

**АКАДЕМИЈА СТРУКОВНИХ СТУДИЈА ШУМАДИЈА
ОДСЕК КРАГУЈЕВАЦ**



ТЕМА ЗАВРШНОГ РАДА

**РАЗВОЈ МОБИЛНИХ АПЛИКАЦИЈА НА ПРИМЕРУ
– ПЛАТФОРМА ЗА ЕФИКАСНО УПРАВЉАЊЕ
ТЕРМНИМА У ФРИЗЕРСКИМ УСЛУГАМА**

- Завршни рад -

**Ментор
Др Хрвоје Пушкар**

**Студент
Лазар Бирташевић 006/2022**

**Крагујевац
10, 2025**

**АКАДЕМИЈА СТРУКОВНИХ СТУДИЈА ШУМАДИЈА
ОДСЕК КРАГУЈЕВАЦ**



ТЕМА ЗАВРШНОГ РАДА
РАЗВОЈ МОБИЛНИХ АПЛИКАЦИЈА НА ПРИМЕРУ
– ПЛАТФОРМА ЗА ЕФИКАСНО УПРАВЉАЊЕ
ТЕРМНИМА У ФРИЗЕРСКИМ УСЛУГАМА

- Завршни рад -

Ментор
Др Хрвоје Пушкар

Студент
Лазар Бирташевић 006/2022

(потпис ментора)

(потпис студента)

Крагујевац
10, 2025

Садржај

1. УВОД	4
1.1. Увод у МОБИЛНЕ АПЛИКАЦИЈЕ	4
1.2. РАЗВОЈ МОБИЛНИХ АПЛИКАЦИЈА	5
1.3. ЗАКЉУЧАК.....	5
2. О АПЛИКАЦИЈИ.....	6
3. ТЕХНОЛОГИЈЕ КОРИШЋЕЊЕ ЗА ИЗРАДУ АПЛИКАЦИЈА	7
3.1. BACKEND GO (GIN API FRAMEWORK)	8
3.2. WEB АПЛИКАЦИЈА REACT I TAILWIND CSS	8
3.3. АНДРОИД АПЛИКАЦИЈА REACT NATIVE.....	9
3.4. БАЗА ПОДАТАКА POSTGRESQL	10
3.5. POSTMAN	10
3.6. DOCKER.....	11
4. ФУНКЦИОНАЛНОСТ АПЛИКАЦИЈЕ.....	12
4.1. ФУНКЦИОНАЛНОСТ ВЕБ АПЛИКАЦИЈЕ	12
4.1.1. Регистрација салона.....	12
4.1.2. Админ панел за власника салона	14
4.1.3. Процес заказивања термина	15
4.2. ФУНКЦИОНАЛНОСТ МОБИЛНЕ АПЛИКАЦИЈЕ	18
4.2.1. Пријава фризера на апликацију.....	18
4.2.2. Преглед термина	19
4.2.3. Подешавање радног времена, услуга и нерадних дана.....	21
5. ОПИС ИМПЛЕМЕНТАЦИЈЕ ПРОЈЕКТА.....	23
5.1. ИМПЛЕМЕНТАЦИЈА BACKEND-A (GO + GIN)	23
5.1.1. Архитектура система	23
5.1.2. Handler слој.....	25
5.1.3. Service слој	26
5.1.4. Repository слој.....	27
5.1.5. Аутентификација и ауторизација	28
5.1.6. Рутирање и API endpoints	29
5.1.7. Email обавештења	30
5.2. БАЗА ПОДАТАКА POSTGRESQL.....	31
5.2.1. Дијаграм релација (ER дијаграм)	31
5.2.2. Напредне функционалности базе података	32
5.2.3. Најважнија табела за заказане термине (табела appointments)	32
5.3. ИМПЛЕМЕНТАЦИЈА ФРОНТЕНД ВЕБ АПЛИКАЦИЈЕ (REACT, TAILWIND CSS)	33
5.3.1. Архитектура апликације	33
5.3.2. Рутирање и навигација.....	34
5.3.3. API комуникација.....	34
5.3.4. Странице апликације.....	35
5.3.5. Feature компоненте	38
5.4. ИМПЛЕМЕНТАЦИЈА МОБИЛНЕ АПЛИКАЦИЈЕ (REACT NATIVE).....	39
5.4.1. Архитектура апликације	39
5.4.2. Аутентификација	40
5.4.3. API слој	41
5.4.4. Екрани апликације	42
5.5. ИМПЛЕМЕНТАЦИЈА DOCKER-A	49
6. ЗАКЉУЧАК	53
7. ЛИТЕРАТУРА.....	54

1. Увод

1.1. Увод у мобилне апликације

Мобилне апликације представљају софтверске програме који се користе на мобилним телефонима, таблетима, паметним сатовима и неизоставан су део живота скоро свих људи на свету. Последњих година конкуренција апликација расте због тога што тржиште постаје све веће. То се дешава због све већег броја програмера и алата за прављење апликација, велике фирме и заједнице константно раде на новим технологијама и олакшавању програмерима производњу мобилних апликација.

Нове технологије омогућиле су производњу различитих врста апликација као што су игрице, друштвене мреже, апликације за финансије, пословање, здравље и пружање услуга. Најпопуларније су друштвене мреже, тренутно Instagram, TikTok и FaceBook доминирају на тржишту мобилних апликација. За посао најпопуларнији је LinkedIn и помоћу њега много људи је дошло до свог првог посла.

Највећи број мобилних апликација се производи за Android и IOS оперативне системе, али такође постоје апликације за Microsoft-ов Windows Mobile оперативни систем и за неке уређаје који користе Linux оперативни систем, али то је веома мали број уређаја.

Типови мобилних апликација:

1. **Нативне апликације** се развијају посебно за одређени оперативни систем, као што су Android или iOS. Ове апликације користе специфичне програмске језике, за Android најчешће Kotlin или Java, а за iOS Swift. Предност је велика брзина и директан приступ хардверским ресурсима уређаја (камера, микрофон, GPS), док је мана то што се морају развијати посебно за сваки систем.
2. **Веб апликације** су апликације којима се приступа преко веб претраживача. Не захтевају инсталацију и најчешће су написане коришћењем HTML-а, CSS-а и JavaScript-а. Главна предност је лако одржавање и универзална доступност, али раде само уз интернет конекцију и не могу у потпуности користити све могућности уређаја.
3. **Хибридне апликације** комбинују карактеристике нативних и веб апликација. Развијају се једном, користећи технологије као што су React Native, Flutter или Ionic, а затим се постављају на више платформи. Хибридне апликације имају предност у бржем развоју и нижим трошковима, јер једна база кода функционише на више оперативних система. У ову групу спада и мобилна апликација коју сам правео за мој завршни рад, коју сам развио у React Native.

1.2. Развој мобилних апликација

Да би се развила модерна и функционална мобилна апликација потребно је познавати процес развоја апликације. Он се састоји из више фаза развоја и сваки је подједнако битан да би апликација као целина била исправна.

За почетак битно је анализирати захтев и дефинисати функционалности и корисничке потребе, ово је веома битно јер сви остали кораци зависе од тога. На основу анализе се пише код и прави функционалност целе апликације.

Следећи корак је имплементација, а то је писање кода, интегрише се база података и сервер. Ово је пресудно за цео рад апликације јер ће она функционисати по логици која је написана.

Након тога долази на ред дизајнирање корисничког интерфејса, веома је битно да кориснички интерфејс буде интуитиван да би сваки корисник могао лако да се снађе и да са лакоћом користи апликацију. Респонзивност апликације је важна зато што нису сви екрани исте величине, због тога је потребно прилагодити кориснички интерфејс за све врсте и величине екрана.

Током производње мобилне апликације обавезан део је тестирање функционалности, без тога није могуће избацити потпуно сигурну апликацију за коришћење зато што може доћи до различитих пропуста. Корисницима је веома битно да апликација коју користе буде функционална и без икаквих грешака јер то може проузроковати велике проблеме ако је апликација направљена за осетљиве и битне податке.

Када је све тестирано и имплементирано, мобилна апликација може бити објављена на Google Play или App Store ако испуњава услове који су тражени. Ово је веома битно да би потенцијални корисници могли да пронађу и користе апликацију. Потребно је редовно ажурирање и одржавање апликације како би корисници били задовољни и да би наставили са активним коришћењем.

Мобилне апликације се данас све више развијају помоћу оквира као што су React Native, Flutter или Ionic, због брзине продукције а и због тога што је тешко направити исте нативне апликације за Android и iOS оперативне системе, и то такође одузима пуно времена и ресурса.

1.3. Закључак

Мобилне апликације данас имају веома битну улогу у животу људи и у пословању предузећа. Њихов напредак је омогућио да скоро сваки пословни процес може бити прилагођен кроз мобилну технологију. Неминовно је да ће у блиској будућности сваки човек на свету користити мобилне апликације за социјализацију, лични напредак и као помоћ у свакодневном животу.

2. О апликацији

За свој дипломски рад одлучио сам да направим мобилну апликацију која ће олакшати и решити проблеме са управљањем фризерских салона и са заказивањем термина. Одлучио сам се за то због тога што је у последњих пар година све више фризерских салона и мислим да је потребан модеран начин за управљање салонима јер ће то помоћи власницима салона и фризерима да лакше организују своје време и да не долази до грешака током резервација. Такође ће много олакшати и муштеријама да брзо и лако закажу услугу и да уштеде време.

Основна идеја апликације је да се направи једноставан и функционалан систем који ће повезати све учеснике у процесу власнике салона, фризере и муштерије. Циљ је да се креира дигитално решење које ће заменити папирне распореде, ручно заказивање и телефонске позиве, који често доводе до забуна и дуплих термина. Мобилна апликација ће омогућити да се сви термини виде у реалном времену, да се свака измена одмах ажурира и да и клијенти и фризерски увек имају преглед својих обавеза.

Поред заказивања термина, апликација ће пружити могућност клијентима да виде услуге које салон нуди, цене, као и профиле фризера са сликама њихових радова. На тај начин, клијент може да изабере тачно оног фризера који му одговара по стилу или препоруци.

Посебан фокус биће на једноставности коришћења. Желим да апликација буде интуитивна, тако да је могу користити сви. Интерфејс ће бити јасан и прегледан, са минималним бројем корака потребних за заказивање термина. Корисници ће добијати обавештења о предстојећим терминима како би се смањио број пропуштених долазака, што је чест проблем у салонима. Такође, власници и фризерски ће имати увид у целокупну историју термина, што ће помоћи у бољем планирању и анализи рада.

На овај начин, апликација ће донети корист свим странама, клијентима ће омогућити једноставно заказивање, фризерима лакшу организацију радног дана, а власницима салона бољи преглед пословања. Увођењем оваквог система, фризерски салони ће постати модернији и ефикаснији, а комуникација између клијената и фризера бржа и једноставнија.

3. Технологије коришћење за израду апликација

За израду апликације користио сам савремене технологије које су тренутно веома актуелне у развоју софтвера. Као развојно окружење користио сам Microsoft Visual Studio Code, који омогућава лак и ефикасан рад са више технологија и коришћење различитих алата потребних током развоја.

Бекенд сам радио у програмском језику Go (Golang), који има велику брзину и стабилност. По својој структури подсећа на C++, али је знатно једноставнији за рад и оптимизован за израду модерних веб сервера. За изградњу API-ја користио сам Gin framework. Одлучио сам се за њега јер пружа високе перформансе, лако руковање рутама и једноставну интеграцију са базама података.

Фронтенд део веб апликације реализован је у React технологији, која омогућава израду динамичких и интерактивних корисничких интерфејса. За стилизовање сам користио Tailwind CSS, који знатно убрзава развој и омогућава прецизну контролу изгледа елемената уз минимално писање кода.

Мобилна апликација је направљена у React Native окружењу, што омогућава развој за обе платформе Android и iOS користећи исти код, чиме се постиже већа ефикасност и лакше одржавање пројекта. Али сам се одлучио да буде само за андроид, касније је могуће урадити и за iOS уз врло мало измена.

Као систем за управљање базом података користио сам PostgreSQL, релациону базу познату по стабилности, безбедности и подршци за комплексне упите.

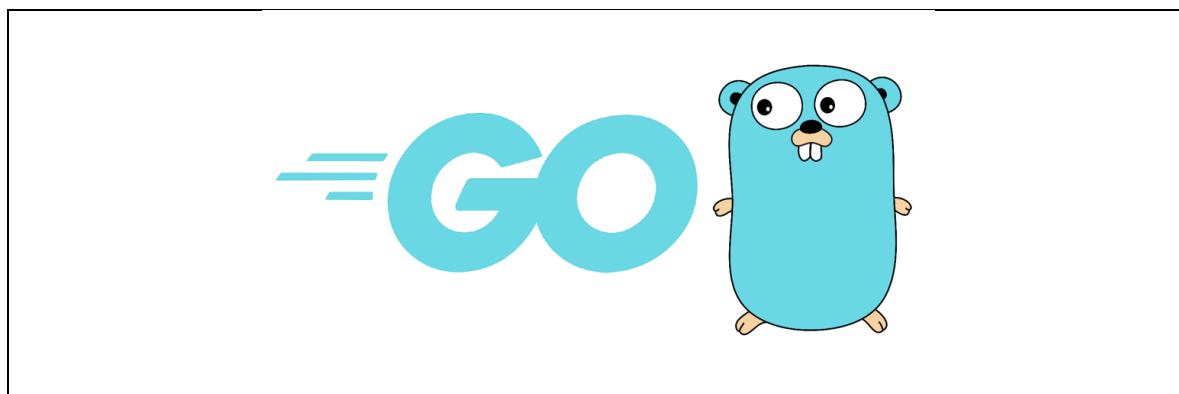
За тестирање API захтева и проверу исправности комуникације између клијентског и серверског дела апликације користио сам Postman, који омогућава детаљно праћење и анализу сваког захтева и одговора.

За брже и лакше покретање апликација користио сам Docker, који служи за контејнеризацију делова система. Он служи да сваки део ради засебно у свом изолованом окружењу.

3.1. Backend Go (Gin API framework)

Go (Golang) је програмски језик познат по високим перформансама, развила га је компанија Google за брз и ефикасан развој софтверских и системских апликација. Код који је написан се директно компајлира у машински језик и због тога има велику брзину извршавања. Дизајниран је тако да извршава паралелно велик број процеса, што је веома битно за апликације које имају пуно корисника и имају велик број истовремених захтева. Одлучио сам се за њега јер апликација коју правим за фризере би у реалном окружењу имала пуно корисника који би истовремено заказивали, а и фризери би константно користили своју мобилну апликацију за преглед термина, због тога је Go савршено решење за ову апликацију.

За серверски део апликације сам користио Gin, то је један од најбољих фрејмворкова за Go. Он омогућава брзо и једноставно прављење API сервера који обрађује HTTP захтеве. Предност је једноставност рутирања, односно путање се лако и брзо дефинишу. Поред тога он подржава и middleware што је веома битно за функције које служе за аутентификацију, логовање и проверу грешака. Комбинација Go и Gin-а је један од најбољих и најпоузданијих backend система.



Слика 1 – лого Go програмског језика

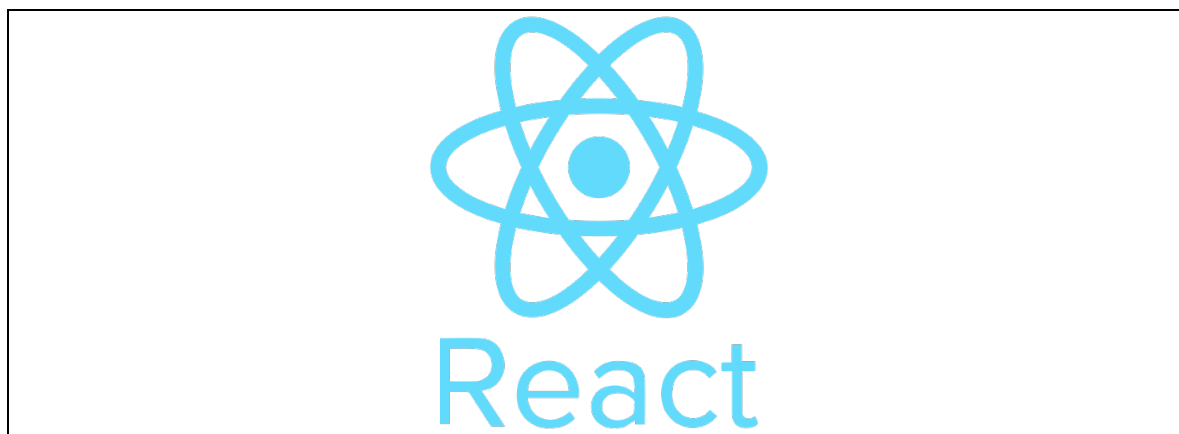
3.2. Web aplikacija React i Tailwind CSS

React је једна од најпопуларнијих библиотка за прављење веб сајтова, развијена је од стране Facebook компаније. Велика предност је брзина промена на екрану, зато што користи Virtual DOM. Уместо да се сваки пут страница учитава поново, React мења само делове који су се променили, због тога све ради брже и течније. Све се гради из различитих компоненти, то су мањи самостални делови који имају своју улогу у систему. Могу да се рециклирају и да се користе на више места у апликацији. То је веома корисно зато што спречава дуплирање кода и непотребно писање.

Он комуницира са backend ари-јем преко HTTP захтева. Компоненте саме извршавају захтеве и не мора цео сајт да се учитава поново, то помаже да не долази до непотребних захтева који могу да успоравају сервер.

За дизајнирање и стилизовање веб апликације сам користио Tailwind CSS, то је савремени CSS фрејмворк за брзо и лепо дизајнирање компоненти. Уместо класичног CSS-а он користи класе где се стилови директно додају у HTML елементе, на тај начин се дизајн креира много брже и не долази до непотребног понављања кода. Tailwind CSS

је веома лако прилагодљив и пружа могућност да се веб апликација дизајнира модерно и да интерфејс буде респонзиван што је веома битно за корисничко искуство.

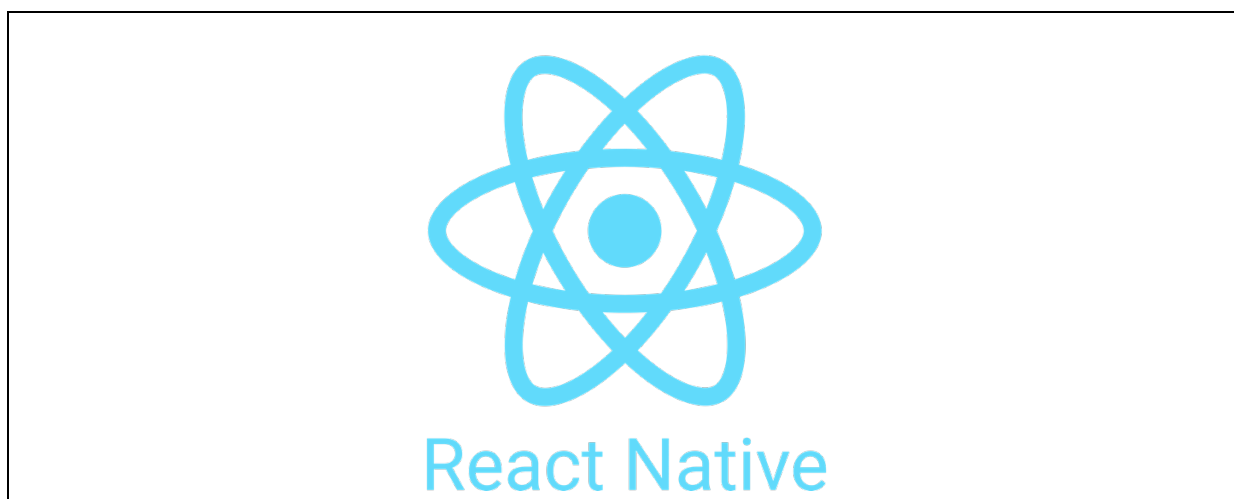


Слика 2 – лого React framework-a

3.3. Андроид апликација React Native

React Native је фрејмворк који се користи за развој мобилних апликација за Android и iOS оперативне системе коришћењем JavaScript и TypeScript програмског језика. Такође је развијен од стране компаније Facebook, са циљем да помогне програмерима да једним кодом направе апликацију за више оперативних система.

Велика предност је то што комбинује функционалности нативних апликација са једноставношћу развоја веб апликација. Такође се користе компоненте и због тога апликације изгледају глатко и интуитивно. Мобилна апликација је повезана са backend сервером путем API позива као и веб апликација. Ово је веома битно да би се користили исти подаци за обе апликације.



Слика 3 – лого React Native framework-a

3.4. Baza podataka PostgreSQL

За базу података сам одабрао PostgreSQL због тога што је веома моћан и стабилан систем за управљање релационим базама података, познат је по поузданости, сигурности и високим перформансама. Већ тридесет година се развија као open source пројекат и једно је од најпопуларнијих решења за складиштење података у модерним апликацијама.

Има подршку за напредне SQL функције, као и могућност за рад са великим количинама података што је веома битно за апликацију коју правим зато што после неког времена ако пуно салона користи апликацију може да дође до чувања великој броја података али то не би утицало на брзину и не би било губитака података. Одличан је за рад са JSON подацима што је веома корисно у комбинацији са API backendom.

У мом пројекту PostgreSQL служи као централно место за складиштење свих података везаних за кориснике, фризере, салоне и термине. Такође подржава индексирање, то су везе између табела и подржава ограничења, то гарантује да подаци у бази увек остану исправни. Ово је веома битно за апликацију која управља терминима да не би дошло до дуплирања термина или непостојећих података.



Слика 4 – лого PostgreSQL-а

3.5. Postman

За тестирање API-ја користио сам Postman, то је апликација која служи за комуникацију између клијента и сервера и веома је битна за тестирање функционалности позива са бекенда. У њему се могу слати HTTP захтеви као што су GET, POST, PUT, DELETE, односно могу се читати, додавати, изменити и обрисати подаци на серверу.

Могуће је чување колекције за тестирање, односно може се направити скуп захтева за све делове система и то олакшава тестирање и одржавање API-ја, јер се сви тестови могу покренути одједном.

Postman pruža uslugu da se u zahtevima koriste promenljive i to može biti veoma korisno za brzo testiranje. U toku pravљења бекенда Postman ми је веома помогао да брзо решим грешке које сам правео и да будем сигуран у свој код да ради.



Слика 5 – лого Postman-a

3.6. Docker

Docker је платформа која помаже при покретању апликације на различитим рачунарима. Она омогућава креирање, тестирање и покретање апликација у контејнерима. Контејнер је изоловани пакет који садржи све што апликација захтева за рад.

Користио сам Docker да ментор који ми помаже око пројекта може да брзо и лако покрене апликације на његовом рачунару. Веома је користан за комплексе системе када има више сервиса. У мом пројекту сам га користио за веб апликацију, бекенд, и базу података.



Слика 6 – лого Docker-a

4. Функционалност апликације

4.1. Функционалност веб апликације

4.1.1. Регистрација салона

Регистрација салона представља први корак у коришћењу веб апликације и намењена је власницима фризерских салона који желе да користе систем за заказивање термина и управљање својим салоном. Процес регистрације је подељен у два корака, први део се односи на унос података власника, а други на унос података о салону.

У првом кораку власник уноси своје личне податке који су потребни за креирање налога. Унос се врши преко једноставне форме у којој се налазе поља за име и презиме, адресу електронске поште, лозинку и број телефона. Након попуњавања свих поља, систем врши проверу да ли су подаци исправно унети и да ли већ постоји налог са истом адресом е-поште. У случају да је све исправно, власник прелази на други корак.

The screenshot shows the 'Kreirajte svoj salon' (Create your salon) registration page. At the top, there is a navigation bar with the BarberBook logo and links for 'Početna', 'Registruj salon', and 'Prijava'. The main heading is 'Kreirajte svoj salon' with a subtext 'Digitalizujte svoj frizerski salon za manje od 5 minuta'. Below this is a progress indicator with two steps, where the first step is active. The section is titled 'Podaci o vlasniku' (Owner's data) with the instruction 'Unesite vaše osnovne podatke za pristup sistemu'. The form contains three input fields: 'Email adresa *' with the placeholder 'owner@example.com', 'Lozinka *' with the placeholder 'Minimum 6 karaktera', and 'Ime i prezime *' with the placeholder 'Marko Petrović'. Below these is a 'Telefon *' field with the placeholder '+381 xx xxx xxx'. At the bottom of the form, there is a progress indicator showing 'Korak 1 od 2' and a 'Sledeći korak' button.

Слика 7 – приказ првог дела регистрације салона

Други корак регистрације је унос података о салону. У овој форми власник уноси назив салона, адресу, број телефона и опис салона. Сви подаци се уносе једноставно, а систем поново врши проверу унетих вредности како би се спречиле грешке.

Након што су сви подаци попуњени, власник кликом на дугме региструј салон завршава процес регистрације.

The screenshot shows a web browser window with the BarberBook logo in the top left. The navigation bar includes links for 'Početna', 'Registruj salon', and 'Prijava'. The main heading is 'Kreirajte svoj salon' with a subtext 'Digitalizujte svoj frizerski salon za manje od 5 minuta'. A progress indicator shows two steps, with the second step 'Podaci o salonu' being active. Below this, the form is titled 'Osnovne informacije o vašem salonu'. It contains four input fields: 'Naziv salona *' with the value 'Salon "Stil"', 'Telefon salona *' with the value '+381 xx xxx xxxxx', 'Adresa *' with the value 'Knez Mihailova 15, Beog', and 'Valuta *' with a dropdown menu showing 'Izaberite valutu'. At the bottom of the form, there are three buttons: 'Nazad', 'Korak 2 od 2', and 'Kreiraj salon'. A footer note states 'Besplatno kreiranje • Bez mesečnih troškova'.

Слика 8 – приказ другог дела регистрације салона

Систем аутоматски чува све податке у бази. Податке о власнику чува у посебној табели у којој се налазе основни подаци као што су име, е-пошта и лозинка, док се подаци о салону чувају у другој табели која садржи назив, адресу, контакт и ид власника.

Након успешне регистрације, власнику се приказује порука о успешном креирању налога и он добија могућност да се пријави у систем. Пријавом на налог, власник стиче приступ административном панелу свог салона, где може да додаје фризеру, дефинише услуге, радно време и врши остале измене везане за функционисање салона.

4.1.2. Админ панел за власника салона

Након успешно извршене регистрације, власник салона се може пријавити у систем и приступити свом административном панелу. Админ панел представља централно место у апликацији преко којег власник управља својим салоном и фризерима. Циљ овог дела апликације је да власнику омогући једноставно руковање подацима, уз прегледан и интуитиван интерфејс.

Главна функција коју власник има је додавање фризера. У овом делу панела власник уноси податке за сваког фризера који ради у салону, као што су име и презиме, адреса е-поште, број телефона. Након уноса, систем аутоматски креира налог за фризера и фризер може да се пријави на мобилну апликацију. На овај начин сваки фризер добија сопствени приступ преко којег може да прати своје заказане термине и мења свој распоред.

The screenshot displays the BarberBook admin panel for a salon owner. The top navigation bar includes the BarberBook logo, a 'Početna' (Home) link, a 'Dashboard' link, and a 'Odjavi se' (Logout) button. The main content area is divided into three primary sections:

- KONTROLNA TABLA (Dashboard):** Titled 'Salon stil', it provides a summary of the salon's status. It includes a button to 'Otvori javnu stranicu' (Open public page) and a note to 'Pregledajte rezultate salona, upravljajte timom i osvežite podatke sa istog mesta.' (View salon results, manage the team, and refresh data from here). Below this are three cards: 'ADRESA' (Karadjordjeva 86), 'KONTAKT' (069697906), and 'TIM' (Nema frizera).
- Podaci o salonu (Salon Data):** A section for updating basic salon information. It includes a 'Uredi podatke' (Edit data) button and a form with fields for 'NAZIV' (Salon stil), 'TELEFON' (069697906), 'ADRESA' (Karadjordjeva 86), and 'VALUTA' (RSD).
- Tim frizera (Barber Team):** A section for managing the team of barbers. It includes a 'Dodaj frizera' (Add barber) button and a large dashed box with a plus icon and the text 'Dodajte svog prvog frizera' (Add your first barber), with a note to 'Kreirajte налог i počnite da dodeljujete termine.' (Create a bill and start assigning appointments).

Слика 9 – приказ админ панела

4.1.3. Процес заказивања термина

Функционалност заказивања термина представља један од најважнијих делова веб апликације. Овај процес омогућава клијентима да на једноставан начин изаберу салон, фризера, услугу и слободан термин, а све се обрађује аутоматски преко бекенд система. Циљ овог дела апликације је да цео процес заказивања буде брз, интуитиван и без могућности грешке.

Први корак је избор фризера. Клијент на почетној страници веб апликације види листу доступних фризера унутар одабраног салона. Сваки фризер има приказано име. Корисник одабиром фризера прелази на следећи корак.

The screenshot shows a mobile app interface for the first step of a booking process. At the top, it says 'KORAK 1 OD 5'. Below that is the title 'Frizer' in a large, bold font. Under the title is the instruction 'Izaberite frizera koji vam najviše odgovara'. A progress bar with five segments is shown, with the first segment filled. Below the progress bar is a dark blue button with the text 'Lazar Frizer' on the left and 'Izabrano' on the right. At the bottom, there are two links: 'Prethodni korak' on the left and 'Sledeći korak' on the right.

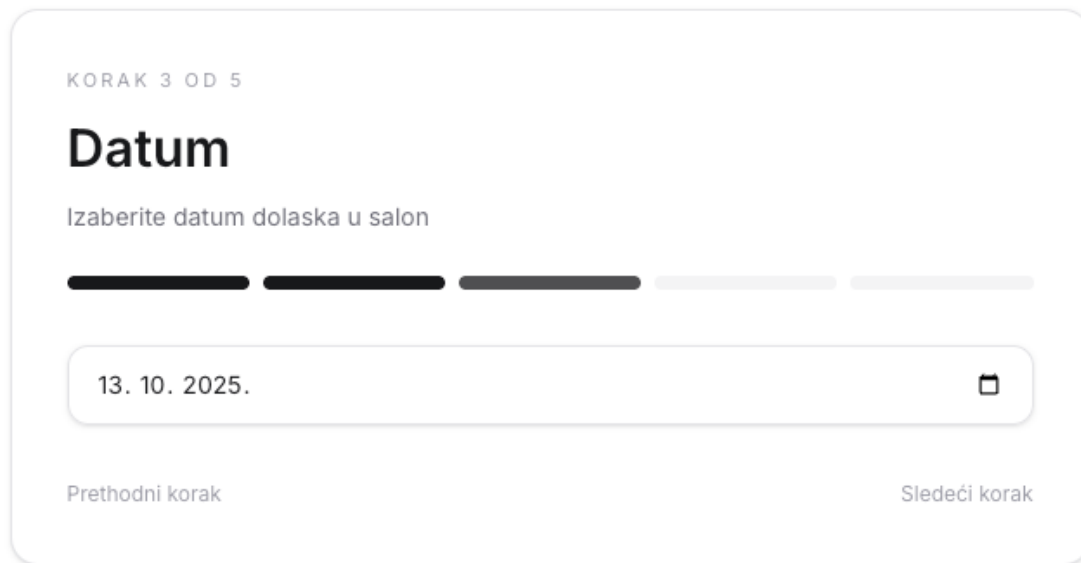
Слика 10 – приказ првог корака резервације термина

Други корак је избор услуге, у овом делу приказују се све услуге које изабрани фризер нуди. Уз сваку услугу приказано је трајање и цена. Клијент одабиром једне од понуђених услуга наставља процес.

The screenshot shows a mobile app interface for the second step of a booking process. At the top, it says 'KORAK 2 OD 5'. Below that is the title 'Usluga' in a large, bold font. Under the title is the instruction 'Odaberite uslugu koju želite da rezervišete'. A progress bar with five segments is shown, with the first two segments filled. Below the progress bar is a dark blue button with the text 'Fade' on the left, 'Trajanje 30 min' below it, and '1.000 RSD' on the right. At the bottom, there are two links: 'Prethodni korak' on the left and 'Sledeći korak' on the right.

Слика 11 – приказ другог корака резервације термина

Трећи корак је избор датума термина. На овом кораку клијенту се приказује календар са данима за заказивање. Клијент одабиром жељеног датума прелази на следећи корак.



KORAK 3 OD 5

Datum

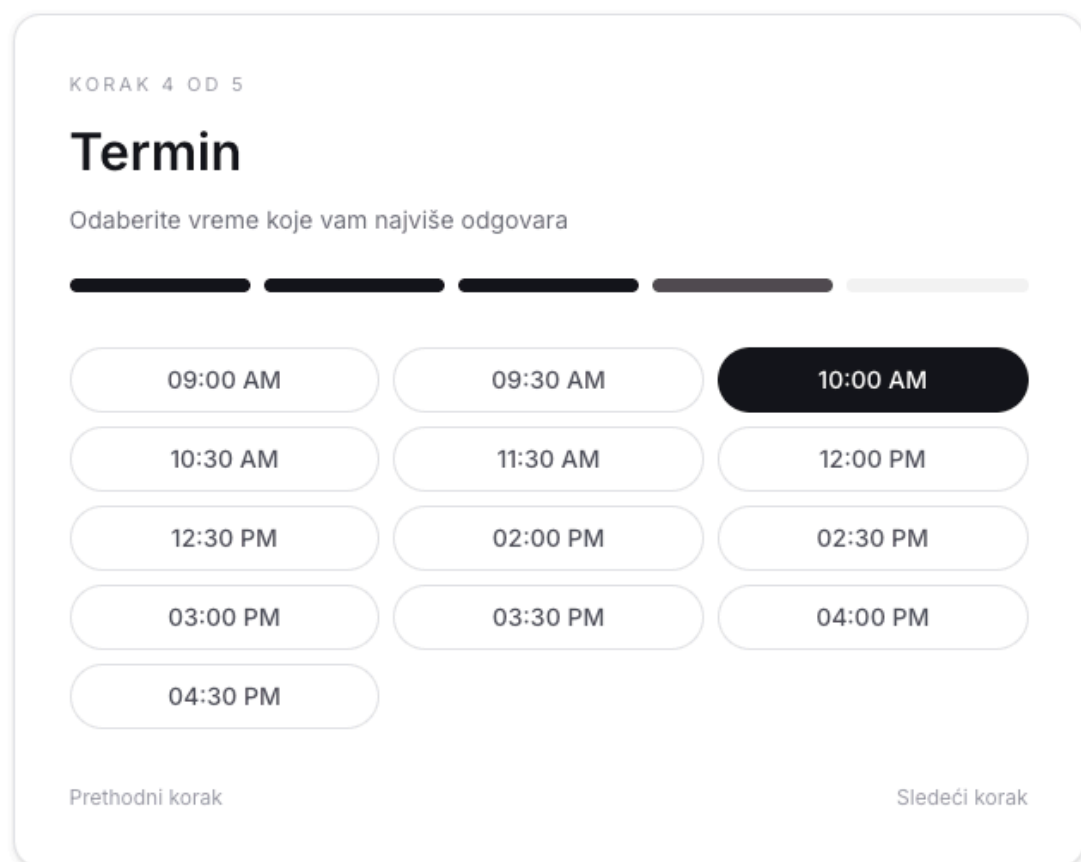
Izaberite datum dolaska u salon

13. 10. 2025.

Prethodni korak Sledeći korak

Слика 12 – приказ трећег корака резервације термина

Четврти корак је избор времена термина. Након што је изабран датум, систем приказује све доступне термина за тај дан, у складу са радним временом фризера и његовим паузама. Клијент бира време које му највише одговара.



KORAK 4 OD 5

Termin

Odaberite vreme koje vam najviše odgovara

09:00 AM	09:30 AM	10:00 AM
10:30 AM	11:30 AM	12:00 PM
12:30 PM	02:00 PM	02:30 PM
03:00 PM	03:30 PM	04:00 PM
04:30 PM		

Prethodni korak Sledeći korak

Слика 13 – приказ четвртог корака резервације термина

Пети корак је унос података клијента и потврда резервације. У овом кораку клијент уноси своје основне податке: име и презиме, број телефона и адресу електронске поште. Поред тога, постоји и поље за опционалну напомену, у коју клијент може унети додатне информације за фризера, као што су посебни захтеви или напомене у вези услуге.

Salon stil

Karadjordjeva 86

069697906

Europe/Belgrade • RSD

KORAK 5 OD 5

Podaci

Unesite kontakt podatke i potvrdite rezervaciju

PODACI O KLIJENTU

Ime i prezime *

Unesite ime i prezime

Telefon *

Unesite broj telefona

Email *

Unesite email adresu

Napomena (opciono)

Dodatne informacije za frizera

Frizer	Lazar
Usluga	Fade
Cena i trajanje	1.000 RSD • 30 min
Termin	13. 10. 2025. • 10:00 AM

Potvrdi rezervaciju

Prethodni korak

Sledeći korak

Слика 14 – приказ петог корака резервације термина

Backend проверава валидност података и креира нови запис у бази података у табели за заказане термине. Свака резервација се повезује са заказаним термином, фризером, услугом и временским интервалом. Након тога сервер шаље потврду да је термин успешно заказан, а клијенту се приказује порука о успешној резервацији.

На страни фризера, термин се одмах појављује у мобилној апликацији, где може да види све детаље резервације. Овим системом се постиже потпуна синхронизација између веб и мобилне апликације.

4.2. Функционалност мобилне апликације

Мобилна апликација помаже фризерима, омогућавајући им да управљају својим терминима, подешавају радно време, паузе, нерадне дане и да дефинишу услуге. Њен циљ је да поједностави организацију свакодневног рада у салону, обезбеди прегледност и ефикасност.

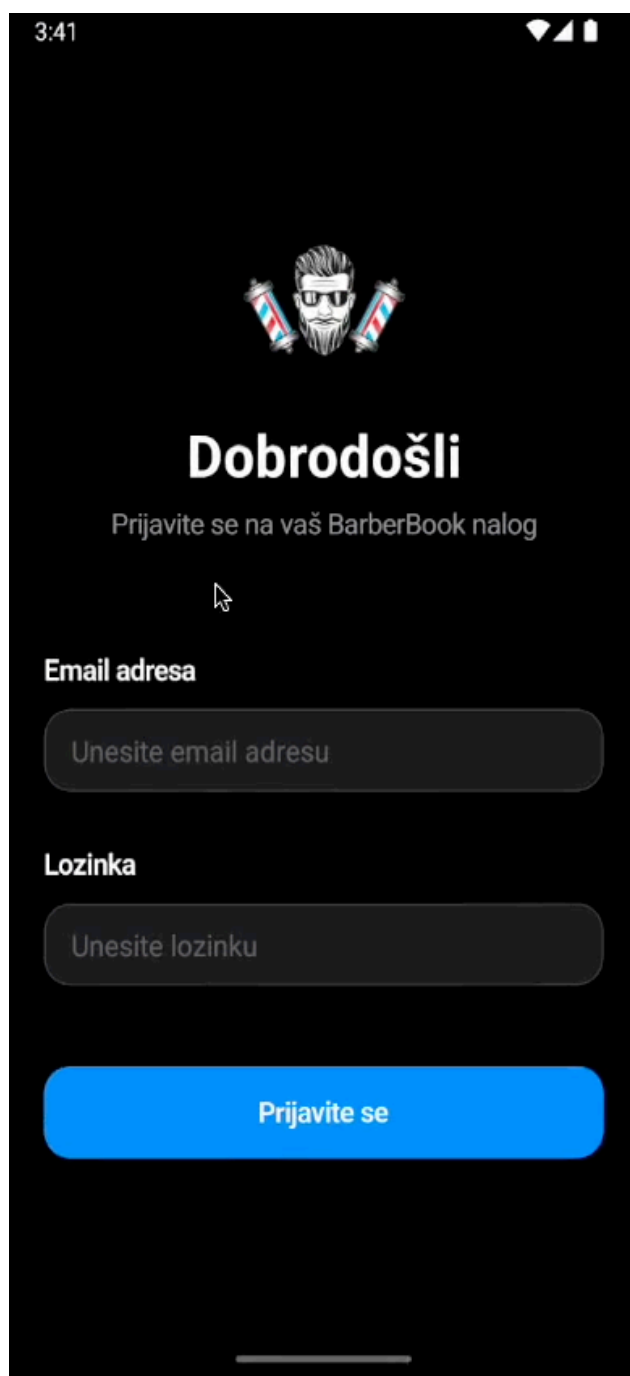
4.2.1. Пријава фризера на апликацију

По покретању мобилне апликације, први екран који се приказује фризеру је екран за пријављивање. Ова функционалност је неопходна како би се обезбедила сигурност података и спречио приступ неовлашћеним корисницима.

Фризер у поља на екрану уноси своју имејл адресу и лозинку, које су му додељене приликом регистрације салона у оквиру веб апликације. Подаци који се уносе на овом екрану шаљу се серверу, где backend систем проверава исправност података.

Уколико су унете информације тачне, сервер враћа позитиван одговор са аутентикационим токеном, након чега апликација чува тај токен локално и омогућава приступ главном делу система. У супротном, кориснику се приказује порука о грешци и он остаје на login страници док не унесе исправне податке.

Након успешне пријаве, фризер се преусмерава на главни екран апликације, где има могућност да управља својим радним временом, подешава услуге, дефинише паузе и нерадне дане, као и да прегледа све термине који су заказани преко веб апликације.



Слика 15 – приказ логин екрана за фризеру

4.2.2. Преглед термина

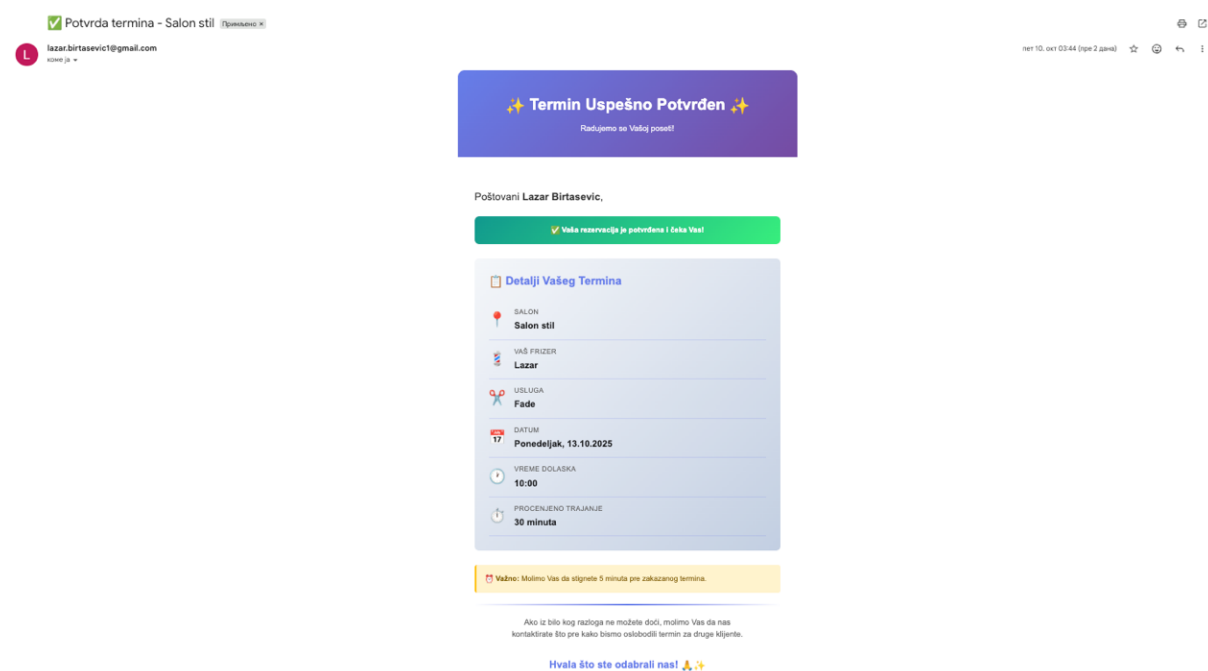
Након успешне пријаве на мобилну апликацију, фризеру се приказује главни екран који садржи преглед свих заказаних термина. Ова функционалност представља један од најважнијих делова апликације, јер омогућава фризеру да има потпун увид у свој распоред.



Слика 16 – приказ резервисаних термина у фризерској апликацији

На екрану се приказују сви термини који су клијенти резервисали преко веб апликације. Сваки термин садржи основне информације као што су име и презиме клијента, услуга коју је клијент одабрао, датум и време заказаног термина, контакт подаци и напомена која може да се напише а и не мора.

Када фризер потврди заказани термин, бекенд систем ажурира статус резервације у бази података и означава га као потврђен. Након тога мустерији на имејл адресу стиже маил са подацима о потврди резервације термина.



Слика 17 – приказ маила за потврђен термин

Уколико фризер откаже термин, мустерији ће на имејл адресу стићи маил у коме пише да је термин отказан.



Слика 18 – приказ маила за отказан термин

4.2.3. Подешавање радног времена, услуга и нерадних дана

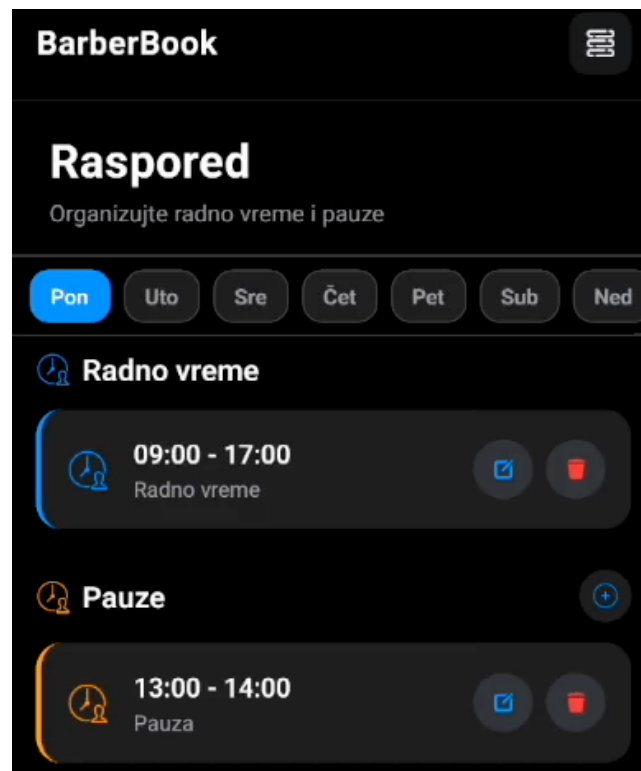
Једна од најзначајнијих функционалности мобилне апликације је могућност да фризер самостално подешава све податке који се односе на његов рад. Овај део система омогућава потпуну флексибилност у организацији пословног распореда, управљању услугама и дефинисању периода када фризер ради а када не ради.

Фризер има могућност да унесе назив услуге, цену и дужину трајања услуге и може подесити да услуга буде активна или неактивна. Такође може изменити податке услуге и да је обрише.



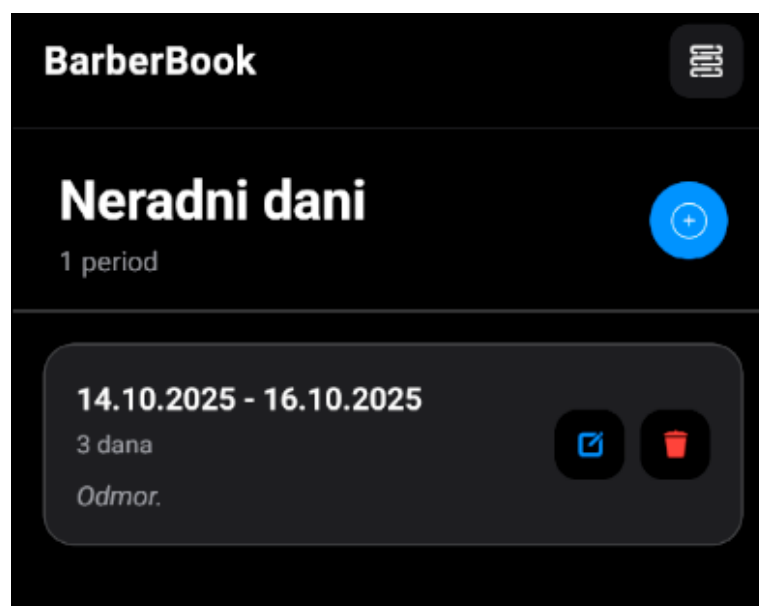
Слика 19 – приказ екрана за услуге у мобилној апликацији

Фризер у мобилној апликацији може подесити своје радно време за сваки дан посебно, и такође за сваки радни дан може убацити паузу између почетка и краја радног времена.



Слика 20 – приказ екрана за радно време и паузе

Мобилна апликација нуди могућност постављања нерадних дана, фризер треба да унесе почетни и крајњи датум да би се нерадни дани сачували. Такође потребно је да унесе разлог због којег не ради у том периоду, то ће се приказати на веб апликацији када мустерије одаберу датум који спада у нерадне дане фризера.



Слика 21 – приказ екрана за нерадне дане

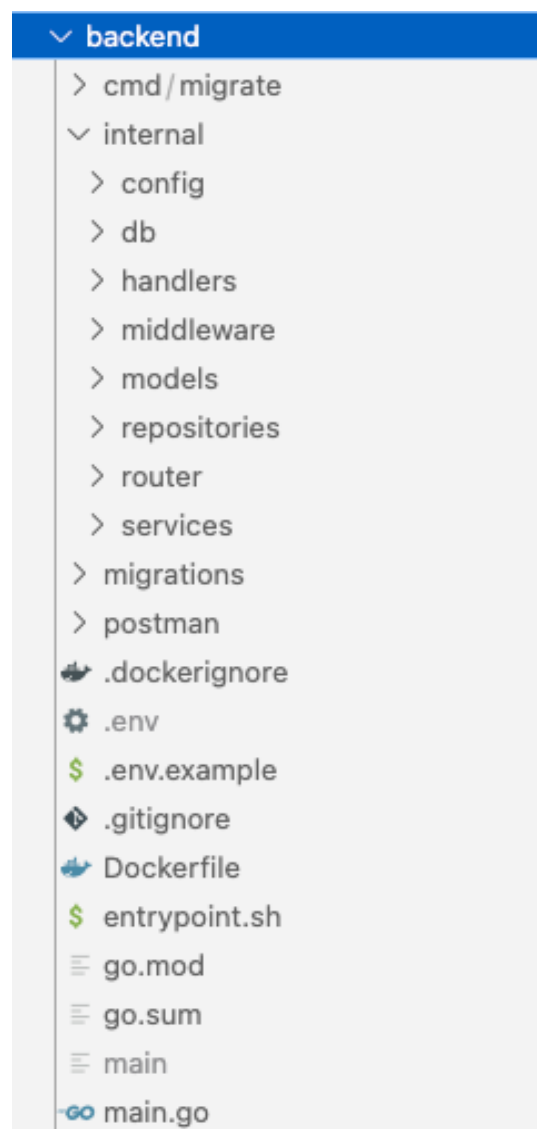
5. Опис имплементације пројекта

Апликација се заснива на клијент–сервер архитектури, која омогућава сигурну комуникацију између различитих делова система. Основу чине три главне целине: веб апликација, мобилна апликација и backend сервер, док се сви подаци централизовано чувају у PostgreSQL бази података.

5.1. Имплементација backend-а (Go + Gin)

5.1.1. Архитектура система

Пројекат је организован по принципу Clean Architecture, са јасно дефинисаним фолдерима.



Слика 22 – приказ стуктуре бекенда

Главни део је main.go јер се преко њега покреће и компајлира цела апликација. Ту се узимају варијабле из .env фајла, повезује се на базу података и затим покреће сервер на порту 8080.

```
package main

import (
    "log"

    "github.com/BirtasevicLazar/BarberBook/backend/internal/config"
    mydb "github.com/BirtasevicLazar/BarberBook/backend/internal/db"
    "github.com/BirtasevicLazar/BarberBook/backend/internal/router"
)

func main() {
    // učitavanje .env fajla
    cfg := config.Load()

    // Povezivanje na bazu
    pool, err := mydb.Connect(cfg.DB)
    if err != nil {
        log.Fatalf("failed to connect to database: %v", err)
    }
    defer pool.Close()

    r := router.New(pool, cfg)

    // pokreće server na :8080
    if err := r.Run(":8080"); err != nil {
        log.Fatal(err)
    }
}
```


5.1.2. Handler слој

Handler служи за обраду HTTP захтеva и враћање одговора. Сваки Handler прима захтев, позива одговарајући сервис и враћа одговор кориснику. Ово је показано на примеру за прављење салона.

```
func (h *SalonHandler) CreateSalon(w http.ResponseWriter, r *http.Request) {
    var req struct {
        Name      string `json:"name"`
        Address    string `json:"address"`
        PhoneNumber string `json:"phone_number"`
        Email      string `json:"email"`
    }

    if err := json.NewDecoder(r.Body).Decode(&req); err != nil {
        RespondWithError(w, http.StatusBadRequest, "Invalid request payload")
        return
    }

    ownerID := r.Context().Value("user_id").(int64)

    salon, err := h.salonService.CreateSalon(r.Context(), req.Name, req.Address,
        req.PhoneNumber, req.Email, ownerID)
    if err != nil {
        if strings.Contains(err.Error(), "already exists") {
            RespondWithError(w, http.StatusConflict, err.Error())
            return
        }
        RespondWithError(w, http.StatusInternalServerError, "Failed to create
salon")
        return
    }

    RespondWithJSON(w, http.StatusCreated, salon)
}
```

Функција прихвата JSON податке који су унети и проверава кроз сервис да ли већ постоји тај салон и да ли су подаци исправно унети, ако је све добро успешно се креира салон.

5.1.3. Service слој

Service слој садржи сву пословну логику апликације. Овај слој проверава податке, примењује пословна правила и управља рад са repository слојем. Као пример изабрао сам проверу доступности термина.

```
func (s *AvailabilityService) IsTimeSlotAvailable(ctx context.Context,
    barberID int64, salonID int64, startTime, endTime time.Time) (bool,
    error) {
    // Провера радног времена
    schedule, err := s.GetBarberSchedule(ctx, barberID, salonID,
    startTime.Weekday())
    if err != nil {
        return false, err
    }
    if !schedule.IsWorking {
        return false, nil
    }
    // Провера одсуства
    timeOffs, err := s.GetBarberTimeOffs(ctx, barberID, salonID, startTime,
    endTime)
    if err != nil {
        return false, err
    }
    if len(timeOffs) > 0 {
        return false, nil
    }
    // Провера постојећих заказивања
    appointments, err := s.GetBarberAppointments(ctx, barberID, startTime,
    endTime)
    if err != nil {
        return false, err
    }
    for _, apt := range appointments {
        if apt.Status == "confirmed" || apt.Status == "pending" {
            if timesOverlap(startTime, endTime, apt.StartTime, apt.EndTime) {
                return false, nil
            }
        }
    }
    return true, nil
}
```

Ова функција проверава да ли је термин слободан за фризера, прво позива GetBarberSchedule да узме радно време за тај дан у случају да фризер ради, затим проверава да ли је на одсуству или не и да ли већ постоје заказивања термина у том временском периоду. Ако ниједан од услова није прекршен враћа true и то значи да је доступан за заказивање.

5.1.4. Repository слој

Овај слој ради директно са базом података, ради CRUD операције и извршава SQL упите према бази. За пример сам узео репозиторијум за креирање салона.

```
type SalonRepository struct {
    db *sql.DB
}

func (r *SalonRepository) Create(ctx context.Context, salon *models.Salon) error {
    query := `
        INSERT INTO salons (name, address, phone_number, email, owner_id,
        created_at, updated_at)
        VALUES ($1, $2, $3, $4, $5, $6, $7)
        RETURNING id`

    err := r.db.QueryRowContext(
        ctx,
        query,
        salon.Name,
        salon.Address,
        salon.PhoneNumber,
        salon.Email,
        salon.OwnerID,
        salon.CreatedAt,
        salon.UpdatedAt,
    ).Scan(&salon.ID)

    return err
}
```

На почетку има конекцију на базу, након тога има функцију Create која прима податке о салону које треба да упише у базу. Онда се извршава SQL упит који уноси податке у табелу у бази података. Ако је све добро err ће бити nil а ако дође до грешке исписаће грешку.

5.1.5. Аутентификација и ауторизација

Систем користи JWT токене за аутентификацију корисника. При успешној пријави, корисник добија access token који се користи за даље захтеве.

```
func (s *AuthService) GenerateToken(userID int64, email, role string) (string, error) {
    claims := jwt.MapClaims{
        "user_id": userID,
        "email":   email,
        "role":    role,
        "exp":     time.Now().Add(time.Hour * 24 * 7).Unix(), // 7 дана
    }

    token := jwt.NewWithClaims(jwt.SigningMethodHS256, claims)
    return token.SignedString([]byte(s.config.JWTSecret))
}
```

Ова функција генерише токен који се користи за аутентификацију корисника и токен траје до 7 дана.

Middleware функција проверава валидност JWT токена и служи да заштити руте које могу да користе само улововани корисници. Ова функција враћа другу функцију, односно прима токен и ако је у реду враћа HTTP handler за неку руту.

```
func AuthMiddleware(jwtSecret string) func(http.Handler) http.Handler {
    return func(next http.Handler) http.Handler {
        return http.HandlerFunc(func(w http.ResponseWriter, r *http.Request) {
            authHeader := r.Header.Get("Authorization")
            if authHeader == "" {
                http.Error(w, "Authorization header required",
                    http.StatusUnauthorized)
                return
            }

            tokenString := strings.TrimPrefix(authHeader, "Bearer ")

            token, err := jwt.Parse(tokenString, func(token *jwt.Token)
                (interface{}, error) {
                    if _, ok := token.Method.(*jwt.SigningMethodHMAC); !ok {
                        return nil, fmt.Errorf("unexpected signing method")
                    }
                    return []byte(jwtSecret), nil
                })
            if err != nil || !token.Valid {
                http.Error(w, "Invalid token", http.StatusUnauthorized)
                return
            }

            claims := token.Claims.(jwt.MapClaims)
            ctx := context.WithValue(r.Context(), "user_id",
                int64(claims["user_id"].(float64)))
            ctx = context.WithValue(ctx, "role", claims["role"].(string))

            next.ServeHTTP(w, r.WithContext(ctx))
        })
    }
}
```

5.1.6. Рутирање и API endpoints

Ово су руте преко којих бекенд извршава неке функционалности, веб и мобилна апликација користе ове руте и шаљу или примају податке преко њих. Неке руте су јавне и доступне свима, док су неке приватне и захтевају аутентификацију.

Аутентификација:

POST /api/auth/register - Регистрација новог корисника

POST /api/auth/login - Пријава корисника

Јавни endpoint-и:

GET /api/public/salons - Листа свих салона

GET /api/public/salons/{salonID} - Детаљи салона

GET /api/public/salons/{salonID}/barbers - Листа фризера у салону

GET /api/public/salons/{salonID}/services - Листа услуга у салону

Салони (захтева аутентификацију):

POST /api/salons - Креирање новог салона

GET /api/salons - Листа салона корисника

GET /api/salons/{salonID} - Детаљи салона

PUT /api/salons/{salonID} - Ажурирање салона

DELETE /api/salons/{salonID} - Брисање салона

Заказивања:

POST /api/appointments - Креирање термина

GET /api/appointments - Листа термина

GET /api/appointments/{appointmentID} - Детаљи термина

PUT /api/appointments/{appointmentID} - Ажурирање термина

DELETE /api/appointments/{appointmentID} - Отказивање термина

5.1.7. Email обавештења

Када фризер потврди или откаже термин, кориснику који је заказао термин на имејл адресу стиже мејл са одговором у зависности од тога да ли је потврђено или отказано. На почетку кода је структура података који су потребни за повезивање на SMTP сервер да би могао да се пошаље мејл. Функција креира наслов мејла и текст поруке са детаљима заказаног термина. На крају позива другу функцију која шаље мејл клијенту.

```
type EmailService struct {
    smtpHost      string
    smtpPort      int
    smtpUsername  string
    smtpPassword  string
    emailFrom     string
}

func (s *EmailService) SendAppointmentConfirmation(appointment
*models.Appointment) error {
    subject := "Потврда заказивања"
    body := fmt.Sprintf(`
        Поштовани,

        Ваш термин је успешно заказан.

        Детаљи термина:
        - Датум и време: %s
        - Салон: %s
        - Фризер: %s
        - Услуга: %s

        Хвала што користите BarberBook!
    `, appointment.StartTime.Format("02.01.2006 15:04"),
        /* остали детаљи */)

    return s.SendEmail(appointment.ClientEmail, subject, body)
}

func (s *EmailService) SendEmail(to, subject, body string) error {
    auth := smtp.PlainAuth("", s.smtpUsername, s.smtpPassword, s.smtpHost)

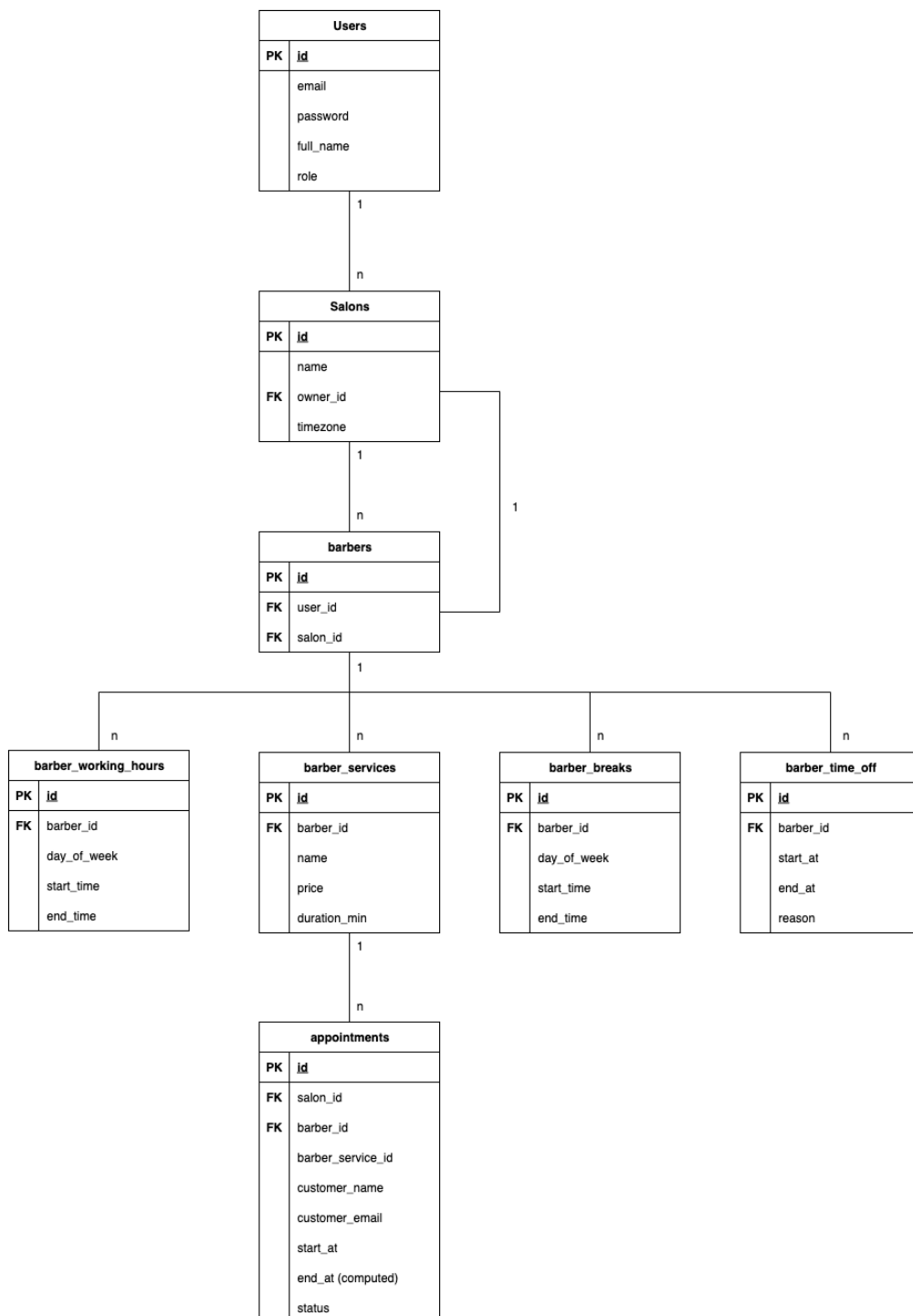
    msg := []byte(fmt.Sprintf("To: %s\r\nSubject: %s\r\n\r\n%s", to, subject,
body))

    addr := fmt.Sprintf("%s:%d", s.smtpHost, s.smtpPort)
    return smtp.SendMail(addr, auth, s.emailFrom, []string{to}, msg)
}
```

5.2. База података PostgreSQL

База података BarberBook апликације представља срце система за управљање салонима за шишање и заказивање термина. Систем користи PostgreSQL релациону базу података са напредним функционалностима као што су Enum типови, Check ограничења, Exclude constraint-и, тригери и индекси за оптималне перформансе.

5.2.1. Дијаграм релација (ER дијаграм)



Слика 23 – приказ дијаграма релација

5.2.2. Напредне функционалности базе података

Аутоматски тригер за израчунавање end_at

Систем користи PostgreSQL тригер за аутоматско израчунавање времена завршетка термина. Када се закаже нови термин или промени време почетка или трајање, тригер се активира. Функција израчунава крај термина тако што сабере време почетка и трајање у минутима. На крају се поље end_at аутоматски попуњава.

```
CREATE OR REPLACE FUNCTION set_appointment_end_at()
RETURNS trigger AS $$
BEGIN
    NEW.end_at := NEW.start_at + make_interval(mins => NEW.duration_min);
    RETURN NEW;
END;
$$ LANGUAGE plpgsql;

DROP TRIGGER IF EXISTS trg_set_appointment_end_at ON appointments;
CREATE TRIGGER trg_set_appointment_end_at
BEFORE INSERT OR UPDATE OF start_at, duration_min ON appointments
FOR EACH ROW
EXECUTE FUNCTION set_appointment_end_at();
```

5.2.3. Најважнија табела за заказане термине (табела appointments)

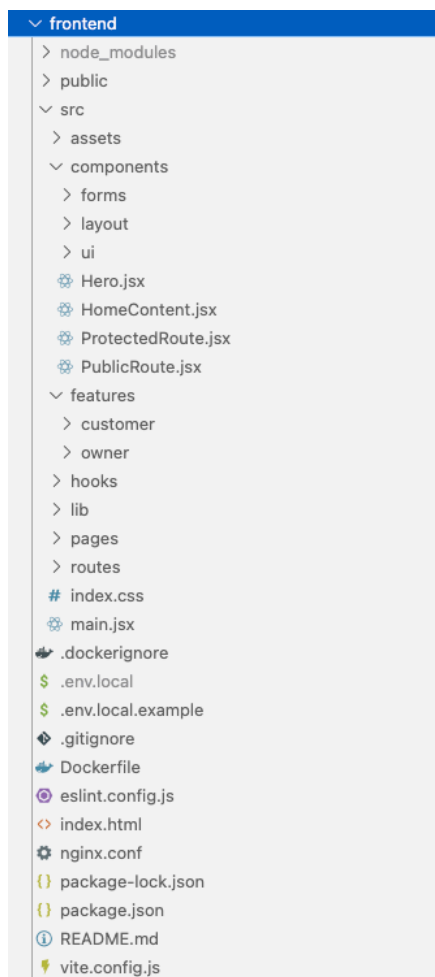
Ид користи UUID који генерише насумичан ид и осигурава да нема преклапања исто ид-ја. Има три спољна кључа везана за салон, фризера и услугу. Price користи CHECK (price >= 0) и тиме сигурно цена неће бити нула или мања. start_at и end_at користе TIMESTAMPTZ за временску зону. Status је ENUM тип и аутоматски се на почетку поставља на 'pending'.

```
CREATE TABLE IF NOT EXISTS appointments (
    id                UUID PRIMARY KEY DEFAULT gen_random_uuid(),
    salon_id          UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
    barber_id          UUID NOT NULL REFERENCES barbers(id) ON DELETE RESTRICT,
    barber_service_id  UUID NOT NULL,
    customer_name      TEXT NOT NULL,
    customer_phone     TEXT,
    customer_email     TEXT,
    price              NUMERIC(10,2) NOT NULL CHECK (price >= 0),
    duration_min       INT NOT NULL CHECK (duration_min > 0),
    start_at           TIMESTAMPTZ NOT NULL,
    end_at             TIMESTAMPTZ NOT NULL,
    status             appointment_status NOT NULL DEFAULT 'pending',
    notes              TEXT,
    created_at         TIMESTAMPTZ NOT NULL DEFAULT now(),
    CHECK (end_at > start_at)
);
```


5.3. Имплементација фронтенд веб апликације (React, Tailwind CSS)

5.3.1. Архитектура апликације

Пројекат сам јасно поделио по фолдерима у зависности од функционалности коју врше. Главни делови су компоненте, странице и lib.



Слика 24 – приказ структуре фронтенда

Главни део је main.jsx јер је то улазна тачка апликације и она служи за извршавање свега. createRoot служи за рендеровање садржаја и за боље перформансе, а document.getElementById('root') монтира апликацију на DOM елемент.

```
import { StrictMode } from 'react'
import { createRoot } from 'react-dom/client'
import './index.css'
import AppRouter from './routes/AppRouter.jsx'

createRoot(document.getElementById('root')).render(
  <StrictMode>
    <AppRouter />
  </StrictMode>,
)
```

5.3.2. Рутирање и навигација

Руте апликације су називи рута или адреса страница које се приказују.

/ - ово представља почетну страницу и то је основна рута на коју се прво улази или на коју се редиректује када се укуца погрешна рута.

/register-salon - је јавна рута за страницу која у себи има форму за регистравање салона, ако је корисник већ пријављен редиректује се на dashboard.

/owner/login – ово је јавна рута за страницу на којој се корисници пријављују на dashboard, ако је корисник већ пријављен одмах се редиректује.

/owner/dashboard – је заштићена рута која захтева аутентификацију и на њу се могу пријавити само корисници који су регистровани.

/s/:salonId – ово је јавна страница салона која у себи има ид салона и на којој клијенти резервишу термине.

- Catch-all – служи за непостојеће руте и редиректује на почетну.

5.3.3. API комуникација

API utility библиотека lib/api.js:

На почетку узима вредност урл-а из .env фајла и њега користи као базу за API руте. Функција buildUrl служи да споји базу са path-ом, а функција api ради тако што поставља header-е, дефинише методе које ће се користити, дефинише тело захтева и токен за аутентификацију.

```
const BASE_URL = import.meta.env.VITE_API_BASE_URL;

function buildUrl(path) {
  if (!BASE_URL) throw new Error('VITE_API_BASE_URL није постављен');
  const p = path.startsWith('/') ? path : `/${path}`;
  return `${BASE_URL}${p}`;
}

export async function api(path, { method = 'GET', headers = {}, body, token } = {}) {
  const isJSON = body && typeof body === 'object' && !(body instanceof FormData);
  const finalHeaders = {
    ...(isJSON ? { 'Content-Type': 'application/json' } : {}),
    ...headers,
  };
  if (token) finalHeaders['Authorization'] = `Bearer ${token}`;

  const res = await fetch(buildUrl(path), {
    method,
    headers: finalHeaders,
    body: isJSON ? JSON.stringify(body) : body,
  });
```

```

    credentials: 'include',
    mode: 'cors',
  });

  const text = await res.text();
  let data;
  try { data = text ? JSON.parse(text) : null; } catch { data = text; }

  if (!res.ok) {
    const err = new Error(data?.error?.message || data?.message ||
res.statusText);
    err.status = res.status;
    err.code = data?.error?.code;
    err.data = data;
    throw err;
  }

  return data;
}

```

5.3.4. Странице апликације

HomePage - Почетна страница

```

import Navbar from '../components/layout/Navbar';
import Hero from '../components/Hero';
import HomeContent from '../components/HomeContent';
import Footer from '../components/layout/Footer';

export default function HomePage() {
  return (
    <div className="min-h-screen">
      <Navbar />
      <Hero />
      <HomeContent />
      <Footer />
    </div>
  );
}

```

Почетна страница представља уводни део веб апликације и служи као први контакт корисника са системом. Компонента HomePage је једноставно структурирана и састоји се из четири основна дела: навигационог менија (Navbar), уводне секције (Hero), главног садржаја (HomeContent) и подножја (Footer).

Свака од ових компоненти има своју посебну улогу. Navbar омогућава лаку навигацију кроз апликацију, Hero визуелно представља први поглед на веб апликацију, HomeContent пружа детаљније информације о услугама и функционалностима, док Footer садржи основне контакт податке и додатне линкове.

OwnerLoginPage - Пријава власника

Функционалност пријаве власника салона направљена је у оквиру компоненте OwnerLoginPage. Ова компонента представља почетну страницу за све власнике салона који имају свој налог у систему и омогућава приступ њиховом административном панелу.

```
export default function OwnerLoginPage() {
  const [email, setEmail] = useState('');
  const [password, setPassword] = useState('');
  const [loading, setLoading] = useState(false);
  const [error, setError] = useState('');
  const navigate = useNavigate();

  async function onSubmit(e) {
    e.preventDefault();
    setError('');
    setLoading(true);

    try {
      const { access_token } = await api('/auth/owner/login', {
        method: 'POST',
        body: {
          email: email.trim().toLowerCase(),
          password
        }
      });

      setToken(access_token);

      try {
        const salon = await api('/owner/me/salon', { token: access_token });
        if (salon?.id) localStorage.setItem('salon_id', salon.id);
      } catch { /* ignore if not found */ }

      navigate('/owner/dashboard');
    } catch (e) {
      setError(e.message || 'Неисправни подаци за пријаву');
    } finally {
      setLoading(false);
    }
  }

  return (
    <Layout>
      <div className="min-h-screen bg-white flex flex-col">
        <div className="flex-1 flex items-center justify-center px-4 py-8">
          <div className="w-full max-w-sm mx-auto -mt-16">
            <div className="text-center mb-8 sm:mb-12">
              <h1 className="text-2xl sm:text-3xl md:text-4xl font-light text-gray-900 mb-3 sm:mb-4 tracking-tight">
                Пријава
              </h1>
              <p className="text-sm sm:text-base text-gray-600 font-light">
                Приступите вашем salon dashboard-y
              </p>
            </div>

            <form onSubmit={onSubmit} className="space-y-6 sm:space-y-8">
              <Input
                label="Email адреса"
                type="email"
                required
                value={email}
              >
```

```

        onChange={e=>setEmail(e.target.value)}
        placeholder="owner@salon.com"
      />

      <Input
        label="Лозинка"
        type="password"
        required
        value={password}
        onChange={e=>setPassword(e.target.value)}
        placeholder="....."
      />

      {error && (
        <div className="p-4 sm:p-6 bg-red-50 border border-red-200
rounded-xl">
          {error}
        </div>
      )}

      <Button type="submit" disabled={loading} className="w-full">
        {loading ? 'Пријављивање...' : 'Пријави се'}
      </Button>
    </form>
  </div>
</div>
</div>
</Layout>
);
}

```

Након што корисник унесе своје податке за пријаву, компонента користи React hook-ове (useState и useNavigate) за управљање стањем форме и навигацију кроз апликацију. Подаци који се уносе су имејл адреса и лозинка, при чему се вредности снимају у стање компоненти преко функција setEmail и setPassword.

Када се кликне дугме Пријави се, активира се функција onSubmit(). Она шаље захтев серверу на рути /auth/owner/login при чему се шаљу унети подаци. Ако сервер врати позитиван одговор из JSON-а се издваја токен за аутентификацију и он се чува локално. Затим се преузимају подаци о салону са руте /owner/me/salon и учитавају се. Када се све то извршило преусмерава се на страницу /owner/dashboard.

5.3.5. Feature компоненте

BookingWidget - Widget за заказивање

Ово је најкомплекснија компонента у апликацији. Implementира multi-step booking процес. Форма је урађена кроз 5 корака и има валидацију на сваком кораку, проверава све унете податке и када су спреми позива се функција book.

```
export default function BookingWidget({ salonId }) {
  const [barbers, setBarbers] = useState([]);
  const [services, setServices] = useState([]);
  const [selectedBarber, setSelectedBarber] = useState(null);
  const [selectedService, setSelectedService] = useState(null);
  const [date, setDate] = useState(() => new Date().toISOString().slice(0, 10));
  const [slots, setSlots] = useState([]);
  const [selectedSlot, setSelectedSlot] = useState(null);
  const [loadingSlots, setLoadingSlots] = useState(false);
  const [error, setError] = useState('');
  const [booking, setBooking] = useState(false);
  const [customer, setCustomer] = useState({ name: '', phone: '', email: '',
notes: '' });
  const [success, setSuccess] = useState(null);
  const [step, setStep] = useState(1);

  const steps = [
    { id: 1, name: 'Фризер', description: 'Изаберите фризера' },
    { id: 2, name: 'Услуга', description: 'Одаберите услугу' },
    { id: 3, name: 'Датум', description: 'Изаберите датум' },
    { id: 4, name: 'Термин', description: 'Одаберите време' },
    { id: 5, name: 'Подаци', description: 'Унесите контакт податке' },
  ];
};
```

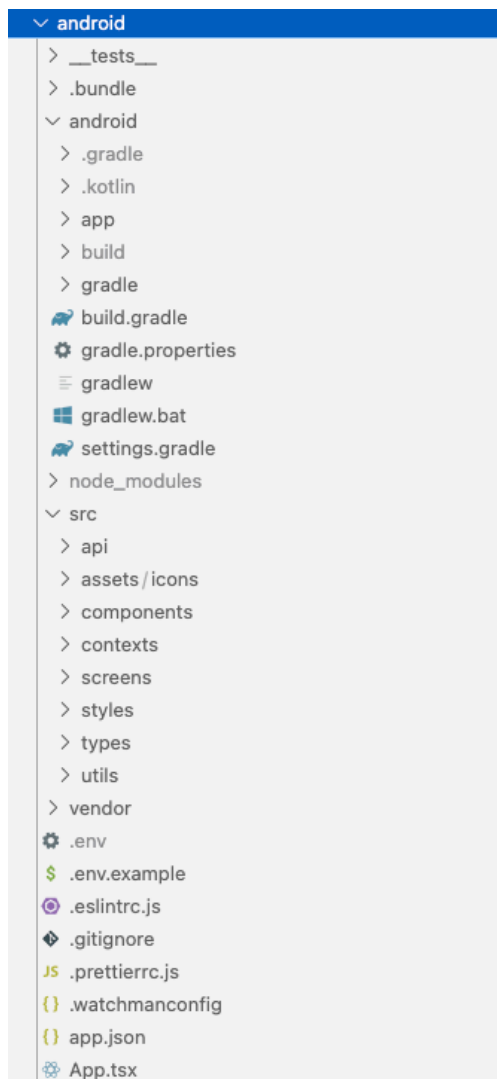
Функција book шаље захтев серверу на руту /public/appointments. Ако су сви послати подаци валидни, добија се одговор за успешно заказивање термина.

```
async function book(startAt) {
  setBooking(true);
  try {
    const res = await api('/public/appointments', {
      method: 'POST',
      body: {
        salon_id: salonId,
        barber_id: selectedBarber,
        barber_service_id: selectedService,
        customer_name: customer.name.trim(),
        customer_phone: customer.phone.trim(),
        customer_email: customer.email.trim(),
        start_at: startAt,
        notes: customer.notes || undefined,
      },
    });
    setSuccess(res);
  } catch (e) {
    setError(e.message);
  } finally {
    setBooking(false);
  }
}
```

5.4. Имплементација мобилне апликације (React Native)

Android апликација представља мобилну апликацију развијену у React Native framework-у, намењена је искључиво за фризере који управљају својим услугама, радним временом, одсуствима и терминима. Апликација обезбеђује брз и интуитиван приступ свим кључним функционалностима директно са њиховог мобилног уређаја.

5.4.1. Архитектура апликације



Слика 25 – приказ структуре мобилне апликације

Структура апликације је подељена по фолдерима који се налазе у /src. Најбитнији фолдери су за компоненте, екране и апи.

Главни део је App.jsx јер се одатле покреће цела апликација, она користи све компоненте и екране и рендерује их у апликацију.

5.4.2. Аутентификација

AuthContext обезбеђује глобално стање аутентификације. Интерфејс AuthState чува токен, тип токена и да ли је апликација тренутно у процесу пријаве или одјаве. Интерфејс AuthContextValue додаје на то функције за пријаву и одјаву корисника.

```
interface AuthState {
  token: string | null;
  tokenType: string;
  isLoading: boolean;
}

interface AuthContextValue extends AuthState {
  login: (email: string, password: string) => Promise<void>;
  logout: () => Promise<void>;
  credentials: { token: string; tokenType: string } | null;
}
```

Имплементација login функције, ово је функција за обраду пријаве корисника у мобилну апликацију, прво уклања празна места и претвара имеј у мала слова да не би дошло до грешке. Након тога шаље POST захтев серверу ка API-ју /auth/barber/login са унетим подацима за пријаву. Када се добије одговор назад узима се токен и тип токена и то се сачува у меморију. И на крају мења токен са новим.

```
const login = useCallback(async (email: string, password: string) => {
  const trimmedEmail = email.trim().toLowerCase();

  // API позив
  const res = await request<LoginResponse>('/auth/barber/login', {
    method: 'POST',
    body: {
      email: trimmedEmail,
      password,
    },
  });

  const token = res.access_token;
  const tokenType = res.token_type ?? 'Bearer';

  // Чување у AsyncStorage
  await AsyncStorage.multiSet([
    [STORAGE_TOKEN_KEY, token],
    [STORAGE_TOKEN_TYPE_KEY, tokenType],
  ]);

  // Ажурирање стања
  setState({ token, tokenType, isLoading: false });
}, []);
```


5.4.3. API слој

Ово је код HTTP клијента који служи за слање API захтева, на врху кода се узима URL API-ја за бекенд сервер. Након тога функција прави URL позив, поставља GET методу као подразумевано ако није подешено, затим додаје JSON header-е ако тело захтева садржи објекат. Ако постоји токен онда аутоматски додаје Authorization header. Затим се користи fetch() да пошаље захтев серверу и када добије одговор претвара га у JSON. На крају ако је све добро враћа JSON резултат као тип T да би могао да се дефинише у TypeScript-у.

```
const BASE_HOST = API_BASE_URL || 'http://localhost:8080';
const API_BASE = `${BASE_HOST}/api/v1`;

export async function request<T = unknown>(
  path: string,
  options: RequestOptions = {}
): Promise<T> {
  const { method = 'GET', body, headers = {}, auth, signal } = options;

  const isJsonBody = body &&
    typeof body === 'object' &&
    !(body instanceof FormData);

  const finalHeaders: Record<string, string> = {
    ...(isJsonBody ? { 'Content-Type': 'application/json' } : {}),
    Accept: 'application/json',
    ...headers,
  };

  // Додавање Authorization header-a
  if (auth?.token) {
    finalHeaders.Authorization = `${auth.tokenType ?? 'Bearer'} ${auth.token}`;
  }

  const response = await fetch(resolveUrl(path), {
    method,
    headers: finalHeaders,
    body: isJsonBody ? JSON.stringify(body) : (body as any),
    signal,
  });

  const rawText = await response.text();
  let data: unknown = null;

  if (rawText) {
    try {
      data = JSON.parse(rawText);
    } catch {
      data = rawText;
    }
  }

  if (!response.ok) {
```

```

const message = typeof data === 'object' && data !== null
  ? ((data as any).error?.message ??
    (data as any).message ??
    response.statusText)
  : response.statusText;

throw new ApiError({
  status: response.status,
  code: (data as any)?.error?.code,
  message,
  details: data,
});
}

return data as T;
}

```

5.4.4. Екрани апликације

ServicesScreen - Управљање услугама.

Ово је структура података која представља модел услуге коју нуди фризер.

```

interface BarberService {
  id: string;
  barberId: string;
  salonId: string;
  name: string;
  price: number;
  currency: string;
  durationMin: number;
  active: boolean;
  createdAt: string;
  updatedAt: string;
}

```

Функције за рад са услугама у мобилној апликацији, користи се прављење услуге, измена и брисање. Све се шаље серверу преко HTTP клијента и оне омогућавају фризеру контролу услуга.

```

// CREATE
const handleCreateService = useCallback(async (values: CreateServicePayload) => {
  try {
    const created = await createBarberService(auth, values);
    setServices(prev => sortServices([created, ...prev]));
    setShowAddModal(false);
    setError(null);
  } catch (e) {
    throw e;
  }
}

```

```

    }
  }, [auth]));

// UPDATE
const handleUpdateService = useCallback(async (
  id: string,
  payload: UpdateServicePayload
) => {
  const updated = await updateBarberService(auth, id, payload);
  setServices(prev => sortServices(
    prev.map(item => (item.id === id ? updated : item))
  ));
  return updated;
}, [auth]);

// DELETE
const handleDeleteService = useCallback(async (id: string) => {
  await deleteBarberService(auth, id);
  setServices(prev => prev.filter(service => service.id !== id));
}, [auth]);

```

ScheduleScreen - Радно време

Ово је структура података која представља модел радног времена и пауза.

```

interface BarberWorkingHour {
  id: string;
  barberId: string;
  salonId: string;
  dayOfWeek: number; // 0=Sunday, 6=Saturday
  startTime: string; // ISO string "2000-01-01T09:00:00Z"
  endTime: string; // ISO string "2000-01-01T17:00:00Z"
  createdAt: string;
  updatedAt: string;
}

interface BarberBreak {
  id: string;
  barberId: string;
  salonId: string;
  dayOfWeek: number;
  startTime: string;
  endTime: string;
  createdAt: string;
}

```

Овај део кода служи за приказ и подешавање радног времена и пауза фризера за сваки дан у недељи. Низ DAY_NAMES садржи називе дана, а функција map() пролази кроз сваки дан и за њега приказује компоненту DayCard. За сваки дан се издвајају радни сати и паузе, који се приказују у одговарајућој картици. Компонента DayCard омогућава фризеру да ажурира радно време, дода или обрише паузу.

```
const DAY_NAMES = [
  'Nedelja', 'Ponedeljak', 'Utorak', 'Sreda',
  'Četvrtak', 'Petak', 'Subota'
];

{DAY_NAMES.map((dayName, dayIndex) => {
  const dayHours = workingHours.filter(wh => wh.dayOfWeek === dayIndex);
  const dayBreaks = breaks.filter(b => b.dayOfWeek === dayIndex);

  return (
    <DayCard
      key={dayIndex}
      dayName={dayName}
      dayIndex={dayIndex}
      workingHours={dayHours}
      breaks={dayBreaks}
      onUpdate={handleUpdateWorkingHours}
      onAddBreak={handleAddBreak}
      onDeleteBreak={handleDeleteBreak}
    />
  );
})}
```

AppointmentsScreen - Термини

Ово је структура података која представља модел заказаних термина.

```
interface Appointment {
  id: string;
  salonId: string;
  barberId: string;
  barberServiceId: string;
  customerName: string;
  customerPhone?: string;
  customerEmail?: string;
  price: number;
  durationMin: number;
  startAt: string; // ISO timestamp
  endAt: string; // ISO timestamp
  status: AppointmentStatus; // 'pending' | 'confirmed' | 'canceled'
  notes?: string;
  createdAt: string;
  updatedAt: string;
}
```

Ово је код који служи за генерисање временских слотова за радни дан фризера на екрану за преглед термина. На почетку се дефинише структура TimeSlot који представља интервал са свим потребним подацима. Након тога пролази се кроз радно време за тај дан и рачуна све временске слотове на основу дужине радног дана и временског интервала.

```
interface TimeSlot {
  time: string;      // "10:00"
  hour: number;      // 10
  minute: number;    // 0
  appointment?: Appointment;
}

// Генерисање временских слотова
const slots: TimeSlot[] = [];
const slotDuration = barberProfile.slotDurationMinutes; // нпр. 15 мин

todayWorkingHours.forEach(wh => {
  // Parse време из ISO формата
  const startMatch = wh.startTime.match(/T(\d{2}):(\d{2})/);
  const endMatch = wh.endTime.match(/T(\d{2}):(\d{2})/);

  const startHour = parseInt(startMatch[1], 10);
  const startMinute = parseInt(startMatch[2], 10);
  const endHour = parseInt(endMatch[1], 10);
  const endMinute = parseInt(endMatch[2], 10);

  // Претварање у минуте
  const startMinutes = startHour * 60 + startMinute;
  const endMinutes = endHour * 60 + endMinute;

  let currentMinutes = startMinutes;

  // Генерисање слотова
  while (currentMinutes + slotDuration <= endMinutes) {
    const hour = Math.floor(currentMinutes / 60);
    const minute = currentMinutes % 60;
    const timeString = `${hour.toString().padStart(2, '0')}:${minute.toString().padStart(2, '0')}`;

    // Проналажење термина који преклапа овај слот
    const appointment = appointments.find(apt => {
      const aptStart = new Date(apt.startAt);
      const aptEnd = new Date(apt.endAt);

      const slotTimeMin = hour * 60 + minute;
      const aptStartMin = aptStart.getHours() * 60 + aptStart.getMinutes();
      const aptEndMin = aptEnd.getHours() * 60 + aptEnd.getMinutes();

      // Слот је заузет ако је унутар термина
      return slotTimeMin >= aptStartMin && slotTimeMin < aptEndMin;
    });

    slots.push({
```

```

        time: timeString,
        hour,
        minute,
        appointment,
    });

    currentMinutes += slotDuration;
  }
});

setTimeSlots(slots);

```

Ово је ScrollView који рендерује и приказује временске слотове кроз редове један испод другог.

```

<ScrollView style={styles.timeline}>
  {timeSlots.map((slot, index) => (
    <View key={index} style={styles.timeSlotRow}>
      <Text style={styles.timeLabel}>{slot.time}</Text>

      {slot.appointment ? (
        <TouchableOpacity
          style={[
            styles.appointmentCard,
            slot.appointment.status === 'pending' && styles.appointmentPending,
            slot.appointment.status === 'confirmed' &&
styles.appointmentConfirmed,
          ]}
          onPress={() => setSelectedAppointment(slot.appointment)}
        >
          <Text style={styles.customerName}>
            {slot.appointment.customerName}
          </Text>
          <Text style={styles.serviceName}>
            {getServiceName(slot.appointment.barberServiceId)}
          </Text>
          <Text style={styles.appointmentDuration}>
            {slot.appointment.durationMin} min
          </Text>
        </TouchableOpacity>
      ) : (
        <TouchableOpacity
          style={styles.emptySlot}
          onPress={() => handleCreateAppointment(slot)}
        >
          <Text style={styles.emptySlotText}>Slobodno</Text>
        </TouchableOpacity>
      )}
    </View>
  ))}
</ScrollView>

```

Акције над терминима:

Функција за потврду термина `handleConfirm` прима ид заказаног термина, и позива API функцију `confirmAppointment`, након тога се поново учитавају термини.

Функција за потврду термина `handleCancel` показује `Alert` који прво поставља питање фризеру да ли је сигуран да жели да откаже термин, ако изабере да покреће се `cancelAppointment` и поново се учитавају термини.

За брисање термина се користи функција `handleDelete` која је слична као отказивање, исто приказује `Alert` и ако се кликне да позива се `deleteAppointment`.

```
// CONFIRM
const handleConfirm = async (appointmentId: string) => {
  setActionLoading(true);
  try {
    await confirmAppointment(credentials, appointmentId);
    await loadAppointments();
    setSelectedAppointment(null);
    Alert.alert('Uspeh', 'Termin je potvrđen');
  } catch (error: any) {
    Alert.alert('Greška', error.message);
  } finally {
    setActionLoading(false);
  }
};

// CANCEL
const handleCancel = async (appointmentId: string) => {
  Alert.alert(
    'Potvrdite',
    'Da li sigurno želite da otkazete ovaj termin?',
    [
      { text: 'Ne', style: 'cancel' },
      {
        text: 'Da',
        style: 'destructive',
        onPress: async () => {
          setActionLoading(true);
          try {
            await cancelAppointment(credentials, appointmentId);
            await loadAppointments();
            setSelectedAppointment(null);
          } catch (error: any) {
            Alert.alert('Greška', error.message);
          } finally {
            setActionLoading(false);
          }
        },
      },
    ],
  );
};
```

```

// DELETE
const handleDelete = async (appointmentId: string) => {
  Alert.alert(
    'Brisanje',
    'Da li sigurno želite da obrišete ovaj termin?',
    [
      { text: 'Ne', style: 'cancel' },
      {
        text: 'Obriši',
        style: 'destructive',
        onPress: async () => {
          setActionLoading(true);
          try {
            await deleteAppointment(credentials, appointmentId);
            await loadAppointments();
            setSelectedAppointment(null);
          } catch (error: any) {
            Alert.alert('Greška', error.message);
          } finally {
            setActionLoading(false);
          }
        },
      },
    ],
  );
};

```


5.5. Имплементација Docker-а

За Backend, Frontend и PostgreSQL базу сам користио Docker и Docker Compose за једноставно покретање тих компоненти апликације. То је веома битно због тога што комплетан пројекат вероватно не би радио када би хтели да га покренемо на другом рачунару.

Docker Compose архитектура

Главни део је дефинисање сервиса а то су база, backend, frontend. Volumes у делу за базу служи за меморију за PostgreSQL податке. Networks служи за комуникацију између контејнера и depends_on дефинише којим редоследом ће се покренути све (база → backend → frontend).

```
version: '3.8'

services:
  # PostgreSQL база података
  db:
    image: postgres:15-alpine
    container_name: barberbook-db
    environment:
      POSTGRES_USER: barberbook
      POSTGRES_PASSWORD: barberbook_password
      POSTGRES_DB: barberbook
    volumes:
      - postgres_data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    networks:
      - barberbook-network
    healthcheck:
      test: ["CMD-SHELL", "pg_isready -U barberbook"]
      interval: 10s
      timeout: 5s
      retries: 5

  # Backend (Go API)
  backend:
    build:
      context: ./backend
      dockerfile: Dockerfile
    container_name: barberbook-backend
    environment:
      DATABASE_URL:
        postgresql://barberbook:barberbook_password@db:5432/barberbook?sslmode=disable
      JWT_SECRET: your-secret-key-here
      SERVER_PORT: 8080
    ports:
      - "8080:8080"
    depends_on:
      db:
        condition: service_healthy
    networks:
```

```

    - barberbook-network
    restart: unless-stopped

# Frontend (React)
frontend:
  build:
    context: ./frontend
    dockerfile: Dockerfile
  container_name: barberbook-frontend
  environment:
    VITE_API_BASE_URL: http://localhost:8080
  ports:
    - "80:80"
  depends_on:
    - backend
  networks:
    - barberbook-network
  restart: unless-stopped

volumes:
  postgres_data:

networks:
  barberbook-network:
    driver: bridge

```

Backend Dockerfile

Дефинише се из два Stage-а, у првом се компајлира Go апликација, користе се верзије за Go, , копирају се потребни фајлови и на крају се генерише извршна main датотека. А у другом кораку се прави финални контејнер и покреће се.

```

# Stage 1: Build
FROM golang:1.21-alpine AS builder

WORKDIR /app

# Копирање dependency фајлова
COPY go.mod go.sum ./
RUN go mod download

# Копирање кода и build
COPY . .
RUN CGO_ENABLED=0 GOOS=linux go build -o main .

# Stage 2: Production
FROM alpine:latest

RUN apk --no-cache add ca-certificates

WORKDIR /root/

```

```
# Копирање извршне датотеке и миграција
COPY --from=builder /app/main .
COPY --from=builder /app/migrations ./migrations
COPY --from=builder /app/.env .

EXPOSE 8080

CMD ["/main"]
```

Frontend Dockerfile

Такође се извршава из два Stage-а, у првом се ради Clean install и након тога се покреће build апликације. У другом кораку се копирају конфигурације Nginx-а.

```
# Stage 1: Build
FROM node:18-alpine AS builder

WORKDIR /app

# Копирање package files
COPY package*.json ./
RUN npm ci

# Копирање кода и build
COPY . .
RUN npm run build

# Stage 2: Production (Nginx)
FROM nginx:alpine

# Копирање build-ованих фајлова
COPY --from=builder /app/dist /usr/share/nginx/html

# Копирање nginx конфигурације
COPY nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80

CMD ["nginx", "-g", "daemon off;"]
```

Ово је Bash скрипта која служи за аутоматско покретање целог система у Docker-у. Прво испишује поруку о покретању и након тога покреће помоћу команде `docker-compose up -d --build`. То покреће све контејнере и гради их ако је потребно. Након десет секунди проверава статус свих сервиса и испишује њихово стање.

```
#!/bin/bash

echo "🚀 Покретање BarberBook система..."

# Build и покретање контејнера
docker-compose up -d --build

# Чека да сервиси буду спремни
echo "⌚ Чекам да сервиси буду спремни..."
sleep 10

# Проверава статус
docker-compose ps

echo "✅ Систем је покренут!"
echo "📱 Frontend: http://localhost"
echo "🔧 Backend API: http://localhost:8080"
echo "🗄️ PostgreSQL: localhost:5432"
```

6. Закључак

Овај завршни рад представља детаљан приказ процеса израде софтверског решења осмишљеног са циљем да модернизује и поједностави рад фризерских салона кроз дигитализацију процеса заказивања термина. Основна идеја рада била је развој потпуно функционалног и повезаног система који обједињује три кључна дела: веб апликацију за власнике и клијенте, мобилну апликацију за фризере и backend сервис који управља комуникацијом и подацима.

Backend система израђен је у програмском језику Go, уз коришћење Gin framework-а за креирање ефикасних и сигурних REST API сервиса. За рад са подацима примењена је PostgreSQL база, која обезбеђује стабилност, интегритет и могућност напредне обраде информација.

Веб апликација је реализована у React технологији, што је омогућило брз и интуитиван кориснички интерфејс, док је React Native коришћен за развој мобилне апликације намењене фризерима, како би им се омогућио директан и једноставан приступ свим заказивањима. На овај начин, постигнута је потпуна повезаност између различитих корисничких улога у систему, клијената, фризера и власника салона.

Посебан акценат у раду стављен је на структуру и архитектуру backend дела, који је подељен на handler, service и repository слојеве. Ова организација кода омогућава лакше одржавање, једноставније тестирање и јасну поделу одговорности.

Цео систем је додатно унапређен применом Docker контејнеризације, чиме је омогућено брзо покретање и лако постављање система на било који сервер. Користећи Docker Compose, сви сервиси база података, backend и frontend се покрећу једном командом, што омогућава једноставну дистрибуцију и стабилност рада у различитим окружењима.

Реализацијом пројекта, повезана су знања из више области, веб и мобилног програмирања, дизајна база података, backend архитектуре и системске администрације. Кроз практичну примену свих тих знања створено је решење које није само академски пример, већ и потпуно функционалан систем који може бити основа за даљи развој.

У будућности, апликација се може проширити функцијама као што су онлајн плаћања, push нотификације и аналитика пословања, што би BarberBook додатно приближило комерцијалним системима за управљање терминским услугама.

Овај рад представља заокружену целину која потврђује способност самосталног планирања, развоја и имплементације сложеног софтверског решења, од иницијалне идеје до стабилног и применљивог производа.

7. Литература

- React. (2025). Званична документација за React библиотеку. [React](#)
- React Native. (2025). Званична документација за React Native радни оквир. [React Native](#)
- Go. (2025). Званична документација за Go програмски језик. [go.dev](#)
- Gin Web Framework. (2025). Документација за Gin, web радни оквир за Go. [Gin Web Framework](#)
- PostgreSQL. (2025). Званична документација за PostgreSQL базу података. [PostgreSQL](#)
- Docker. (2025). Званична документација за Docker платформу. [Docker Documentation](#)
- Tailwind CSS. (2025). Званична документација за Tailwind CSS радни оквир. [Tailwind CSS](#)
- MDN Web Docs. (2025). Ресурси за web програмере, укључујући JavaScript и TypeScript. [MDN Web Docs](#)