Web aplikacija za najam brodica

Didak, Karlo

Undergraduate thesis / Završni 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:200:982344

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2025-03-21**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

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

Stručni studij

WEB APLIKACIJA ZA NAJAM BRODICA

Završni rad

Karlo Didak

Osijek, 2024.

SADRŽAJ

1. UVOD 1

1.1 Zadatak završnog rada 2

2. KORIŠTENE TEHNOLOGIJE 3

2.1 HTML 3

2.2 CSS 5

2.3 React 6

2.4 Java 7

2.5 Framework 7

2.6 Springboot 8

2.7 MySQL 9

3. SLIČNA RJEŠENJA 10

4. OPIS WEB APLIKACIJE 11

4.1. Naslovna / Početna stranica 11

4.2. Stranica za brodice 13

4.3. Role stranice 15

5. PROGRAMSKO RJEŠENJE 18

5.1 Postavljanje/strukturiranje frontend koda 18

5.2. Postavljanje backenda 21

5.3. Početna stranica 23

5.2. Stranica za ispisivanje brodica 26

5.3. Najam brodice 26

5.4. Pregled najmova 33

5.5 Obavijesti 35
5.6. Validacija/kontrola podataka 36
ZAKLJUČAK 41
LITERATURA 42
SAŽETAK 43
SUMMARY 44

1. UVOD

Ovaj završni rad obrađuje temu izrade web aplikacije za najam brodica, s ciljem rješavanja problema gužvi i dugotrajnih čekanja tijekom ljetne sezone i turističkih vrhunaca. Implementacija aplikacije koja olakšava proces najma brodica zasigurno bi rezultiralo uštedom vremena i povećanjem korisničkog zadovoljstva.

Aplikacija je usmjerena na korisnike koji žele iznajmiti svoju brodicu drugim korisnicima (unajmiteljima) i na korisnike koji traže brodicu za najam (iznajmljivači). Za svaku dostupnu brodicu postoji sučelje koje sadrži podatke o cijeni zauzeća te drugim parametrima te specifične brodice kako bi se korisniku dala što bolja informacija. Osim sučelja za pregled dostupnih brodica, aplikacija također omogućuje dodavanje svog broda, pregled svojih najmova, postavljanje svoje dozvole za upravljanje brodicom i druga sučelja s relevantnim funkcionalnostima.

Kroz niz poglavlja detaljno će se objasniti proces izrade navedene web aplikacije. Prvo poglavlje nosi naziv "KORIŠTENE TEHNOLOGIJE" u kojem će biti opisane tehnologije korištene pri izradi web aplikacije. Ukratko, bit će obrađena njihova povijest, razlozi zašto su odabrane, te njihova sintaksa i struktura.

Nakon toga, u kratkom poglavlju "SLIČNA RJEŠENJA" bit će predstavljena već postojeća aktivna web aplikacija koja nudi slične mogućnosti, te koja je poslužila kao inspiracija za izgled aplikacije ovog završnog rada.

U poglavlju "OPIS WEB APLIKACIJE" bit će opisana struktura korisničkog sučelja, sve uloge koje aplikacija ima te dijelovi korisničkog sučelja koji su specifični za pojedinu rolu.

Poglavlje "OBJAŠNJENJE KODA" prolazit će kroz strukturu korisničkog sučelja i sve uloge u detalje, pružajući uvid u rad pojedinog dijela koda i objašnjavajući kako funkcionira. Ovaj pregled omogućit će čitateljima bolje razumijevanje implementacije aplikacije i tehničkih detalja koji su uključeni u njezino funkcioniranje.

1.1 Zadatak završnog rada

Zadatak završnog rada je izrada web aplikacije za najam brodica, koristeći tehnologije Spring Boot i React, uz upotrebu ostalih tehnologija relevantnih za razvoj ovakve vrste web aplikacije. Cilj je omogućiti pristup aplikaciji svim korisnicima, bez obzira jesu li registrirani ili neregistrirani. Registriranim korisnicima, odnosno unajmiteljima, treba omogućiti postavljanje vlastitih brodica, dok bi iznajmitelji trebali moći izvršiti najam nekog od ponuđenih brodica.

2. KORIŠTENE TEHNOLOGIJE

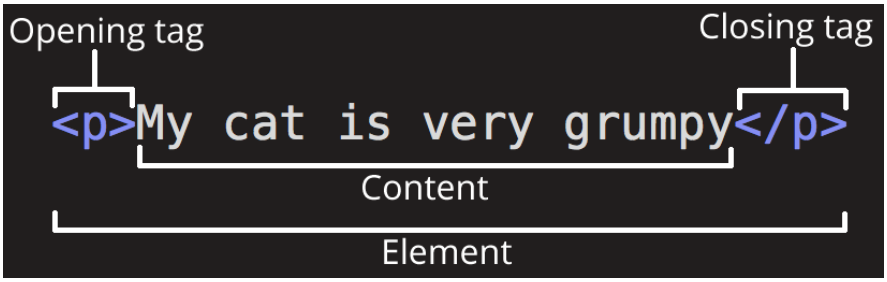
Danas postoje razne tehnologije i alati za razvoj web aplikacija. Na fakultetu se uče temelji za izradu web aplikacije kroz upotrebu tehnologija poput HTML, CSS, JavaScript, PHP te druge. Radi lošijeg praćenja modernog načina osmišljavanja web aplikacije nabrojanim tehnologijama i kako bi ovaj završni rad bio što kvalitetnije i efikasnije izrađen prethodnim tehnologijama je dodan tržišno, široko zastupljen programski okvir (eng. *framework*) Spring Boot i React.

2.1 HTML

HTML(eng. *HyperText Markup Language*) je najosnovniji dio web programiranja. HTML je napravio Tim Berners-Lee. Prvu verziju je napravio 1991. godine, a danas koristimo petu verziju. Velika rasprostranjenost i popularnost HTML-a proizlazi iz njegove jednostavnosti učenja i upotrebe i zbog toga što je od početka zamišljen da bude besplatan i svima dostupan.[1] Još neke prednosti HTML-a su to da ima veliku količinu dostupnih resursa te se izvorno može izvoditi na svakom pregledniku. Unatoč mnogim prednostima, HTML ima i određene nedostatke, ali koji ga nisu spriječili u njegovom razvoju i širenju. Naime, zbog nedostatka široke dinamičke funkcionalnosti ne smatra se programskim jezikom i uglavnom se koristi za statičke web stranice. Također, sve komponente moraju biti izrađene zasebno čak i ako koriste slične elemente te ponašanje preglednika može biti nepredvidivo, budući da stariji preglednici možda neće biti kompatibilni s novijim značajkama. [2]

Svaki HTML element se sastoji od 3 dijela (Slika 2.1.):

- Oznake otvaranja (eng. *Opening tag*) - oznaka je omotana znakom manje i veće. Na primjer koristili bi oznaku za stvaranje paragrafa.
- Sadržaj (eng. *Content*) - ovo je izraz koji drugi korisnici vide.
- Oznake zatvaranja (eng. *Closing tag*) - ista kao i početna oznaka, ali s kosom crtom ispred naziva elementa. Na primjer, za završetak paragrafa. [3]



Slika 2.1. Sintaksa HTML [4]

Primjeri elemenata koji imaju mogućnost prikazati tekst, slike i druge sadržaje su: <head>, <title>, <body>, <header>, <foot>, <article>, <section>, <p>, , , <img> itd.

U HTML-u unutar oznaka otvaranja elementa mogu se dodati atributi (eng. *attribute)* što je prikazano slikom 2.2. Atribut sadrži dodatne informacije za element unutar kojeg se nalazi (npr. adresa slike, naslov...).

Slika 2.2. Sintaksta dodavanja atributa HTML elementu [4]

HTML omogućuje stvaranje linkova koji povezuju web stranice jedne s drugima što je označeno slovima HT (eng. *HyperText*) u nazivu. Ovi linkovi se mogu postaviti kao atribut u određeni element tako da on vodi na određenu web stranicu. Kao atribut se mogu postaviti i razni stilski izrazi pomoću kojih se uređuje element.

Kako bi bio što veći broj mogućnosti pri uređivanju elemenata koristimo druge jezike za uređivanje poput CCS.

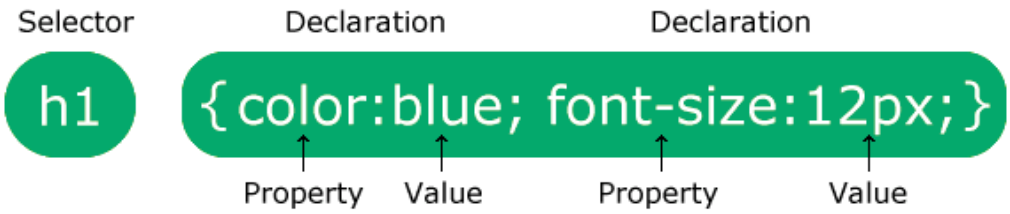
2.2 CSS

CSS (eng. *Cascading Style Sheets*) je stilski jezik koji se koristi za opisivanje izgleda dokumenata napisanih u HTML-u ili XML-u (eng. *Extensible Markup Language*). Pomoću CSS-a se mogu opisati i zadati kako će se svaki pojedini element prikazivati.

Jedan od ključnih problema koje je CSS riješio proizlazi iz činjenice da je HTML zamišljen i napravljen da se koristi za oblikovanje web stranice. S obzirom na to da su se od HTML 3.2 uveli elementi poput <font> i atributi za boju, nastali su veliki problemi za programere. Razvijanje većih web stranica zahtijevalo je ručno postavljanje fontova i boja za svaku stranicu, što je rezultiralo dugotrajnim i skupim procesom. Osim toga, kod je postao složeniji za održavanje i teži za čitanje.

Zbog toga je World Wide Web Consortium (W3C) kreirao CSS zajedno s HakonWium Lie i Bert Bos.

Sintaksa CSS-a se sastoji od selektora (eng. *selector*) i deklaracijskog bloka (okužen uglastim zagradama). Selektor pokazuje HTML element koji se želi stilizirati. Unutar deklaracijskog bloka se nalaze CSS deklaracije (eng. *declaration*) kojim se uređuju HTML i svaka je pojedinačno odvojena točkom zarezom. Svaka deklaracija se sastoji od naziva CSS svojstva (eng. *property*) i vrijednosti (eng. *value*), odvojene dvotočkom. Primjer deklaracije se može vidjeti na slici 2.3.



Slika 2.3. Sintaksa CSS [5]

2.3 React

React je besplatna JavaScript biblioteka otvorenog koda za izgradnju korisničkih sučelja.

JavaScript je skriptni programski jezik, koji se izvršava u web pregledniku na strani korisnika pored HTML i CSS.

React održavaju Meta (prijašnji Facebook) i zajednica programera i tvrtki. Pomoću Reacta se programira puno brže nego koristeći obični JavaScript. Ubrzano se programira tako što se kreiraju komponente koje se mogu ponovo koristiti. Ove komponente se slažu poput Lego kockica tako da dobijemo konačno sučelje.

Prednost Reacta je u tome što ne mora uvijek ispočetka učitavati novu stranicu. Inače kada korisnik želi pristupiti određenoj web stranici, on klikne na njenu lokaciju (URL) te preglednik šalje zahtjev za tu web stranicu nakon čega ju učitava korisniku. Za bilo koju web stranicu na koju korisnik odluči pristupiti cijeli proces se ponavlja. Ovaj pristup je inače dobar i funkcionira, ali ako stranica ima veliku količinu podataka onda je situacija drugačija. Učitavanje cijele web stranice naprijed natrag bilo bi suvišno i stvorilo bi loše korisničko iskustvo.

React omogućava drugačiji pristup. React omogućava učitavanje jednostranične aplikacije, SPA (eng. *Single-Page Application*). SPA učitava samo jedan HTML dokument na prvom zahtjevu korisnika. Nakon toga učitava samo onaj dio web stranice koji se treba promijeniti koristeći JavaScript. [6]

Za razliku od Angulara, React ne slijedi striktna pravila za organizaciju koda. Programeri si mogu konstruirati raspored datoteka i implementirati React kako žele. Može se implementirati proizvoljna količina elemenata Reacta, što predstavlja njegovu fleksibilnost.

Pored ovih prednosti React ima ostale prednosti zbog kojih ga vrijedi koristiti.

2.4 Java

Java je objektno orijentiran programski jezik. Razvio ga je James Gosling iz tvrtke Sun Microsystems, a jezik je objavljen 1995. godine.

Java je dizajnirana tako da ima što manje ovisnosti o implementaciji i kao jezik koji se može pokrenuti na bilo kojoj platformi koja podržava Javu. Kada se napiše program u Javi za računalo vrlo ga je lako prebaciti na mobitel. Takav programski jezik nosi oznaku WORA (eng. *write once, run anywhere*) što je bio primarni cilj Jamesu pri kreiranju Jave. Sintaksa je slična programskim jezicima C i C++, ali Java ima manje niskorazinskih objekata.

Od 2019. godine Java je jedan od najrasprostranjenijih i korištenih programskih jezika na svijetu. Kada se pogleda da je Java na tržištu preko 25 godina i da se i dalje bira/koristi, ispred Pythona, Rubyja, PHP-a te ostalih programskih jezika mora se reći da je vrlo impresivan programski jezik.[7]

2.5 Framework

Framework je struktura koja se postavlja kao baza, kako se programiranje softvera ne bi počelo potpuno od nule. *Framework* su tipično povezani sa specifičnim programskim jezikom i svaki je dizajniran za obavljanje određenih zadataka.

Framework se može objasniti s primjerom građenja kuće. Osoba može sama praviti temelje i okvir kuće, no trebalo bi puno vremena, ali je izvedivo. Kada bi to napravio netko drugi, uštedjelo bi se puno vremena i truda, pogotovo ako je osoba koja to radi profesionalac.

Kao i kuća čiji je temelj napravio profesionalac, *framework* su napravili drugi softver developeri i inženjeri, koji su ga istestirali. Na taj način je osigurano da je temelj softvera u kojem se koristi *framework* dobar.

Kuća naravno nije gotova samo s temeljem i okvirom. Isto tako na *framework* u razvoju nekog softvera se mogu dodavati razne funkcije višeg nivoa.

Pored toga što je *framework* pouzdan kao baza softvera i što mu je lako dodati funkcionalnost, *framework* ima još neke prednosti:

- sigurniji programski kod

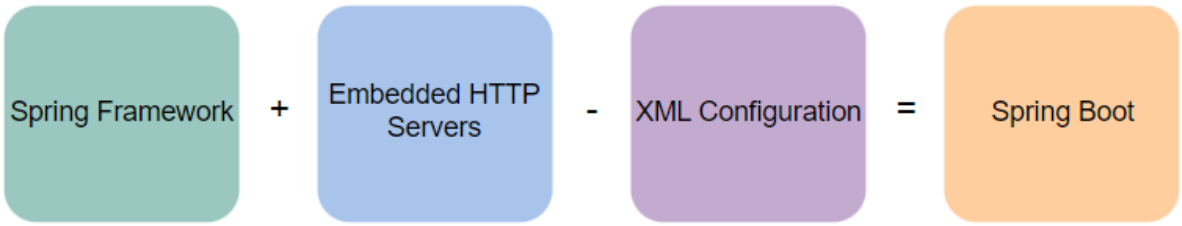
- jednostavnije testiranje i debagiranje
- izbjegavanje dupliciranje koda
- jednostavnije fokusiranje na kod koji je specifičan projektu
- jednostavan programski kod za adaptaciju

Često se pojmovi *framework* i biblioteka poistovjećuju. *Framework* je poput okvira s kojim programer dalje može raditi. *Framework* poziva kod programera, a kod biblioteka programer poziva kod. Isto se može reći da je kod programera u kontroli kada se koristi biblioteka, a kada se koristi *framework* onda je *framework* u kontroli. [8]

Framework često uključuje biblioteke, a biblioteke se koriste za ispunjavanje funkcija

2.6 Springboot

Spring Boot je Java *framework* otvorenog koda kojeg je razvio Pivotal 2014. i koji pojednostavljuje razvoj i postavljanje Java web aplikacija. Spring Boot se sastoji od Spring programskog okvira te ugrađenih servera bez XML konfiguracije, što je prikazno slikom 2.4. XML konfiguracija nije potrebna zbog toga što Spring Boot koristi konvenciju umjesto konfiguracijskog softverskog dizajna.



Slika 2.4. Graf strukture Spring Boota [9]

Konvencije kodiranja skup su smjernica za određeni programski jezik koje preporučuju programski stil, prakse i metode za svaki aspekt programa napisanog na tom jeziku. Razlika u konvenciji i konfiguraciji softverskog dizajna je u tome što s konvencijom želimo postići pravila, oblik koji se može uvesti od strukture umjesto da je potreban određeni kod, kao što je potrebno s konfiguracijom.

Prednosti Spring Boota su:

- Omogućava jednostavniji i ubrzan razvoj aplikacija te time i produktivnost
- Omogućuje veliki broj pluginova (proširenja)
- Lagano testira aplikacije pomoću ugrađenih HTTP servera
- Nema potrebe za XML konfiguracijom
- Minimizira pisanje šablonskog koda (eng. *Boilerplate code*, kod koji je korišten na više različitih mjesta s jako malo i bez ikakvih promjena)

Problem sa Spring Boot-om, točnije, njegova limitacija je u tome što nekad uključuje ovisnosti (eng. *dependencies*, biblioteke koje omogućuju pristup određenim mogućnostima, funkcijama) koje se ne koriste u programu/aplikaciji i zbog toga se može dogoditi da imamo aplikaciju koja je bespotrebno veća nego što bi trebala biti. [9]

2.7 MySQL

MySQL je sustav za upravljanje relacijskom bazom podataka (RDBMS, eng. *Relational Database Management System*) otvorenog koda. Dobio je naziv po kćerki suosnivača MySQL-a Michaela Wedeniusa, My, i po SQL, akronimu za *Structured Query Language*.

SQL je programski jezik koji omogućuje rad s bazama podataka. Često dolazi do miješanja pojmova RDBMS, MySQL i SQL, u smislu da se poistovjećuju, ali to samo pokazuje veliku zastupljenost MySQL-a. Danas MySQL koriste mnoge velike tvrtke poput Facebooka, Twittera, YouTuba, Googla itd.

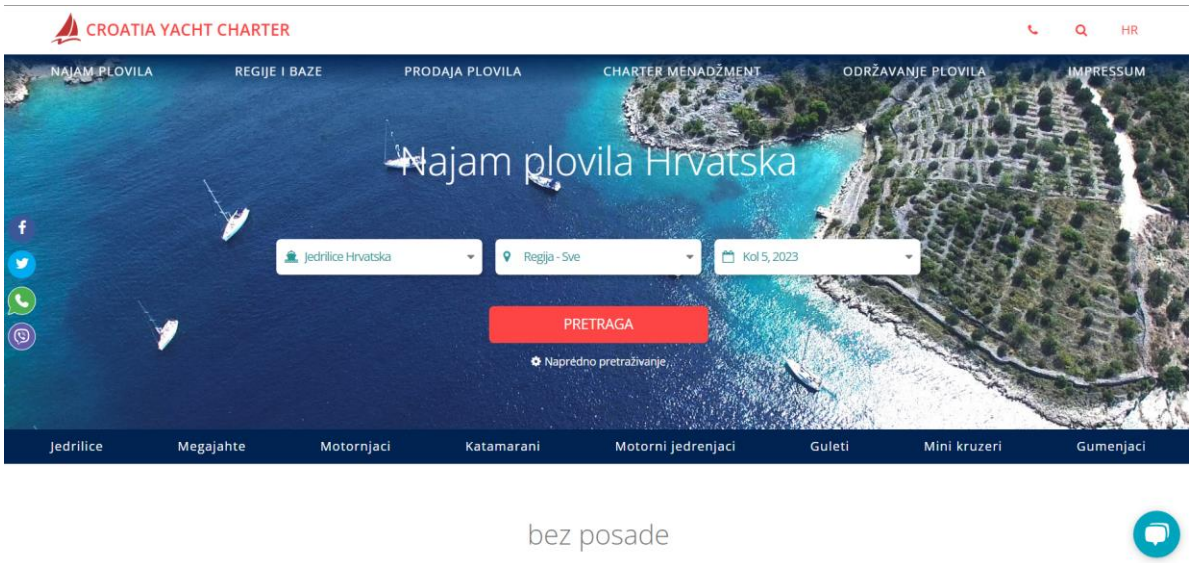
Općenito baza podataka je zbirka podataka strukturiranih i pohranjenih u tablice. Također baza podataka se koristi za upravljanje spremljenim podacima (pohranjivanje, dohvaćanje, ažuriranje, brisanje).

MySQL omogućuje jednostavniji softverski okvir za stvaranje, održavanje i interakciju s bazama podataka.

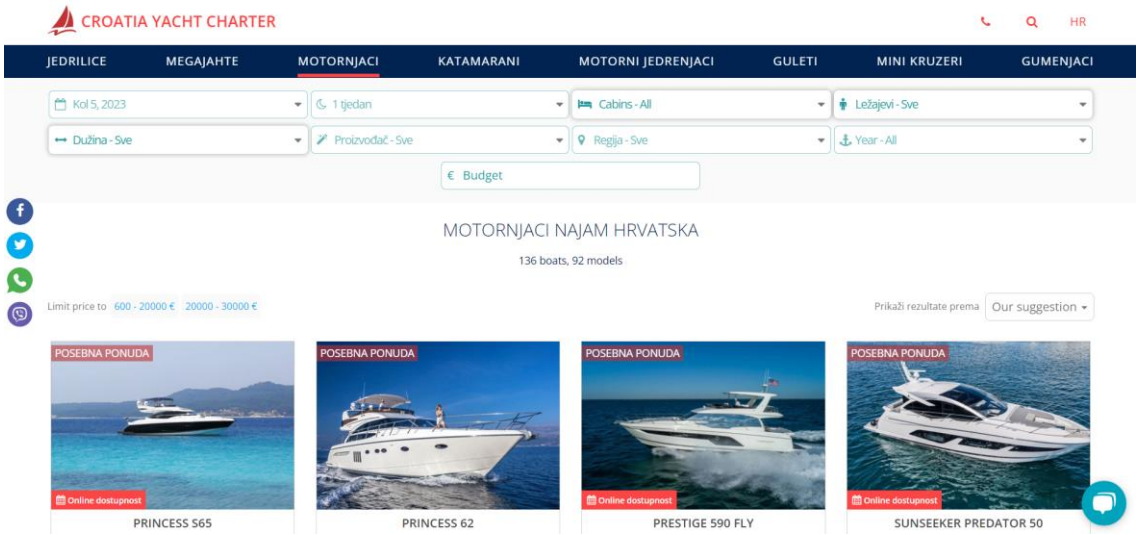
3. SLIČNA RJEŠENJA

Danas postoje web aplikacije za sve što je potrebno, postoje i one koje nemaju pravu svrhu tako da je teško izmisliti potpuno novu ideju za web aplikaciju.

Prije izrade web aplikacije za završni rad pregledane su već postojeće i aktivne web stranice na području Hrvatske koje se bave najmom brodica. Stranica koja je najviše potakla inspiraciju za izradu web aplikacije, je bila Croatia Yacht Charter koja se može vidjeti na slikama 3.1. i 3.2.



Slika 3.1. - Croatia Yacht Charter [10]

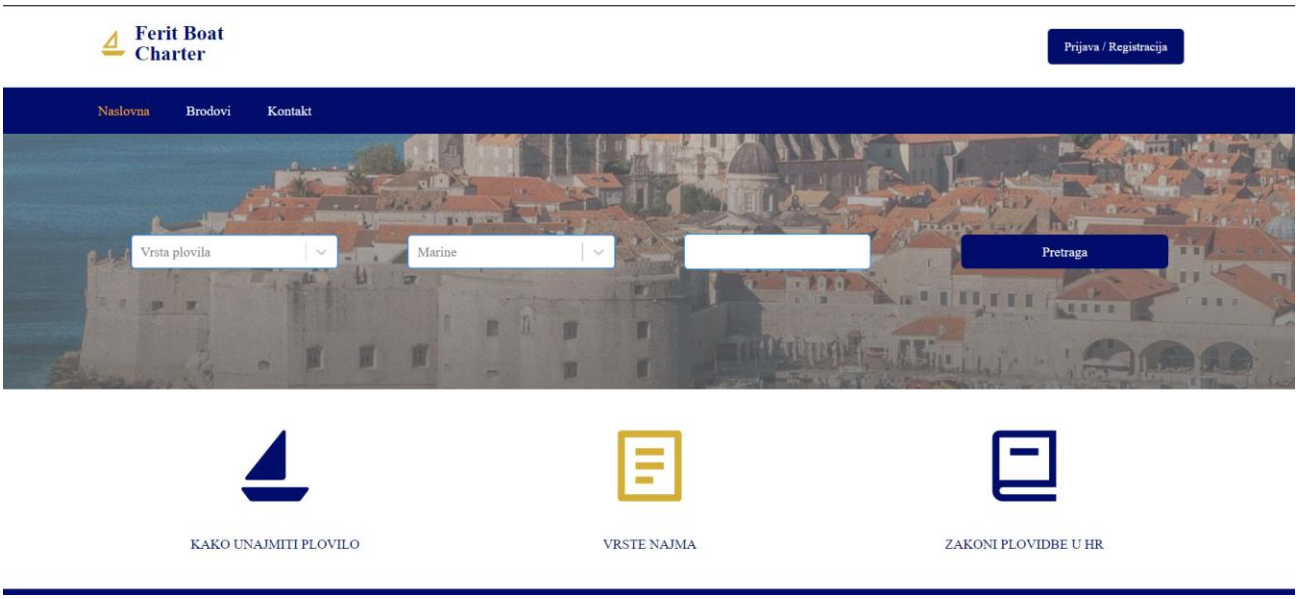


Slika 3.2. - Croatia Yacht Charter – pregled brodova [11]

4. OPIS WEB APLIKACIJE

Web aplikacija završnog rada je zamišljena da se koristi, implementira, u tvrtkama za najam brodica. Omogućuje jednostavno postavljanje brodica za najam te pretragu dostupnih brodica za iznajmljivanje te nenametljivo nudi podatke esencijalne za poznavanje pri iznajmljivanju brodica.

4.1. Naslovna / Početna stranica



Slika 4.1. – Naslovna stranica

Prva stranica koja se vidi kada se pristupi Ferit Boat Charter je naslovna stranica.

Na vrhu naslovne stranice osim ikone i naziva može se vidjeti u gornjem desnom kutu dugme koje vodi na ekran za prijavu ili registraciju korisnika. Kada se korisnik prijavi, odnosno registrira, tipka postaje dugme za odjavu i dugme za pregled svojih najmova ili dugme za iznajmljivanje svoje brodice (ovisno o vrsti korisnika koji se prijavi). Ispod prvog dijela naslovne stranice se nalazi glavi izbornik. Ova dva dijela koja se nalaze na vrhu stranice zajedno čine zaglavlje(eng. *header*), a oni s podnožjem (eng. *footer)*, koje je prikazano na slici 4.2., ispunjavaju trajni izgled stranice.

Slika 4.2. – Podnožje

Podnožje se sastoji od naziva i opisa stranice/tvrtke na lijevoj strani. Na sredini se nalazi izbornik s istim članovima kao i u zaglavlju stranice te desno od izbornika pišu osnovni podaci za kontakt (broj mobitela, adresa, e-mail i radno vrijeme).

Na samoj sredini naslovne stranice ima forma za pretragu i veliki izbornik.

Forma za pretragu se sastoji od tri polja za odabir:

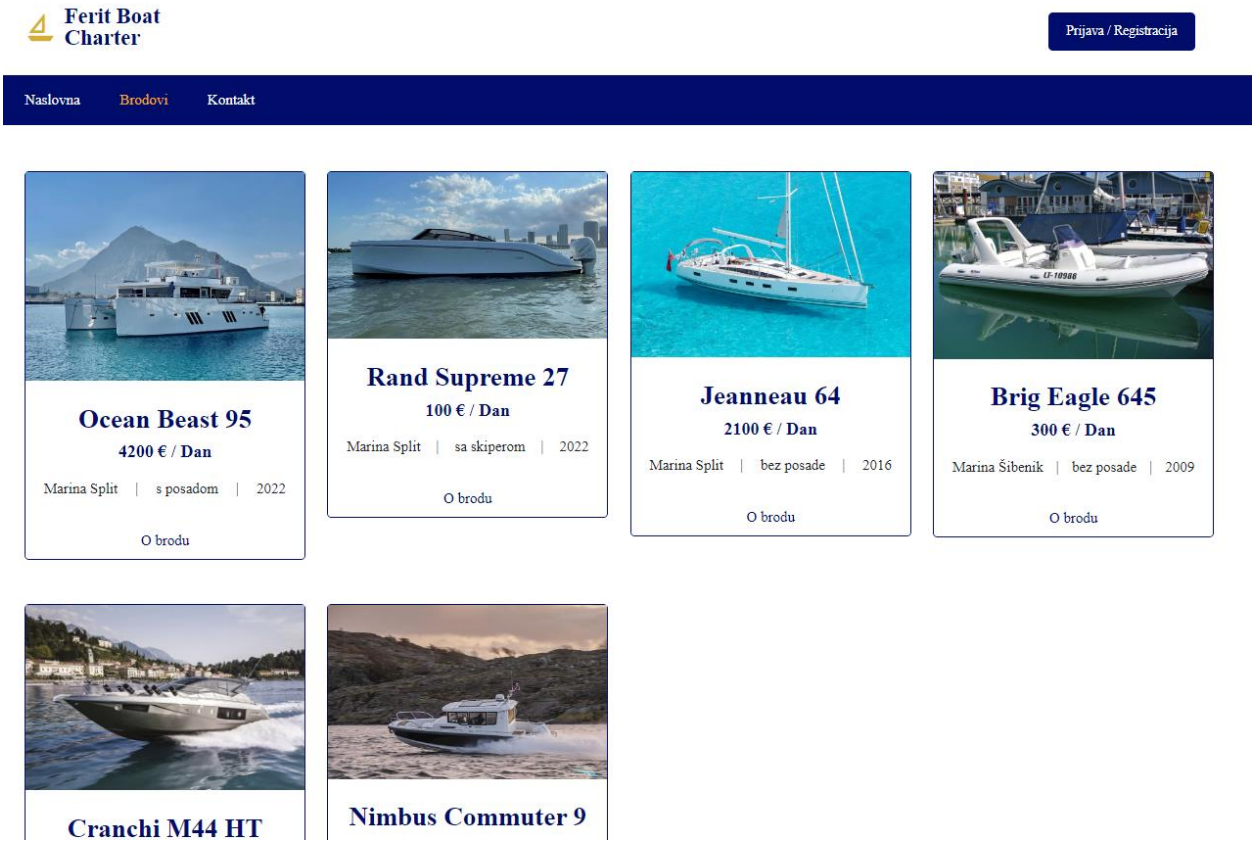
1. Vrsta broda koja se želi unajmiti
2. Marine u kojoj se brod želi preuzeti
3. Datume od kojeg, do kojeg se želi unajmiti brod

Na kraju forme od tri polja nalazi se tipka koja pokreće pretragu i preusmjerava na stranicu koja prema unesenim podacima ispisuje dostupne brodove.

Ispod navedene forme nalazi se izbornik od tri člana: Kako unajmiti plovilo, Vrste najma i Zakoni plovidbe u HR. Svaka od ova 3 izbora vode na pojedinačnu stranicu s tekstom koji opisuje pojedinu temu.

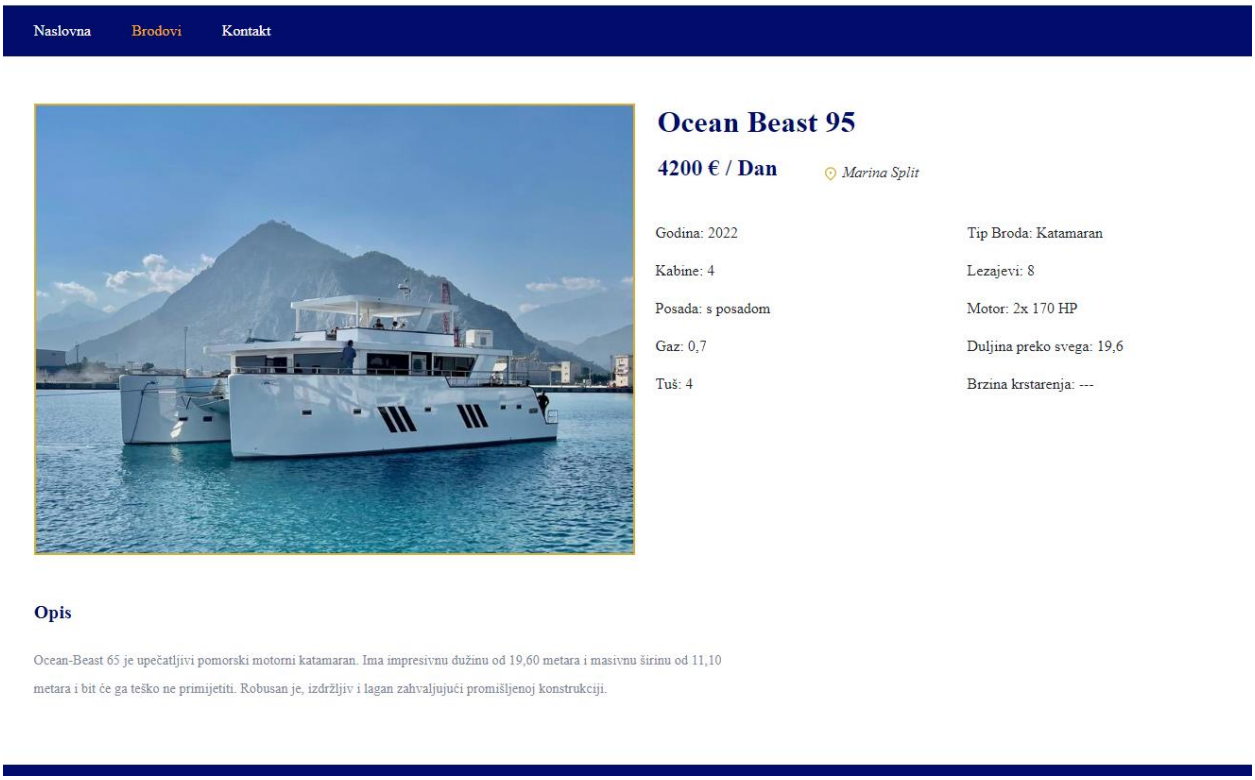
4.2. Stranica za brodice

Na ovoj se stranici, prikazanoj na slici 4.3, ispisuju svi brodovi koji se nalaze unutar baze podataka. Svaki brod pojedinačno se nalazi u svom prozoru u kojem se mogu iščitati naziv broda, cijena za najam, mjesto gdje se nalazi, posada, te godina proizvodnje.



Slika 4.3 – Stranica za brodice

Na dnu svakog prozora broda nalazi se tipka „O brodu“ s kojom se prelazi na stranicu na kojoj se nalaze svi detalji broda (Slika 4.4.).



Slika 4.4. Stranica za pojedinačni pregled broda

Na stranici „O brodu“ se može vidjeti slika broda u punoj veličini ispod koje se nalazi tekstualni opis broda, a na desnoj polovici stranice imaju svi podaci vezani za taj brod.

4.3. Role stranice

Kada se prijavljuje koristeći tipku u desnom kutu zaglavlja (Slika 4.1.) prelazi se na ekran za prijavu, odnosno registraciju (Slika 4.5., 4.6.).

Slika 4.5. Forma za prijavu

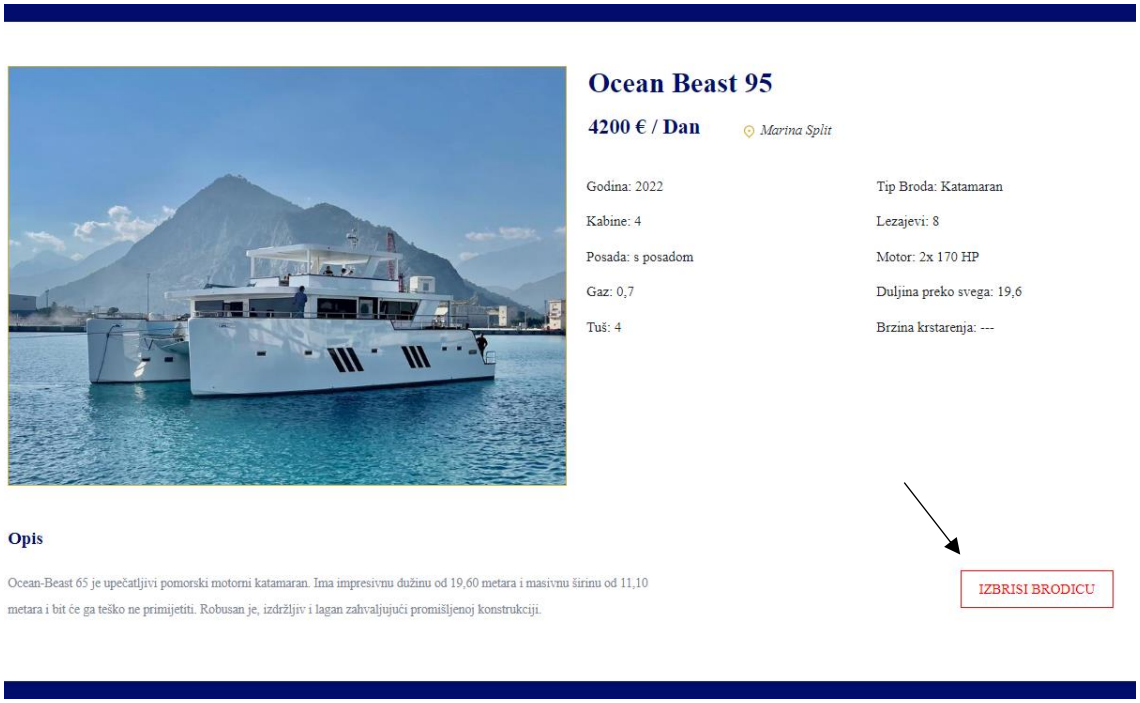
Slika 4.6. Forma za registraciju

Web stranica ima četiri role: admin, posjetitelj, iznajmitelj i unajmitelj. One se razlikuju u mogućnostima koje svaka rola pruža.

Unoseći i izborom podataka u poljima za registraciju i klikom na tipku „Registracija“ pravi se novi korisnik koji se zapisuje u bazu podataka. Već prethodno registrirani korisnici se prijavljuju upisujući svoje podatke u polja sučelja sa slike 4.5. i ako podaci odgovaraju onima u bazi podataka korisnik dobiva pristup web stranici.

Admin je rola koju imaju samo odobrene osobe (zaposlenici). Admin ima mogućnost dodati nove brodove u bazu podataka kako bi bili dostupni korisnicima za najam te izbrisati bilo koju brodicu koja je postavljena na stranici. Brisanje brodice se radi klikom na tipku „Izbrisi brodicu“ pokazanom strelicom na slici 4.8.

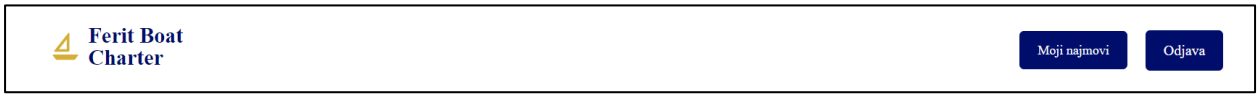
Slika 4.7. Prikazuje kako izgleda zaglavlje kada se prijavi admin



Slika 4.8. Stranica za pojedinačni pregled broda kada je korisnik, iznajmitelj prijavljen

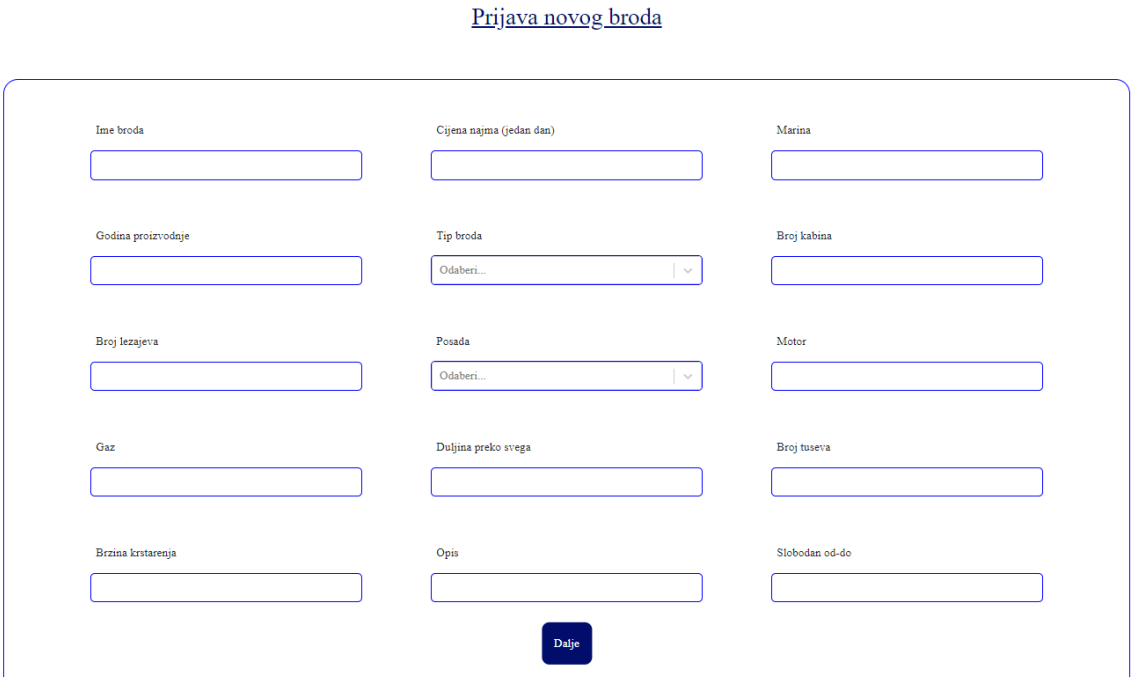
Posjetitelj je rola koju svaka osoba ima kad se nalazi na web stranici, a nije prijavljena.

Iznajmitelj je korisnik koji ima mogućnost iznajmiti brod koji je u ponudi postavljen na stranici. On ima i mogućnost pregleda svojih najmova klikom na dugme „Moji najmovi“ prikazanom na slici 4.9.



Slika 4.9. Prikazuje kako izgleda zaglavlje kada se prijavi iznajmitelj

Unajmitelj je osoba koja može postaviti svoj brod kako bi ga drugi korisnici (iznajmitelji) mogli iznajmiti. Slika 4.10. prikazuje formu pomoću koje se dodaje novi brod. Unajmitelj može i brisati brodicu, ali za razliku od admina koji može izbrisati bilo koju, unajmitelj može izbrisati samo onu koju je on postavio.

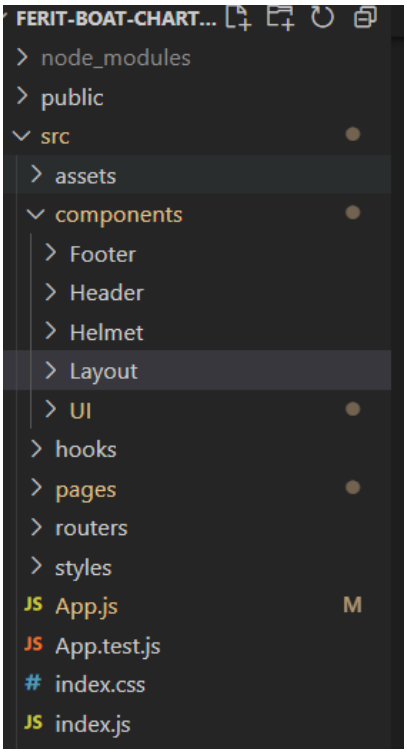


Slika 4.10. Forma za dodavanje novog broda

5. PROGRAMSKO RJEŠENJE

5.1 Postavljanje/strukturiranje frontend koda

Izrada i kodiranje web aplikacije se započinje postavljanjem odgovarajućeg rasporeda datoteka unutar Visual Studio Coda, uređivača koda. (Slika 5.1.).



Slika 5.1. Prikazuje raspored *frontend* dijela aplikacije

Zatim se postavlja raspored (eng. *layout*) web aplikacije koji odgovara SPA, aplikaciji s jednom stranicom (eng. *Single Page Application*) unutar datoteke koja je nazvana *Layout* (Slika 5.2.).

```
import React, { Fragment } from "react";

import Header from "../Header/Header";
import Footer from "../Footer/Footer";
import Routers from "../../routers/Routers";

const Layout = () => {
  return (
    <Fragment>
      <Header />
      <div>
      <Routers />
      </div>
      <Footer/>
    </Fragment>


  );
};

export default Layout;
```

Slika 5.2. Kod za raspored *frontenda* aplikacije

Layout se sastoji od *Fragment* elementa, unutar kojeg se nalaze stalni *Header* i *Footer* te između njih element *Routers* koji su prikazani na slici 5.3., koji omogućuju prikazivanje ostalog sadržaja.

```
const Routers = () => {
    return (
        <Routes>
            <Route path="/*" element={<Navigate to="/home" />} />
            <Route path="/home" element={<Home/>} />
            <Route path="/boats" element={<BoatListing />} />
            <Route path="/contact" element={<Contact/>} />
            <Route path="/boats/:slug1/:slug2" element={<BoatDetails />} />
            <Route path="/kako-unajmiti-plovilo" element={<KakoUnajmitiPlovilo />} />
            <Route path="/zakoni-plovidbe-u-hr" element={<ZakoniPlovidbeUHR />} />
            <Route path="/vrste-najma" element={<VrsteNajma />} />
            <Route path="/prijava" element={<LogRegSelect />} />
            <Route path="/registracija" element={<Register />} />

            <Route element={<ProtectedRoutersUnajmiteljAdmin />} >
                <Route path="/novi-brod-forma" element={<NewBoatForm/>} />
                <Route path="/dodajSliku/:slug" element={<PictureAdd />} />
            </Route>

            <Route element={<ProtectedRoutersIznajmljivac />} >
                <Route path="/kosarica/:slug1" element={<Kosarica />} />
                <Route path="/prijeKosarica/:slug1" element={<PrijeKosarica />} />
                <Route path="/mojiNajmovi" element={<MojiNajmovi/>} />
                <Route path="/urediNajam/:slug1/:slug2" element={<Uredi/>} />
            </Route>
        </Routes>
    );
};
```

Slika 5.3. Rute (eng. *Routes*) za usmjeravanje kroz aplikaciju

Mijenjanjem URL-a dolazi i do odgovarajuće promjene izgleda aplikacije. Neke od *Route* elemenata su postavljeni unutar nadređenih *Route* elemenata tako da njima mogu pristupiti samo odgovarajući korisnici nakon što su se prijavili.

Na slici 5.4. je prikazana datoteka unutar koje se radi provjera statusa korisnika i prema tome odobrava pristup željenoj stranici ili šalje korisnika na stranicu za prijavu. Status korisnika predstavlja vrstu korisnika:

- Status 0 predstavlja admin korisnika
- Status 1 predstavlja iznajmljivača
- Status 2 predstavlja unajmitelja

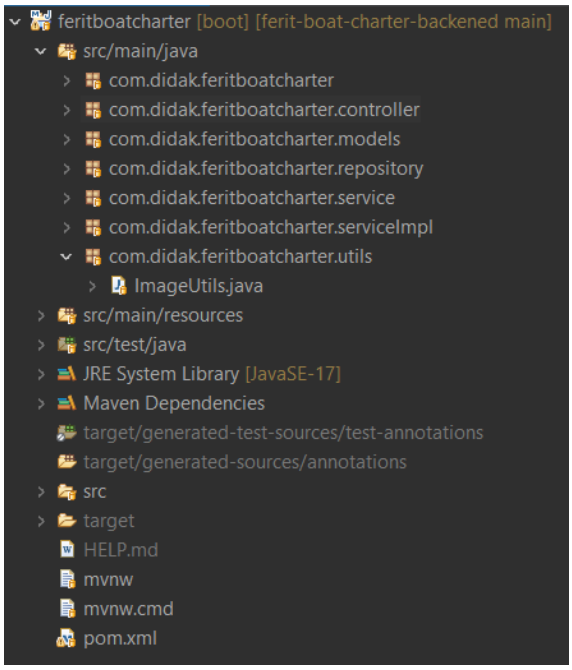
Kao i u ovoj datoteci dalje u aplikaciji će se koristit status za određivanje pristupa određenim mogućnostima.

```
const ProtectedRoutersUnajmiteljAdmin = () => {

    if (localStorage.getItem('statusUser') == 2) {
        return <Outlet/>;
    }else if (localStorage.getItem('statusUser') == 0) {
        return <Outlet/>;
    }else{
        return <LogRegSelect/>;
    }

}

export default ProtectedRoutersUnajmiteljAdmin;
```

Slika 5.4. Metoda koja omogućuje pristup rutama ovisno o prijavljenom korisniku

5.2. Postavljanje backenda

Backend dio koda web aplikacije je strukturiran u mape (eng. *folder*) s istim „baznim“ dijelom naziva (com.didak.feritboatcharter) koji su odgovarali aplikaciji, a svaki naziv mape završava drugačije, odnosno prema tome koju vrstu datoteka je sadržavao (.controller, .models, .service...). Slika 5.5. prikazuje *backend* strukturu.



Slika 5.5. Raspored *backend* dijela aplikacije

Controller datoteke služe za pozivanje REST API poziva. Postoji više vrsta poziva, ali ova web aplikacija koristio samo GET koji služi za dohvaćanje podataka iz baze podataka i POST koji služi za slanje, zapisivanje podataka u bazu. Slika 5.6. prikazuje primjer *controller* datoteke.

```java
@RestController
@RequestMapping("/brod")
@CrossOrigin
public class BoatController {

    @Autowired
    private BrodService brodService;

    @GetMapping("/getAll")
    public List<Brod> getAll(){
        return brodService.getAllBrod();
    }

    @GetMapping("/getBrodWithId")
    public Brod getBrodWithId(@RequestParam int id){
        return brodService.getBrodWithId(id);
    }

    @GetMapping("/getBrodWithName")
    public Brod getBrodWithName(@RequestParam String ime){
        Brod temp = brodService.getBrodWithName(ime);
        return temp;
    }
```

Slika 5.6. Primjer *controller* datoteke

Model datoteke sadrže klase objekata (Brod, Najam, Marina, Korisnik...) koje se pozivaju i zapisuju kroz rad u aplikaciji.

```java
@Entity
public class Brod {
    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int id;

    private int userId;
    private String ime;
    private String tip;
    private String regija;
    private int godina;
    private int kabine;
    private int lezajevi;
    private String gaz;
    private String brzina;
    private String duljinaPrekoSvega;
    private int tus;
    private int cijena;
    private String posada;
    private String motor;
    private String opis;
    private Date slobodanOd;
    private Date slobodanDo;
```

Slika 5.7. Prikazuje klasu Broda koja se nalazi unutar .models mapi

Service datoteke su sučelja (eng. *interface*) koja služe samo za pozivanje metoda, a *serviceImpl* mape sadrže datoteke koje implementiraju metode zapisane u service.

```java
public interface BrodService {
    public List<Brod> getAllBrod();
    public List<Brod> getBrodForDropDownForm(String tipBroda, String regija, String slobodanOd, String slobodanDo);
    public Brod getBrodWithId(int id);
    public Brod getBrodWithName(String ime);
    public List<Brod> getBrodWithRegijaTipBroda(String tipBroda, String regija);
    public List<Brod> getBrodWithRegija(String regija);
    public List<Brod> getBrodWithTipBroda(String tipBroda);
    public List<Brod> getBrodWithDatumi(String slobodanOd, String slobodanDo);

    public Brod addNewBrod(Brod brod);
    public String uploadImage(MultipartFile file, String ime);

    public void deleteBrod(String ime);
}
```

Slika 5.8. Primjer *service* datoteke, odnosno sučelja u .service folderu

```java
@Override
public List<Brod> getAllBrod() {
    return brodRepository.findAll();
}

@Override
public Brod getBrodWithId(int id) {
    String sql = "select id, brzina, cijena, duljina_preko_svega, gaz, godina, ime, slika, kabine, lezajevi, motor, opis, posada, "
            + "regija, tip, tus, slobodan_do, slobodan_od from brod where id = ?";

    return jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(Brod.class) ,new Object[] {id});
}
```

Slika 5.9. Primjer datoteke unutar *serviceImpl* mape

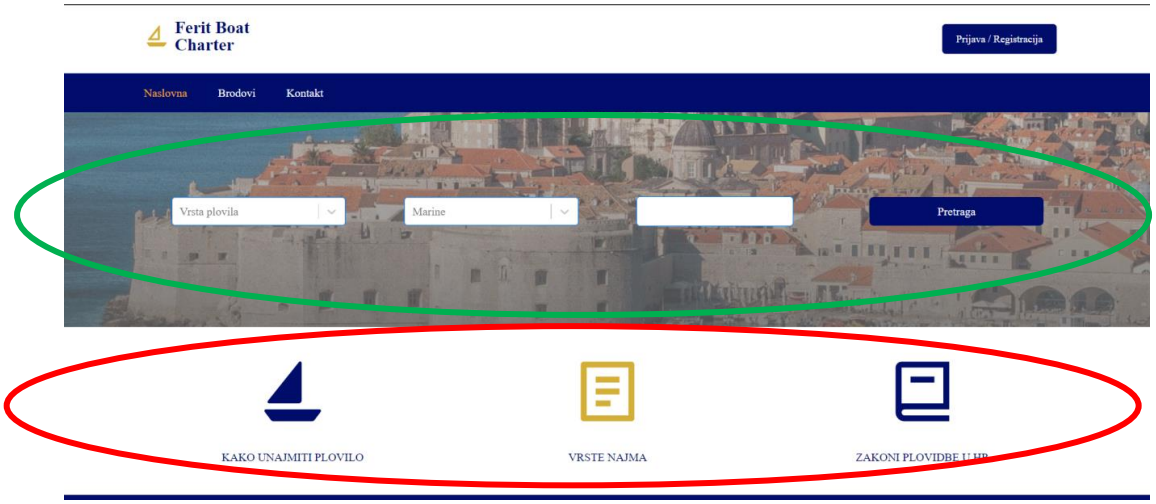
5.3. Početna stranica

Početna stranica, pod nazivom *Home*, je prva stranica koja se otvori kada se pristupi aplikaciji. Ona je strukturirana s dva elementa, odnosno datoteke, stranice, koje se pozivaju unutar *Home* datoteke (Slika 5.10.).

```java
import DropDownBoatForm from "../components/UI/DropDownBoatForm";
import BasicInformation from "./BasicInformation";

const Home = () => {
    return(
        <div>
            <DropDownBoatForm/>
            <BasicInformation/>
        </div>
    )
}

export default Home;
```

Slika 5.10. Kod za raspored početnog ekrana

Ta dva elementa su *DropDownBoatForm* (zaokruženo zelenom bojom) i *BasicInformation* (zaokruženo crvenom bojom) (Slika 5.11.).



Slika 5.11. Početna stranica

Zadaća *DropDownBoatForm* je brza pretraga brodica. Pretraga je ubrzana jer se mogu odabirati tri parametra prema kojima se sužava izbor brodica. Ta tri parametra su vrsta plovila (brodice), koju korisnik traži, marina u kojoj se brodica nalazi te datum od kojeg do kojeg bi korisnik želio iznajmiti brodicu. Slika 5.12. prikazuje kod *DropDownBoatForm*.

```
<FormGroup className="form__group">

<Select
    placeholder="Marine"
    className="basic-single"
    classNamePrefix="select"
    isClearable={isClearable}
    isSearchable={isSearchable}
    name="color"
    onChange={(update) => {
      setOdabranaMarina(update);
      console.log(odabranaMarina);
      }}
    options={marine}
/>

</FormGroup>

<FormGroup className="form__group">

<DatePicker
  dateFormat={'dd-MM-yyyy'}
  minDate={new Date()}
  selectsRange={true}
  startDate={startDate}
  endDate={endDate}
  onChange={(update) => {
  setDateRange(update);
  console.log(startDate);
  }}
  isClearable={true}
/>
</FormGroup>
```

Slika 5.12. Kod forme za traženje brodica na početnoj stranici

Odabir željenih parametara se radi koristeći dva padajuća izbornika unutar kojih se učitavaju podaci zapisani u bazi te jednim biračem datuma (eng. *DatePicker*). Učitavanje podataka iz baze se radi pomoću *fetch* nakon čega se rezultati poziva mapiraju na varijablu *temp* preko koje se postavlja vrijednost varijable koja se ispisuje u padajućem izborniku (Slika 5.13).

```
useEffect(()=>{
  fetch("https://ferit-boat-charter-backened-production.up.railway.app/marina/getAll")
  .then(res=>res.json())
  .then((result)=>{

    let temp = result.map((d) => ({
      value: d.id,
      label: d.naziv
    }))
    setMarine(temp);
  }
)
},[])
```

Slika 5.13. Primjer poziva marina iz baze podataka s fetch

Ovaj poziv na *backend* se radi pozivajući sučelje *MarinaRepository* koje je prošireno koristeći *JpaRepository* pa omogućuje pozivanje metode *findAll* koja automatski dohvaća sve podatke iz baze (Slika 5.14).

```
@Autowired
private MarinaRepository marinaRepository;

@Override
public List<Marina> getAllMarina() {
    return marinaRepository.findAll();
}
```

Slika 5.14. Metoda za dohvaćanje svih marina iz baze podataka

Nakon što se odaberu željeni parametri i spreme se u odgovarajuće varijable one se postave u URL koji će dalje preusmjeriti na stranicu s odgovarajućim brodicama.(Slika 5.15.)

```
  if (selectedBoat !== undefined && selectedBoat !== null) {
    url = url + ("?brod=" + selectedBoat.label);
  }
  if (selectedMarine !== undefined && selectedMarine !== null ) {
    if(url.indexOf("?") !== -1) url = url + ("&marine=" + selectedMarine.label);
      else url = url + ("?marine=" + selectedMarine.label);
  }

  if (selectedStartDate !== undefined && selectedStartDate !== null) {
    if (selectedEndDate !== undefined && selectedEndDate !== null) {
      if(url.indexOf("?") !== -1) url = url + ("&startDatum=" + selectedStartDate + "&endDatum=" + selectedEndDate);
      else url = url + ("?startDatum=" + selectedStartDate + "&endDatum=" + selectedEndDate);
    }
  }

  console.log(selectedStartDate)
  console.log(selectedEndDate)
  navigate(url);
}
```

Slika 5.15. If petlja za slaganje URL-a

Drugi element na početnoj stranici, pod imenom *BasicInformation* je jednostavni element, *Container*, koji se sastoji od jednog retka u kojemu se nalaze tri stupca. Svaki od ta tri stupca sadrži linkove (element *Link*) s ikonama koje preusmjeravaju na stranice s odgovarajućim tekstom.

5.2. Stranica za ispisivanje brodica

Ova stranica pored svog posebnog HTML rasporeda poziva podatke iz baze kao i *DropDownBoatForm* ali umjesto *fetch* koristi *axios* kao tehniku poziva podataka brodica. Kao i *DropDownBoatForm* na *backendu* koristi *JpaRepository* proširenje za poziv svih brodova iz baze. Podatke dobivene iz baze dalje mapira na element *BoatItem*.

Podaci primljeni u *BoatItem* se dalje raspoređuju i ispisuju u HTML-u te se ime broda prosljeđuje linku za otvaranje stranice za detaljni pregled brodice. Stranica za detaljni prikaz brodice, na koju se dolazi klikom na dugme „O brodu“, je već opisana u poglavlju 4.2.

5.3. Najam brodice

Najam brodice se radi klikom na dugme „UNAJMI BRODICU“, prikazano slika 5.16., koje je dostupno u *BoatItem* elementu kada je prava vrsta korisnika prijavljena.

Provjera koja vrsta korisnika je prijavljena se radi tako što se pozove status korisnika iz lokalne pohrane (eng. *localStorage)* (Slika 5.17.) te se u *if* petlji usporedi (Slika 5.18.). Ako je status jednak 1 onda se postavlja da se prikaže dugme „UNAJMI BRODICU“.

Slika 5.16. Dugme „Unajmi brodicu“

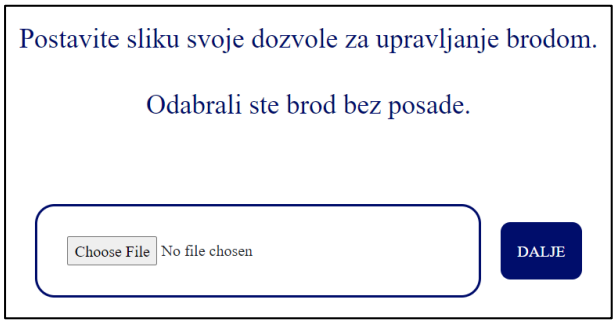
```
const tempStatus = localStorage.getItem('statusUser')
```

Slika 5.17. Pozivanje statusa korisnika iz lokalne memorije i postavljanje njegove vrijednosti na *tempStatus* varijablu

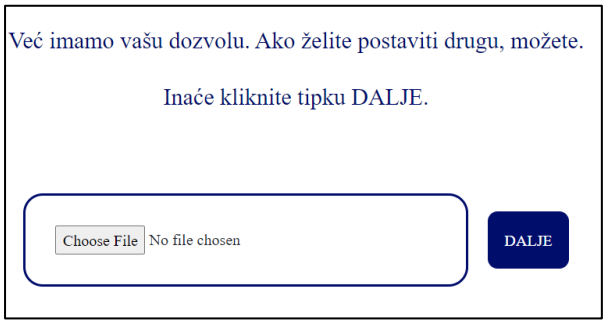
```
useEffect(() => {
  if (tempStatus == 1) {
    setLook1(<IznajmitButton ime={slug1} posada={brod.posada}/>)
  }
},[brod])
```

Slika 5.18. Postavljanje izgleda stranice ovisno o prijavljenosti korisnika

Ako brodica koju korisnik želi iznajmiti nema posade ni skipera onda dugme „UNAJMI BRODICU“ vodi na stranicu za dodavanje dozvole za upravljanje brodicom (Slika 5.19. i 5.20.). U suprotnom, dugme vodi direktno na košaricu, na stranicu za odabiranje trajanja najma i njegovo plaćanje (Slika 5.21.).



Slika 5.19. Forma za dodavanje dozvole



Slika 5.20. . Forma za dodavanje dozvole

Slika 5.21. Forma za upis novog najma

Dozvola za upravljanje brodicom se dodaje kao slika. Na svaku promjenu varijable odabrane s dugmom „*Choose File*“ prolazi se kroz funkciju *HandleChange* koja primljenu sliku postavlja na varijablu *file*. Dalje se varijabla *file* zajedno s korisnikovim id-om postavlja kao podatak unutar *data* varijable koja se prosljeđuje na *backend*. (Slika 5.22.)

```
const handleSubmitDozvola = (e) => {
    e.preventDefault()
    if (file.length != 0 && korisnikPostojecaDozvola == false){
    const data = new FormData();
    data.append("image", file);
    data.append("id", userId);
    fetch("https://ferit-boat-charter-backened-production.up.railway.app/korisnik/addDozvola", {
            method:"POST",
            body: data,
        }).then(()=>{
            navigate(`/kosarica/${ime.ime}`)
        })
    }else if (korisnikPostojecaDozvola == true) {
        navigate(`/kosarica/${ime.ime}`)
    }else{
        nemaDozvole();
    }
}

function handleFileChange(e){
    if(e.target.files && e.target.files[0]) setFile(e.target.files[0]);
}
```

Slika 5.22. Metoda za dodavanje dozvole

Na *backend* strani aplikacije slika se prima u obliku *Mutipart* vrste varijable, *file*, a korisnikov id u obliku *int*, pod nazivom id, u *addDozvola* funkciji u *controlleru*. Dalje unutar *addDozvola* funkcije se poziva iz *korisnikService*, funkcija *addDozvola* koja prima proslijeđene varijable *file* i id iz roditeljske funkcije. (Slika 5.23.)

```
@PostMapping("/addDozvola")
public void addDozvola(@RequestParam("image")MultipartFile file, @RequestParam("id")int id) {
    korisnikService.addDozvola(file, id);
}
```

Slika 5.23 Metoda za dodavanje dozvole u bazu podataka u *korisnikController*

Unutar *korisnikServiceImpl* funkcija *addDozvola* primljenu varijablu kodira u *String* vrstu varijable, koristeći *Base64* klasu te pozivajući statičke metode za kodiranje. Slika zapisana u tom obliku je dalje predana *jdbcTemplate update* funkciji koja sliku, te id korisnika mapira na SQL upit, koji ažurira bazu podataka dodajući sliku dozvole gdje je odgovarajući id korisnika. (Slika 5.24.)

```
@Override
public void addDozvola(MultipartFile file, int id) {
    String img = null;

        try {
            img = Base64.getEncoder().encodeToString(file.getBytes());
        } catch (IOException e) {
            e.printStackTrace();
        }

        String sql = "UPDATE korisnik SET dozvola= ? WHERE id= ? ";

        jdbcTemplate.update(sql, img, id);

}
```

Slika 5.24. Metoda koja sliku pretvara u *String* i zapisuje ju u bazu sa SQL upitom

Slika 5.25. prikazuje kod za provjeru posade broda i za dugme „UNAJMI BRODICU“.

```
const IznajmitButton = ({ime, posada}) => {

    const [urlNajam, setUrlNajam] = useState();

    useEffect(() => {
    if (posada == "bez posade") {
        setUrlNajam(`/prijeKosarica/${ime}`)
    }else {
        setUrlNajam(`/kosarica/${ime}`)
    }
},[posada])

    return(
        <Link to={urlNajam}>
            <button className="contact-button">UNAJMI BRODICU</button>
        </Link>
    )
};

export default IznajmitButton;
```

Slika 5.25. Metoda koja provjerava vrstu posade broda

Na stranici košarica, odabire se datum od kojeg do kojeg se želi iznajmiti brod. Glavna prepreka kod kodiranja ovog dijela najma je bila kako filtrirati datume kada je brod dostupan, jer je brod dostupan

za najam samo određeni period. Kasnije kada se određeni brod iznajmi, onda i u tim periodima neće biti slobodan.

Pristup koji je korišten za rješavanje ove prepreke je, da se oduzimaju svi datumi koji ne odgovaraju rasponima kad je brod slobodan, odnosno da se ti datumi uopće ne mogu odabrati u biraču datuma.

Prvo je u bazu dodana tablica Najam koja se sastoji od *najam_id, brod_id, cijena, zauzet_od, zauzet_do i korisnik_id*. U ovu tablicu se zapisuju najmovi i preko nje se može pratiti koji je brod iznajmljen, koji je korisnik napravio najam i za koji period.

Proces uklanjanja datuma u biraču datuma nedostupnih za tu brodicu je započet pozivanjem podataka iz baze podataka za brodicu koja se želi iznajmiti (Slika 5.26.).

```
const getBrod = {
    method:"GET",
    url: "https://ferit-boat-charter-backened-production.up.railway.app/brod/getBrodWithName",
    params:{ime: slug1},
  }

  useEffect(() => {
  axios.request (getBrod).then((response) => {
    setBrod(response.data);
    setCompareDate1(new Date(response.data.slobodanOd));
    setCompareDate2(new Date(response.data.slobodanDo));
  }).catch((error) => {
    console.error(error);
  });
}, []);
```

Slika 5.26. Metoda za pozivanje broda iz baze podataka

Odmah unutar poziva na *backend* se postavlja vrijednost datuma od kad je brod slobodan na varijablu *compareDate1* i datum do kad je brod slobodan na *compareDate2* (zaokruženo crveno na slici 5.26.).

Poziv podataka iz baze podataka za brodicu na *backend* strani aplikacije izgleda kao prikazan na slici 5.27. Koristeći SQL SELECT upit te *queryForObject* metodu iz *jdbcTemplate* pozivaju se potrebni podaci iz baze podataka.

```java
@Override
public Brod getBrodWithName(String ime) {
    String sql = "select id, brzina, cijena, duljina_preko_svega, gaz, godina, ime, slika, kabine, lezajevi, motor, opis, posada, "
            + "regija, tip, tus, slobodan_do, slobodan_od from brod where ime = ?";
    Brod tBrod = null;

    try {
        tBrod = jdbcTemplate.queryForObject(sql, new BeanPropertyRowMapper<>(Brod.class) ,new Object[] {ime});
    } catch (EmptyResultDataAccessException e) {
        return null;
    }

    return tBrod;
}
```

Slika 5.27. Metoda koja poziva sve podatke broda preko SQL upita

Zatim je napravljena funkcija *getDatesInbetween*, koja pronalazi sve datume između dva ulazna datuma funkcije i zapisuje ih u brojčanom obliku u polje koje vraća (Slika 5.28.).

```javascript
const getDatesInbetween = (compareDate1, compareDate2) => {
    let dateList = [];
    let currentDate = new Date(compareDate1);

    while(currentDate <= compareDate2){
        dateList.push(currentDate.getTime());
        currentDate.setDate(currentDate.getDate() + 1);
    }
    return dateList;
}
```

Slika 5.28 Funkcija za pronalaženje svih datuma između dva ulazna datuma

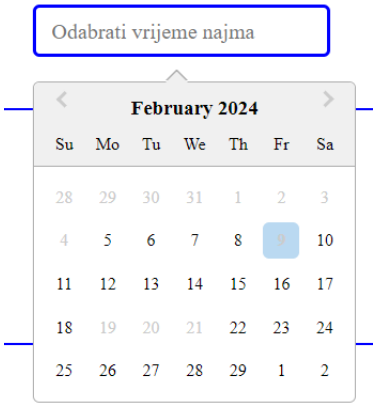
Rezultate funkcije *getDatesInbetween* kojoj se predaje datum prvog dana u mjesecu dva mjeseca u natrag i datum od kad je brod slobodan zapisuju se u varijablu *datesExcludeSlobodanOd*, a rezultate funkcije *getDatesInbetween* kojoj se predaju datum do kad je brod slobodan i datum šest mjeseci u naprijed zapisuju u varijablu *datesExcludeSlobodanDo*. (Slika 5.29.)

```javascript
var date = new Date();
var today = dateToNumber(Date());
const datesExcludeSlobodanOd  = getDatesInbetween(new Date(date.getFullYear(), date.getMonth()-2, 1), compareDate1)
const datesExcludeSlobodanDo = getDatesInbetween(compareDate2, date.setMonth(date.getMonth() + 6))
const datesExludeBeforeToday = getDatesInbetween(compareDate1, new Date())
```

Slika 5.29. Prikazuje inicijaliziranje varijabli za pohranu datuma te dodjeljivanje vrijednosti pozivanjem metode *getDatesInbetween*

Problem koji se pojavio pri uklanjanju datuma je kada datum dana na koji se želi napraviti novi najam nalazi poslije dana od kada je brod slobodan. U ovom slučaju i dalje je moguće odabrati dane koji su već prošli, odnosno dane između datuma od kada je brod slobodan i dana kada se pravi novi

najam. Primjer ovog slučaja se može vidjeti na slici 5.30. na kojoj se mogu odabrati datumi 5.,6.,7., iako se najam pravi 8. Kako bi se ovaj problem riješio, ponovo se poziva funkciju *getDatesInbetween* kojoj se predaje datum od kad je brod slobodan i datum današnjeg dana (*new Date()* automatski poziva datum današnjeg dana).



Slika 5.30. Primjer birača datuma kada je moguće odabrati dane prije današnjeg

Preostalo je još oduzeti raspon datuma za postojeće najmove tog broda i datume za postojeće najmove tog korisnika, jer korisnik ne može imati više najmova istovremeno.

Nakon pozivanja svih najmova iz tablice Najam iz baze podataka koji imaju odgovarajući id broda i svih najmova koji imaju odgovarajući id korisnika koristeći *for* petlju prolazi se kroz polje najmova koje je bilo povratna informacija s *backenda* (Slika 5.31.).

```
useEffect(() => {
  var tmp = [...datesExcludeSlobodanOd, ...datesExcludeSlobodanDo]
  var tmp2 = [...tmp, ...datesExludeBeforeToday]
    if(najmovi !== undefined){
        for (let index = 0; index < najmovi.length; index++) {
            var zauzetOdTemp = new Date(najmovi[index].zauzetOd)
            var zauzetDoTemp = new Date(najmovi[index].zauzetDo)
            var datesTmp = getDatesInbetween(zauzetOdTemp, zauzetDoTemp)
            finalEx = finalEx.concat(datesTmp);
        }
    }

    if (allUserNajam !== undefined) {
      for (let index = 0; index < allUserNajam.length; index++) {
        var zauzetOdUserTemp = new Date(allUserNajam[index].zauzetOd)
        var zauzetDoUserTemp = new Date(allUserNajam[index].zauzetDo)
        var datesUserTmp = getDatesInbetween(zauzetOdUserTemp, zauzetDoUserTemp)
        finalEx = finalEx.concat(datesUserTmp);
      }
  }

  setFinalExclude([...finalEx, ...tmp2])
}, [najmovi])
```

Slika 5.31. Kod koji prolazi kroz polje najmove i zapisuje nepotrebne datume u polje

Pri svakom prolasku kroz polje najmova datumi najma zauzetOd i zauzetDo se postavljaju u privremene varijable koje se dalje prosljeđuju funkciji *getDatesInbetween* kako bi se pronašli datumi između. Svi dobiveni datumi se zapisuju u polje *finalEx* koristeći metodu *concat* pri svakom prolasku kroz *for* petlju.

Nakon *for* petlji za najmove odgovarajućeg broda i sve najmove odgovarajućeg korisnika polje *finalEx* se spaja s poljem tmp2 koje je spoj svih prethodno dobivenih polja datuma (*datesExcludeSlobodanOd*, *datesExcludeSlobodanDo*, *datesExcludeBeforeToday* – Slika 5.29.) u polje *finalExclude* koje se predaje atributu *excludeDates* unutar birača datuma, *DatePickera*. Na ovaj način se onemogućuje odabir datuma koji su već zauzeti i koji ne odgovaraju rasponima zapisanim u bazu.

Unutar košarice još se prema broju dana koliko traje najam odmah ispisuje ukupna cijena koja se treba platit za cijeli najam.

Na kraju se s gumbom „Plaćanje“ (Slika 5.21.) šalju podaci za novi najam na *backend* te zapisuju u bazu podataka u tablicu Najam.

5.4. Pregled najmova

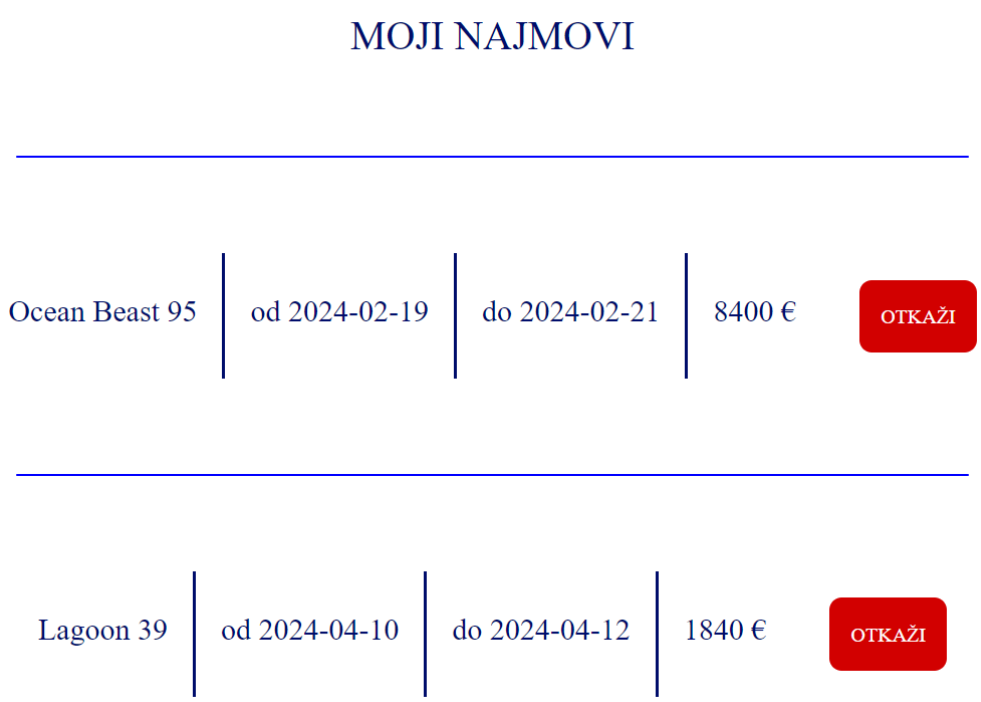
Korisnik iznajmitelj kada pregledava svoje najmove u slučaju da nema upisanih najmova bit će dočekan sa stranicom na slici 5.32.

Slika 5.32. Izgled dijela stranice kada korisnik nema brodica u najmu

Na stranici MojiNajmovi koristeći uvjetno prikazivanje (eng. *conditional rendering*) odnosno uvjetni operator, uvjet ? točno : netočno (Slika 5.33.), mijenja se prikaz između teksta „NEMATE NITI JEDNU REZERVACIJU“ (Slika 5.32.) i ispisivanja najmova korisnika (Slika 5.34.).

```
<h2 className="najamContainer" > MOJI NAJMOVI </h2>

{najamCheck ? <p></p> : <h5 className="najamContainer" id="nemaRezervacije" >NEMATE NITI JEDNU REZERVACIJU</h5>}

{najamBrod.map((item) => (
    <MojiNajmoviItem item={item} key={item.id} />
))}
```

Slika 5.33. Prikazuje kod koji mijenja izgled stranice ovisno o broju trenutnih najmova korisnika



Slika 5.34. Izgled stranice „Moji najmovi“ kada korisnik ima upisane najmove

MojiNajmoviItem element koji se poziva u MojiNajmovi ispisuje sve najmove te za svaki ispisuje dugme „Otkaži“. Dugme „Otkaži“ omogućuje otkazivanje najma odnosno njegovo brisanje iz baze podataka.

Prema uputama mentorice da iznajmljivač brodice mora platit kaznu ako otkaže najam pet dana prije početka najma uvodi se varijabla *daysToNajam* koja ima vrijednost razlike datuma od kad počinje najam i trenutnog datuma, odnosno sadrži koliko dana ima do početka najma.

```
useEffect(() => {
    var diffInMilliseconds = Math.abs(today - zauzetOdFull);
    var diffInDays  = Math.ceil(diffInMilliseconds / (1000 * 60 * 60 * 24));
    setDaysToNajam(diffInDays)
  },[])
```

Slika 5.35. Metoda koja računa koliko dana ima do najma

Kako bi korisnici bili sigurni da žele platiti kaznu kada otkazuju najam postavljeno je da se broji broj klikova na dugme „Otkaži“. Tako na prvi klik dugmeta korisnik dobije obavijest „Do najma ima manje od 5 dana! Ako otkažete sad platit će te kaznu otkazivanja.“, a tek na drugi klik briše najam.

```
const handleOtkazivanje = () => {
  if(daysToNajam <= 5){
      clickCount = clickCount + 1;
  }
  if(daysToNajam > 5) clickCount = 2;

 if (clickCount == 2) {
  const data = new FormData();
  data.append("najamId", najamId);
  fetch("https://ferit-boat-charter-backened-production.up.railway.app/najam/deleteNajam", {
          method:"POST",
          body: data,
      }).then(()=>{
          notifyUspjesnoOtkazivanjeNajma();
          window.location.reload()
      })
  }else if (clickCount = 1) {
      notifyPlacanjeKazne();
  }
```

Slika 5.36. Prikazuje funkciju koja se poziva na klik dugma „OTKAŽI“

```
@Override
public void deleteNajam(int najamId) {
    String sql = "DELETE FROM najam WHERE najam_id= ?";
    jdbcTemplate.update(sql, najamId);

}
```

Slika 5.37. *Backend* kod za brisanje najma

5.5 Obavijesti

Obavijesti, kao na primjer za otkazivanje najma, su napravljene koristeći *toast*. Koriste se pri obavještavanju korisnika o uspješnoj registraciji ili prijavi (*toast.success*), obavještavanju korisnika da nije ispunio sva polja pri unosu novog broda (*toast.warning*) te pri brisanju (*toast.error*).

```
const notifyUspjesanNajam = () => toast.success("Brod uspješno iznajmljen!")
const notifyNajamDatum = () => toast.warning("Datumi najma moraju biti određeni!")
```

Slika 5.38. Prikazuje inicijaliziranje *toast* obavijesti

5.6. Validacija/kontrola podataka

S obzirom na to da ova web aplikacija ima puno unosa različitih vrsta podataka bila to registracija novog korisnika, prijava starog, unos novog broda ili dodavanje slike dozvole korisnika, potrebne su mnoge i razne validacije podataka pri njihovom unosu.

Kod registracije novog korisnika imaju tri vrste validacije/provjere:

1. Da li je polje popunjeno/odabrano
2. Da li su e-mail i šifra pravilnog oblika
3. Da li je e-mail već korišten za pravljenje računa

Na slici 5.39. je funkcija koja provjerava je li dužina upisanog imena jednaka nuli, odnosno je li polje prazno.

```
export function validateImeiPrezime(tempImeiPrezime){
  if(tempImeiPrezime.length == 0){
    notifyIme();
    return false;
  }else{
    return true;
  }
}
```

Slika 5.39 Kod za provjeru upisanosti imena

Na slici 5.40. funkcija provjerava je li vrsta korisnika odabrana. Ako nije odabrana niti jedna vrsta korisnika onda će status predan funkciji za provjeru biti jednak -1 zbog toga što je bazna vrijednost postavljena na -1. U slučaju da je polje za ime prazno ili niti jedna vrsta korisnika odabrana korisnik će dobiti adekvatnu *toast* obavijest.

```
export function validateStatus(tempStatus){
    if(tempStatus == -1){
      notifyVrstaRacuna();
      return false;
    }else if (tempStatus == 1) {
      return true;
    }else if (tempStatus == 2) {
      return true;
    }
    return false;
}
```

Slika 5.40. Kod za provjeru odabrane vrste računa

Šifra se provjera funkcijom *validate* koja gleda da li šifra sadrži bar jedno veliko slovo, jedno malo slovo, jedan broj, jedan specijalni znak i da je minimalno dužine od 8 znakova (Slika 5.41).

```
export function validate(value) {
  const lower = new RegExp('(?=.*[a-z])');
  const upper = new RegExp('(?=.*[A-Z])');
  const number = new RegExp('(?=.*[0-9])');
  const special = new RegExp('(?=.*[!@#\$%\^&\*])');
  const length = new RegExp('(?=.{8,})')

      var lowerCheck = false;
      var upperCheck = false;
      var numberCheck = false;
      var specialCheck = false;
      var lengthCheck = false;

        if(lower.test(value)){
            console.log("Mala slova: DOBRA")
            lowerCheck = true
          }
          else{
            console.log("Mala slova: FALE")
          }

          if(upper.test(value)){
            console.log("Velika slova: DOBRA")
            upperCheck = true
          }
          else{
            console.log("Velika slova: FALE")
          }
```

Slika 5.41. Funkcija za provjeru formata upisane šifre

Slika 5.42. prikazuje funkciju koja provjerava format upisanog e-maila.

```
export function validateMail(tempMail){
  var mailPattern = /^[^ ]+@[^ ]+\.[a-z]{2,3}$/;

  if (tempMail.match(mailPattern)) {
    return true;
  }else{
    notifyMail();
    return false;
  }
}
```

Slika 5.42. Funkcija za provjeru formata e-maila

Kada validacija za popunjenost i pravilan oblik prođe onda se na *backendu* provjerava korištenost upisanog e-maila. U slučaju da je e-mail već korišten ispisuje se obavijest i ne dopušta se unos novog korisnika s tim e-mailom.

```
const handleSubmit = () => {
  if(validateImeiPrezime(ime) == true){
    if(validateMail(mail)==true){
        if(validate(sifra) == true) {
          if(validateStatus(status) == true){
            const korisnik = {ime, mail, sifra, status}
              const provjeraKoristenjaMaila = {
                method:"GET",
                url: "https://ferit-boat-charter-backened-production.up.railway.app/kori
                params:{email: mail}
              }

              axios.request(provjeraKoristenjaMaila).then((response) => {
                if (response.data == true) {
                  notifyMailVecKoristen();
                }else if (response.data == false) {
                  fetch("https://ferit-boat-charter-backened-production.up.railway.app/k
                    method:"POST",
                    headers:{"Content-Type":"application/json"},
                    body:JSON.stringify(korisnik)
                  }).then(()=>{
                    notifySucces();
                    console.log("New KORISNIK added")
                    navigate('/prijava')
                  })
                }
              }).catch((error) => {
                console.error(error);
              })
          }
      }
```

Slika 5.43. Prikazuje cijelu funkciju unosa novog korisnika sa svim provjerama pri kliku dugma za registraciju

Kod unosa novog broda (*NewBoatForm*) provjerava se je li dužina svih polja, varijabli, jednaka nuli odnosno jesu li prazna (Slika 5.44.), a gotovo sva polja su ograničena koliko se maksimalno može unijeti simbola u njih koristeći *maxLength* atribut (Slika 5.45.).

Kod nekih polja ograničena je i vrsta unosa, kao što je kod polja za unos cijene, gdje se mogu unijeti samo brojevi (Slika 5.46.).

```
const validateNewBoatForma = (e) => {

    if (ime.length == 0) {
        notifyPrazno();
        e.preventDefault();
    }else if (cijena.length == 0) {
        notifyPrazno();
        e.preventDefault();
    }else if (regija.length == 0) {
        notifyPrazno();
        e.preventDefault();
    }else if (godina.length == 0) {
        notifyPrazno();
        e.preventDefault();
    }else if (tip == null) {
        notifyPrazno();
        e.preventDefault();
    }else if (kabine.length == 0) {
        notifyPrazno();
        e.preventDefault();
    }else if (lezajevi.length == 0) {
```

Slika 5.44. Provjera upisanosti podataka u formi za novi brod

```
="" maxLength={20} ></input>

)} placeholder="" maxLength={6}></input>

eholder="" maxLength={15}></input>
```

Slika 5.45. Prisutnost *maxLenght* atributa za ograničavanje broja slova ili brojeva u polju za upis

```
setCijena(e.target.value.replace(/\D/g, ""))}
```

Slika 5.46. Kod koji postavlja vrijednost cijene i pri tome ne dopušta upis slova

Iste tehnike, provjere dužine varijable kako bi se provjerilo da li je odabrana i provjera da li već postoji zapisana u bazi koristi se pri dodavanju slike broda i dozvole za upravljanje brodicom (Slika 5.47.).

```
if (file.length != 0 && korisnikPostojecaDozvola == false){
```

Slika 5.47. *If* petlja koja provjerava prisutnost trenutne odabrane slike dozvole i one upisane u bazu podataka

Navedene validacije i provjere su rađene na *frontendu*, ali aplikacija ima i provjere na *backendu*. Prije prikaza brodica, forma za traženje brodica na početnom ekranu prolazi kroz metodu *getBrodForDropDownForm* na *backendu* u kojoj dolazi do provjere popunjenosti varijabli. (Slika 5.48.) Ovisno o njihovoj popunjenosti poziva se SQL upit.

```
@Override
public List<Brod> getBrodForDropDownForm(String tipBroda, String regija, String slobodanOd, String slobodanDo) {

    String sql;

    if(tipBroda == null && slobodanOd.equals("1970-01-01") && slobodanDo.equals("1970-01-01")) {
        sql = "SELECT * FROM `brod` WHERE regija = ?";
        return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {regija});

    }else if(regija == null && slobodanOd.equals("1970-01-01") && slobodanDo.equals("1970-01-01")) {
        sql = "SELECT * FROM `brod` WHERE tip = ?";
        return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {tipBroda});

    }else if(regija == null && tipBroda == null) {
        sql = "SELECT * FROM `brod` WHERE `slobodan_od`<= ? AND `slobodan_do`>= ?";
        return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {slobodanOd, slobodanDo});

    }else if (tipBroda == null) {
         sql = "SELECT * FROM `brod` WHERE regija = ? AND `slobodan_od`>= ? AND `slobodan_od`<= ?";
         return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {regija, slobodanOd, slobodanDo});

    }else if(regija == null) {
         sql = "SELECT * FROM `brod` WHERE tip = ? AND `slobodan_od`>= ? AND `slobodan_od`<= ?";
         return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {tipBroda, slobodanOd, slobodanDo});

    }else if(slobodanOd != "1970-01-01" && slobodanDo != "1970-01-01") {
        sql = "SELECT * FROM `brod` WHERE tip = ? and regija = ?";
        return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {tipBroda, regija});

    }else {
        sql = "SELECT * FROM `brod` WHERE tip = ? AND regija = ? AND `slobodan_od`>= ? AND `slobodan_od`<= ?";
        return jdbcTemplate.query(sql, new BeanPropertyRowMapper<>(Brod.class), new Object[] {tipBroda, regija, slobodanOd, sloboda
    }
}
```

Slika 5.48. Metoda za dohvaćanje odgovarajućih brodica

ZAKLJUČAK

Kroz ovaj završni rad cilj je bio napraviti kompletnu, sveobuhvatnu, web aplikaciju za najam, odnosno charter brodica.

Glavni ciljevi rada aplikacije su bili mogućnost ispisa svih dostupnih brodica, njihov najam te mogućnost dodavanja novih brodica. Svi ciljevi postavljeni na početku rada su uspješno ostvareni.

Tijekom rada na ovoj aplikaciji, glavna prepreka je bila rad s datumima i omogućavanje dodavanja slika. Iako rad s datumima nije potpuno novo područje, rad s njima na *frontend* dijelu aplikacije i u ovoj mjeri predstavljao je izazov. Prepreka s datumima je bila kako ih sve adekvatno organizirat da su lako dostupni u svim dijelovima aplikacije te kako ukloniti datume koji nisu uključeni za tu brodicu ili korisnika. Problem organiziranja je riješen s dobro organiziranom bazom podataka i tablicama unutar nje. Pri uklanjanju datuma koji nisu dostupni kod upisivanja novog najma morao se promijeniti način razmišljanja, s obzirom na očekivani pristup. Novo naučeno kod rada s datumima je rad s poljima na *frontendu*, njihovo spajanje. Do tada nepoznata sintaksa [..., ...] spajanja više polja u jedno ili dodavanje nove vrijednosti u polja koristeći *concat* metodu je postalo novo znanje.

Dodavanje slika je predstavljalo prepreku, na koji način, u kojem obliku, se slika sprema u bazu podataka. Kroz proces pronalaženja rješenja, klasa *Base64* unutar Jave koja služi za pozivanje statičkih metoda za pozivanje kodera i dekodera, se pokazala kao najefikasnije rješenje.

Područje primjene ove web aplikacije je u bilo kojoj tvrtki za najam brodica. Prednost aplikacije je ta što se uz manje promjene izgleda aplikacije, odnosno korisničkog sučelja može upotrijebiti i u drugim tvrtkama u industriji iznajmljivanja (npr. iznajmljivanje automobila).

LITERATURA

[1] „ A Brief History of HTML“ Pristupljeno: [8.10.2022.] Dostupno na:

https://www.washington.edu/accesscomputing/webd2/student/unit1/module3/html_history.html

[2] „HTML(Hypertext Markup Language)“ Pristupljeno: [8.10.2022.] Dostupno na:

https://www.theserverside.com/definition/HTML-Hypertext-Markup-Language

[3] „What is HTML ? Hypertext Markup Language Basics Explained“ Pristupljeno: [8.10.2022.] Dostupno na: https://www.hostinger.com/tutorials/what-is-html

[4] „HTML Basics“ Pristupljeno: [8.10.2022.] Dostupno na:

https://developer.mozilla.org/en-US/docs/Learn/Getting_started_with_the_web/HTML_basics

[5] „CSS Syntax“ Pristupljeno: [10.10.2022.] Dostupno na:

https://www.w3schools.com/css/css_syntax.asp

[6] „What is React.js? Uses, Examples, & More“ Pristupljeno: [10.10.2022.] Dostupno na:

https://blog.hubspot.com/website/react-js

[7] Pristupljeno: [10.10.2022.] Dostupno na:

https://www.ibm.com/topics/java

[8] Pristupljeno: [10.10.2022.] Dostupno na:

https://www.codecademy.com/resources/blog/what-is-a-framework/

[9] Pristupljeno: [14.10.2022.] Dostupno na:

https://www.educative.io/answers/what-is-spring-boot

[10] Pristupljeno: [14.10.2022.] Dostupno na:

https://www.croatia-yacht-charter.com/hr/

[11] Pristupljeno: [14.10.2022.] Dostupno na:

https://www.croatia-yacht-charter.com/hr/motornjaci/

SAŽETAK

"Ferit Boat Charter" je korisnički pristupačna web aplikacija dizajnirana za najam brodova, s ciljem pojednostavljenja procesa pregledavanja, iznajmljivanja i objavljivanja brodova za najam. Dostupnost određenih značajki ovisi o ulozi korisnika unutar aplikacije. Ima četiri vrste korisnika: posjetitelj (neprijavljeni korisnik), administrator, iznajmljivač i unajmitelj (vlasnik broda koji želi postaviti svoj brod na raspolaganje za najam). Sa svojim jednostavnim dizajnom, aplikacija osigurava lagan pristup svim svojim funkcionalnostima.

Kako bi postigla svoje ciljeve, "Ferit Boat Charter" koristi modernu tehnologiju i trendove tržišta. *Frontend* aplikacije je razvijen koristeći HTML, CSS i JavaScript React biblioteku, dok backend koristi Javu sa Spring Boot okvirom. Prikupljanje i pohranjivanje podataka upravlja se putem MySQL baze podataka.

Korištenjem suvremene tehnologije i korisnički prijateljskog sučelja, "Ferit Boat Charter" pojednostavljuje proces najma brodova, pružajući zadovoljstvo kako iznajmljivačima, tako i vlasnicima brodova.

Ključne riječi: brodice, moderne tehnologije, najam, React, Spring Boot

SUMMARY

"Ferit Boat Charter" is a user-friendly web application designed for boat rentals, aiming to simplify the process of browsing, renting, and listing boats for rent. The accessibility of specific features depends on the user's role within the application, which includes four types: visitor (unregistered user), admin, renter, and boat owner listing their boat for rent. With its straightforward design, the application ensures easy access to all its functionalities.

To achieve its goals, "Ferit Boat Charter" leverages modern technology and market trends. The frontend of the application is developed using HTML, CSS, and the JavaScript React library, while the backend utilizes Java with the Spring Boot framework. Data retrieval and storage are managed through a MySQL database.

By employing current technology and a user-friendly interface, "Ferit Boat Charter" streamlines the boat rental process, catering to the needs of both renters and boat owners.

Keywords: boats, modern technology, React, renting, Spring Boot

ŽIVOTOPIS

Karlo Didak rođen je 10. veljače 2001. godine u Vitezu, Bosna i Hercegovina. Svoje školovanje započeo je u osnovnoj školi Vitez te ga nastavlja u srednjoj školi Vitez u smjeru gimnazija u Vitezu. Svoje srednjoškolsko obrazovanje završava vrlo dobrim uspjehom i polaže državnu maturu u Zagrebu, Republika Hrvatska. Nakon uspješno položene državne mature upisuje preddiplomski stručni studij, smjer računarstvo na FERIT u Osijeku, Republika Hrvatska.

_____

Potpis