# IZRADA SUSTAVA ZA UPRAVLJANJE AKTIVNOSTIMA U VODENOM TURIZMU

**Galiman, Marko**

**VISOKO UČILIŠTE ALGEBRA**

ZAVRŠNI RAD

# IZRADA SUSTAVA ZA UPRAVLJANJE AKTIVNOSTIMA U VODENOM TURIZMU

Marko Galiman

Zagreb, Veljača 2023.

*„Pod punom odgovornošću pismeno potvrđujem da je ovo moj autorski rad čiji niti jedan dio nije nastao kopiranjem ili plagiranjem tuđeg sadržaja. Prilikom izrade rada koristio sam tuđe materijale navedene u popisu literature, ali nisam kopirao niti jedan njihov dio, osim citata za koje sam naveo autora i izvor te ih jasno označio znakovima navodnika. U slučaju da se u bilo kojem trenutku dokaže suprotno, spreman sam snositi sve posljedice uključivo i poništenje javne isprave stečene dijelom i na temelju ovoga rada".*

*U Zagrebu, 28.2.2023.*

# Predgovor

Sve zahvale profesoru i mentoru Danielu Bele na prenesenom znanju kroz sve tri godine preddiplomskog studija te na pomoći prilikom izrade završnog rada.

A Gradišćanska 24, HR-10000 Zagreb
T (01) 2222 182
E student@algebra.hr
www.algebra.hr

ALGEBRA

Temeljem članka 8. Pravilnika o završnom radu i završnom ispitu na preddiplomskom studiju Visokog učilišta Algebra sačinjena je ova

# Potvrda o dodjeli završnog rada

kojom se potvrđuje da student Marko Galiman, JMBAG 0321013570, OIB 30638245278 u šk. godini 2021./2022., studij: Primjenjeno računarstvo - Preddiplomski studij, smjer: Programsko inženjerstvo, od strane povjerenstva za provedbu završnog ispita, dana 18.01.2022. godine, ima odobrenu izradu završnog rada

s temom: **Izrada sustava za upravljanje aktivnostima u vodenom turizmu**

i sažetkom rada: Student će svojim radom prikazati napredno korištenje inovativnih tehnologija u kontekstu razvoja sustava za upravljanje turističkim djelatnostima na vodi. Rezultat ovog završnog rada je sustav koji omogućuje interaktvino i napredno koristenje tehnologije u svrhu upravljanja vodenim turizmom.

Mentor je: Daniel Bele.

Odobrenjem završnog rada studentu je omogućen upis kolegija "Izrada završnog projekta/Praksa" te je sukladno članku 8. Pravilnika o završnom radu i završnom ispitu dužan najkasnije do početka nastave ljetnog semestra u sljedećoj školskoj godini, uspješno obraniti završni rad uspješnim polaganjem završnog ispita.

U protivnom student može zatražiti novog mentora/icu i temu te ponovo upisati kolegij "Izrada završnog projekta/Praksa" budući da rad koji nije predan i obranjen na završnom ispitu u roku određenom Pravilnikom završnom radu i završnom ispitu prestaje vrijediti. Izrada novog završnog rada se izvodi sukladno rokovima određenima za školsku godinu u kojoj je studentu određen novi mentor/ica i dodijeljen novi završni rad.

| Potpis studenta: | Potpis mentora: | Potpis predsjednika povjerenstva: |
|---|---|---|

Ova potvrda izdaje se u 4 (četiri) primjerka od kojih 3 (tri) idu kao prilog završnom radu.

# Sažetak

Hrvatska je zemlja koja se nalazi na Jadranskom moru, a poznata je po prekrasnim plažama i turizmu na obali. Iz tog se razloga mnogi gosti odlučuju na aktivni turizam. No organizacija aktivnosti poput iznajmljivanja brodova, glisera i ostalog, iznimno je loša te bi digitalni sustav olakšao i učinio upravljanje aktivnostima dostupnijima i jednostavnijima.

Osim toga, komunikacija među zaposlenicima nije dobra i često se pojavljuju nesporazumi u vezi s rezervacijama i dostupnošću brodova. Također, u slučaju kvarova brodova klijenti nisu na vrijeme obaviješteni i trenutačni sustav rezerviranja brodova je vrlo nepraktičan i podložan greškama.

Implementacijom digitalnog sustava, koji je jednostavan, interaktivan i koji povezuje klijente s ponudama, poboljšao bi se proces. Taj sustav bi imao višestruke funkcionalnosti, poput rezervacija za brodove i druge aktivnosti, mogućnost pregleda dostupnih brodova za radnike, karta kampova s vodenim aktivnostima, prijava kvarova, mogućnost ocjenjivanja usluga, poboljšane komunikacije između zaposlenika, mogućnost odabira skipera i skupljanje kupona za popuste. Sustav bi, također, automatski izračunavao dostupnost brodova i aktivnosti na temelju informacija o rezervacijama u bazi podataka.

**Ključne riječi:** Organizacija poslovanja, interaktivan, komunikacija, rezervacija, aktivnosti.

# Abstract

Croatia is a country located on the Adriatic Sea and is known for its beautiful beaches and coastal tourism. In the organization of activities such as renting boats, speedboats and other activites. Business organization is poor and a digital system would facilitate and make the management of available activities simpler.

In addition, communication between employees is not good and there are constant misunderstandings regarding reservations and ship availability. Also in case of ship breakdowns, customers are not notified in time and the current ship reservation system is impractical and prone to errors.

Implementing a digital system that is simple, interactive and connects customers with offers would improve the process. This system would have multiple functionalities, such as booking boats and other activities, the ability to view available boats for workers, maps of camps with water activities, fault reporting, service rating capabilities, improved communication between employees, the ability to select a skipper and collection. discount coupons. The system would also automatically calculate the availability of ships and activities based on the booking information in the database.

**Key words:** Business organization, interactive, communication, reservation, activity.

# Sadržaj

# 1. Uvod

Organizacija aktivnosti kao što su iznajmljivanje brodova, glisera i drugih aktivnosti, ključni je aspekt obalnog turizma u Hrvatskoj. Međutim, postojeći sustav upravljanja ovim djelatnostima često se pokazuje lošim i treba ga poboljšati. Jedan od velikih problema je nepostojanje digitalnog sustava koji bi upravljanje dostupnim aktivnostima učinio jednostavnijim i dostupnijim.

Jedan od glavnih problema postojećeg sustava je loša komunikacija među zaposlenicima. To dovodi do stalnih nesporazuma u vezi s rezervacijama i dostupnošću brodova, što na kraju dovodi do zbunjenosti i frustracije i za zaposlenike i za klijente.

Kako bi se riješili navedeni problemi, predlaže se implementacija digitalnog sustava koji je jednostavan, interaktivan i povezuje korisnike s dostupnim ponudama. Ovaj sustav bi imao više funkcionalnosti, uključujući mogućnost rezerviranja brodova i drugih aktivnosti, pregled dostupnih brodova za radnike, pristup kartama kampova s vodenim aktivnostima, prijavu kvarova, ocjenjivanje usluge, poboljšanje komunikacije među zaposlenicima, pretplatu na novosti, odabir skipera i skupljanje kupona za popust.

Radi svega navedenog, ovaj završni rad bavi se problemom organizacije poslovanja za aktivnosti na moru i rješenjem tih problema koje će riješiti aplikacija zvana “Watersports Manager“. Ovaj rad se sastoji od pet poglavlja.

Prvo poglavlje je uvod u kojem se ukratko opisuje rad i o čemu će sve biti riječ u radu.

Drugo poglavlje opisuje uočene probleme u području aktivnosti vodenog turizma i opisuje ciljeve aplikacijskog rješenja.

Treće poglavlje opisuje implementaciju cjelovitog softverskog rješenja, opisuju se korištene tehnologije, povijesti tehnologije i glavne značajke tehnologije. Isto tako, usporedba s drugim jezicima u web okruženju. Opisuje se model baze podataka i web API.

Četvrto poglavlje je implementacija sustava u kojem se opisuje arhitektura projekta gdje će se navesti pristupne točke, servisi, primjena oblikovnih obrazaca kod razvoja web servisa, arhitektura i raslojavanje, pohrana aplikacijskih podataka, komunikacija s pristupnim točkama web API-a te navigacija među komponentama aplikacije. Isto tako, opisivat će se korisničke funkcionalnosti poput glavnog sučelja, prijava korisnika, mape, rezervacija plovila i sustava za ocjenjivanje.

Peto i zadnje poglavlje opisivat će analizu uspješnosti aplikacije gdje ćemo vidjeti koliko je funkcionalna i stabilna aplikacija.

# 2. Problematika vodenog turizma i ciljevi

Trenutni sustav upravljanja navedenim aktivnostima često se pokazuje zastarjelim i treba ga poboljšati. Jedan od velikih problema je nepostojanje modernog, digitalnog sustava koji bi upravljanje dostupnim aktivnostima učinio jednostavnijim i dostupnijim kako korisnicima tako i zaposlenicima. To je problem s kojim se treba pozabaviti kako bi se poboljšalo cjelokupno iskustvo turističkih aktivnosti na obalama Hrvatske.

Jedan od glavnih problema s trenutnim sustavom je nedostatak centralne, digitalne platforme za upravljanje rezervacijama i komunikacijama s korisnicima. Radnici trenutno međusobno komuniciraju pozivima i porukama putem mobitela, što nije samo neučinkovito, već je i podložno pogreškama. Informacije često ne dopiru do željenih primatelja što dovodi do zbunjenosti i frustracije i zaposlenika i kupaca. Osim toga, uobičajeno je da se važne informacije vezane uz kvarove plovila, probleme s rezervacijama i probleme s navigacijom do najbližeg dostupnog kampa za vodene sportove zaborave ili ne priopće ispravno.

Još jedan problem s trenutnim sustavom je nedostatak informacija u stvarnom vremenu o raspoloživosti brodova i aktivnosti. Kupci često moraju nazvati ili osobno posjetiti lokaciju kako bi se raspitali o dostupnosti, što može biti dugotrajno i frustrirajuće. Osim toga, trenutni sustav ručnog knjigovodstva sklon je pogreškama i može dovesti do dvostrukih ili prebukiranih rezervacija.

Kako bi se riješili ovi problemi, predlaže se implementacija digitalnog sustava koji je jednostavan, interaktivan i povezuje korisnike s dostupnim ponudama. Ovaj sustav bi imao više funkcionalnosti, uključujući mogućnost rezerviranja brodova i drugih aktivnosti, pregled dostupnih brodova za radnike, pristup kartama kampova s vodenim aktivnostima, prijavu kvarova, ocjenjivanje usluge, poboljšanje komunikacije među zaposlenicima, odabir skipera te skupljanje kupona za popust.

Sustav bi također automatski izračunavao dostupnost brodova i aktivnosti na temelju podataka o rezervaciji u bazi podataka. To bi kupcima osiguralo ažurne informacije o aktivnostima i raspoloživosti brodova, smanjujući zbunjenost i povećavajući zadovoljstvo. Dodatno, sustav bi također zaposlenicima pružio platformu jednostavnu za korištenje i upravljanje rezervacijama, poboljšavajući njihovu učinkovitost i djelotvornost.

Novi sustav bi također bio velika korist za djelatnike koji rade u djelatnosti vodenog turizma. Imali bi središnju platformu na kojoj bi lako mogli pristupiti svim informacijama koje su im

potrebne, poput raspoloživosti brodova, rezervacija kupaca i kvarova brodova. Također bi mogli međusobno komunicirati u stvarnom vremenu, što bi im pomoglo u brzom rješavanju problema. Imali bi i jasan pregled cijelog procesa, što bi im znatno pomoglo.

Zaključno, organizacija aktivnosti poput iznajmljivanja brodova, glisera i drugih aktivnosti ključna je za uspjeh obalnog turizma u Hrvatskoj. Međutim, utvrđeno je da trenutni sustav nedostaje u mnogim područjima. Implementacijom digitalnog sustava koji je jednostavan, interaktivan i povezuje korisnike s ponudama, uvelike bi se unaprijedio proces upravljanja ovim aktivnostima i povećalo zadovoljstvo korisnika. Ovaj sustav bi imao više funkcionalnosti i automatski bi izračunavao raspoloživost brodova i aktivnosti na temelju podataka o rezervaciji u bazi podataka. Također bi poboljšala komunikaciju i rad zaposlenika. Sve u svemu, to bi omogućilo učinkovitiji i djelotvorniji sustav za upravljanje obalnim turističkim aktivnostima.

# 3. Implementacija cjelovitog softverskog rješenja

Implementacija novog softverskog sustava može biti zastrašujući zadatak, ali uz pravilno planiranje i izvođenje, može donijeti značajne prednosti poslovanju. U ovom završnom radu bit će opisani koraci koje treba poduzeti prilikom implementacije novog softverskog sustava u području aktivnosti vodenog turizma, kao što su iznajmljivanje čamaca, glisera i drugih aktivnosti.

Prvi korak u implementaciji novog softverskog sustava je provođenje temeljite analize postojećeg sustava. To uključuje prepoznavanje trenutnih procesa, procedura i bolnih točaka u trenutnom sustavu. Ova analiza omogućit će jasno razumijevanje ograničenja trenutnog sustava i područja koja trebaju poboljšanja.

Drugi korak je utvrđivanje specifičnih zahtjeva za novi softverski sustav. To uključuje određivanje funkcionalnosti koje su potrebne, kao što su rezervacija brodova, pregled dostupnih brodova za radnike, pristup kartama kampova s vodenim aktivnostima, prijava kvarova, ocjenjivanje usluge, poboljšanje komunikacije među zaposlenicima, odabir skipera i prikupljanje kupona za popust. Također je važno uzeti u obzir specifične potrebe poslovanja, kao što su broj zaposlenika, broj kupaca i broj plovila i dostupnih aktivnosti.

Treći korak je dizajn i testiranje novog softverskog sustava. To uključuje rad s dobavljačem na dizajniranju i prilagodbi softvera kako bi zadovoljio specifične zahtjeve poslovanja. Također je važno provesti testiranje kako bi se osiguralo da sustav ispravno radi i da zadovoljava potrebe poslovanja.

Posljednji korak je implementacija novog softverskog sustava i edukacija zaposlenika. To uključuje instaliranje softvera, njegovo konfiguriranje i pružanje obuke zaposlenicima o tome kako koristiti novi sustav. Također je važno osigurati stalnu podršku i održavanje kako bi se osiguralo da sustav i dalje zadovoljava potrebe poslovanja.

Zaključno, implementacija novog softverskog sustava u području aktivnosti vodenog turizma može donijeti značajne koristi za poslovanje. Provođenjem detaljne analize postojećeg sustava, utvrđivanjem specifičnih zahtjeva, dizajniranjem i testiranjem novog sustava te pružanjem obuke i stalne podrške, tvrtka može osigurati da se novi sustav uspješno implementira i da zadovoljava potrebe poslovanja.

## 3.1. Korištene tehnologije .NET, SQL i Angular

.NET, SQL i Angular su moćna kombinacija za izradu web aplikacija. .NET pruža stabilno i pouzdano okruženje za izradu web aplikacija, SQL pruža učinkovit i pouzdan način za pohranu i dohvaćanje podataka, a Angular pruža moćan i učinkovit način za izgradnju korisničkih sučelja. Zajedno, oni pružaju sveobuhvatno rješenje za izgradnju robusnih i skalabilnih web aplikacija. U sljedećem poglavlju opisat će se glavne značajke tehnologija i usporedba s njihovim konkurencijama.

### 3.1.1. Glavne značajke i usporedba s drugim jezicima

**.NET** je besplatni softverski okvir otvorenog koda za više platformi koji je prvi put predstavio Microsoft 2002. Prvenstveno se koristi za izradu aplikacija temeljenih na sustavu Windows, ali se može koristiti i za razvoj aplikacija za druge platforme uključujući iOS, Android, i Linux. .NET je sveobuhvatan okvir koji uključuje širok raspon značajki i alata za podršku razvoju visokokvalitetnih, skalabilnih i robusnih aplikacija.

Jedna od glavnih značajki .NET-a je njegova podrška za više programskih jezika. Programeri mogu koristiti C#, Visual Basic, F# i druge programske jezike za izradu aplikacija s .NET-om. To znači da programeri mogu odabrati jezik koji im najviše odgovara, a da i dalje imaju pristup istom skupu značajki i alata koje nudi .NET. [1]

Druga važna značajka .NET-a je podrška za web usluge. .NET olakšava izradu i korištenje web usluga, aplikacija kojima se može pristupiti putem interneta. To organizacijama olakšava integraciju njihovih aplikacija s aplikacijama drugih organizacija, čime se poboljšava suradnja i učinkovitost.

.NET također nudi bogat skup biblioteka i API-ja, što programerima olakšava izradu aplikacija sa širokim rasponom funkcionalnosti. Na primjer, .NET framework uključuje biblioteke za rad s bazama podataka, rukovanje XML-om i upravljanje sigurnošću. To znači da programeri ne moraju pisati sav kod od nule, već umjesto toga mogu koristiti već postojeći kod za izradu svojih aplikacija.[2]

Sada usporedimo .NET i Javu. Java je još jedna popularna platforma koja se koristi za izradu širokog spektra aplikacija, uključujući desktop, web i mobilne aplikacije. I .NET i Java imaju svoje snage i slabosti, a izbor između njih često ovisi o specifičnim zahtjevima projekta.

Jedna ključna razlika između .NET-a i Jave je ta što se .NET primarno koristi za izradu aplikacija temeljenih na sustavu Windows, dok se Java može koristiti za izradu aplikacija za

više platformi, uključujući Windows, Mac, Linux i druge. To Javu čini boljim izborom za organizacije koje trebaju izraditi aplikacije koje će raditi na više platformi.

Druga je razlika u tome što je Java objektno orijentirani programski jezik, dok .NET podržava više programskih jezika, uključujući objektno orijentirane jezike poput C#. To znači da Java programeri imaju dosljednu programsku paradigmu, dok .NET programeri imaju slobodu odabira jezika koji im najviše odgovara.

Što se tiče performansi, općenito se smatra da je .NET brži od Jave, iako razlika u većini slučajeva nije značajna. Java je, s druge strane, poznata po svojim sigurnosnim značajkama i često se koristi za izradu aplikacija koje obrađuju osjetljive podatke.[3]

Zaključno, i .NET i Java su moćne platforme za izgradnju širokog spektra aplikacija. Odabir između njih često ovisi o specifičnim zahtjevima projekta, uključujući ciljanu platformu, zahtjeve izvedbe i sigurnosna razmatranja. Bez obzira na izbor, i .NET i Java pružaju programerima opsežan skup značajki i alata za podršku razvoju visokokvalitetnih, skalabilnih i robusnih aplikacija.

**SQL** (Structured Query Language) standardni je programski jezik koji se koristi za upravljanje relacijskim bazama podataka. Prvi put je predstavljen 1970-ih i od tada je postao jedan od najčešće korištenih sustava za upravljanje bazama podataka u svijetu.

Jedna od glavnih značajki SQL-a je njegova podrška za strukturirane podatke. SQL baze podataka pohranjuju podatke u tablice, pri čemu svaka tablica sadrži niz redaka i stupaca. To olakšava organiziranje i upravljanje velikim količinama podataka, kao i izvođenje složenih upita i analizu podataka.

Druga ključna značajka SQL-a je njegova podrška za transakcije. Transakcije su skup operacija baze podataka koje se izvode kao jedna radna jedinica. To znači da ako bilo koja od operacija u transakciji ne uspije, cijela se transakcija vraća unatrag, čime se osigurava dosljednost i cjelovitost podataka.[4]

SQL također nudi bogat skup funkcija za upravljanje i manipulaciju podacima, što programerima olakšava izvođenje uobičajenih zadataka baze podataka kao što su sortiranje, filtriranje i prikupljanje podataka. To znači da programeri ne moraju pisati složeni kod za izvođenje ovih zadataka, nego umjesto toga mogu koristiti već postojeće funkcije za izvođenje istih operacija.[5]

Sada usporedimo SQL i NoSQL. NoSQL, što znači Not only SQL, nova je vrsta sustava za upravljanje bazom podataka koji se pojavio posljednjih godina. Za razliku od SQL baza podataka, NoSQL baze podataka ne zahtijevaju da podaci budu pohranjeni u fiksnoj strukturi. Umjesto toga, podržavaju različite strukture podataka, uključujući parove ključ-vrijednost, strukture temeljene na dokumentima i grafikonima.

Jedna ključna razlika između SQL-a i NoSQL-a je skalabilnost. NoSQL baze podataka su dizajnirane da budu visoko skalabilne, što ih čini prikladnim za rukovanje velikim količinama podataka, posebno u distribuiranom okruženju. SQL baze podataka, s druge strane, općenito su manje skalabilne, iako još uvijek mogu obraditi veliku količinu podataka uz pravu konfiguraciju i optimizaciju.

Druga razlika je razina složenosti. SQL baze podataka zahtijevaju strukturiraniji pristup upravljanju podacima, što ih može učiniti složenijima za korištenje i održavanje. NoSQL baze podataka, s druge strane, često su jednostavnije za korištenje i održavanje jer ne zahtijevaju strogu strukturu podataka.

Što se tiče performansi, NoSQL baze podataka općenito su brže od SQL baza podataka, osobito kada se radi s velikim količinama nestrukturiranih podataka. Međutim, SQL baze podataka još uvijek se preferiraju za aplikacije koje zahtijevaju složene transakcije i analizu podataka, budući da pružaju robusniji skup funkcija za upravljanje podacima i analizu.[6]

Zaključno, i SQL i NoSQL imaju svoje snage i slabosti, a izbor između njih često ovisi o specifičnim zahtjevima projekta. Bez obzira na izbor, i SQL i NoSQL pružaju programerima skup snažnih alata za upravljanje i analizu podataka, a onda i za izgradnju visokokvalitetnih, skalabilnih i robusnih aplikacija.

**Angular** je popularan open-source JavaScript okvir za izgradnju dinamičnih i interaktivnih web aplikacija. Google ga je prvi put objavio 2010. i od tada je postao jedan od najčešće korištenih okvira za izradu jednostraničkih aplikacija. Angular pruža opsežan skup značajki i alata za podršku razvoju visokokvalitetnih, skalabilnih i robusnih web aplikacija.

Jedna od glavnih značajki Angulara je njegovo dvosmjerno povezivanje podataka. Ova značajka omogućuje da se promjene napravljene u modelu automatski odražavaju u prikazu i obrnuto. Ovo eliminira potrebu da programeri ručno ažuriraju pogled kada se naprave promjene u modelu, čime se poboljšava učinkovitost i smanjuje vjerojatnost grešaka.[7]

Još jedna važna značajka Angulara je njegova modularna arhitektura. Angular aplikacije izgrađene su korištenjem skupa komponenti za višekratnu upotrebu, što olakšava upravljanje i

održavanje koda. Ovo također olakšava ponovnu upotrebu komponenti u različitim projektima, čime se smanjuje vrijeme i trud potreban za izgradnju novih aplikacija.

Angular također nudi bogat skup biblioteka i API-ja, što programerima olakšava dodavanje širokog raspona funkcionalnosti njihovim aplikacijama. Na primjer, Angular uključuje biblioteke za rad s HTTP-om, rukovanje obrascima i upravljanje animacijama. To znači da programeri ne moraju pisati sav kod od nule, nego umjesto toga mogu koristiti već postojeći kod za izradu svojih aplikacija.[8]

Sad će se usporediti Angular i React. React je još jedna popularna JavaScript biblioteka za izradu korisničkih sučelja i naširoko se koristi za izradu jednostraničkih aplikacija. I Angular i React imaju svoje snage i slabosti, a izbor između njih često ovisi o specifičnim zahtjevima projekta.

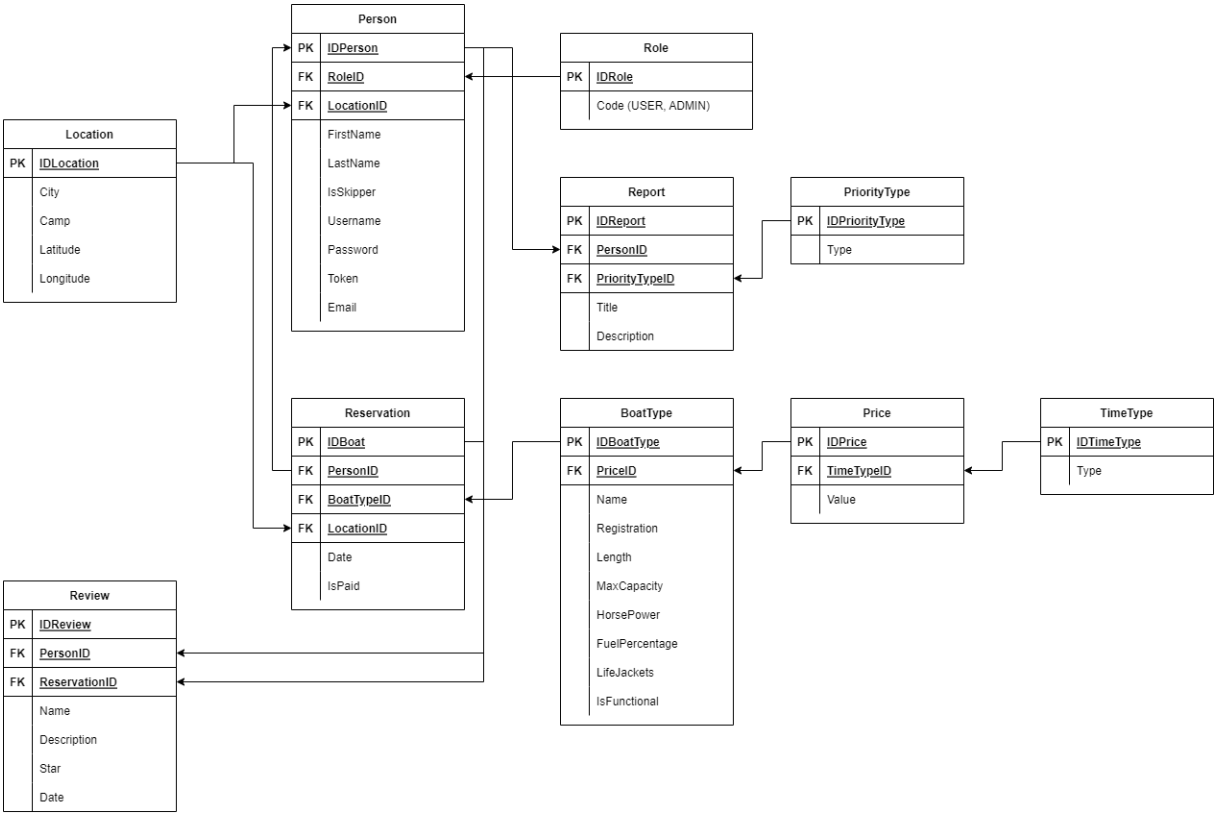
Jedna ključna razlika između Angulara i Reacta je ta što je Angular kompletan okvir, dok je React biblioteka. To znači da Angular pruža opsežan skup značajki i alata za podršku razvoju web aplikacija, dok se React posebno fokusira na izgradnju korisničkih sučelja.

Druga je razlika u tome što React koristi virtualni DOM, što mu omogućuje učinkovito ažuriranje korisničkog sučelja bez ponovnog renderiranja cijele stranice. To čini React aplikacije brzima i responzivnima, čak i kada se radi s velikim količinama podataka. Angular, s druge strane, pruža sveobuhvatniji skup značajki i alata, što olakšava izradu složenih aplikacija.

Što se tiče performansi, React se općenito smatra bržim od Angulara, iako razlika u većini slučajeva nije značajna. Angular je, s druge strane, poznat po dvosmjernom povezivanju podataka, što olakšava upravljanje protokom podataka u aplikaciji.[9]

Zaključno, i Angular i React moćni su alati za izgradnju dinamičnih i interaktivnih web aplikacija. Izbor između njih često ovisi o specifičnim zahtjevima projekta, uključujući razinu složenosti, zahtjeve izvedbe i specifične potrebe aplikacije. Bez obzira na izbor, i Angular i React pružaju programerima sveobuhvatan skup značajki i alata za podršku razvoju visokokvalitetnih, skalabilnih i robusnih web aplikacija.

## 3.2. Model baze podataka




Slika 3.1 WatersportsManagerDB Diagram

Iznad je prikazan dijagram baze podataka koji je korišten u projektu, a sastoji se od jedanaest tablica koje su međusobno povezane stranim ključevima i konfigurirane sa strane .NET-a.

Person tablica ima glavni ključ *IDPerson*, strane ključeve *RoleID* i *LocationID*, u sebi ima stringove *FirstName* i *LastName* gdje se sprema ime i prezime registriranog korisnika, *IsSkipper* boolean gdje se sprema informacija o tome je li korisnik skiper ili nije, pod *Username* se zapisuje uneseno korisničko ime, pod *Password* se sprema lozinka korisnika u kriptiranome obliku, pod *Token* se sprema korisnikov token koji ima tijekom korištenja aplikacije te njime odlučujemo koliko korisnik može dugo koristiti aplikaciju prilikom ponovne prijave u sustav, a pod *Email* se sprema mail korisnika koji unese tijekom registracije.

Role tablica ima glavni ključ *IDRole* i *Code* gdje je zapisano je li korisnik *USER* ili *ADMIN*.

Location tablica u sebi ima glavni ključ *IDLocation* i atribute *City*, *Camp*, *Latitude* i *Longitude* gdje su zapisani podaci o lokacijama, u kojem gradu se nalaze, naziv kampa i njegove koordinate.

PriorityType tablica u sebi ima glavni ključ *IDPriorityType* i *Type* gdje se odabire razina hitnosti reporta (niska, normalna ili visoka hitnost).

Report tablica u sebi ima glavni ključ *IDReport* i dva strana ključa *PersonID* i *PriorityTypeID*, ima *Title* gdje se zapisuje naziv reporta i *Description* gdje se zapisuje opis reporta.

Review tablica u sebi ima glavni ključ *IDReview* i dva strana ključa *PersonID* i *BoatID*, ima *Name* gdje se zapisuje naziv ocjene, pod *Description* se zapisuje opis ocjene, pod *Star* se zapisuje 1-5 zadovoljstvo korisnika i pod *Date* datum kreirane ocjene.

TimeType tablica u sebi ima glavni ključ *IDTimeType* i *Type* gdje se zapisuje željeno vrijeme aktivnosti.

Price tablica u sebi ima glavni ključ *IDPrice*, strani ključ *TimeTypeID* i *Value* gdje se zapisuje cijena aktivnosti.

BoatType tablica u sebi ima glavni ključ *IDBoaType* i strani ključ *PriceID*, ima i *Name* gdje se zapisuje naziv plovila, *Registration* gdje se zapisuje registracija plovila, *Length* gdje se zapisuje dužina plovila, *MaxCapacity* u kojem se zapisuje koliko ljudi može biti na plovilu, *HorsePower* zapisuje snagu motora od plovila, *FuelPercantage* zapisuje koliki postotak goriva ima plovilo, *LifeJackets* zapisuje koliko sigurnosnih prsluka plovilo ima te na kraju *IsFunctional* zapisuje je li plovilo u ispravnom stanju za plovidbu.

Finalna Reservation tablica u sebi ima primarni ključ *IDBoat* i tri strana ključa *PersonID*, *BoatTypeID*, *LocationID* i atribute *Date* za datum rezervacije i *IsPaid* kako bi se provjerilo je li korisnik platio.

Primarni i stani ključevi te atributi i njihove funkcionalnosti konfigurirani su kroz .NET.

# 4. Implementacija sustava

U ovom dijelu opisat će se na koji način se koristio .NET za poslužiteljski dio, Angular za klijentski dio i SQL za bazu podataka. Bit će opisana Arhitektura projekta, na koji način su implementirane pristupne točke, navigacija među komponentama, kreacija baze podataka i pohrana podataka. Bit će opisane korisničke funkcionalnosti kao što su glavno sučelje, prijava korisnika i mapa kampova.

## 4.1. Arhitektura projekta

Arhitektura projekta se sastoji od tri glavna dijela. Prvi dio je baza podataka u kojoj su kreirane tablice i u kojoj se spremaju podatci korisnika, plovila i rezervacija. Drugi dio je .NET poslužiteljski, u kojem su implementirani modeli, pristupne točke za dohvaćanje, mijenjanje i brisanje podataka iz baze. Uz to, tamo su podešene sigurnosne postavke kod kreiranja baze i kreiranja tokena za korisnike. Treći i finalni dio je Angular klijentski, tamo su implementirane komponente preko kojih se izmjenjuju podatci iz baze kroz sučelje i gdje je kreiran dizajn web aplikacije.

### 4.1.1. Pristupne točke

Pristupne točke su skup protokola, rutina i alata koji omogućuju različitim softverskim aplikacijama da međusobno komuniciraju. Pristupne točke određuju kako softverske komponente trebaju međusobno komunicirati i mogu se smatrati mostom između različitih aplikacija, omogućujući im razmjenu podataka i funkcionalnosti.

Jednostavnije rečeno, Pristupne točke su skup pravila koji dopušta da dvije različite softverske aplikacije međusobno komuniciraju. Omogućuje jednoj aplikaciji pristup funkcionalnosti i podacima druge aplikacije ili sustava, bez potrebe za razumijevanjem osnovnog koda ili infrastrukture. Pristupne točke se široko koriste u razvoju softvera, omogućujući programerima da integriraju različite aplikacije i usluge za stvaranje novih proizvoda ili značajki.

Na primjer, sljedeći isječak koda predstavlja pristupnu točku koja omogućuje korisniku da dohvati sve reporte iz baze podataka.

```
namespace WatersportsManager.API.Controllers
{
    [ApiController]
    [Route("[controller]")]
    public class ReportController : Controller
    {
        private readonly IReportService _reportService;
        public ReportController(IReportService reportService)
        {
            _reportService = reportService;
        }
        [HttpGet(Name = nameof(GetReports))]
        public async Task<IReadOnlyList<ReportDto>>
GetReports(CancellationToken cancellationToken)
        {
            return await _reportService.GetReports(cancellationToken);
        }
    }
}
```

Kôd 4.1 Pristupna točka za reporte

### 4.1.2. Servisi

Općenito, servis je vrsta softverske komponente koja radi u pozadini i obavlja određene zadatke ili funkcije. Usluge su dizajnirane da budu dugotrajne i rade neovisno o interakciji korisnika, često pružajući neku vrstu funkcionalnosti ili resursa koji mogu koristiti druge aplikacije ili servisi.

Servisi se obično koriste u razvoju softvera za pružanje funkcionalnosti koju može koristiti više aplikacija, bez potrebe za pisanjem istog koda više puta. To omogućuje veću ponovnu upotrebu koda i pojednostavljuje održavanje i ažuriranja. Servisi se također mogu koristiti za poboljšanje performansi i skalabilnosti na zasebnu uslugu, oslobađajući glavnu aplikaciju za obradu drugih zahtjeva.

U softverskom sustavu servisi se mogu organizirati na različite načine, ovisno o specifičnoj arhitekturi i dizajnu sustava. Na primjer, servisno orijentirana arhitektura (SOA) organizira

funkcionalnost u skup modularnih, neovisnih servisa koji se mogu kombinirati i ponovno koristiti za stvaranje složenijih sustava.

Ukratko, servis je softverska komponenta koja pruža određene funkcije ili resurse, radi neovisno o interakciji korisnika i može je koristiti više aplikacija ili servisa. Servisi su čest i važan sastavni blok u modernom razvoju softvera, omogućuju ponovnu upotrebu koda, skalabilnost i poboljšane performanse.

Na primjer, sljedeći isječak koda predstavlja kreiranje modela za prikazivanje podataka o reportima.

```
namespace WatersportsManager.Application.Reports.Models
{
    public class ReportDto
    {
        public int Id { get; set; }
        public string Title { get; set; }
        public string Description { get; set; }
        public string Person { get; set; }
        public string PriorityType { get; set; }
    }
}
```

Kôd 4.2 Report model

Ova klasa ima nekoliko svojstava:

- *Id* svojstvo koje predstavlja identifikacijski broj reporta.
- *Title* svojstvo koje predstavlja naslov reporta.
- *Description* svojstvo koje predstavlja opis reporta.
- *Person* svojstvo koje predstavlja ime osobe kojoj je dodijeljen report.
- *PriorityType* svojstvo koje predstavlja vrstu prioriteta reporta.

Ova se klasa koristi za predstavljanje DTO (Data Transfer Object) za entitet report. DTO je jednostavan objekt koji sadrži podatke i može se koristiti za prijenos podataka između slojeva ili između sustava. U ovom slučaju, DTO se koristi za prijenos podataka sa sloja postojanosti na servisni sloj ili prezentacijski sloj. Koristi se kao jednostavan spremnik podataka i najbolja je praksa korištenja DTO-a za odvajanje problema različitih slojeva.

Kada je kreiran DTO model, može se krenuti s kreiranjem potrebnog servisa koji će koristiti taj model. Na primjer, sljedeći isječak koda predstavlja kreiranje report servisa kojeg će pristupna točka za dohvaćanje reporta koristiti kako bi dohvatila sve reporte iz baze.

```
namespace WatersportsManager.Application.Reports
{
    public class ReportService : IReportService
    {
        private readonly WatersportsManagerDbContext _dbContext;
        public ReportService(WatersportsManagerDbContext dbContext)
        {
            _dbContext = dbContext;
        }

        public async Task<IReadOnlyList<ReportDto>>
        GetReports(CancellationToken token)
        {
            return await _dbContext.Reports.AsNoTracking()
               .Select(report => new ReportDto
               {
                   Id = report.Id,
                   Title = report.Title,
                   Description = report.Description,
                   Person = report.Person.ToString(),
                   PriorityType = report.PriorityType.Type
               }).ToListAsync(token);
        }
    }
}
```

Kôd 4.3 Report servis

Navedeni kod se izvršava asinkrono, što znači da se više servisa odjedanput mogu pokrenuti bez ikakvog problema. Ovdje se servis povezuje s bazom kako bi mogao komunicirati s istom i dohvatiti sve tražene reporte.

Nadalje, sljedeći isječak koda predstavlja validacijski filter koji prolazi kroz sve argumente radnje u kontekstu, za svaki argument provjerava vrstu argumenta i dobiva odgovarajući

validator za tu vrstu od pružatelja usluge. Ako postoji validator za trenutni argument, stvorit će kontekst provjere valjanosti i potvrditi će argument, a ako provjera valjanosti ne uspije, vratit će odgovor o pogrešci s detaljima problema provjere valjanosti.

```
namespace WatersportsManager.API.Filters
{
    public class ValidationFilter : IAsyncActionFilter
    {
        private readonly IServiceProvider _serviceProvider;
        public ValidationFilter(IServiceProvider serviceProvider)
        {
            _serviceProvider = serviceProvider;
        }
        public async Task OnActionExecutionAsync(
        ActionExecutingContext context, ActionExecutionDelegate next)
        {
            foreach (var actionArgument in context.ActionArguments)
            {
                Type actionArgumentType = actionArgument.Value?.GetType();
                if (actionArgumentType is null)
                    continue;

                Type paramValidatorType = typeof(IValidator<>)
                .MakeGenericType(actionArgumentType);
                IValidator validator = (IValidator)_serviceProvider
                .GetService(paramValidatorType);
                if (validator is null)
                    continue;

                IValidationContext validationContext =
                (IValidationContext)Activator.CreateInstance(
                typeof(ValidationContext<>).MakeGenericType(actionArgumentT
                ype), actionArgument.Value);
```

```
                ValidationResult validationResult = await validator
                .ValidateAsync(validationContext);
                if (!validationResult.IsValid)
                {
                    var validationErrorsDictionary = validationResult
                    .ToDictionary();
                    ValidationProblemDetails problemDetails = new(
                    validationErrorsDictionary)
                    {
                        Type =
                        "https://tools.ietf.org/html/rfc7231#section-6.5.1"
                    };
                    problemDetails.Extensions.Add(
                    "traceId", Activity.Current?.Id ?? context
                    .HttpContext.TraceIdentifier);
                    context.Result = new BadRequestObjectResult(
                    problemDetails);
                    return;
                }
            }
            await next();
        }
    }
}
```

Kôd 4.4 Validacijski filter

Ovaj kod se koristi za dodavanje sloja provjere valjanosti pristupnoj točki. On provjerava ulaze koji su proslijeđeni kroz kontrolore kako bi osigurao da su valjani, a ako nisu, vratit će odgovor o pogrešci.

Sljedeći isječak koda predstavlja validator za kreiranje vrste plovila i koristi se kako bi osigurao da su podaci proslijeđeni modelu *CreateBoatTypeDto* valjani.

```
namespace WatersportsManager.Application.BoatTypes.Validators
{
    public class CreateBoatTypeValidator :
    AbstractValidator<CreateBoatTypeDto>
    {
        public CreateBoatTypeValidator(IPriceService priceService)
        {
            RuleFor(boatType =>
                  boatType.Name).NotEmpty().MaximumLength(100);
            RuleFor(boatType =>
                  boatType.Registration).NotEmpty().MaximumLength(100);
            RuleFor(boatType =>
                  boatType.FuelPercentage).NotEmpty().
                  GreaterThanOrEqualTo(0).LessThanOrEqualTo(100);
            RuleFor(boatType =>
                  boatType.PriceId).MustAsync(priceService.PriceExists)
                .WithMessage(boatType
                  => $"Price with id '{boatType.PriceId}' does not exist");
        }
    }
}
```

Kôd 4.5 Validacijski sloj za kreiranje vrstu plovila

Ovaj kod ima nekoliko pravila definiranih pomoću biblioteke *FluentValidation* koja je pomoćni dodatak ovom kodu kako bi se lakše validirala polja. Ta se pravila koriste kako bi se osigurala valjanost ulaznih podataka.

Prvo pravilo osigurava da svojstvo *Name* ulaznih podataka nije prazno i da je njegova maksimalna duljina 50 znakova.

Drugo pravilo osigurava da svojstvo *Registration* ulaznih podataka nije prazno i da je njegova maksimalna duljina 100 znakova.

Treće pravilo osigurava da svojstvo *FuelPercentage* ulaznih podataka nije prazno i da je veće ili jednako od 0 i manje ili jednako od 100.

Četvrto pravilo osigurava da je svojstvo *PriceId* ulaznih podataka valjano. Ova se metoda koristi za provjeru postoji li cijena s navedenim ID-om ili ne.

Ova se klasa koristi za provjeru valjanosti podataka proslijeđenih klasi *CreateBoatTypeDto*. Ona i osigurava da su podaci valjani, a ako nisu, vratit će poruku o pogrešci.

Napokon, sljedeći isječak koda predstavlja primjer koda koji se koristi za registraciju servisa i validatora koji se koriste u cijeloj aplikaciji. Također im omogućuje jednostavan pristup i ponovnu upotrebu u drugim dijelovima aplikacije.

```
namespace WatersportsManager.Application
{
    public static class DependencyInjection
    {
        public static IServiceCollection AddApplication(
            this IServiceCollection services, IConfiguration configuration)
        {
            services.AddScoped<IBoatTypeService, BoatTypeService>();
            services.AddScoped
            <IValidator<UpdateBoatTypeDto>, UpdateBoatTypeValidator>();

            services.AddDbContext<WatersportsManagerDbContext>(options =>
            {

                  options.UseSqlServer(configuration.GetConnectionString("W
                  atersportsManagerDb"));
            });
            return services;
        }
    }
}
```

Kôd 4.6 Primjer dependency injection koda

### 4.1.3. Kreacija SQL tablice kroz .NET

Kreacija SQL tablice kroz .NET se izvršava ORM ili Object-Relational Mapping tehnikom. To je tehnika koja programerima softvera omogućuje interakciju s bazom podataka korištenjem objektno orijentiranog pristupa.

Jednostavno rečeno, to znači da umjesto pisanja složenih SQL upita za interakciju s bazom podataka, programeri mogu koristiti kod koji se ponaša i izgleda kao normalne konstrukcije programskog jezika za rad s podacima.

ORM okviri pružaju sloj apstrakcije između baze podataka i koda aplikacije. Ovaj sloj preslikava tablice u bazi podataka iz klase koja je napravljena u aplikaciji, preslikava retke u objekte, a stupce u svojstva tih objekata.

Korištenjem ORM okvira programeri mogu pisati manje koda, raditi sa svojim podacima na intuitivniji i poznatiji način. To može dovesti do bržeg razvoja i koda koji se lakše održava.

Kada se kreiraju tablice, potrebno je krenuti od kodnog dijela, što će pokazati sljedeći primjer.

```
namespace WatersportsManager.Domain
{
    public class BoatType : BaseNameEntity
    {
        public string? Registration { get; set; }
        public double? Length { get; set; }
        public int? MaxCapacity { get; set; }
        public int? HorsePower { get; set; }
        public int? FuelPercentage { get; set; }
        public int? LifeJackets { get; set; }
        public bool IsFunctional { get; set; }

        public virtual Price? Price { get; set; }
        public int? PriceId { get; set; }

        public ICollection<Reservation> Reservations { get; set; }
            = new List<Reservation>();
    }
}
```

Kôd 4.7 Primjer modela klase za tip plovila

U ovom primjeru će se prikazati klasa koja ima nekoliko svojstava:

- *Registration* svojstvo koje predstavlja registraciju plovila.
- *Length* svojstvo koje predstavlja duljinu broda.
- *MaxCapacity* svojstvo koje predstavlja najveći broj ljudi koje brod može primiti.
- *HorsePower* svojstvo koje predstavlja konjske snage broda.
- *FuelPercentage* svojstvo koje predstavlja postotak goriva preostalog u čamcu.
- *LifeJackets* svojstvo koje predstavlja broj prsluka za spašavanje na brodu.

- *IsFunctional* svojstvo koje pokazuje je li brod funkcionalan ili ne.

Klasa također ima dva svojstva vezana uz cijenu broda:

- *Price* svojstvo klase *Price* koje predstavlja cijenu broda.
- *PriceId* svojstvo koje predstavlja identifikacijski broj cijene.

Općenito, klasa *BoatType* koristi se za predstavljanje vrste čamca i njegovih svojstava kao što su registracija, duljina, maksimalni kapacitet, konjske snage, postotak goriva, prsluci za spašavanje i funkcionalnost. Također ima svojstva vezana uz cijenu i rezervacije broda.

Kada je model klase pripremljen, potrebno je definirati konfiguraciju, što će prikazati sljedeći isječak koda.

```
namespace WatersportsManager.Application.Persistence.Configurations
{
    public class BoatTypeConfiguration : IEntityTypeConfiguration<BoatType>
    {
        public void Configure(EntityTypeBuilder<BoatType> builder)
        {
            builder.ToTable(nameof(BoatType));

            builder.HasOne(boatType => boatType.Price)
                .WithMany(price => price.BoatTypes)
                .HasForeignKey(boatType => boatType.PriceId)
                .OnDelete(DeleteBehavior.Cascade);

            builder.HasKey(x => x.Id);
        }
    }
}
```

Kôd 4.8 Konfiguracija za tip plovila

U ovom kodu putem ORM tehnike implementira se konfiguracija za odnos između *BoatType* i *Price* modela, gdje će se sve to automatski primijeniti u bazi podataka.

Kada je konfiguracija gotova, treba se kreirati kontekst, što je prikazano sljedećim kodnim isječkom.

```
namespace WatersportsManager.Application. Persistence
{
	public class WatersportsManagerDbContext: DbContext
	{
		public DbSet<BoatType> BoatTypes { get; set; }
		protected override void OnModelCreating (ModelBuilder
		modelBuilder)
	{
		BoatTypeModels (modelBuilder);
		base.OnModelCreating (modelBuilder);
		modelBuilder.ApplyConfigurationsFromAssembly(typeof(BoatTypeCon
		figuration).Assembly);
		private static void BoatTypeModels (ModelBuilder modelBuilder)
		{
			modelBuilder.Entity<BoatType>(). HasData(
			new BoatType()
			{
				Id = 1,
				PriceId = 7,
				Name = "QS 675", Registration = "RV000", Length =
				6.75, MaxCapacity = 8,
				Horsepower = 120,
				FuelPercentage = 100,
				LifeJackets = 10,
				IsFunctional = true
			});
		}
	}
}
```

Kôd 4.9 Definiranje konteksta

U ovom kodu je kreirana klasa koja komunicira s bazom, ona također primjenjuje sve konfiguracije za entitete koji se koriste u kontekstu baze podataka i nadodaje nekoliko početnih tablica prilikom kreiranje baze.

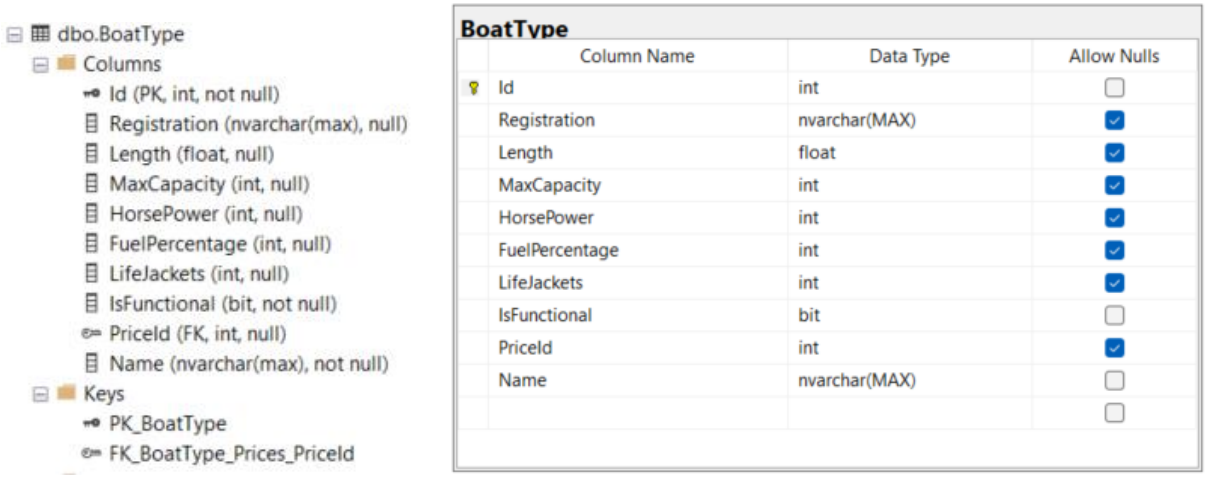
U sljedećem isječku koda prikazan je primjer proširenja konteksta koji služi za postavljanje veze s bazom podataka.

```
builder.Services.AddDbContext<WatersportsManagerDbContext>(options
      =>options.UseSqlServer(builder.Configuration.GetConnectionString
      ("WatersportsManagerDb")));
```

Kôd 4.10 Proširenje konteksta

Nakon što se sve navedeno konfigurira, kroz Package Manager Console se pozovu dvije naredbe. Prva je *add-migration* u kojoj se generira migracija baze, a druga *update-database* u kojoj se migracija baze umetne u željeni server.

Napokon, u sljedećem se primjeru provjerava je li tablica uspješno kreirana.




Slika 4.1 Tablica koja predstavlja vrstu plovila

Kao što možemo vidjeti, primarni ključ *Id_BoatType* je uspješno postavljen. Isto tako je i strani ključ *FK_BoatType_Prices_PriceId* uspješno postavljen. Konfiguracije su također uspješno postavljene i ova tablica je sada sa sigurnošću dostupna za korištenje.

### 4.1.4. Registracija i autentifikacija korisnika

Registracija korisnika i provjera autentičnosti uobičajene su značajke mnogih softverskih aplikacija, osobito onih koje uključuju korisničke račune i kontrolu pristupa.

Registracija korisnika je proces kreiranja novog korisničkog računa unutar aplikacije. Obično to uključuje prikupljanje nekih osnovnih podataka od korisnika, kao što su njegovo ime, email i lozinka, te njihovo sigurno pohranjivanje u bazu podataka.

Autentifikacija je proces provjere je li korisnik ono za što se predstavlja. To obično uključuje traženje od korisnika da unese svoje korisničko ime i lozinku, koji se zatim uspoređuju s pohranjenim vjerodajnicama kako bi se osiguralo da odgovaraju. Ako su vjerodajnice ispravne, korisniku se odobrava pristup aplikaciji ili njezinom određenom dijelu.

Zajedno, registracija korisnika i provjera autentičnosti korisnicima omogućuju način kreiranja i pristupa vlastitim računima unutar aplikacije, omogućujući im da prilagode svoje iskustvo i pristup značajkama koje su ograničene na registrirane korisnike. Oni također pružaju način za razvojne programere aplikacija da kontroliraju pristup određenim dijelovima aplikacije ili podataka, osiguravajući da su osjetljive informacije zaštićene i dostupne samo ovlaštenim korisnicima. U nastavku će biti predstavljeni kodovi koji opisuju primjenu registracije i autentifikacije u poslužiteljskom i klijentskom dijelu.

#### 4.1.4.1. Primjena u poslužiteljskom dijelu

Sljedeći isječak koda predstavlja provjeru valjanosti modela, osobe i lozinke. Ako navedene stavke nisu valjane, onda javi grešku. Ako je sve ispravno, poziva se *_personService.CreateJwtToken(person)* za stvaranje tokena. Zatim dodjeljuje token korisniku i vraća odgovor s tokenom i porukom "Uspješna prijava!".

```
[HttpPost("authenticate")]
public async Task<IActionResult> AuthenticatePerson(
      [FromBody] PersonDto model)
        {
            if (model == null)
                return BadRequest();

            Person person = await _personService.AuthenticatePerson(model);

            if (person == null)
                return NotFound(new { Message = "Person Not Found!" });

            if (!PasswordHasher.VerifyPassword(
                  model.Password, person.Password))
                return BadRequest(new {
                  Message = "Password is Incorrect"});
            model.Token = _personService.CreateJwtToken(person);
            return Ok(new
```

```
            {
                Token = model.Token,
                Message = "Login Success!"
            });
        }
```

Kôd 4.11 Autentifikacija osobe

Sljedeći isječak koda predstavlja provjeru prilikom kreiranja korisnika, odnosno postoji li korisnik s istim mailom i korisničkim imenom. Ako postoji javlja grešku, ako ne postoji, onda se korisnik kreira.

```
[HttpPost("register")]
public async Task<IActionResult> RegisterPerson(
[FromBody] CreatePersonDto model, CancellationToken cancellationToken)
        {
            if (model == null)
                return BadRequest();
            if (await _personService.CheckUsernameExistAsync(
                  model.Username))
                return BadRequest(new {
                  Message = "Username Already Exist!" });
            if (await _personService.CheckEmailExistAsync(model.Email))
                return BadRequest(new {
                  Message = "Email Already Exist!" });
            var password = _personService.CheckPasswordStrength(
                              model.Password);
            if(!string.IsNullOrEmpty(password))
                return BadRequest(new { Message = password });

            model.Password = PasswordHasher.HashPassword(model.Password);
            model.Token = "";
            await _personService.CreatePerson(model, cancellationToken);

            return Ok(new { Message = "User Registered!" });
        }
```

Kôd 4.12 Registracija osobe

U ovom kodu se svaka kreirana lozinka kriptira i sprema u bazu podataka. Ovo je uobičajena praksa kako bi se osiguralo da obične tekstualne lozinke nisu pohranjene u bazi podataka, što bi predstavljalo sigurnosnu ranjivost.

Ukratko, ovaj kod je metoda za rukovanje zahtjevima za registraciju korisnika. Provjerava valjane unose zahtjeva kao što su model koji nije null, jedinstveno korisničko ime, email, jačinu lozinke te stvara novu osobu s kriptiranom lozinkom u bazi podataka i vraća odgovarajuće odgovore.

Sljedeći isječak koda predstavlja kreiranje JWT tokena za korisnika.

```
public string CreateJwtToken(Person person)
        {
            var jwtTokenHandler = new JwtSecurityTokenHandler();
            var key = Encoding.ASCII.GetBytes("veryverysecret.....");

            var identity = new ClaimsIdentity(new Claim[]
            {
                new Claim(ClaimTypes.Role, $"{(
                  person.RoleId == 2 ? "ADMIN" : "USER")}"),
                new Claim(ClaimTypes.Name, $"{
                  person.FirstName} {person.LastName}"),
                new Claim("Id", $"{person.Id}")
            });

            var credentials = new SigningCredentials(
            new SymmetricSecurityKey(key), SecurityAlgorithms.HmacSha256);

            var tokenDescriptor = new SecurityTokenDescriptor
            {
                Subject = identity,
                Expires = DateTime.Now.AddDays(1),
                SigningCredentials = credentials
            };
```

```
            var token = jwtTokenHandler.CreateToken(tokenDescriptor);
            return jwtTokenHandler.WriteToken(token);
        }
```

Kôd 4.13 Kreiranje tokena

JWT token je niz znakova koji sadrži kodirane informacije u standardiziranom formatu. Sastoji se od tri dijela odvojena točkama: zaglavlje, sadržaj i potpis. Zaglavlje sadrži informacije o vrsti tokena i algoritmu koji se koristi za njegovo potpisivanje. Korisni teret sadrži stvarne podatke ili zahtjeve koji se prenose, kao što su korisnički ID, ime ili dopuštenja. Potpis se koristi za provjeru da token nije neovlašteno mijenjan i da ga je generirala osoba od povjerenja.

JWT tokeni se obično koriste za autentifikaciju i autorizaciju u web aplikacijama i pristupnim točkama. Kada se korisnik prijavi, poslužitelj generira JWT token i šalje ga natrag klijentu, obično u kolačiću ili u tijelu odgovora. Klijent zatim uključuje token u sljedeće zahtjeve, bilo u zaglavlju ili kao parametar upita, kako bi dokazao svoj identitet i dobio pristup zaštićenim resursima.

JWT tokeni su popularni jer su bez statusa, što znači da poslužitelj ne treba pratiti nikakve podatke o sesiji ili korisničke informacije nakon početne autentifikacije. Ovo može pojednostaviti dizajn i implementaciju web aplikacija i pristupnih točaka, kao i poboljšati izvedbu i skalabilnost. Međutim, važno je osigurati da se JWT tokeni sigurno generiraju i da su ispravno potvrđeni na strani poslužitelja kako bi se spriječio neovlašteni pristup ili petljanje.[10]

#### 4.1.4.2. Primjena u klijentskom dijelu

Sljedeći isječak koda predstavlja Angular klasu servis pod nazivom *AuthService* koja upravlja procesom autentifikacije korisnika. Ona kreira i omogućuje korisniku da se ulogira u sistem, postavlja korisniku JWT token i provjerava  je li korisnik dobro unio podatke i je li korisnik postojeći.

```
export class AuthService {
  private baseUrl: string = "https://localhost:7185/Person/"
  private personPayload: any;
  constructor(private http: HttpClient, private router: Router) {
```

```
  this.personPayload = this.decodedToken()
}
signUp(signUpObj: any): Promise<any> {
  return new Promise((resolve, reject) => {
    this.http.post<any>(`${this.baseUrl}register`, signUpObj).subscribe({
      next: (response) => {
        resolve(response);
      },
      error: (err) => {
        reject(err);
      }
    });
  });
}
login(loginObj: any): Promise<any> {
  return new Promise((resolve, reject) => {
    this.http.post<any>(
    `${this.baseUrl}authenticate`, loginObj).subscribe({
      next: (response) => {
        this.storeToken(response.token);
        this.personPayload = this.decodedToken();
        resolve(response);
      },
      error: (err) => {
        reject(err);
      }
    });
  });
}
storeToken(tokenValue: string) {
  localStorage.setItem('token', tokenValue)
}
decodedToken() {
  const jwtHelper = new JwtHelperService();
  const token = this.getToken()!;
```

```
    console.log(jwtHelper.decodeToken(token));
    return jwtHelper.decodeToken(token);
  }
  getToken() {
    return localStorage.getItem('token')
  }
}
```

Kôd 4.14 Autentikacijski servis

Sljedeći isječak koda predstavlja kod koji definira klasu *AuthGuard* koja provjerava je li korisnik prijavljen prije nego što mu dopusti pristup zaštićenim rutama u web aplikaciji. Ako korisnik nije prijavljen, prikazuje se poruka o pogrešci i preusmjerava ga na stranicu za prijavu.

```
export class AuthGuard implements CanActivate {
  constructor(private auth : AuthService, private router: Router, private
  toast: NgToastService) {
  }
  canActivate():boolean {
    if (this.auth.isLoggedIn()) {
      return true
    } else {
      this.toast.error
            ({ detail: "ERROR", summary: "Please Login First!" });
      this.router.navigate(['login'])
      return false;
    }
  }
}
```

Kôd 4.15 Zaštita ruta

Sljedeći isječak koda obrađuje pogreške povezane s autentifikacijom, kao što su istekli tokeni, prikazivanjem poruke upozorenja i odjavom korisnika. Presretač to postiže implementacijom sučelja *HttpInterceptor* i korištenjem metode *getToken()* iz *AuthService* za dohvaćanje tokena. Općenito, ovaj presretač pomaže u osiguravanju dosljednog ponašanja pri postavljanju HTTP zahtjeva s autorizacijom.

```
@Injectable()
export class TokenInterceptor implements HttpInterceptor {
  constructor(private auth: AuthService, private toast: NgToastService,
      private router: Router) { }
  intercept(request: HttpRequest<unknown>, next: HttpHandler):
      Observable<HttpEvent<unknown>> {
    const myToken = this.auth.getToken();
    if (myToken) {
      request = request.clone({
        setHeaders: { Authorization: `Bearer ${myToken}` }
      })
    }
    return next.handle(request).pipe(
      catchError((err: any) => {
        if (err instanceof HttpErrorResponse) {
          if (err.status === 401) {
            this.toast.warning({ detail: "Warning",
                  summary: "Token is expired, Please Login again" });
            this.auth.signOut()
          }
        }
        return throwError(() => err)
      })
    );
  }
}
```

Kôd 4.16 Presretač tokena

Sljedeći isječak koda predstavlja Angular komponentu, a Angular komponente su građevni blokovi Angular aplikacije koji su odgovorni za rukovanje korisničkim sučeljem i logikom interakcije. Mogu se ponovno koristiti, modularni su i mogu se kombinirati za stvaranje složenih korisničkih sučelja.

```
onLogin() {
    if (this.loginForm.valid) {
      Send the object to database
      this.auth.login(this.loginForm.value).then((res: any) => {
        this.loginForm.reset();
        this.auth.storeToken(res.token);
        const tokenPayload = this.auth.decodedToken();
        this.personStore.setFullNameForStore(tokenPayload.unique_name);
        this.personStore.setRoleForStore(tokenPayload.role);
        this.toast.success({ detail: "SUCCESS",
            summary: res.message, duration: 5000 });
        this.router.navigate(['/home']);
      }).catch((err: any) => {
        this.toast.error({ detail: "ERROR",
            summary: err.error.message, duration: 5000 });
      });
    } else {
      Throw the error using toaster with required fields
      ValidateForm.validateAllFormFields(this.loginForm);
      this.toast.error({ detail: "ERROR",
            summary: "Your form is invalid!", duration: 5000 });
    }
  }
```

Kôd 4.17 Komponenta za login

Ovaj kod je funkcija u Angular komponenti koja rukuje prijavom korisnika slanjem korisničkih podataka za prijavu na poslužitelju, pohranjivanjem vraćenog tokena i prikazivanjem poruka o uspjehu ili pogrešci ovisno o rezultatu.

### 4.1.5. Komunikacija s pristupnim točkama web API-ja

Sada se napokon može ostvariti komunikacija s pristupnim točkama kako bi se ostvarila cjelina sustava. U sljedećem isječku koda predstavlja se Angular servis koji je odgovoran za rukovanje komunikacijom s pozadinskom RESTful pristupnom točkom koja se odnosi na reporte. Odgovoran je i za slanje HTTP zahtjeva poslužitelju i rukovanje primljenim odgovorima.

```
@Injectable({
  providedIn: 'root'
})
export class ReportService {
  private baseUrl: string = 'https://localhost:7185/Report/'
  constructor(private http: HttpClient) { }
  getReports(): Promise<any> {
    return new Promise((resolve, reject) => {
      this.http.get<any>(this.baseUrl).subscribe({
        next: (res) => {
          setTimeout(() => {
            resolve(res);
          }, 500);
        },
        error: (err) => {
          reject(err);
        }
      })
    })
  }
}
```

Kôd 4.18 Klijentski kod pristupa pristupnoj točki za reporte

Sljedeći isječak koda predstavlja kreiranje Angular komponente koja je odgovorna za prikaz podataka od reporta, ona dohvaća podatke iz poslužiteljskog dijela pomoću klase servis i koristi ih za renderiranje prikaza. Ova je komponenta vezana uz određenu rutu i koristi se za značajku reports.

```
@Component({
  selector: 'app-reports',
  templateUrl: './reports.component.html',
  styleUrls: ['./reports.component.scss']
})
```

```
export class ReportsComponent implements OnInit {
  @Output() reportUpdated = new EventEmitter<any[]>();
  public reports: any = [];
  reportToEdit?: Report;
  constructor(
    private reportApi: ReportService
  ) { }
  ngOnInit(): void {
    this.getReports();
  }
  getReports() {
    this.reportApi.getReports()
      .then(res => {
        this.reports = res;
      });
  }
}
```

Kôd 4.19 Klijentski kod za prikaz reporta

### 4.1.6. Navigacija među komponentama aplikacije

Navigacija se odnosi na proces kretanja između različitih prikaza ili stranica unutar aplikacije. Navigacija u Angularu se obično postiže korištenjem Angular Routera, koji je ugrađena usluga koja pruža deklarativan način za definiranje stanja aplikacije i navigaciju između njih.

Komponente, s druge strane, građevni su blokovi korisničkog sučelja Angular aplikacije. Komponente se mogu ponovno koristiti, modularne su i sadrže prezentacijsku logiku aplikacije. Obično se koriste za definiranje pojedinačnih prikaza ili odjeljaka korisničkog sučelja aplikacije i mogu sadržavati predložak koji definira strukturu pogleda i klasu koja definira ponašanje i podatke za prikaz.

Komponente se mogu ugnijezditi jedna u drugu kako bi se stvorila složenija sučelja i mogu međusobno komunicirati koristeći ulaze, izlaze i servise. U Angularu se komponente mogu smatrati UI ekvivalentom funkcija u programskom jeziku, gdje svaka komponenta enkapsulira određenu funkcionalnost ili ponašanje aplikacije.[11]

Sljedeći isječak koda predstavlja primjer kako se rute definiraju i konfiguriraju u Angular aplikaciji. Niz ruta sadrži nekoliko objekata od kojih svaki definira put i komponentu. Modul

Angular Router omogućuje programerima jednostavno definiranje i upravljanje rutama aplikacije.

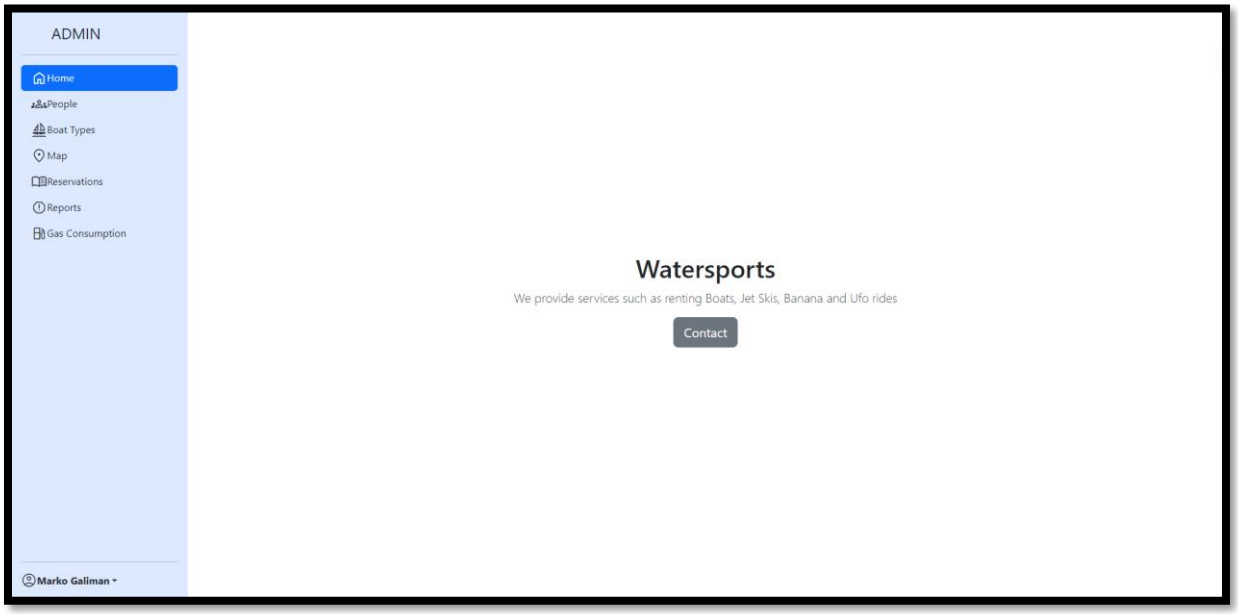
```
const routes: Routes = [
  { path: '', redirectTo: 'home', pathMatch: 'full' },
  { path: 'login', component: LoginComponent },
  { path: 'signup', component: SignupComponent },
  { path: '', component: LayoutComponent, canActivate: [AuthGuard],
      children: [
      { path: 'home', component: HomeComponent,
            canActivate: [AuthGuard] },
      { path: 'people', component: PeopleComponent,
            canActivate: [AuthGuard] },
      { path: 'boat-types', component: BoatTypesComponent,
            canActivate: [AuthGuard] },
      { path: 'map', component: MapComponent,
            canActivate: [AuthGuard] },
      { path: 'reservations', component: ReservationsComponent,
            canActivate: [AuthGuard] },
      { path: 'reports', component: ReportsComponent,
            canActivate: [AuthGuard] },
      { path: 'gasConsumption', component: GasConsumptionComponent,
            canActivate: [AuthGuard] },
      { path: 'skippers', component: SkippersComponent,
            canActivate: [AuthGuard] },
      { path: 'reviews', component: ReviewsComponent,
            canActivate: [AuthGuard] },
      { path: 'coupons', component: CouponsComponent,
            canActivate: [AuthGuard] },
      { path: 'editPerson', component: EditPersonComponent,
            canActivate: [AuthGuard] }
    ]
  },
];
```

Kôd 4.20 Navigacijski modul

## 4.2. Korisničke funkcionalnosti

U ovom dijelu se opisuje i prezentira dizajn stranice, koje sve korisničke funkcionalnosti postoje i na koji se način koriste.

### 4.2.1. Glavno sučelje




Slika 4.2 Stranica za administratora

Suvremeni svijet i njegove potrebe zahtijevaju prilagodljiva i funkcionalna softverska rješenja. Svi segmenti poslovanja, od turizma do financija, koriste tehnologiju kako bi poboljšali svoje poslovanje i pružili kvalitetne usluge svojim klijentima. Stoga, izrada kvalitetnog sučelja za upravljanje poslovanjem jedna je od ključnih stvari koja određuje uspjeh i učinkovitost poslovanja.

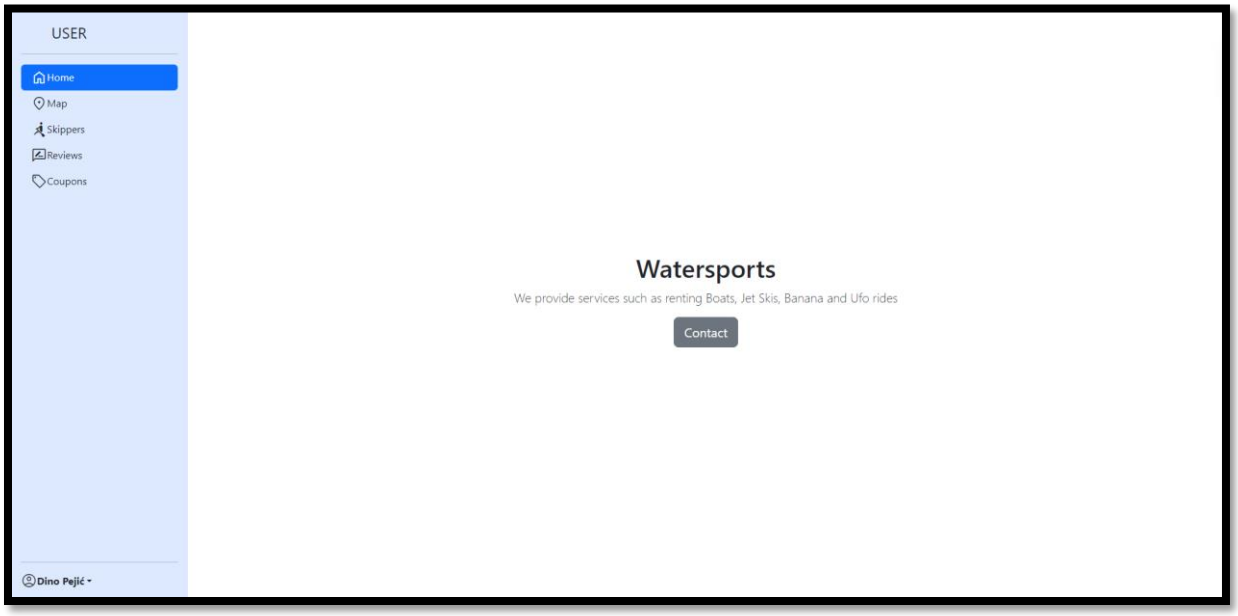
Sadržaj glavnog sučelja:

Glavno sučelje pruža jednostavno i intuitivno korištenje te funkcionalnost za upravljanje poslovanjem. Navigacijska traka s lijeve strane sadrži sve opcije koje omogućuju lako navigiranje kroz cijelo sučelje. S druge strane, glavni sadržaj s desne strane prikazuje podatke i funkcionalnosti koje su odabrane u navigacijskoj traci.

Navigacijski izbornik mijenja se ovisno o tome je li korisnik USER ili ADMIN, što znači da će svaki korisnik dobiti iskustvo prilagođeno njegovim potrebama i ulogama.

Funkcionalnosti:

- Kada je korisnik postavljen kao ADMIN, Home opcija prikazuje naziv i kratak opis firme, što je korisno za promociju i prezentaciju poslovanja.
- Contact opcija nas vodi na stranicu za kontaktiranje djelatnika firme, što pomaže u povećanju komunikacije s klijentima i rješavanju njihovih pitanja i problema.
- People opcija omogućuje pregled svih korisnika i opciju za brisanje i uređivanje usera, što je važno za sigurnost i praćenje aktivnosti korisnika.
- Boat Types opcija pruža pregled svih plovila i opciju za uređivanje, što je korisno za upravljanje flotom i pružanje informacija klijentima.
- Map opcija prikazuje kartu okolice Rovinja s prikazanim kampovima, što pomaže u planiranju i rezervaciji smještaja.
- Reservations opcija omogućuje kreiranje i uređivanje rezervacija, što je važno za praćenje dostupnosti smještaja i planiranje poslovanja.
- Reports opcija pruža pregled svih reporta i opciju za uređivanje, što je korisno za analizu poslovanja i donošenje odluka.
- Gas Consumption opcija omogućuje kalkulaciju potrošnje goriva koje je kupac potrošio, što je važno za lakše kalkuliranje potrošnje goriva.




Slika 4.3 Stranica za korisnika

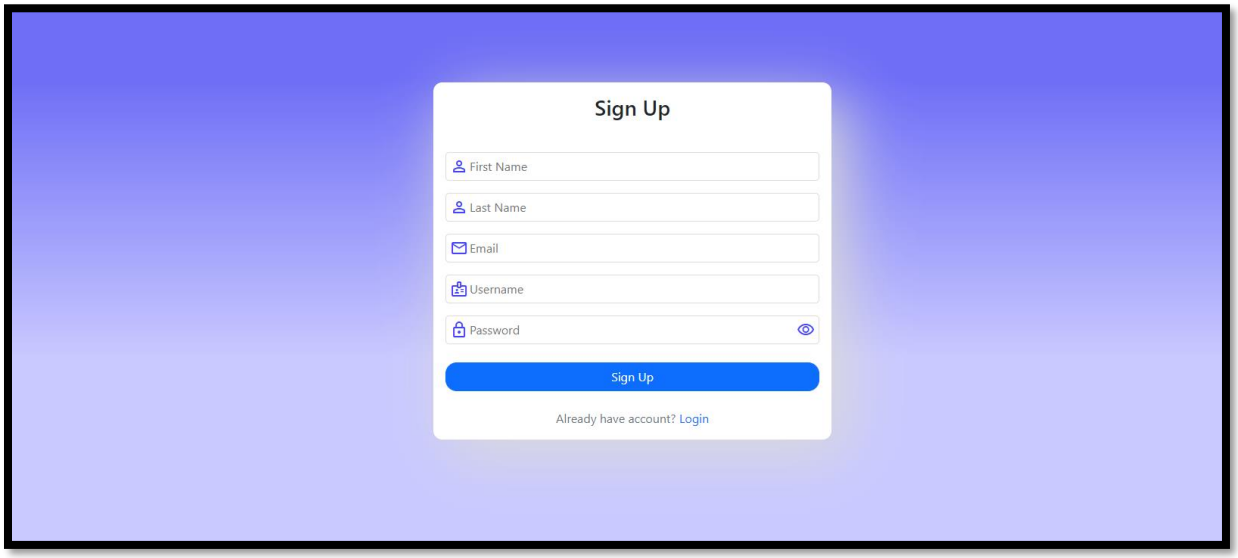
Kada je korisnik USER, postoje dva zajednička izbornika, Home i Map. Home opcija pruža informacije o firmi i uslugama koje pruža. Map opcija pruža pregled okolice Rovinja i kampova u kojima se nalaze poslovnice. Izbornik Skippers omogućuje pregled dostupnih skipera, što je

korisno za planiranje avanture na moru i odabir iskusnog skipera. Izbornik Reviews pruža mogućnost pregleda i kreiranja ocjena usluga i djelatnika. Ove ocjene su korisne za druge korisnike koji se žele informirati o kvaliteti usluga i djelatnika. Izbornik Coupons omogućuje pregled svog kupona za jeftiniju uslugu. Ovaj izbornik omogućuje korisnicima da vide svoje aktivne popuste i da ih koriste pri rezervaciji aktivnosti.

U dnu sučelja, lijevo, prikazano je korisničko ime i prezime te opcija za odjavu iz aplikacije. Ova opcija omogućuje korisnicima da se brzo i jednostavno odjave iz aplikacije i zaštite svoje privatne informacije.

U konačnici, sučelje korisničkoj verziji aplikacije omogućuje jednostavno i intuitivno korištenje svih funkcionalnosti koje su korisnicima potrebne za planiranje avanture na moru. Funkcionalnosti poput pregleda dostupnih skipera, kreiranja ocjena i pregleda kupona omogućuju korisnicima da planiraju svoju avanturu na moru temeljem relevantnih informacija i koriste najbolje usluge i popuste.

### 4.2.2. Prijava korisnika




Slika 4.4 Registracija korisnika

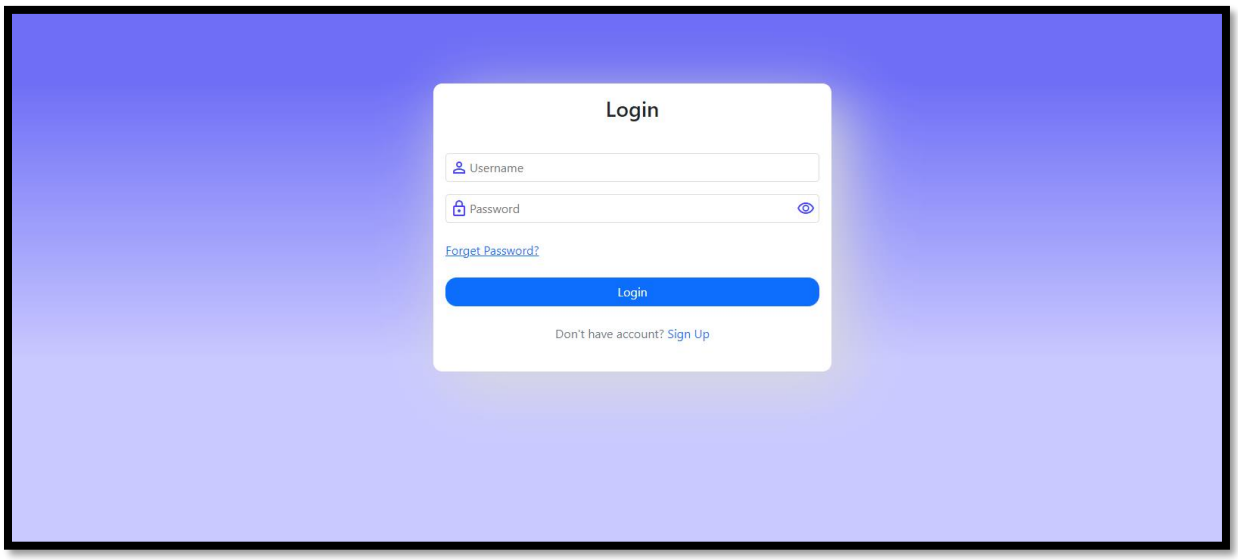
Registracija i prijava u aplikaciju su ključni koraci za korištenje aplikacije. Registracija omogućava novom korisniku da se prijavi u aplikaciju i da ima pristup svim funkcijama koje su dostupne. Bez registracije, korisnik neće moći koristiti aplikaciju.

Proces registracije:

Registracija se odvija u nekoliko jednostavnih koraka. Prvi korak je unos osnovnih informacija o korisniku, uključujući ime, prezime, email i željeno korisničko ime. Zatim, korisnik unosi

svoju lozinku. Lozinka bi trebala biti snažna i jedinstvena, što znači da bi trebala sadržavati mješavinu velikih i malih slova, brojeva i specijalnih znakova.

Nakon što su unesene sve informacije, korisnik je registriran i može ući u aplikaciju putem prijave.




Slika 4.5 Prijava korisnika

Prijava u aplikaciju:

Nakon što se korisnik registrira, moći će se prijaviti u aplikaciju putem svojeg korisničkog imena i lozinke. Ako korisnik zaboravi svoju lozinku, moći će ju resetirati putem emaila.

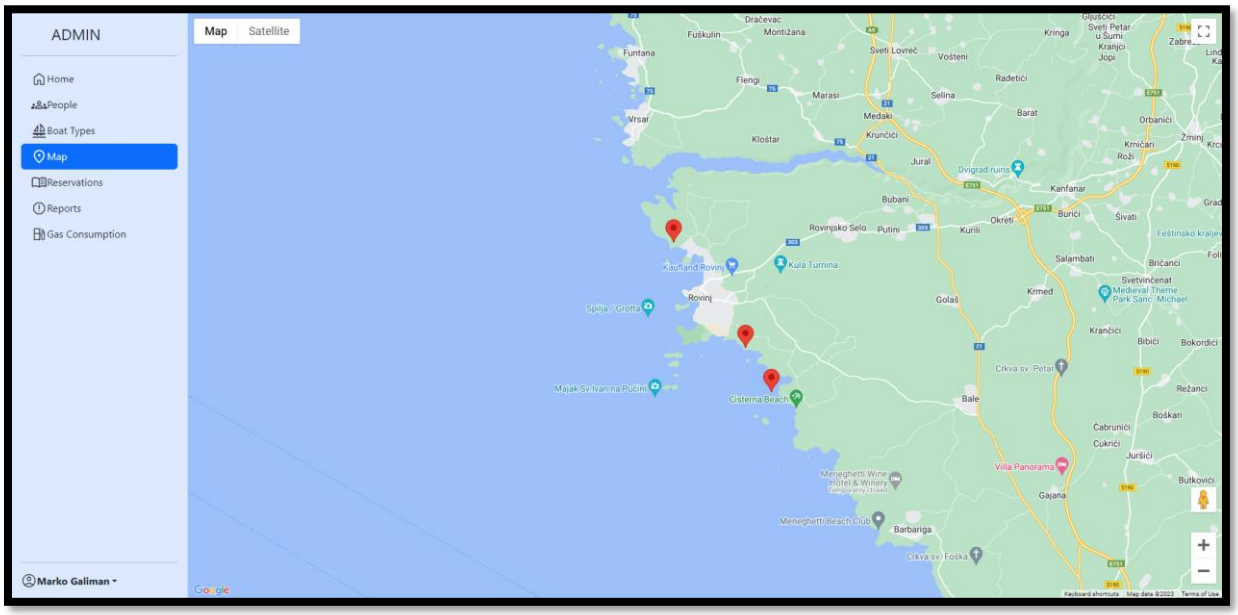
U aplikaciji se također automatski sprema opcija "Zapamti me", što omogućuje korisniku da se automatski prijavi u aplikaciju prilikom svakog dolaska na stranicu. Ova opcija također osigurava da korisnik ne mora svaki put ponovno unositi svoje podatke prijave.

Registracija i prijava su važni dijelovi svake aplikacije. Oni omogućuju korisnicima da se registriraju i pristupe svim funkcijama aplikacije. Aplikacija koja nudi jednostavan proces registracije i prijave povećava vjerojatnost da će korisnik ostati aktivan i koristiti aplikaciju redovito. Iz tog je razloga važno osigurati da je proces registracije i prijave što jednostavniji i intuitivniji kako bi se povećala vjerojatnost da će korisnici ostati dugotrajni korisnici aplikacije.

Korisnicima treba omogućiti da unesu potrebne informacije bez ikakvih poteškoća, što uključuje svoje ime i prezime, email, željeno korisničko ime i snažnu lozinku. Također, aplikacija osigurava sigurnost korisničkih podataka korištenjem naprednih sigurnosnih tehnologija i protokola, kao što je enkripcija podataka i autentifikacija korisnika.

Registracija i prijava su ključni za uspjeh aplikacije jer omogućuju korisnicima da pristupe svim funkcijama i da ih koriste na najbolji način. Ako proces registracije i prijave nije jednostavan i intuitivan, korisnici će se osjećati frustrirano i vjerojatno će napustiti aplikaciju, što će u konačnici utjecati na njezin uspjeh.

### 4.2.3. Mapa



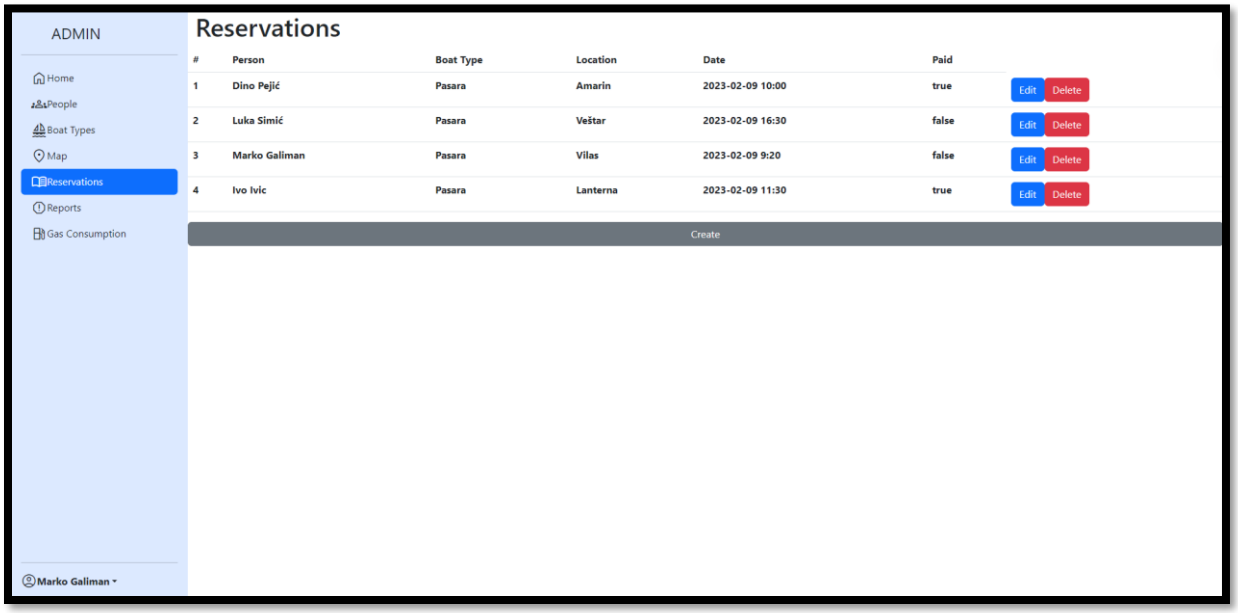

Slika 4.6 Mapa

Mapa sadrži razne informacije o kampovima, mjestima gdje se nalaze, aktivnostima koje se nalaze oko tih lokacija i tako dalje.

Osim toga, funkcija prikazivanja mape također pomaže korisnicima da shvate lokaciju kampova i steknu bolju sliku o okolnostima u kojima se nalaze. To može biti korisno za one koji planiraju putovanje u nepoznato područje i žele dobiti širi uvid u mjesto koje žele posjetiti.

Ova funkcija je jednostavna i responzivna, što omogućuje korisnicima da brzo i jednostavno koriste mapu.

### 4.2.4. Rezervacija plovila




Slika 4.7 Prikaz rezervacija

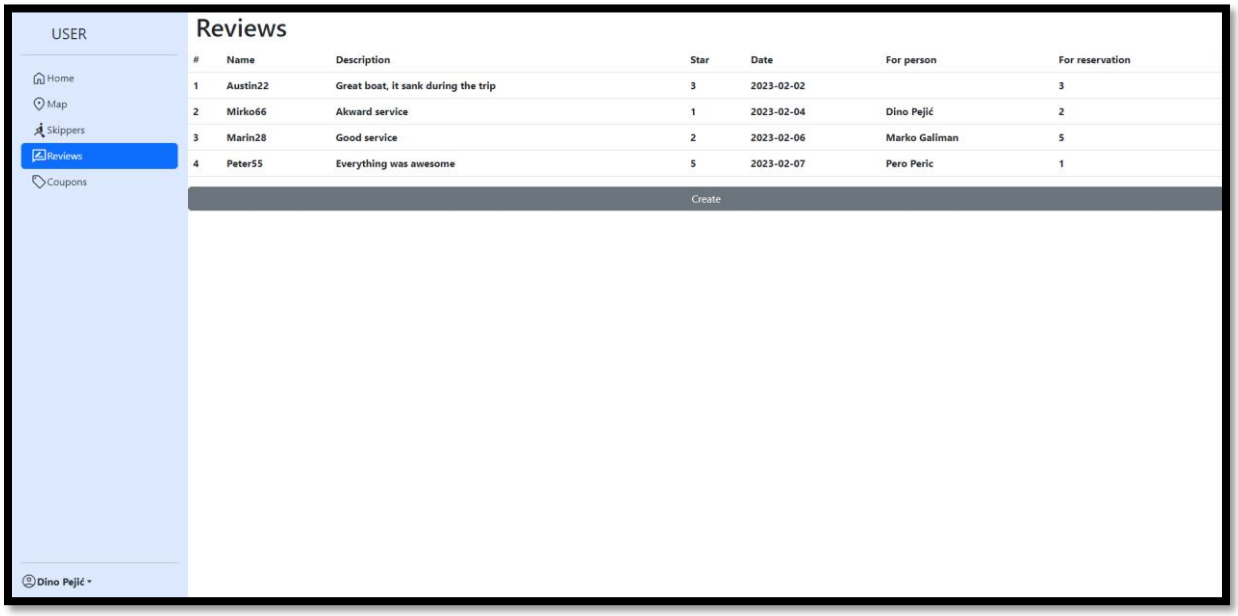
Tablica za prikaz rezervacija također može imati funkciju filtriranja, što omogućuje korištenje različitih kriterija za sortiranje rezervacija (po vrsti broda, lokaciji, datumu itd.). To će pomoći pri pronalasku željene rezervacije brže i jednostavnije.

Tablica također može imati opciju uređivanja, što omogućuje djelatniku da ažurira ili izmijeni bilo koju informaciju o rezervaciji, poput izmjene datuma ili vrste broda. To će osigurati da svi podaci o rezervacijama budu uvijek aktualni i točni.

Tablica također može imati opciju brisanja, što omogućuje brzo i jednostavno brisanje nepotrebnih ili neispravnih rezervacija. To će osigurati manji broj pogrešaka ili zabuna u vezi s rezervacijama.

Tablica za prikaz rezervacija je ključni dio svake aplikacije za iznajmljivanje brodova. To omogućuje brzo i jednostavno kreiranje, pregled i uređivanje rezervacija, čime se osigurava točnost i ažuriranost informacija o rezervacijama. Povećava se učinkovitost rada i osigurava zadovoljstvo korisnika.

### 4.2.5. Sustav ocjenjivanja




Slika 4.8 Prikaz ocjena

Ova funkcija omogućuje korisnicima da iznesu svoje mišljenje i doživljaj u vezi rezervacije i usluga koje su dobili. Ove ocjene su vrlo važne za daljnji razvoj aplikacije i osiguravaju da su usluge koje se pružaju uvijek na visokom nivou.

Kako bi se osiguralo da što više korisnika koristi ovu funkciju, ona je napravljena tako da bude jednostavna i laka za korištenje. Dizajn stranice za davanje ocjena je čist i jednostavan, sa što manje elemenata koji bi mogli odvratiti pažnju korisniku.

Kako bi se osigurala iskrenost i vjerodostojnost ocjena, dozvoljava se anonimno davanje ocjena. Korištenjem korisničkog imena, osigurava se da su ocjene povezane sa stvarnim korisnikom i da su one stvarno proizvod korisničkog iskustva.

Funkcija davanja ocjena omogućuje da korisnici iznesu svoje mišljenje i da se osigura poboljšanje usluge.

# 5. Analiza uspješnosti aplikacije

Analiza uspješnosti aplikacije bitan je dio svakog projekta razvoja softvera. U današnjem brzom svijetu, gdje tvrtke neprestano nastoje poboljšati svoju učinkovitost i pružiti bolja korisnička iskustva, ključno je analizirati izvedbu web aplikacija kako bi se osigurala njihova vrhunska izvedba. Web aplikacija "Watersports Manager" izvrstan je primjer aplikacije koja je optimizirana za performanse, što je čini vrijednim alatom za tvrtku koja se bavi vodenim sportovima koja je koristi.

Jedna od ključnih značajki web aplikacije "Watersports Manager" je njena modularnost. Aplikacija je izgrađena koristeći modularnu arhitekturu, što znači da su njezine različite komponente dizajnirane da rade zajedno na besprijekoran i učinkovit način. Ovaj modularni dizajn olakšava nadogradnju ili zamjenu pojedinačnih komponenti aplikacije bez utjecaja na cjelokupnu izvedbu ili funkcionalnost sustava. Dodatno, modularna arhitektura omogućuje skalabilnost, što znači da aplikacija može podnijeti povećan promet i potražnju korisnika kako posao raste.

Još jedna značajna prednost aplikacije "Watersports Manager" je jednostavnost korištenja. Korisničko sučelje aplikacije je jednostavno i intuitivno, što radnicima olakšava korištenje i navigaciju, čak i ako imaju malo ili nimalo tehničkog znanja. Ova jednostavnost korištenja je ključna jer omogućuje radnicima da se više usredotoče na klijente, a manje na navigaciju aplikacijom, što dovodi do povećane učinkovitosti i zadovoljstva korisnika.

Unatoč brojnim prednostima aplikacije, ona ima neke male nedostatke. Na primjer, dok je korisničko sučelje jednostavno i lako za korištenje, može se poboljšati u smislu estetike. Osim toga, dokumentacija aplikacije mogla bi biti sveobuhvatnija, što bi programerima i osoblju za podršku olakšalo rješavanje problema i nadogradnju.

Ova web aplikacija je izvrstan primjer dobro osmišljene i optimizirane aplikacije koja je pomogla tvrtki za vodene sportove povećati učinkovitost, prihode i zadovoljstvo kupaca. Modularnost i jednostavnost korištenja aplikacije čine je vrijednim alatom za tvrtku, dok analiza performansi jamči da radi na vrhunskoj učinkovitosti. Iako postoje neki mali nedostaci, oni se mogu riješiti daljnjim razvojem i optimizacijom. Općenito, web aplikacija „Watersports Manager" je izvrstan model za dizajniranje web aplikacija za poboljšanje poslovnih rezultata i zadovoljstva korisnika.

# Zaključak

Zaključno, implementacija novog softverskog sustava u području aktivnosti vodenog turizma može donijeti značajne koristi za poslovanje. Provođenjem detaljne analize postojećeg sustava, određivanjem specifičnih zahtjeva, projektiranjem i testiranjem novog sustava te pružanjem obuke i stalne podrške, tvrtka može osigurati da novi sustav bude uspješno implementiran i da zadovolji potrebe poslovanja.

Usvajanje digitalnog sustava u području aktivnosti vodenog turizma može pružiti mnoge pogodnosti tvrtkama koje nude ove usluge. Novi softverski sustav može usmjeriti i pojednostaviti upravljanje dostupnim aktivnostima, olakšavajući upravljanje rasporedom zaposlenika, rezervacijama kupaca, rezervacijama brodova i više. Digitalizacijom upravljanja aktivnostima vodenog turizma, poslovanje može smanjiti troškove, poboljšati učinkovitost i povećati prihode.

Implementacijom novog digitalnog sustava za aktivnosti vodenog turizma, tvrtke mogu poboljšati svoje poslovanje i postići veći uspjeh. Novi softverski sustav može pomoći tvrtki da učinkovitije upravlja svojim resursima, uključujući brodove, zaposlenike i klijente. Sustav također može poboljšati komunikaciju između zaposlenika i kupaca, čineći cijeli proces učinkovitijim i jednostavnijim. Osim toga, novi softverski sustav može pružiti vrijedne podatke koji mogu pomoći tvrtkama da donose informirane odluke i poboljšaju svoje ponude.

Sve u svemu, implementacija novog softverskog sustava u području aktivnosti vodenog turizma ključni je korak za tvrtke koje žele optimizirati svoje poslovanje i ostati konkurentne na današnjem tržištu. Uz pravilno planiranje, izvođenje i stalnu podršku, tvrtka može osigurati da novi sustav zadovoljava njezine potrebe i pruža vrijedne prednosti i za tvrtku i za njene klijente.

# Popis kratica

| | |
|---|---|
| ADO *Active Data Object* | Aktivni podatkovni objekt |
| API *Applicaiton Program Interface* | Aplikacijsko programsko sučelje |
| ASP *Active Server Pages* | Aktivne stranice poslužitelja |
| ACID *Atomicity Consistency Isolation Durability* | Atomičnost Konzistencija Izolacija Trajnost |
| BCL *Base Class Library* | Biblioteka osnovnih klasa |
| CLI Command Line Interface | Sučelje naredbenog retka |
| CLR *Common Language Runtime* | Zajednički jezik za izvođenje |
| DOM *Document Object Model* | Objektni model dokumenta |
| DPI *Dots per Inch* | Točke po inču |
| IBM *International Business Machines Corporation* | Međunarodna poslovna mašinerija korporacija |
| ID *Identifier* | Identifikator |
| JDBC *Java Database Connectivity* | Povezivost Java sa bazom podataka |
| JS *JavaScript* | JavaScript |
| ORM *Object-Relational Mapping* | Objektno-relacijsko preslikavanje |
| PLINQ *Parallel LINQ* | Paralelni LINQ |
| SPA *Single-page Application* | Aplikacija na jednoj stranici |
| SQL *Structured Query Language* | Strukturirani jezik upita |
| TS *TypeScript* | TypeScript |
| TPL *Task Parallel Library* | Paralelna biblioteka zadataka |
| WCF *Windows Communication Foundation* | Windows okvir za komunikacije |
| WF *Windows Workflow Foundation* | Windows okvir za radne tokove |
| WPF *Windows Presentation Foundation* | Windows okvir za prezentacije |
| XML *eXtensible Markup Language* | Proširivi jezik za označavanje |

# Popis slika

# Popis kôdova

# Literatura


[1] Ashcraft, A. (2022) Parallel Programming and Concurrency with C# 10 and .NET 6 : A modern approach to building faster, more responsive, and asynchronous .NET applications using C# [online]. Dostupno na: https://www.amazon.com/Parallel-Programming-Concurrency-NET-asynchronous-ebook/dp/B0B25CW5NF, pristupano veljača 2023.

[2] Whitaker, RB. The C# Player's Guide: 3rd Edition [online]. Dostupno na: https://www.amazon.com/C-Players-Guide-3rd/dp/0985580135, pristupano sječanj 2023.

[3] Sierra, K. Head First Java: 2nd Edition [online]. Dostupno na: https://www.amazon.com/Head-First-Java-Kathy-Sierra/dp/0596009208 ,pristupano veljača 2023.

[4] Beaulieu, A. Learning SQL: Generate, Manipulate, and Retrieve Dana [online]. Dostupno na: https://www.amazon.com/Learning-SQL-Generate-Manipulate-Retrieve-ebook/dp/B085HDSWR3, pristupano veljača 2023.

[5] DeBarros, A. Practical SQL, 2nd Edition: A Beginner's Guide to Storytelling with Dana [online]. Dostupno na: https://www.amazon.com/Practical-SQL-2nd-Beginners-Storytelling-ebook/dp/B08K7LJRQX, pristupano sječanj 2023.

[6] Sadalage, PJ. NoSQL Distilled: A Brief Guide to the Emerging World of Polyglot Persistence [online]. Dostupno na: https://www.amazon.com/NoSQL-Distilled-Emerging-Polyglot-Persistence-ebook/dp/B0090J3SYW, pristupano veljača 2023.

[7] Bampakos, A. Angular Projects: Build modern web apps by exploring Angular 12 with 10 different projects and cutting-edge technologies, 2nd Edition [online]. Dostupno na: https://www.amazon.com/Angular-Projects-exploring-cutting-edge-technologies-ebook/dp/B08XXNBKNP , pristupano sječanj 2023.

[8] Ayaz, MA., Obaid, N. Angular Cookbook: Over 80 actionable recipes every Angular developer should know [online]. Dostupno na: https://www.amazon.com/Angular-Cookbook-actionable-recipes-developer-ebook/dp/B08VTWYJ7H, pristupano sječanj 2023.

[9] Wieruch, R. The Road to React: The React.js with Hooks in JavaScript Book (2023 Edition) [online]. Dostupno na: https://www.amazon.com/Road-learn-React-pragmatic-React-js-ebook/dp/B077HJFCQX, pristupano veljača 2023.

[10] Web stranica, https://jwt.io/introduction, pristupano sječanj 2023.

[11] Web stranica, https://angular.io/guide/component-overview, pristupano sječanj 2023.

ALGEBRA
VISOKO
UČILIŠTE

# IZRADA SUSTAVA ZA UPRAVLJANJE AKTIVNOSTIMA U VODENOM TURIZMU

Pristupnik: Marko Galiman, 0321013570

Mentor: mag. ing. comp. Daniel Bele