true);
 else setPLozinkaError(false);
 if (
   provijeriMail.test(email) &&
   lozinka.length >= 6 &&
   pLozinka == lozinka &&
   ime.length > 0 &&
   prezime.length > 0
 ) {
   await axios.post("/api/auth/register",
        {ime: ime,
         prezime: prezime,
         email: email,
         lozinka: lozinka,
         slika: profilePicture,},
        {headers: {
           Accept: "application/json",
```

```
            "Content-Type": "application/json",},}
        ).then((res) => {
       const data: { message?: string; error?: string; id?: number} =
            res.data;
          router.push("/activation/" + data.id);
        }).catch((error) => {
          console.log(`Došlo je do pogreške! | Poruka: ${error}`);
          setEmailExists(true);
        });}
};
```

Nakon primanja zahtjeva, poslužitelj kriptira lozinku, generira aktivacijski kod te sprema novog korisnika u bazu koji još nije aktiviran. Nakon uspješnog spremanja u bazu, šalje se email sa aktivacijskom kodom na korisnikovu email adresu. U nastavku je prikazan programski kod na strani poslužitelja.

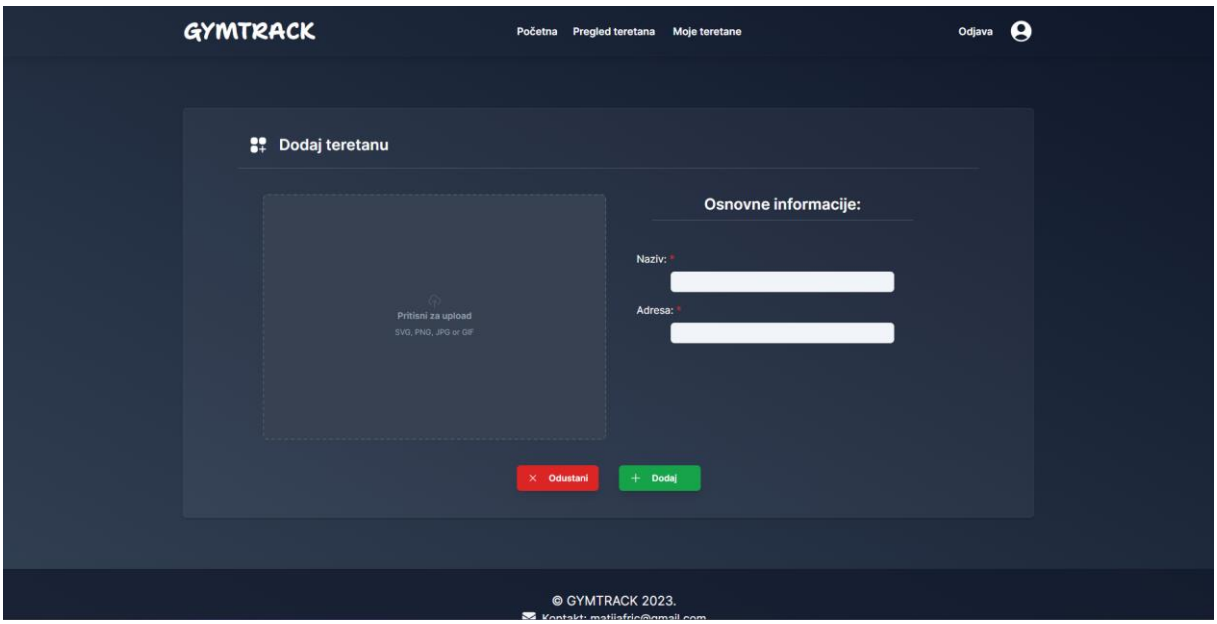
```
…
if (req.method === "POST") {
    try {
      console.info("API zahtjev za registraciju novog korisnika!");
      const { ime, prezime, email, lozinka, slika } = req.body;
      const id = v4();
      const hash = await bcrypt.hash(lozinka, 12);
      const activationCode = Math.floor(Math.random() * 900000) +
100000;
      const dbResponse = await prisma.korisnik.create({
        data: {id: id, ime: ime, prezime: prezime, email: email,
slika: slika, lozinka: hash, aktivacijski_kod: activationCode,
aktiviran: 0, },
      });
      await transporter.sendMail({
        ...mailOptions, to: email, subject: "Aktivacijski kod", text:
"Aktivacijski kod",
        html: `<h1>Aktivacijski kod </h2> <p style="font-
size:18px;font-weight:normal">Hvala Vam na registraciji, kako bi
```

```
nastavili sa prijavom Vaš aktivacijski kod je: ${activationCode}.
</p>`,});

      return res.status(200).json({ message: "Kreiran je novi
korisnik!", id: id });

    } catch (error) {

      console.error(

        `Greška kod API zahtjeva za registraciju novog korisnika |
Poruka ${error}`);

      return res.status(400).json({

        error: `Greška kod API zahtjeva za registraciju novog
korisnika | Poruka ${error}`,});}

  }
```

...

6.7.2. Kreiranje teretane

Svaki prijavljeni korisnik ima mogućnost dodavanja vlastite teretane ispunjavanjem standardizirane forme. Nakon što je korisnik kreirao teretanu, on postaje njen vlasnik. U nastavku je prikazan ekran na kojem se kreira teretana te dio programskog koda na klijentskoj i poslužiteljskoj strani.



Slika 17: Ekran za kreiranje teretane [autorski rad]

Nakon što je korisnik popunio potrebna polja te pritisnuo gumb „Dodaj“, šalje se zahtjev na poslužitelj sa unesenim podacima pomoću funkcije koja je prikazana u nastavku.

```
  const handleOnSubmit = async () => {
    const id = session.data.user.id;
    if (naziv.length > 0 && adresa.length > 0) {
      await axios.post("/api/gym/" + id + "/owner",
          {naziv: naziv, adresa: adresa, slika: gymPicture,},
          {headers: {
              Accept: "application/json",
              "Content-Type": "application/json",},
          }
        ).then((res) => {
          router.push("./owner");})
        .catch((error) => {
          console.log(`Došlo je do pogreške! | Poruka: ${error}`);});
    } else setErrorMessage(true);
  };
```

Nakon što je poslužitelj zaprimio zahtjev za kreiranje teretane, pokreće se transakcija koja prvo kreira teretanu u bazi podataka te nakon toga dodaje korisnika teretane u tablicu *korisnik_teretana* sa ulogom vlasnika.

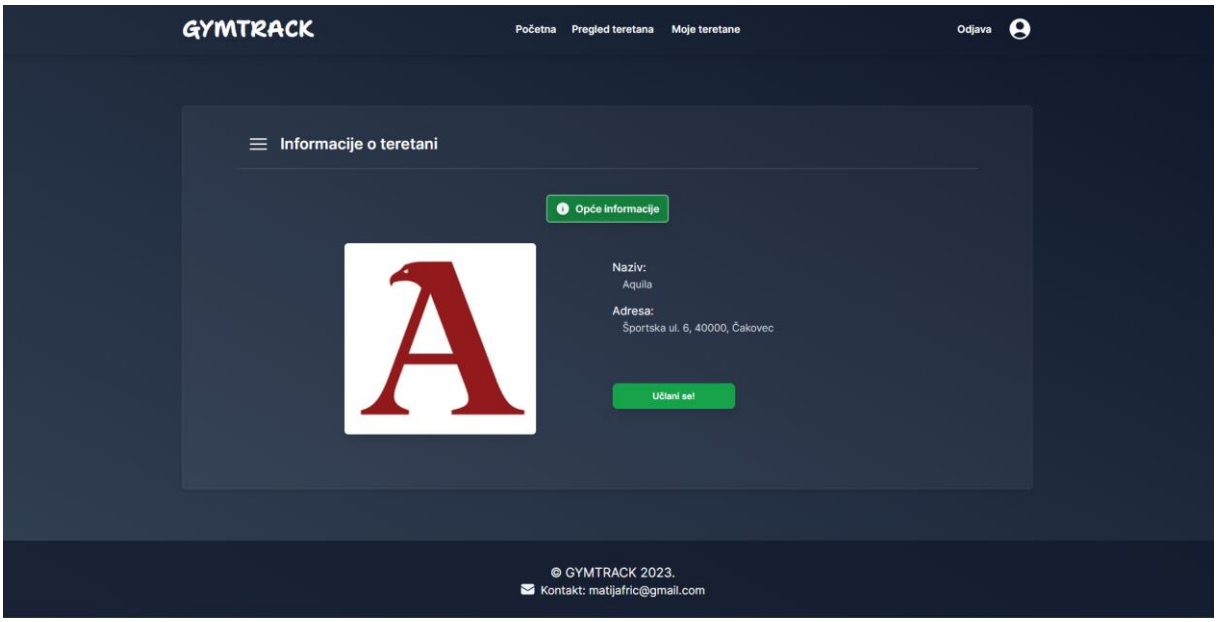
…

```
else if (req.method === "POST") {
    try {
      if (session) {
        console.log("API zahtjev za kreiranje teretane!");
        const { id } = req.query;
        const { naziv, adresa, slika } = req.body;
        const gymId = v4();
        const [teretana, kor_ter] = await prisma.$transaction([
          prisma.teretana.create({
            data: {
              id: gymId,
              naziv: naziv,
```

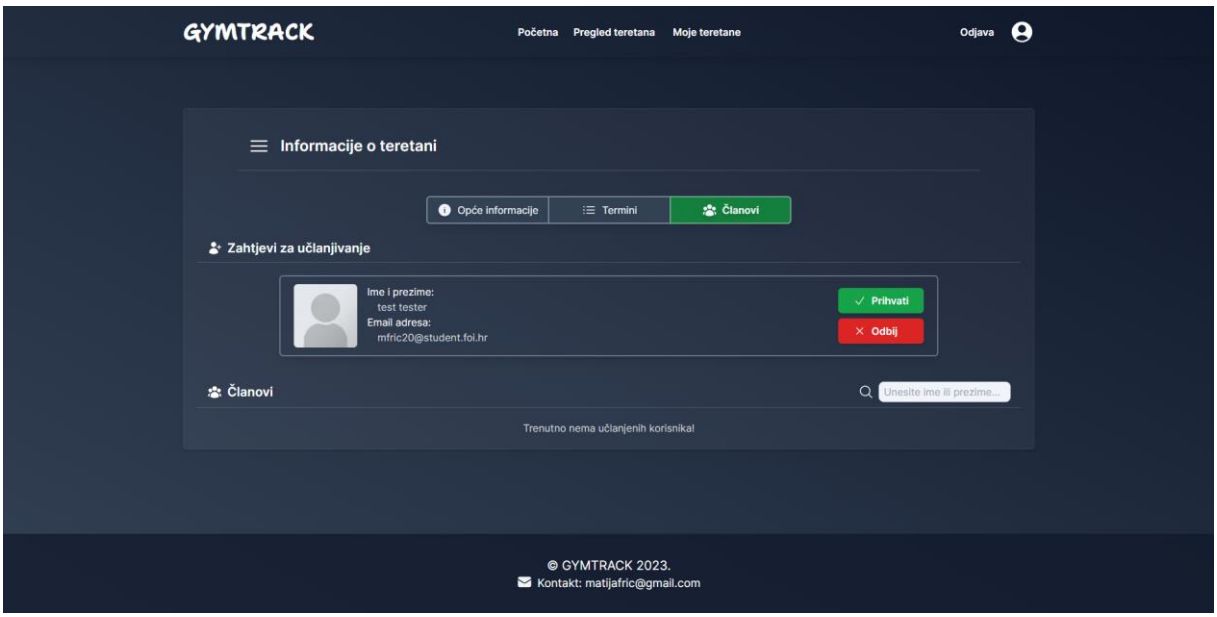
```
            adresa: adresa,
            slika: slika,},}),
          prisma.korisnik_teretana.create({
            data: {
              korisnik_id: id as string,
              teretana_id: gymId,
              uloga_id: 3,
              odobren: true,},}),]);
        return res.status(200).json({ message: "Uspješno!" });
      } else {return res.status(401).json({ error: "Nema sesije!" });}
    } catch (error) {
      console.error(
        `Greška kod API zahtjeva za kreiranje teretane | Poruka ${error}`);
      return res.status(400).json({
        error: `Greška kod API zahtjeva za kreiranje teretane  | Poruka ${error}`,});}
  }
…
```

6.7.3. Učlanjivanje u teretanu

Učlanjivanje u teretanu sastoji se od dva koraka, prvi korak je od strane korisnika koji se želi učlaniti u teretanu dok je drugi korak od strane vlasnika ili djelatnika teretane. Korisnik koji se želi učlaniti u teretanu mora pronaći željenu teretanu na pregledu teretana i pritisnuti na gumb za učlanjivanje na prikazu detalja teretane (slika 18). Vlasnik ili djelatnik teretane ima pregled svih pristiglih zahtjeva za učlanjivanje te tako može prihvatiti ili odbiti zahtjev od korisnika koji se želi učlaniti (slika 19).



Slika 18: Ekran za učlanjivanje u teretanu [autorski rad]



Slika 19: Ekran za prikaz zahtjeva za učlanjivanje u teretanu [autorski rad]

U nastavku je prikazana funkcija na klijentskoj strani koja se poziva nakon što je korisnik pritisnuo na gumb „Učlani se!“. Funkcija šalje zahtjev na poslužitelj sa Id-om teretana u koju se korisnik želi učlaniti.

```
const handleJoinClick = async () => {
  await axios.post("/api/gym/join",
      {gymId: gymId,},
      {headers: {
          Accept: "application/json",
          "Content-Type": "application/json",},}
```

```
      ).then((res) => {
        console.log(res);
      })
      .catch((error) => {
        console.log(`Došlo je do pogreške! | Poruka: ${error}`);
      });
    loadGymInfo();
  };
```

Poslužitelj nakon dobivenog zahtjeva sprema prijavu u tablicu *korisnik_teretana* uz atribut *odobren* koji je postavljen na *false*.

```
…
if (req.method === "POST") {
    try {
      if (session) {
        console.info("API zahtjev za učlanjivanje u teretanu!");
        const { gymId } = req.body;
        const dbResponse = await prisma.korisnik_teretana.create({
          data: {korisnik_id: session.user.id,
            teretana_id: gymId,
            uloga_id: 1,
            odobren: false,},});
        return res.status(200).json({ message: "Uspješno poslan zahtjev!" });
      } else {return res.status(401).json({ error: "Nema sesije!" });}
    } catch (error) {
      console.error(
        `Greška kod API zahtjeva za učlanjivanje u teretanu | Poruka ${error}`);
      return res.status(400).json({
        error: `Greška kod API zahtjeva za učlanjivanje u teretanu | Poruka ${error}`,});}
  }
…
```

Nakon što je zahtjev za učlanjivanje uspješno kreiran, vlasnik ili djelatnik teretane može prihvatiti ili odbiti zahtjev. U nastavku je prikazana funkcija koje se poziva kada vlasnik ili djelatnik teretana prihvati zahtjev za učlanjivanje. Funkcija šalje zahtjev na poslužitelj sa id-om korisnika kojem se prihvaća zahtjev.

```
  const handleAcceptClick = async () => {
    const id = router.query.id;
    await axios.put("/api/gym/" + id + "/members/applicationMembers",
        {userId: user.id,},
        {headers: {
            Accept: "application/json",
            "Content-Type": "application/json",},}
      ).then((res) => {
        loadGymMembers("");
        loadGymApplications();})
      .catch((error) => {
        console.log(`Došlo je do pogreške! | Poruka: ${error}`);});
  };
```

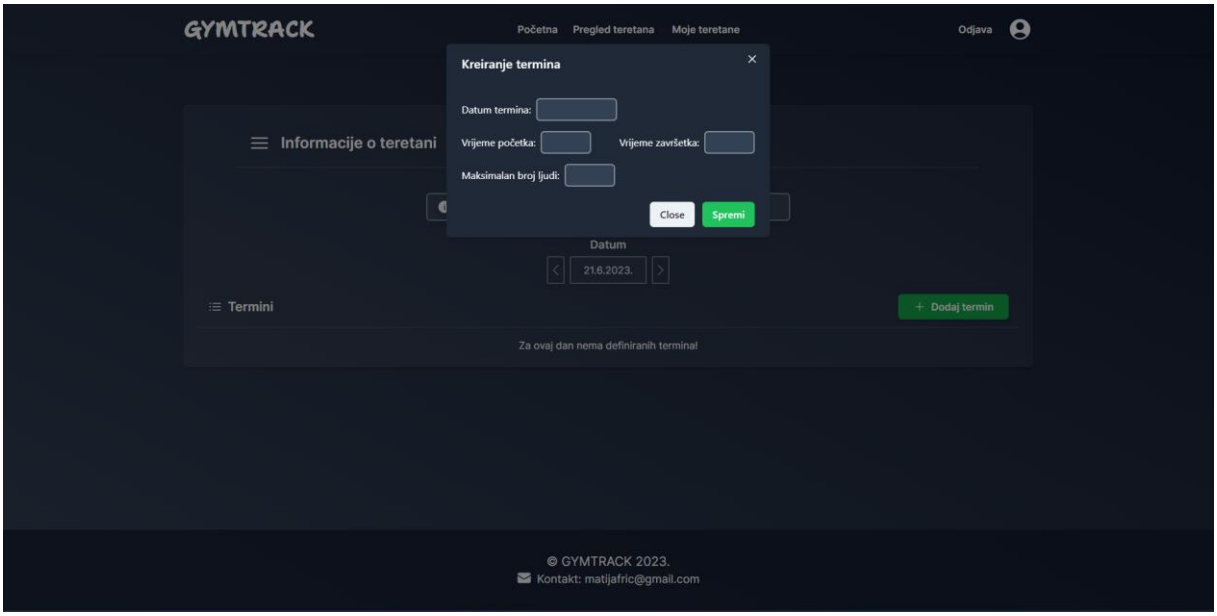
Nakon što je zahtjev za prihvaćanje prijave za učlanjivanje poslan, poslužitelj ažurira prijavu korisnika u tablici *korisnik_teretana* te postavlja atribut *odobren* na *true.*

```
…
else if (req.method === "PUT") {
    try {
      if (session) {
        console.info(
          "API zahtjev za prihvaćanje prijava korisnika u teretane!");
        const { id } = req.query;
        const { userId } = req.body;
        if (id) {
          const dbResponse = await prisma.korisnik_teretana.update({
            where: {
              korisnik_id_teretana_id: {
                teretana_id: id as string,
```

```
                korisnik_id: userId as string,},},
            data: {odobren: true,},});


    return res.status(200).json({ updated: JSON.stringify(dbResponse)
});}
      } else {return res.status(401).json({ error: "Nema sesije!" });}
    } catch (error) {
      console.error(
        `Greška kod API zahtjeva za prihvaćanje prijava korisnika
teretane | Poruka ${error}`);
      return res.status(400).json({
        error: `Greška kod API zahtjeva za prihvaćanje prijava
korisnika teretane  | Poruka ${error}`,});}
  }
…
```

6.7.4. Kreiranje termina

Svaki vlasnik ili djelatnik teretane može kreirati termin ispunjavanjem jednostavne forme. Nakon što je termin kreiran, on će biti vidljiv u popisu termina te će članovi teretane imati mogućnost prijave na kreirani termin.



Slika 20: Ekran za kreiranje termina [autorski rad]

Pritiskom na gumb „Spremi“ poziva se funkcija koja validira unesene podatke te šalje zahtjev poslužitelju za kreiranje novog termina. U nastavku je prikazan dio funkcije koji šalje zahtjev ukoliko nema pogrešaka u unesenim podacima.

…

```
    if (errorMess.length === 0) {
      await axios.post("/api/gym/" + gymId + "/terms/term",
          { startDate,
            startTime,
            endTime,
            maxNumber,
            createdBy: session.data.user.id,},
          { headers: {
              Accept: "application/json",
              "Content-Type": "application/json",},}
        ).then((res) => {
          onClose();
          loadCurrentTerms();})
        .catch((error) => {
          console.log(`Došlo je do pogreške! | Poruka: ${error}`);});
    } else setErrorMessage(errorMess.substring(0, errorMess.length -
2));
```

…

Nakon što je poslužitelj zaprimio zahtjev za kreiranje termina, datumi se pretvaraju u *string* te se u tablici *termin* kreira novi zapis.

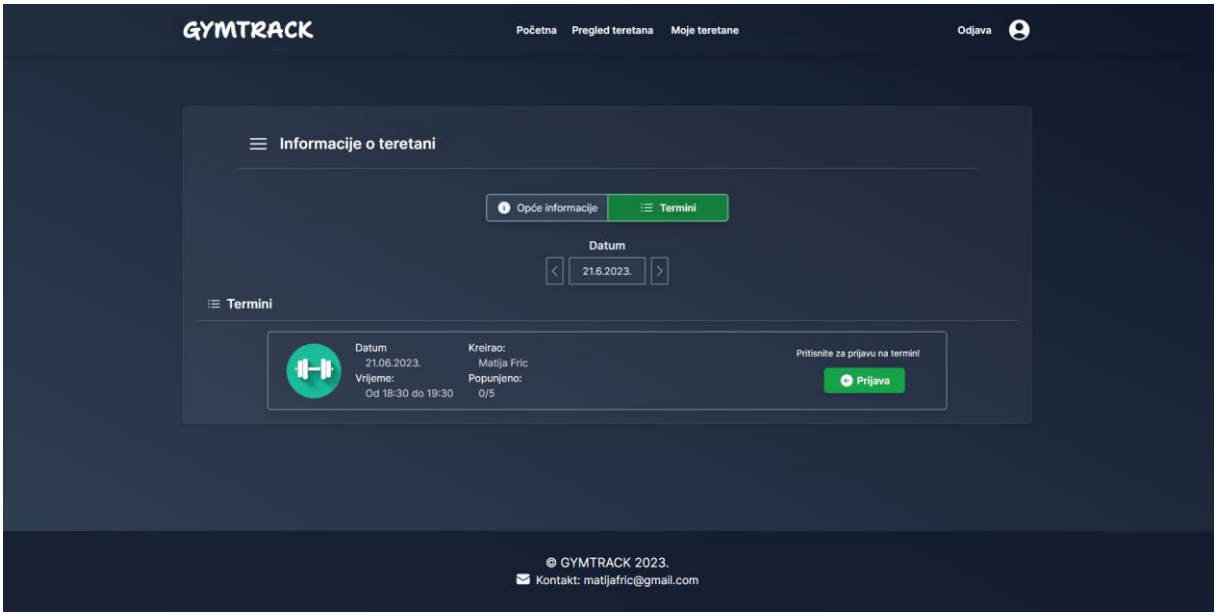
…

```
if (req.method === "POST") {
    try {
      if (session) {
        console.info("API zahtjev za kreiranje termina!");
        const { id } = req.query;
        const { startDate, startTime, endTime, maxNumber, createdBy }
= req.body;
```

```
      let startDateString = new Date(startDate).toLocaleDateString();
      let startTimeString = new Date(startTime).toLocaleTimeString();
      let endTimeString = new Date(endTime).toLocaleTimeString();
      const maxNumberInt = parseInt(maxNumber);
      startDateString = startDateString.replaceAll(" ", "");
      startTimeString = startTimeString.substring(0,
      startTimeString.length - 3);
      endTimeString = endTimeString.substring(0, endTimeString.length
- 3);
      const newTerm = await prisma.termin.create({
        data: {datum: startDateString,
             pocetak: startTimeString,
             kraj: endTimeString,
             maksimalan_broj: maxNumberInt,
             obrisan: false,
             kreirao: createdBy as string,
             teretana_id: id as string,},});
       if (newTerm)
         return res.status(200).json({ message: "Termin je dodan!"});
       else
         return res.status(500).json({ error: "Greška kod dodavanja
termina!" });
       } else {return res.status(401).json({ error: "Nema sesije!" });}
     } catch (error) {
       console.error(
         `Greška kod API zahtjeva za kreiranje termina! | Poruka
${error}`);
       return res.status(400).json({
         error: `Greška kod API zahtjeva za kreiranje termina! | Poruka
${error}`,});}
   }
…
```

6.7.5. Prijava na termin

Korisnik koji je učlanjen u teretanu ima mogućnost pregleda svih kreiranih termina za odabrani datum. Ukoliko se korisnik odluči prijaviti na neki od termina, to može učiti pritiskom na gumb „Prijava“ na ekranu na kojem su prikazani termini (slika 21).



Slika 21: Ekran za prijavu na termin [autorski rad]

U nastavku je prikazan funkcija koja se poziva ukoliko je korisnik pritisnuo na gumb „Prijava“. Funkcija šalje zahtjev poslužitelju uz id termina na koji se korisnik prijavljuje.

```
const handleTermApplication = async () => {
  const id = router.query.id;
  await axios.post("/api/gym/" + id + "/terms/member", {
      data: {termId: term.id,},
      headers: {
        Accept: "application/json",
        "Content-Type": "application/json",},
    }).then((res) => {
      if (res.data.message == "Termin je pun!")
        console.log(res.data.message);
      else loadCurrentTerms();
      modalApply.onClose();
    }).catch((error) => {console.log(`Došlo je do pogreške! |
    Poruka: ${error}`);});};
```

Nakon što je poslužitelj zaprimio zahtjev za prijavom na termin, provjerava se ako je termin pun. Ukoliko termin nije popunjen, kreira se novi zapis u tablici *korisnik_termin*.

```
...
else if (req.method === "POST") {
    try {
      if (session) {
        console.info("API zahtjev za prijavu na termin!");
        const { termId } = req.body.data;
        const maxNumber = await prisma.termin.findFirst({
          select: {maksimalan_broj: true,},
          where: {id: termId,},
        });
        const currentNumber = await prisma.korisnik_termin.count({
          where: {termin_id: termId,},
        });
        if (maxNumber.maksimalan_broj <= currentNumber)
          return res.status(200).json({ message: "Termin je pun!" });
        const newApplication = await prisma.korisnik_termin.create({
          data: {korisnik_id: session.user.id,
            termin_id: parseInt(termId as string),},});
        if (newApplication)
          return res.status(200).json({ message: "Uspješna
prijava!});
        else return res.status(500).json({ error: "Dogodila se
greška!" });
      } else {
        return res.status(401).json({ error: "Nema sesije!" });}
    } catch (error) {console.error(
`Greška kod API zahtjeva za prijavu na termin! | Poruka ${error}`);
      return res.status(400).json({error: `Greška kod API zahtjeva za
prijavu na termin!  | Poruka ${error}`,});}
  }
...
```

7. Zaključak

Razvoj Web aplikacija vrlo je popularan posao današnjice te primjena *TypeScript-a* kao nadogradnje na *JavaScript* također raste u popularnosti. *TypeScript* nudi brojne prednosti te pruža programerima lakše razumijevanje programskog koda te smanjenje broja pogrešaka zbog mogućnosti tipizacije. U ovome radu detaljno je opisan *TypeScript* programski jezik sa primjerima programskog koda.

Izrada Web aplikacija pomoću paketa tehnologija (eng. *full-stack*) pridonosi smanjenju potrebe za učenjem i korištenjem više različitih biblioteka i okvira na klijentskoj i poslužiteljskoj strani, čime se smanjuje kompleksnost izrade same aplikacije i povećava produktivnost programera. Ovaj rad uspoređuje te navodi prednosti i nedostatke najpopularnijih okvira za razvoj pomoću paketa tehnologija.

NextJS jedan je od najpopularnijih okvira za razvoj pomoću paketa tehnologije te omogućuje kreiranje skalabilnih Web aplikacija visoke kvalitete. U ovom radu je detaljno opisan *NextJS* sa svim svojim mogućnostima uz primjere programskog koda te je kroz praktičan primjer razvijena kompletna Web aplikacija za podršku poslovanja teretane.

Zaključak ovog rada jest da *TypeScript* u kombinaciji sa *NextJS-om* omogućuje programerima izgradnju kvalitetnih Web aplikacija kao i mogućnost razvoja modernih i skalabilnih aplikacija koje su prilagođene suvremenim korisničkim zahtjevima.

Popis literature

[1] Leon Shklar and Richard Rosen, *Web Application Architecture: Principles, Protocols and Practices*. Wiley & Sons, Incorporated, John, 2009.

[2] Ralph F. Grove, *Web-Based Application Development*. Jones And Bartlett Publishers, 2010.

[3] Cal Henderson, *Building Scalable Web Sites*, 1st Edition. 2006.

[4] Ethan Brown, *Web Development with Node and Express*, First Edition. O'Reilly Media, 2014.

[5] Luke Wroblewski, *Mobile First*. New York: Jeffrey Zeldman, 2011.

[6] Calum Ryan, "An accessibility-first design approach," Dec. 02, 2021.

[7] Kent Beck, *Test-Driven Development By Example*. Addison Wesley, 2002.

[8] MDN contributors, "HTML: HyperText Markup Language," Mar. 13, 2023. https://developer.mozilla.org/en-US/docs/Web/HTML (accessed Mar. 27, 2023).

[9] MDN Contributors, "CSS: Cascading Style Sheets," Feb. 26, 2023. https://developer.mozilla.org/en-US/docs/Web/CSS (accessed Mar. 27, 2023).

[10] MDN contributors, "What is JavaScript?," Mar. 05, 2023. https://developer.mozilla.org/en-US/docs/Learn/JavaScript/First_steps/What_is_JavaScript (accessed Mar. 27, 2023).

[11] Cory Gackenheimer, *Introduction to React*. Apress, 2015.

[12] Chinmayee Deshpande, "What is Angular?: Architecture, Features, and Advantages," Feb. 24, 2023. https://www.simplilearn.com/tutorials/angular-tutorial/what-is-angular (accessed Mar. 27, 2023).

[13] vuejs.org, "What is Vue?," 2023. https://vuejs.org/guide/introduction.html (accessed Mar. 27, 2023).

[14] npm trends, "angular/cli vs react vs vue," 2023. https://npmtrends.com/@angular/cli-vs-react-vs-vue (accessed Mar. 27, 2023).

[15] Coursera, "What Does a Back-End Developer Do?," Mar. 24, 2023. https://www.coursera.org/articles/back-end-developer (accessed Mar. 28, 2023).

[16] Jeff Forcier, Paul Bissex, and Wesley Chun, *Python Web Development with Django*. Boston: Pearson Education, Inc., 2009.

[17] Matt Stauffer, *Laravel: Up and Running*, First Edition. O'Reilly Media, Inc., 2017.

[18] npm trends, "@nestjs/cli vs next vs nuxt," 2023. https://npmtrends.com/@nestjs/cli-vs-next-vs-nuxt (accessed Mar. 28, 2023).

[19] William Dawson, "Nuxt, Next, Nest! Confused?," Sep. 16, 2019. https://codersera.com/blog/nuxt-next-nest-confused/ (accessed May 31, 2023).

[20] Rich Kurtzman, "Advantages and disadvantages of Nuxt.js," Oct. 17, 2022. https://dev.to/richkurtzman/advantages-and-disadvantages-of-nuxtjs-13ml (accessed May 31, 2023).

[21] Stanislav Naborshchikov and Mateusz Tkacz, "Advantages and disadvantages of NestJS," Jun. 10, 2022. https://themobilereality.com/blog/advantages-and-disadvantages-of-nestjs (accessed May 31, 2023).

[22] Rich Kurtzman, "Advantages and disadvantages of Next.js," Sep. 08, 2022. https://dev.to/richkurtzman/advantages-and-disadvantages-of-nextjs-5hg6 (accessed May 31, 2023).

[23] Boris Cherny, *Programming TypeScript*, First Edition. O'Reilly Media, Inc., 2019.

[24] Vercel, "NextJS docs," 2023, Accessed: May 31, 2023. [Online]. Available: https://nextjs.org/docs

Popis slika