Академија струковних
студија Шумадија
Одсек Крагујевац

Завршни рад

ПРЕДМЕТ МОБИЛНЕ АПЛИКАЦИЈЕ

Ментор:
Проф др. Хрвоје Пушкарић

Студент:
Лазар Бирташевић 006/2022

Крагујевац 2025.

Академија струковних
студија Шумадија
Одсек Крагујевац

Završni rad

ПРЕДМЕТ МОБИЛНЕ АПЛИКАЦИЈЕ

Ментор:

Проф др. Хрвоје Пушкарић

Студент:

Лазар Бирташевић 006/2022

_____ _____

(потпис ментора) *(потпис студента)*

Крагујевац 2025.

Садржај

1. Увод

1.1. Мобилне апликације

1.1.1. Увод у мобилне апликације

Мобилне апликације представљају софтверске програме дизајниране за рад на телефонима, таблетима и другим уређајима. Њихов развој и примена постали су неизоставни део савременог живота, јер омогућавају корисницима да лако приступе услугама, информацијама и комуникацији било када и било где.

Развој мобилних технологија, као што су Android и iOS оперативни системи, омогућио је производњу различитих типова апликација – од игара и друштвених мрежа, до апликација за електронско пословање, образовање и здравље.

За компаније, мобилне апликације су постале средство за побољшање корисничког искуства, директан контакт с клијентима и унапређење услуга. С друге стране, корисници очекују да све што им је потребно буде доступно веома брзо, једноставно и по мери. Управо та очекивања утичу на стални развој мобилних технологија и нових решења.

1.1.2. Подела мобилних апликација

Постоје три основна типа мобилних апликација:

1. **Нативне апликације (Native apps)**
   Развијају се 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 или Objective-C.
   Предност им 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 (Web apps)**
   То су апликације којима се приступа преко веб прегледача. Не захтевају инсталацију и најчешће су написане коришћењем HTML-a, CSS-a и JavaScript-a.
   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 (Hybrid apps)**
   Комбинују карактеристике нативних и веб апликација. Развијају се једном, користећи оквире као што су **React Native**, **Flutter** или **Ionic**, а затим се пласирају 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 правио за мој завршни рад, коју сам развио у React Native.

1.1.3. Развој мобилних апликација

Процес развоја мобилних апликација обухвата више фаза:

- **Анализа захтева** – дефинисање функционалности и корисничких потреба.
- **Дизајн корисничког интерфејса (UI/UX)** – израда изгледа апликације.
- **Имплементација** – писање кода, интеграција са сервером и базом података.
- **Тестирање и оптимизација** – провера исправности рада апликације.
- **Објављивање и одржавање** – постављање на Google Play / App Store и редовно ажурирање.

Мобилне апликације данас се све чешће развијају помоћу **крос-платформских оквира**, који омогућавају да се исти код користи и за Android и за iOS системе, што убрзава развој и поједностављује одржавање.

1.1.4. Значај мобилних апликација у савременом пословању

Мобилне апликације данас играју важну улогу у комуникацији између компанија и њихових корисника. Оне омогућавају директнији и прилагођенији приступ сваком клијенту, уз могућност праћења потреба и понашања корисника ради унапређења услуга.

У услужним делатностима, попут фризерских салона, апликације омогућавају:

- аутоматизовано заказивање термина,
- праћење распореда,
- лакшу комуникацију између клијената и фризера,

Управо из ових разлога мобилне апликације су постале саставни део модерног пословања, а њихова примена у услужним делатностима, као што су фризерски салони, представља практичан пример дигиталне трансформације.

1.1.5. Закључак поглавља

Мобилне апликације данас имају централну улогу у животу људи и у пословању предузећа. Њихов технолошки напредак омогућио је да готово сваки пословни процес може бити оптимизован кроз мобилну технологију. Ово поглавље представља основу за даље објашњење и анализу мобилне апликације *BarberBook* коју сам правио, је пример примене савремених технологија у реалном систему управљања терминима и клијентима у фризерским салонима.

1.2. Коришћене технологије

За развој апликације применио сам савремене технологије које омогућавају креирање динамичних, интерактивних, стабилних web и мобилних апликација. Комбинација различитих технологија за frontend, backend и базе података омогућила је развој система који функционише на више платформи — web, Android и iOS. У наставку су описане кључне технологије које су примењене у овом раду:

- React
- Tailwind CSS
- Go (Golang)
- Gin framework
- PostgreSQL
- React Native
- Docker
- Postman
- Git и GitHub

1.2.1. React и Tailwind CSS

React је једна од најпопуларнијих JavaScript библиотека за развој динамичних корисничких интерфејса. Његова основна предност је компоненцијални приступ, који омогућава да се сложене web странице граде из мањих, независних делова односно компоненти. Свака компонента у оквиру апликације има сопствену логику и изглед, што омогућава лако одржавање и поновну употребу кода.

React користи концепт Virtual DOM-а, што омогућава брзо ажурирање само оних делова странице који су промењени, без потребе за поновним учитавањем целе странице. Овим се постижу високе перформансе и брзо корисничко искуство, што је посебно важно код апликација које често комуницирају са сервером, као што је систем за заказивање термина.

Уз React, за стиловање интерфејса коришћен је Tailwind CSS, модеран framework који омогућава брзо и ефикасно дизајнирање изгледа апликације. Tailwind CSS омогућава да се дизајн прилагођава различитим величинама екрана (responsive design) и обезбеђује леп визуелни идентитет целог система.

React и Tailwind CSS заједно омогућавају брз развој, лако одржавање и визуелно леп интерфејс, што значајно унапређује корисничко искуство.

1.2.2. Go (Golang) и Gin Framework

Go (Golang) је модерни програмски језик развијен од стране компаније Google, познат по својој брзини, стабилности и једноставности.
У пројекту који сам радио, Go је коришћен за развој backend дела апликације, односно за имплементацију REST API-ја који обрађује захтеве са web и мобилне апликације и управља базом података.

Backend апликација је организована у више слојева:

- **Handler слој** – прихвата HTTP захтеве и шаље одговоре,
- **Service слој** – садржи пословну логику апликације,
- **Repository слој** – директно комуницира са базом података.

За изградњу web сервера коришћен је Gin framework, који омогућава лако руковање рутама (routes), middleware и JSON одговорима. Go омогућава високе перформансе, јер је компајлирани језик и има уграђену подршку за конкурентност преко goroutines, што омогућава обраду више захтева истовремено без пада перформанси.

У оквиру backend-а примењен је и JWT (JSON Web Token) механизам за аутентификацију корисника, што обезбеђује сигуран приступ свим заштићеним ресурсима система.

1.2.3. PostgreSQL

За чување и управљање подацима коришћен је PostgreSQL, један од најмоћнијих система за управљање релационим базама података.
PostgreSQL је изабран због своје стабилности, брзине, подршке за сложене упите и потпуне компатибилности са Go библиотекама за базу (као што су `gorm` или `pgx`).

Моја база BarberBook_db садржи више повезаних табела које складиште податке о:

- корисницима,
- фризерима,
- терминима,
- услугама,
- оценама и рецензијама.

Сви односи између табела дефинисани су преко примарних и страних кључева.
PostgreSQL такође подржава миграције и верзионирање шеме базе, што омогућава лако ажурирање структуре података током развоја.

1.2.4. React Native

Мобилна апликација за фризере развијена је у React Native технологији.
React Native омогућава развој крос-платформских нативних апликација, што значи да исти код ради и на Android и на iOS уређајима.
Апликација је повезана са backend API-јем и омогућава фризерима преглед и управљање заказаним терминима у реалном времену.

Главне предности React Native-а:

- Један код за више платформи,
- Нативне перформансе,
- Једноставно тестирање и одржавање.

React Native користи исту логику као и web апликација у React-у, што значајно убрзава развој и обезбеђује визуелну и функционалну доследност између мобилне и web верзије система.

1.2.5. Docker

Да би цео систем био лако покретљив и независан од оперативног система, примењен је **Docker**.
Docker омогућава да сваки део апликације (frontend, backend и база података) ради у свом изолованом контејнеру, без потребе за ручним инсталацијама зависности.

Фајл `docker-compose.yml` дефинише све сервисе:

- `frontend` – React web апликација,
- `backend` – Go сервер,
- `db` – PostgreSQL база,
- (по жељи) `nginx` – reverse proxy за production окружење.

На овај начин се систем може покренути једноставном командом:

```
docker-compose up -d
```

Docker је посебно користан за демонстрацију рада пројекта, јер професор може лако да покрене комплетан систем без сложеног подешавања окружења.

1.2.6. Postman

За тестирање backend API-ја коришћен је алат **Postman**.
У оквиру пројекта постоји Postman колекција `BarberBook.postman_collection.json` која садржи све дефинисане HTTP захтеве за тестирање функционалности као што су:

- регистрација и пријава корисника,
- додавање и брисање термина,
- добијање листе салона и фризера.

Postman омогућава једноставно тестирање без потребе за frontend интерфејсом и обезбеђује проверу исправности свих API одговора.

1.2.7. Git и GitHub

За контролу верзија и сарадњу у развоју коришћен је **Git**, а цео пројекат је постављен на платформи **GitHub**:
https://github.com/BirtasevicLazar/BarberBook

Git омогућава:

- праћење измена кроз commit историју,
- креирање грана за различите функционалности,
- враћање на претходне верзије кода,
- лако дељење пројекта са ментором и другим сарадницима.

GitHub такође служи као јавна документација пројекта — садржи README датотеку, Docker конфигурацију и Postman колекцију, што олакшава увид у структуру и функционалност система.

1.2.8. Закључак

Комбинација ових технологија омогућила је развој савременог и скалабилног система за управљање фризерским терминима. React и React Native обезбеђују богато корисничко искуство, Go и PostgreSQL стабилност и перформансе, Docker једноставну инсталацију, а GitHub и Postman транспарентност и лакше тестирање. Све заједно чини апликацију пример модерне full-stack апликације у реалној примени.

2. О апликацији

2.1. Увод

Апликација коју сам правио представља модерно решење за дигитално управљање терминима у фризерским салонима. Развијена је као систем који омогућава лакшу комуникацију између клијената и фризера, са циљем да процес заказивања постане једноставан, брз и потпуно аутоматизован.

Апликација је реализована у виду веб апликације и мобилне апликације, које међусобно комуницирају преко заједничког backend API-ја написаног у Go (Golang) програмском језику. Сви подаци се чувају у PostgreSQL бази података, док је за визуелни приказ и интеракцију са корисницима коришћен React за веб и React Native за мобилну апликацију.

- **Веб апликација** је намењена **клијентима**, који путем једноставног интерфејса могу да:
  - прегледају расположиве фризере и услуге,
  - изаберу жељени термин и резервишу га,
  - прате статус свог заказивања и историју претходних термина.
- **Мобилна апликација** је намењена **фризерима**, који преко свог налога могу да:
  - виде све активне и предстојеће термине,
  - потврде или одбију заказивања,
  - мењају статус термина (нпр. завршен, отказан),
  - управљају распоредом и прегледају детаље о клијентима.
- **Backend API** обрађује све захтеве, комуницира са базом података и води рачуна о валидности података и сигурности комуникације.

У току развоја користио сам **Postman** за тестирање API захтева, као и **GitHub** за верзионисање кода.

Цео систем је контејнеризован помоћу **Docker-a**, што омогућава лако покретање и преносивост на различитим окружењима без потребе за додатним конфигурацијама.

Апликација **BarberBook** је осмишљена тако да буде **скалабилна**, што значи да се у будућности може проширити новим функцијама, као што су:

- систем обавештења,
- могућност онлајн плаћања,
- оцена и рецензија фризера,
- статистика о раду и анализе броја термина.

На овај начин, BarberBook не представља само алат за заказивање, већ потпуни систем за дигитално управљање салоном и унапређење услуга.

2.2. Циљ апликације

Основни циљ апликације је унапређење и дигитализација процеса заказивања термина у фризерским салонима. Традиционални начин рада путем бележница, телефонских позива и ручних записа често доводи до непотребних грешака, недоследности и губитка времена. BarberBook решава овај проблем увођењем једноставног и аутоматизованог система који омогућава клијентима и фризерима потпуну контролу над заказивањем.

Циљ апликације је да:

1. **Омогући клијентима** да у неколико корака закажу термин код жељеног фризера, без потребе за позивима или чекањем на одговор.
2. **Омогући фризерима** да преко мобилне апликације лако управљају распоредом и виде све заказане термине у реалном времену.
3. **Смањи број грешака и дуплих термина**, јер систем аутоматски блокира већ резервисане термине.
4. **Побољша организацију рада у салону** и омогући бољу комуникацију између клијената и запослених.
5. **Олакша вођење евиденције**, јер се сви подаци чувају централизовано у бази података и лако се могу прегледати и анализирати.
6. **Омогући раст и проширење пословања**, кроз интеграцију нових функција као што су онлајн плаћања, систем лојалности и статистика успешности рада.

Поред функционалних циљева, BarberBook има и едукативни циљ да демонстрира примену савремених технологија као што су React, Go, PostgreSQL и Docker у реалном пројекту, показујући могућност изградње комплетног система заснованог на модерној архитектури.

2.3. Разлози за одабир теме

Тему сам изабрао јер сам желео да повежем знање стечено на предмету Мобилне апликације са областима веб програмирања и радом са базама података**.**

Мој циљ је био да направим нешто више од обичне мобилне апликације, да развијем **комплетан систем** који обухвата и веб и мобилни део, повезан преко заједничког backend-а и базе података. На тај начин сам желео да применим различите технологије и стекнем практично искуство које одражава реалне захтеве савремених софтверских решења.

Ова тема је изабрана и због своје актуелности. У последњих неколико година, све више услужних делатности (укључујући и фризерске салоне) прелази на онлајн системе заказивања како би се побољшала ефикасност пословања и комуникација са клијентима. Због тога ова апликација има реалну практичну вредност и може се у будућности даље развијати и користити у стварним условима.

Поред практичног значаја, избор теме је био мотивисан и жељом за усавршавањем у развоју комплексних система који укључују више технологија и међусобну комуникацију између различитих платформи (веб и мобилне). Развој оваквог пројекта представљао је одличну

прилику да унапредим своје знање из области backend програмирања у Go-у, рада са PostgreSQL базом**,** као и из React и React Native окружења.

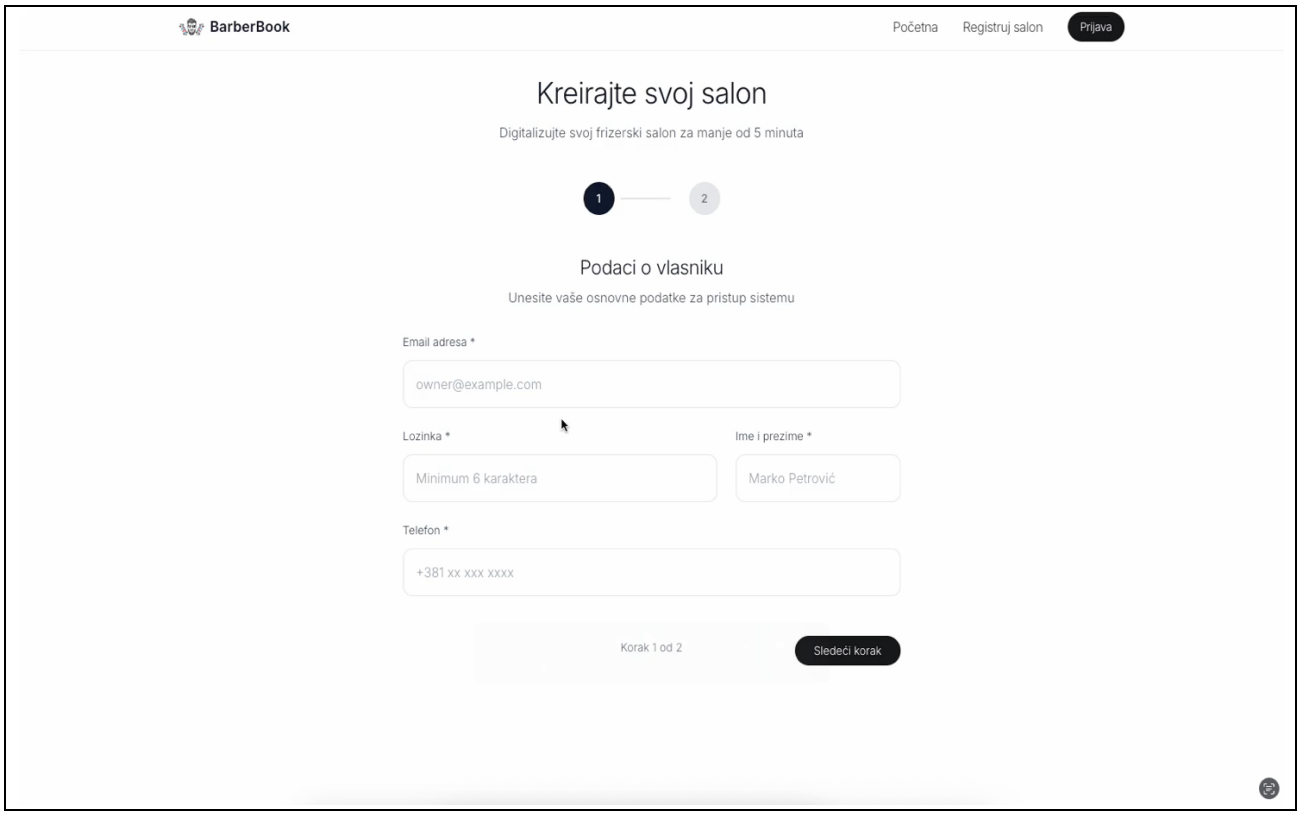
Кроз овај пројекат имао сам прилику да објединим више области, дизајн интерфејса, рад са API-јем, руковање подацима, тестирање и деплој на сервер, што чини овај завршни рад значајним искуством у мом професионалном развоју.

2.4. Функционалност веб апликације

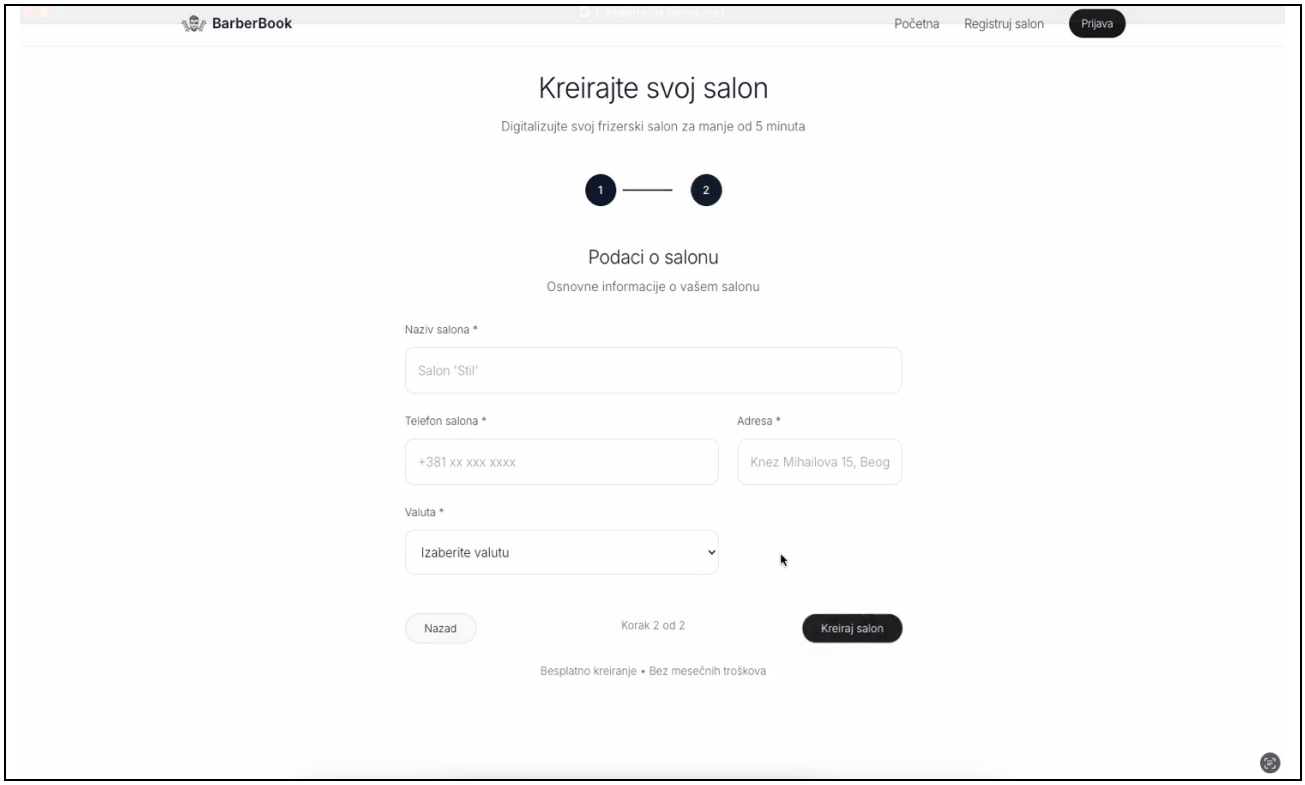
2.4.1. Регистрација салона

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



Други корак регистрације односи се на унос података о самом 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.



Систем тада 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 повезани идентификатор власника.

Након успешне регистрације, власнику се приказује порука о успешној 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 фризере, дефинише услуге, радно време и врши остале измене везане за функционисање салона.

2.4.2. Админ панел за власника салона

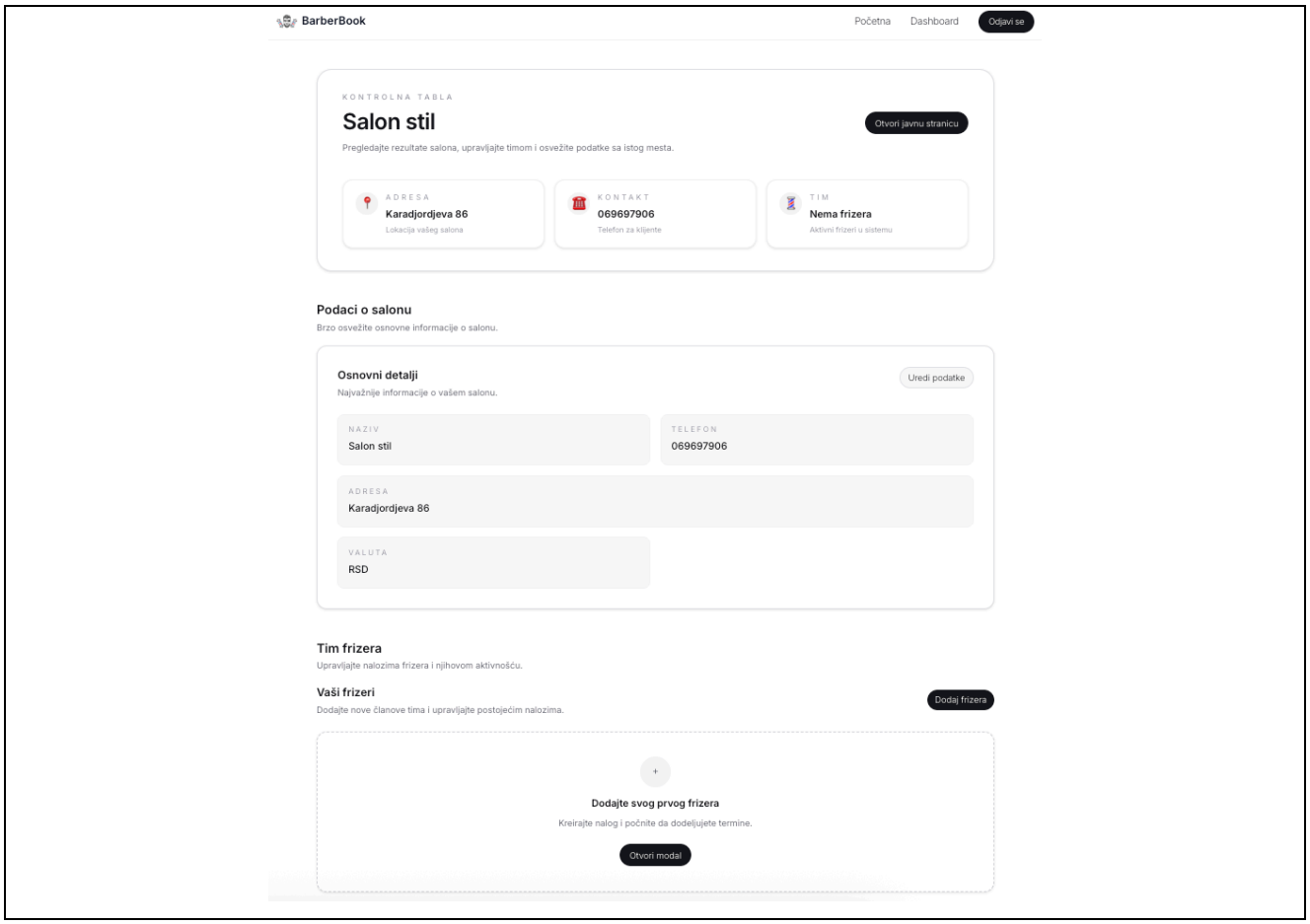
Након успешно извршене регистрације, власник салона се може пријавити у систем и приступити свом административном панелу. Админ панел представља централно место у апликацији преко којег власник управља својим салоном и фризерима. Циљ овог дела апликације је да власнику омогући једноставно руковање подацима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

У административном панелу власник има могућност да види све битне информације о свом салону, као и да врши измене које се одмах чувају у систему.

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.

Интерфејс административног панела је једноставан за коришћење. Сви подаци се приказују у прегледним табелама и формама, а измене се врше кликом на одговарајуће дугме. При свакој промени, систем шаље захтев ка серверу, који обрађује податке и чува их у бази, тако да су све измене одмах видљиве.

Захваљујући овом делу апликације, власник салона може у сваком тренутку да управља свим аспектима пословања, од фризера и услуга до распореда рада и заказаних термина. На тај начин, административни панел представља централну контролу целог система и омогућава ефикасно управљање салоном без потребе за додатним софтвером или ручним евиденцијама.

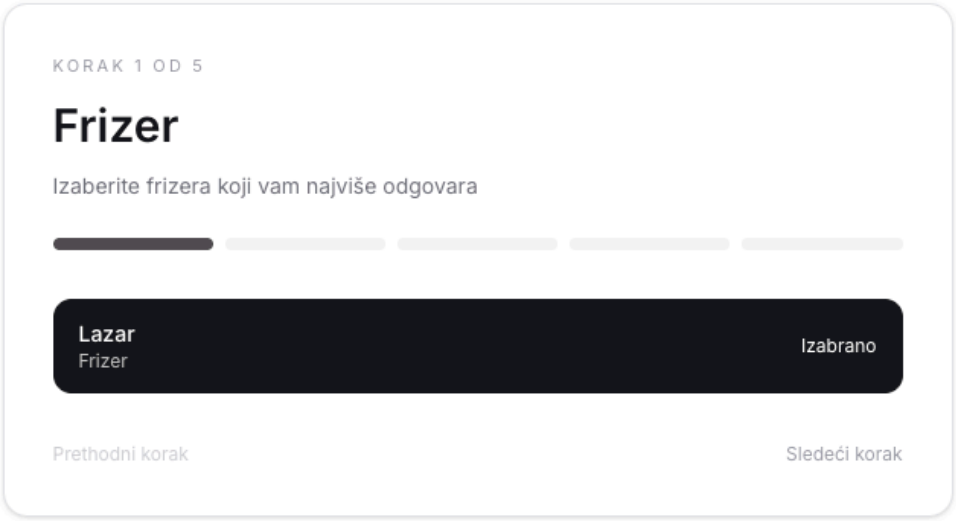


2.4.3. Процес заказивања термина

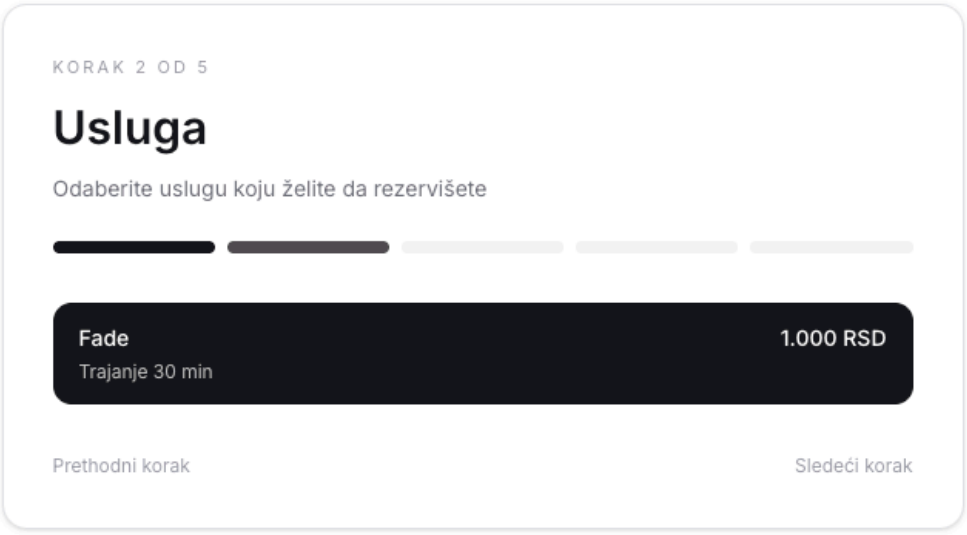
Функционалност заказивања термина представља један од најважнијих деолова апликације. Овај процес омогућава клијентима да на једноставан начин изаберу салон, фризера, услугу и слободан термин, а све се обрађује аутоматски преко backend система. Циљ овог дела апликације је да цео процес заказивања буде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

Процес заказивања се састоји од више корака, који се извршавају редом, тако да корисник у сваком кораку уноси одређене податке. Кораци су дизајнирани тако да се подаци постепено прикупљају и валидирају пре него што се резервација потврди.

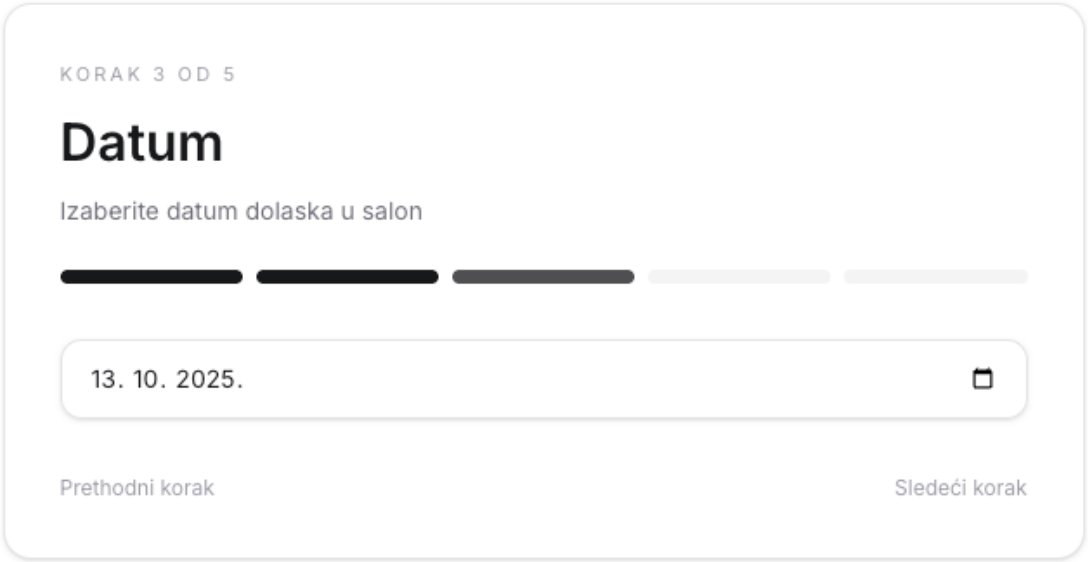
Први корак је **избор фризера**. Клијент на почетној страници веб апликације види листу доступних фризера унутар одабраног салона. Сваки фризер има приказано своје име. Корисник одабиром фризера прелази на следећи корак.



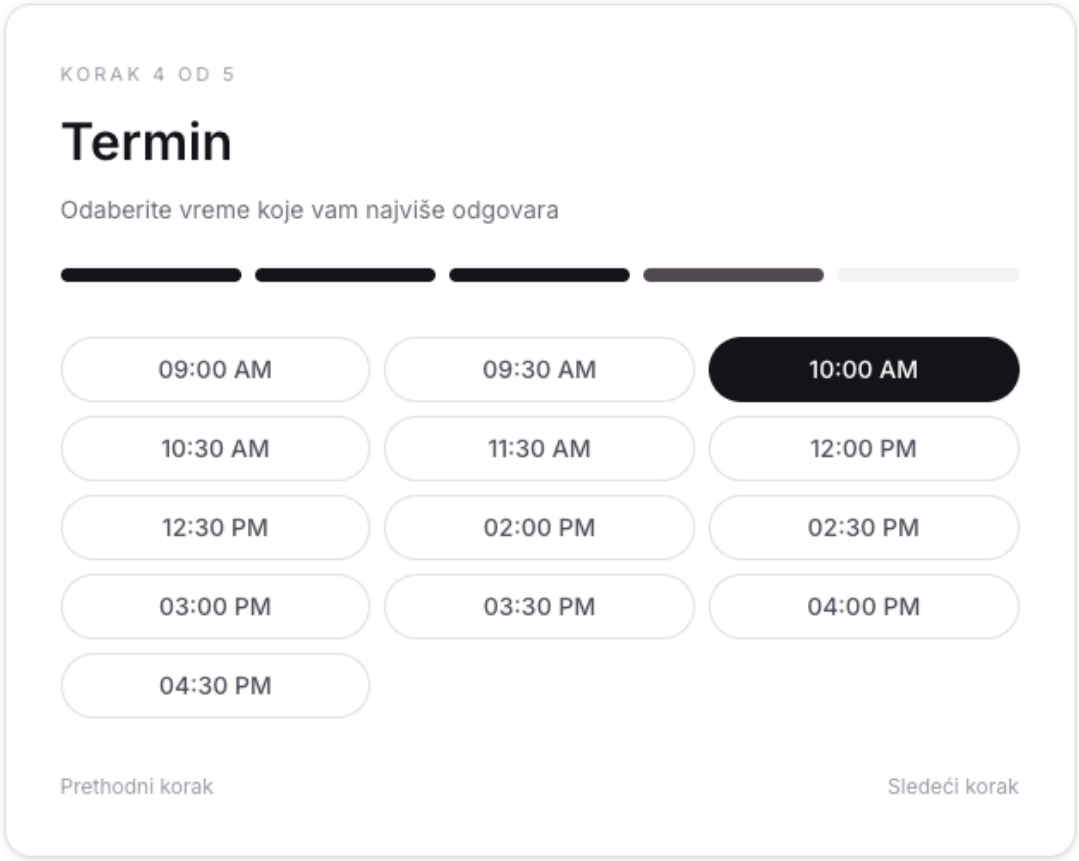
Други корак је **избор услуге**. У овом делу приказују се све услуге које изабрани фризер нуди, као што су шишање, фарбање, стилизовање или бријање. Уз сваку услугу приказано је трајање и цена. Клијент одабиром једне од понуђених услуга наставља процес.



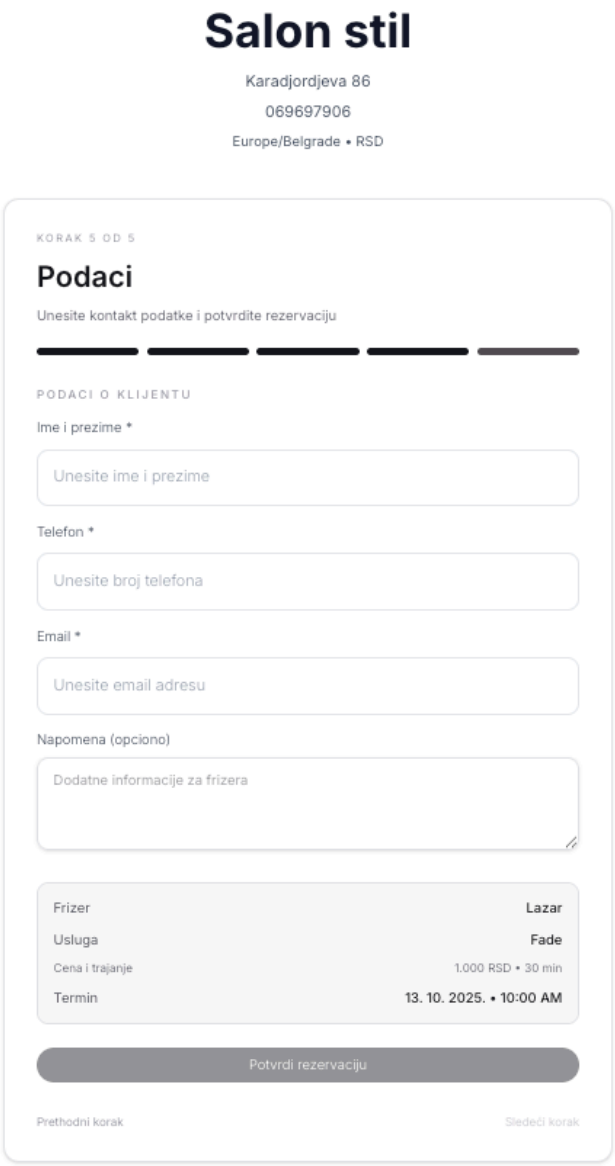
Трећи корак је **избор датума термина**. На овом кораку клијенту се приказује календар са данима за заказивање. Клијент одабиром жељеног датума прелази на следећи корак.



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



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.



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

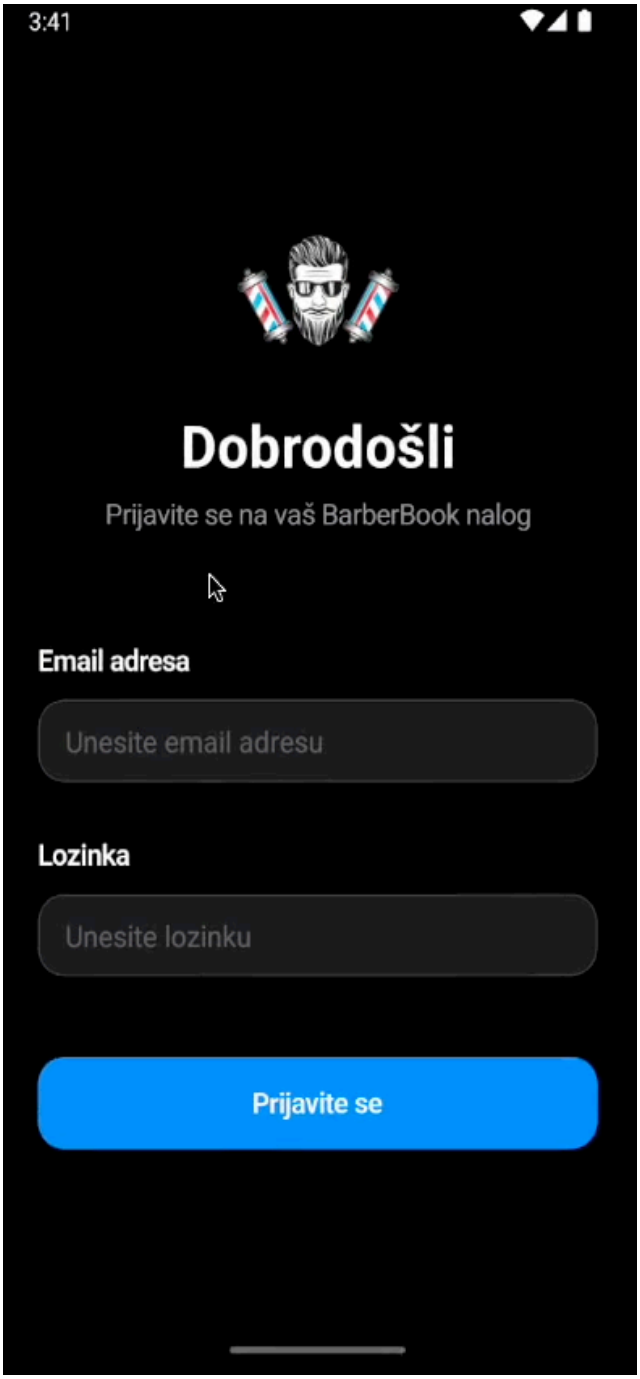
На страни фризера, термин се одмах појављује у мобилној апликацији, где може да види све детаље резервације. Овим системом се постиже потпуна синхронизација између веб и мобилне апликације.

2.5. Функционалност мобилне апликације

Мобилна апликација **BarberBook** представља радни алат фризера, 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, као и да у потпуности синхронизује податке са веб апликацијом.

2.5.1. Пријава фризера на апликацију

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



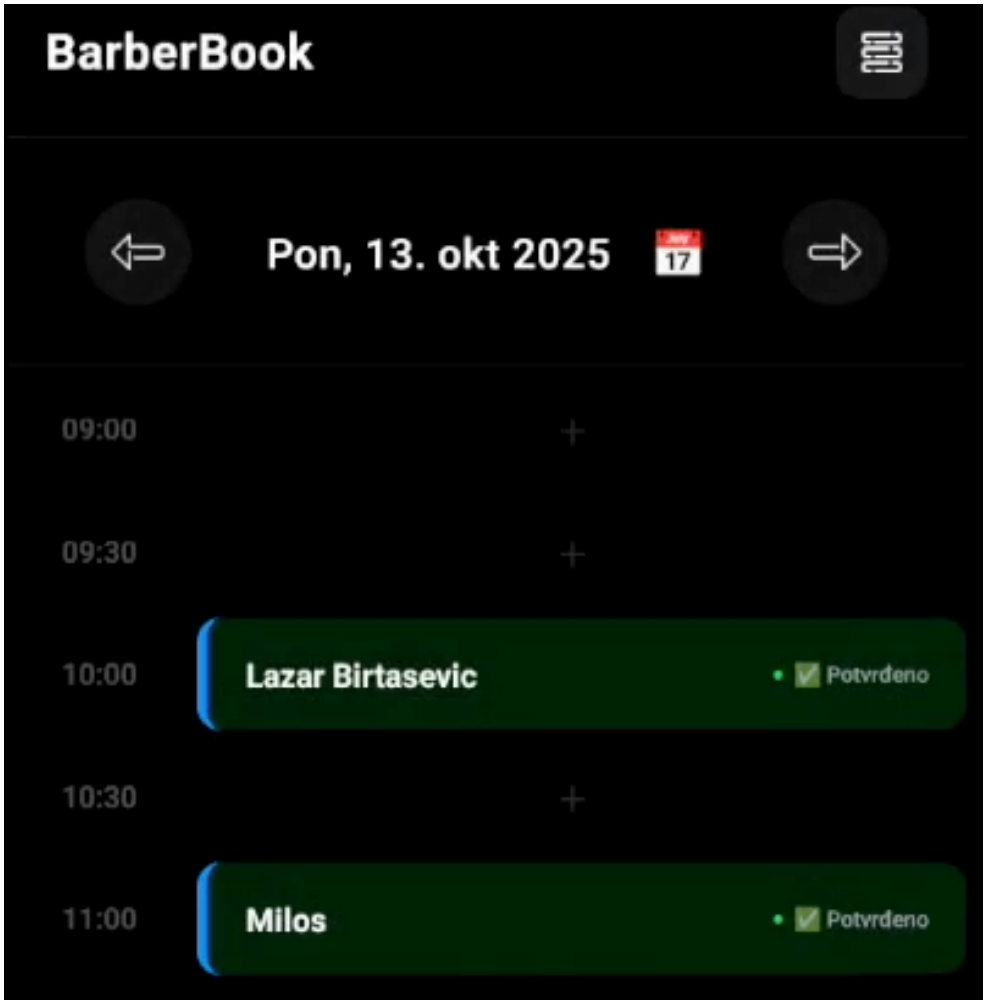
Фризер у поља на екрану уноси своју **имејл адресу** и **лозинку**, које су му додељене приликом регистрације салона у оквиру веб апликације. Подаци који се уносе на овом екрану шаљу се серверу преко **REST API** позива, где backend систем урађен у **Go** програмском језику (са Gin framework-ом) проверава 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.

2.5.2. 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 дневни, недељни или месечни распоред.



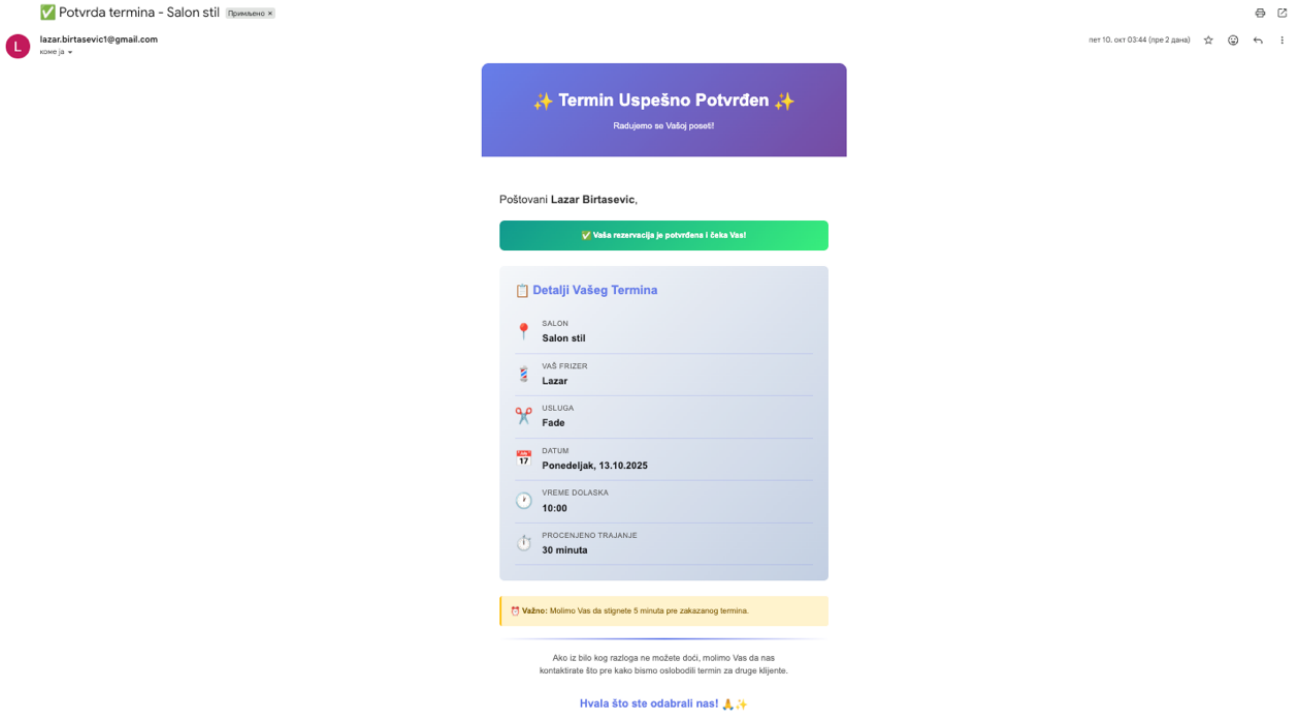
На екрану се приказују сви термини који су клијенти резервисали преко веб апликације. Сваки термин садржи основне информације као што су:

- име и презиме клијента,
- одабрана услуга,
- датум и време термина,
- контакт подаци клијента,
- и евентуална напомена коју је клијент оставио приликом заказивања.

Фризер има могућност да за сваки појединачни термин изврши акције као што су **потврда** или **отказивање термина**. Ове функције омогућавају фризеру да у реалном времену управља свим резервацијама, како би се обезбедила прецизна организација рада и благовремена комуникација са клијентима.

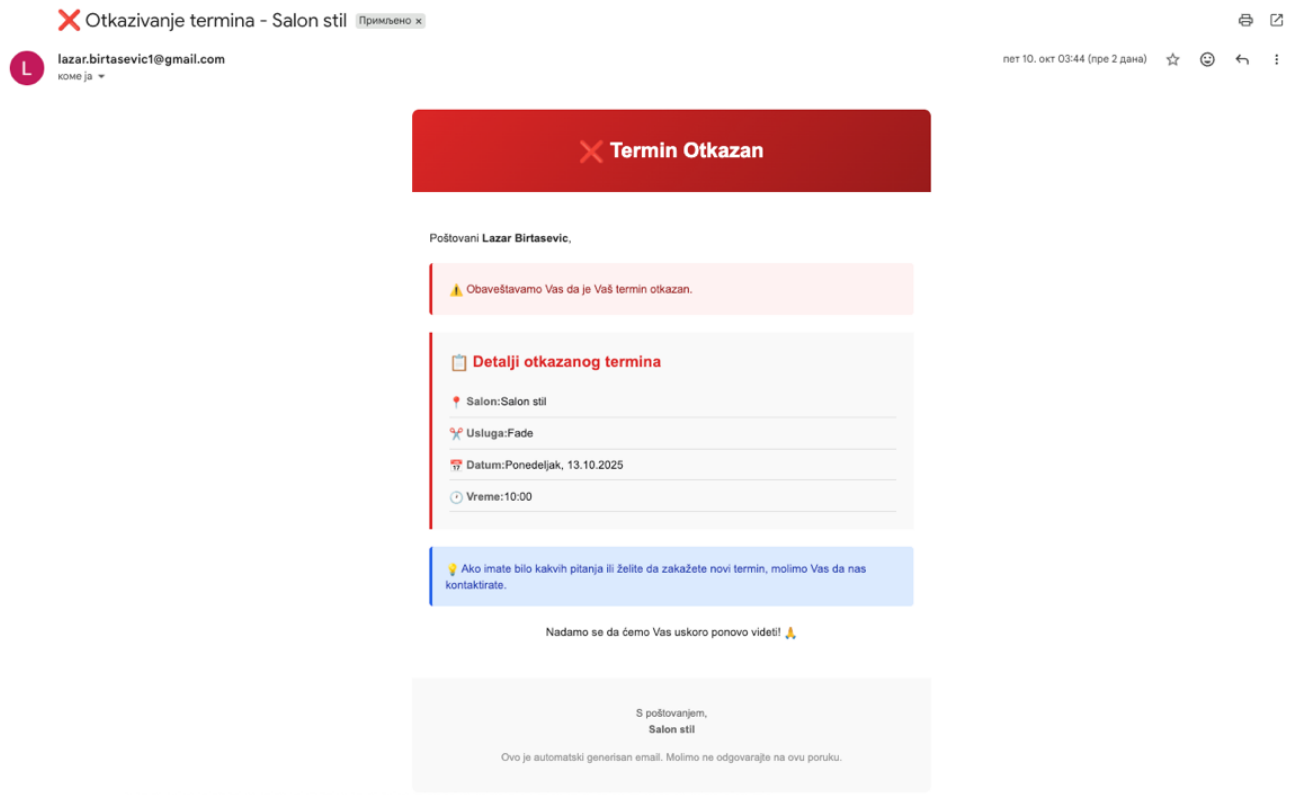
Када фризер **потврди термин**, систем аутоматски ажурира статус резервације у бази података и означава да је термин званично прихваћен. Потврђивањем термина фризер потврђује да ће услуга бити извршена у предвиђено време, чиме се клијенту пружа сигурност да је његово заказивање успешно прихваћено.

Након потврде термина, систем аутоматски шаље **мејл обавештење клијенту** на адресу коју је оставио приликом резервације. У том обавештењу клијент добија све релевантне информације о свом термину, датум, време, назив услуге, име фризера и адресу салона. На овај начин клијент има јасну потврду да је његово заказивање успешно и може лакше да планира свој долазак.



Уколико фризер из било ког разлога не може да прими клијента у заказаном термину, има могућност да изврши **отказивање термина**. Након што фризер потврди отказивање, систем аутоматски мења статус термина у отказан и шаље клијенту имејл обавештење са информацијом да је његов термин отказан, као и разлог који је фризер навео (нпр. ванредна ситуација, технички проблем, промена распореда и сл.).

Овим поступком обезбеђује се правовремено информисање клијента и избегавају се неспоразуми или неугодности које би могле настати уколико би клијент дошао у термин који више није активан.



Поред управљања постојећим резервацијама, мобилна апликација омогућава и **додавање нових термина директно од стране фризера**. Ова опција је посебно корисна у ситуацијама када клијент није користио веб платформу за заказивање, већ се заказивање врши лично или путем телефона. Фризер у том случају може ручно унети нови термин, одабрати датум, време, услугу и унети основне податке клијента, чиме се обезбеђује да сви термини буду уједначено евидентирани у систему.

Сви подаци који се приказују у прегледу термина повезани су са базом података преко **REST API** комуникације. На тај начин, све промене које фризер изврши у мобилној апликацији (потврда, отказивање или додавање термина) одмах се одражавају и на веб апликацији, чиме се постиже потпуна синхронизација између клијентске и фризерске стране.

Кориснички интерфејс је дизајниран тако да буде једноставан и прегледан, како би фризери могли лако да управљају својим распоредом без потребе за додатним објашњењима или сложеним командама.

2.5.3. Подешавање радног времена, услуга и нерадних дана

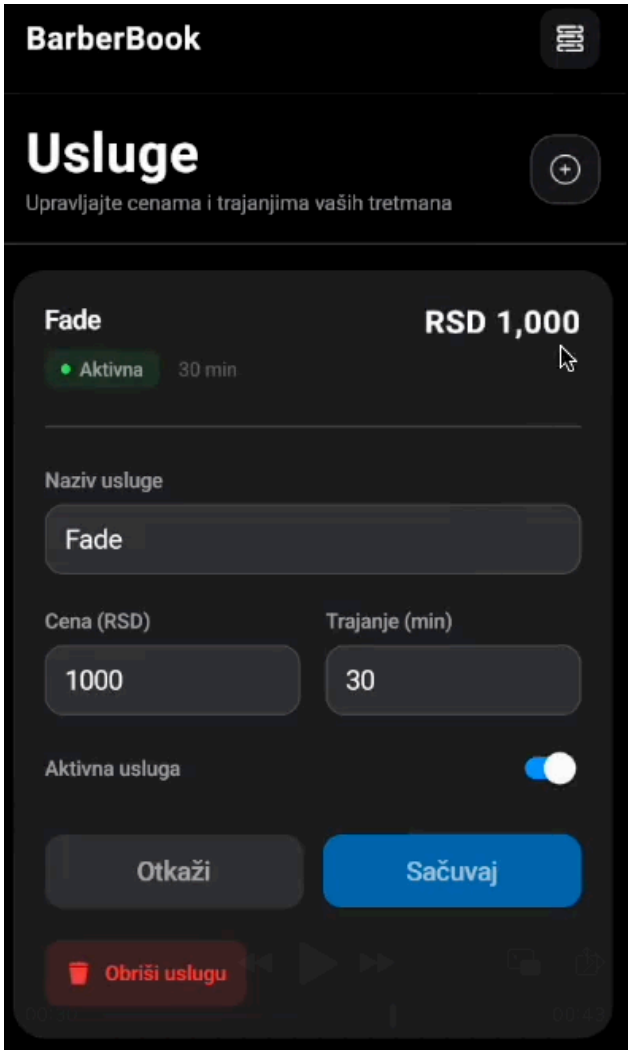
Једна од најзначајнијих функционалности мобилне апликације **BarberBook** јесте могућност да фризер самостално подешава све параметре који се односе на његов рад. Овај део система омогућава потпуну флексибилност у организацији пословног распореда, управљању услугама и дефинисању периода када фризер не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

Подешавање услуга

Фризер има могућност да у апликацији дефинише све услуге које пружа клијентима.
За сваку услугу уносе се следећи подаци:

- **Назив услуге** (нпр. шишање, брада, фарбање и сл.)
- **Цена услуге** у динарима
- **Трајање услуге** у минутима

Фризер може означити да ли је услуга активна или неактивна. Неактивне услуге се не приказују клијентима у процесу заказивања преко веб апликације. Ово омогућава фризеру да привремено искључи услуге које тренутно не пружа, без потребе да их потпуно обрише из система.

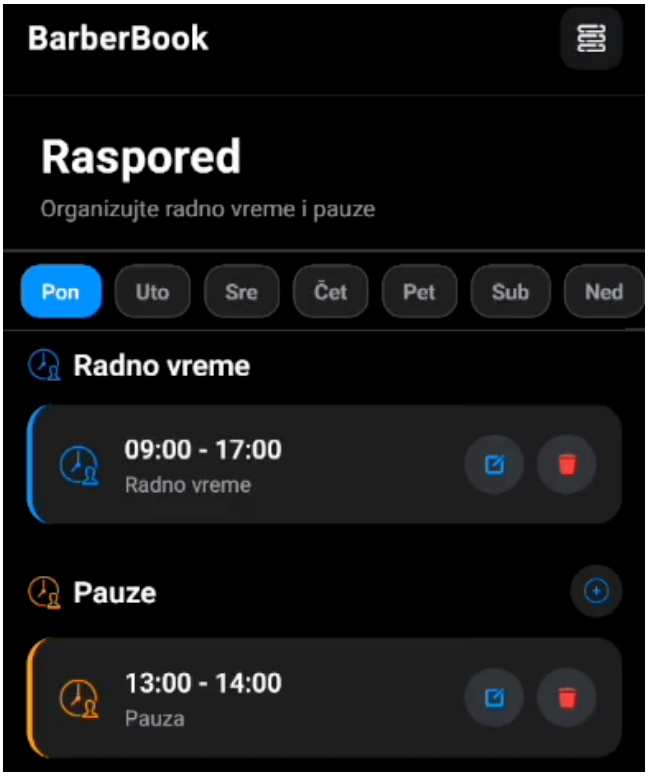


Подешавање радног времена и пауза

Фризер у апликацији може прецизно подесити своје **радно време** за сваки дан у недељи. Сваки дан се конфигурише посебно, па је могуће, на пример, радити дуже током викенда или имати различите смене током радних дана.

Поред радног времена, фризер има могућност да унесе и **паузу**, односно временски интервал током дана када не прима клијенте. На тај начин се обезбеђује да током процеса заказивања клијент не може изабрати време које се поклапа са паузом.

Сви подаци о радном времену и паузама се чувају у бази података и користе на веб апликацији како би систем знао који су термини доступни за заказивање.



Подешавање нерадних дана

Апликација омогућава и дефинисање **нерадних дана**, односно периода када фризер неће радити. Приликом уноса нерадног дана, фризер задаје:

- **датум почетка и краја нерадног периода**,
- као и **разлог због ког неће радити** (нпр. одмор, болест, празник и сл.).

Ови подаци се користе на страни веб апликације приликом заказивања термина. Ако клијент покуша да изабере датум који се налази у нерадном периоду, систем ће приказати обавештење са разлогом који је фризер унео. На тај начин клијент одмах зна да тог дана услуге нису доступне, чиме се спречавају могуће грешке и неспоразуми.

Сви подаци које фризер уноси у овом делу апликације синхронизовани су са backend сервером преко **REST API** комуникације. Свака промена (додавање, измена или брисање) аутоматски се чува у бази података, тако да је веб апликација увек ажурирана и приказује тренутно стање распореда и доступности.

3. Опис имплементације пројекта

3.1. Архитектура система

Апликација се заснива на клијент–сервер архитектури, која омогућава поуздану и скалабилну 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**.**

Веб апликација представља део система намењен клијентима. Она омогућава претраживање салона, избор фризера, преглед услуга и заказивање термина. Развијена је помоћу **React** библиотеке, која омогућава креирање брзих и интерактивних корисничких интерфејса. Комуникација са сервером врши се путем **REST API** позива, при чему се подаци размењују у **JSON** формату.

Мобилна апликација, развијена у **React Native** окружењу, намењена је фризерима. Она омогућава преглед свих заказаних термина, потврђивање или отказивање резервација, додавање нових термина, као и подешавање радног времена, пауза и услуга. Мобилна апликација је у потпуности повезана са истим backend сервером као и веб апликација, што омогућава тренутну синхронизацију свих података.

Backend сервер, развијен у програмском језику **Go** уз коришћење **Gin framework-a**, представља централни део система који управља свим логичким операцијама. Сервер обрађује захтеве које шаљу веб и мобилна апликација, врши аутентикацију корисника, чува и преузима податке из базе, као и слање имејл обавештења клијентима.

PostgreSQL база података чува све податке у вези корисника, салона, фризера, услуга и термина. База је дизајнирана тако да омогућава брз приступ подацима, поузданост и интегритет, као и лаку надоградњу система у будућности.

Комуникација између компоненти одвија се путем **HTTP протокола** преко REST API интерфејса. Backend сервер прима захтеве од клијентских апликација, обрађује их и враћа одговор у JSON формату. На тај начин систем обезбеђује стабилну и сигурну размену података између свих делова.

Систем је такође дизајниран тако да омогућава лако одржавање и проширивање функционалности. Захваљујући модуларној структури, могуће је накнадно додати нове услуге, типове корисника или интеграције без потребе за великим изменама постојећег кода.

У наредним потпоглављима биће детаљније објашњени појединачни делови система: backend, frontend и база података, као и начини њихове међусобне комуникације.

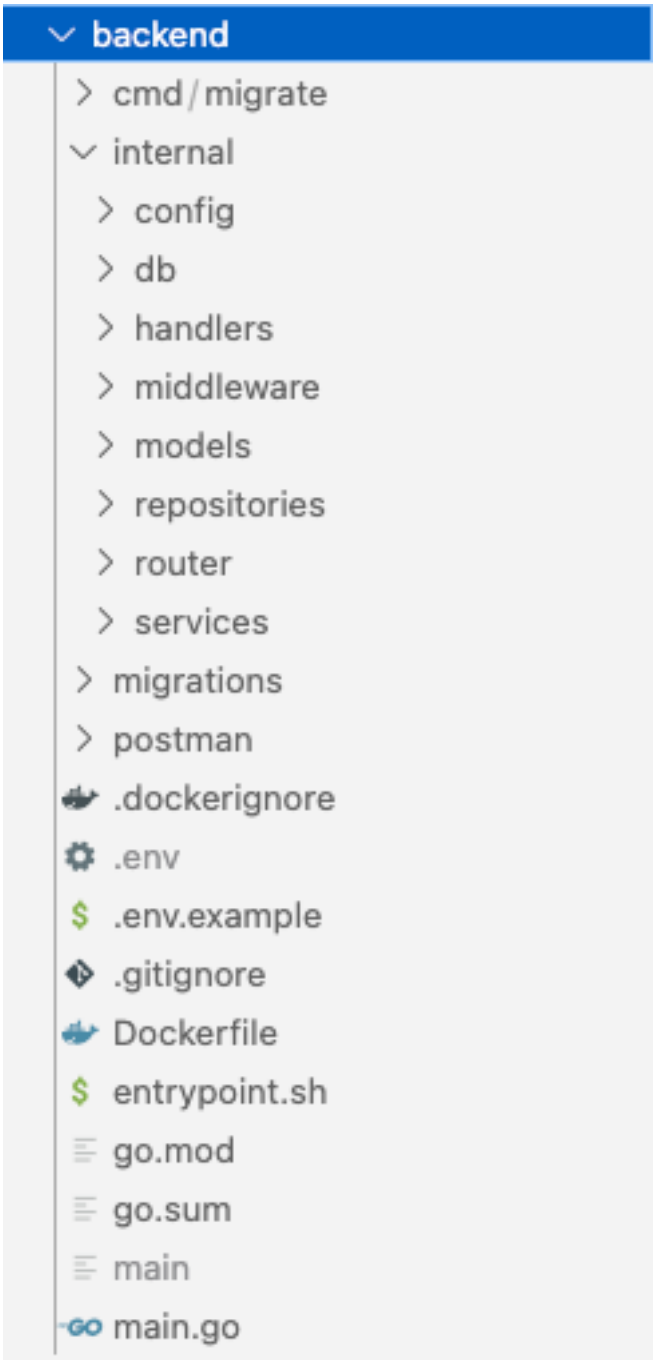
3.2. Структура backend-а (Go + Gin)

3.2.1. Увод

Backend систем BarberBook апликације представља RESTful API сервис развијен у програмском језику Go (Golang), који омогућава комплетно управљање салонима за шишање, фризерима, клијентима и заказивањем термина. Систем је имплементиран користећи модерне архитектурнеPattern-е и best practice-е за развој скалабилних веб апликација.

3.2.2. Архитектура система

Пројекат је организован по принципу Clean Architecture, са јасно дефинисаним слојевима одговорности:



3.2.3. Handler слој (Презентациони слој)

Handler слој је одговоран за обраду HTTP захтева и враћање одговора. Сваки handler функција prima HTTP захтев, позива одговарајући сервис и враћа резултат кориснику.

Пример: Креирање салона

```go
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)
}
```

3.2.4. Service слој (Пословна логика)

Service слој садржи сву пословну логику апликације. Овај слој валидира податке, примењује пословна правила и координира рад са repository слојем.

Пример: Провера 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
        }
```

3.2.5. Repository слој (Слој приступа подацима)

Repository слој апстрахује комуникацију са базом података. Овај слој извршава SQL упите и мапира резултате у Go структуре.

Пример: Репозиторијум за салоне

```
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
}

func (r *SalonRepository) GetByID(ctx context.Context, id int64) (*models.Salon, error)
{
    salon := &models.Salon{}
    query := `
        SELECT id, name, address, phone_number, email, owner_id, created_at, updated_at
        FROM salons
        WHERE id = $1`

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

    if err == sql.ErrNoRows {
        return nil, nil
    }

    return salon, err
}
```

3.2.6. Аутентификација и ауторизација

JWT (JSON Web Token) аутентификација

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

Имплементација генерисања токена:

```
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))
        }
```

Middleware за аутентификацију

Middleware функција проверава валидност JWT токена за заштићене руте.

```
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))
        })
    }
        }
```

3.2.7. Модели података

Основни ентитети:

Модел корисника:

```
type User struct {
    ID           int64     `json:"id"`
    Email        string    `json:"email"`
    PasswordHash string    `json:"-"`
    FirstName    string    `json:"first_name"`
    LastName     string    `json:"last_name"`
    PhoneNumber  string    `json:"phone_number"`
    Role         string    `json:"role"` // 'barber', 'salon_owner'
    CreatedAt    time.Time `json:"created_at"`
    UpdatedAt    time.Time `json:"updated_at"`
       }
```

Модел салона:

```
type Salon struct {
    ID          int64     `json:"id"`
    Name        string    `json:"name"`
    Address     string    `json:"address"`
    PhoneNumber string    `json:"phone_number"`
    Email       string    `json:"email"`
    OwnerID     int64     `json:"owner_id"`
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`
}
```

Модел заказивања:

```
type Appointment struct {
    ID          int64     `json:"id"`
    ClientID    int64     `json:"client_id"`
    BarberID    int64     `json:"barber_id"`
    SalonID     int64     `json:"salon_id"`
    ServiceID   int64     `json:"service_id"`
    StartTime   time.Time `json:"start_time"`
    EndTime     time.Time `json:"end_time"`
    Status      string    `json:"status"` // 'pending', 'confirmed', 'cancelled'
    Notes       string    `json:"notes,omitempty"`
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`
}
```

3.2.8. Кључне функционалности

Управљање заказивањима

Систем за заказивање термина представља срце апликације. Процес заказивања укључује:

1. Провера доступности фризера
2. Валидација радног времена
3. Провера одсуства
4. Провера постојећих термина
5. Креирање термина

Имплементација креирања термина:

```
func (s *AppointmentService) CreateAppointment(ctx context.Context,
    clientID, barberID, salonID, serviceID int64, startTime time.Time)
(*models.Appointment, error) {

    // Добијање трајања услуге
    service, err := s.GetServiceByID(ctx, serviceID)
    if err != nil {
        return nil, err
    }

    endTime := startTime.Add(time.Duration(service.Duration) * time.Minute)

    // Провера доступности
    available, err := s.availabilityService.IsTimeSlotAvailable(ctx,
        barberID, salonID, startTime, endTime)
    if err != nil {
        return nil, err
    }

    if !available {
        return nil, errors.New("time slot not available")
    }

    appointment := &models.Appointment{
        ClientID:  clientID,
        BarberID:  barberID,
        SalonID:   salonID,
        ServiceID: serviceID,
        StartTime: startTime,
        EndTime:   endTime,
        Status:    "pending",
        CreatedAt: time.Now(),
        UpdatedAt: time.Now(),
    }

    if err := s.repo.Create(ctx, appointment); err != nil {
        return nil, err
    }

    // Слање email обавештења
    s.emailService.SendAppointmentConfirmation(appointment)

    return appointment, nil
}
```

Управљање радним временом

Сваки фризер има дефинисано радно време за сваки дан у недељи.

Модел радног времена:

```
type BarberSchedule struct {
    ID        int64     `json:"id"`
    BarberID  int64     `json:"barber_id"`
    SalonID   int64     `json:"salon_id"`
    DayOfWeek int       `json:"day_of_week"` // 0=Sunday, 6=Saturday
    StartTime string    `json:"start_time"`  // HH:MM формат
    EndTime   string    `json:"end_time"`    // HH:MM формат
    IsWorking bool      `json:"is_working"`
    CreatedAt time.Time `json:"created_at"`
    UpdatedAt time.Time `json:"updated_at"`
}
```

Постављање радног времена:

```
func (h *BarberScheduleHandler) SetSchedule(w http.ResponseWriter, r *http.Request) {
    var schedules []struct {
        DayOfWeek int    `json:"day_of_week"`
        StartTime string `json:"start_time"`
        EndTime   string `json:"end_time"`
        IsWorking bool   `json:"is_working"`
    }

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

    barberID, _ := strconv.ParseInt(chi.URLParam(r, "barberID"), 10, 64)
    salonID, _ := strconv.ParseInt(chi.URLParam(r, "salonID"), 10, 64)

    // Валидација да фризер припада салону
    if err := h.service.ValidateBarberSalon(r.Context(), barberID, salonID); err != nil
{
        RespondWithError(w, http.StatusForbidden, "Barber not associated with salon")
        return
    }

    // Постављање радног времена за сваки дан
    for _, schedule := range schedules {
        err := h.service.SetBarberSchedule(r.Context(), barberID, salonID,
            schedule.DayOfWeek, schedule.StartTime, schedule.EndTime,
schedule.IsWorking)
        if err != nil {
            RespondWithError(w, http.StatusInternalServerError,
                "Failed to set schedule")
            return
        }
    }

    RespondWithJSON(w, http.StatusOK, map[string]string{
        "message": "Schedule updated successfully",
    })
}
```

Управљање одсуствима

Систем омогућава фризерима да означе периоде када нису доступни (одмори, боловања, приватни разлози).

Модел одсуства:

```
type BarberTimeOff struct {
    ID        int64     `json:"id"`
    BarberID  int64     `json:"barber_id"`
    SalonID   int64     `json:"salon_id"`
    StartDate time.Time `json:"start_date"`
    EndDate   time.Time `json:"end_date"`
    Reason    string    `json:"reason,omitempty"`
    CreatedAt time.Time `json:"created_at"`
    UpdatedAt time.Time `json:"updated_at"`
}
```

Управљање услугама

Сваки салон може да дефинише услуге које нуди, а фризери могу да изаберу које услуге пружају.

Модел услуге:

```
type Service struct {
    ID          int64     `json:"id"`
    SalonID     int64     `json:"salon_id"`
    Name        string    `json:"name"`
    Description string    `json:"description,omitempty"`
    Duration    int       `json:"duration"` // у минутима
    Price       float64   `json:"price"`
    CreatedAt   time.Time `json:"created_at"`
    UpdatedAt   time.Time `json:"updated_at"`
}

type BarberService struct {
    ID        int64     `json:"id"`
    BarberID  int64     `json:"barber_id"`
    SalonID   int64     `json:"salon_id"`
    ServiceID int64     `json:"service_id"`
    Price     float64   `json:"price"` // Може да преопредели цену
    CreatedAt time.Time `json:"created_at"`
}
```

3.2.9. Рутирање и API endpoints

Структура рутирања

Систем користи chi router за дефинисање HTTP рута.

Преглед главних API endpoint-а:

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

3.2.10. Email обавештења

Систем шаље email обавештења за кључне догађаје.

```
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)
}
```

3.2.11. Закључак

BarberBook backend систем представља робустно и скалабилно решење за управљање салонима за шишање. Кључне карактеристике система укључују:

1. Модуларна архитектура - Јасна подела одговорности између слојева
2. RESTful API - Стандардизован интерфејс за комуникацију
3. JWT аутентификација - Сигурна аутентификација корисника
4. Комплексна логика заказивања - Провера доступности, радног времена и одсуства
5. Email нотификације - Аутоматска обавештења корисника
6. PostgreSQL база података - Релациона база са миграцијама
7. Docker подршка - Једноставно deployovanje апликације

Систем је развијен са фокусом на одржавање, проширивост и перформансе, користећи best practice-е Go програмског језика и модерне архитектурне Pattern-е.

3.3. Структура базе података (PostgreSQL)

3.3.1. Увод

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 суEnum типови, Check ограничења, Exclude constraint-и, тригери и индекси за оптималне перформансе.

3.3.2. Разлози за избор PostgreSQL-а

PostgreSQL је изабран као систем за управљање базом података из следећих разлога:

1. ACID компатибилност - Гаранција конзистентности података
2. Напредне функционалности - Подршка за JSON, пуно-текстуалну претрагу, геопросторне податке
3. Enum типови - Дефинисање кастом типова података
4. Exclude constraints - Спречавање преклапања термина
5. Тригери и функције - Аутоматизација пословне логике
6. Перформансе - Напредне могућности индексирања
7. Open source - Бесплатан и са великом заједницом

3.3.3. Верзија и екстензије

Систем користи PostgreSQL 15+ са следећим екстензијама:

```
-- Екстензија за генерисање UUID вредности
CREATE EXTENSION IF NOT EXISTS pgcrypto;

-- Екстензија за спречавање преклапања временских интервала
CREATE EXTENSION IF NOT EXISTS btree_gist;

-- Екстензија за case-insensitive текстуалне колоне (email)
CREATE EXTENSION IF NOT EXISTS citext;
```

Објашњење екстензија:

pgcrypto - Омогућава генерисање UUID вредности помоћу функције gen_random_uuid(), што обезбеђује јединствене идентификаторе за све записе

btree_gist - Потребна за креирање EXCLUDE constraint-а са временским интервалима, спречава преклапање термина

citext - Case-insensitive TEXT тип, користи се за email адресе да би user@example.com и USER@example.com били третирани као исти

3.3.4. ENUM типови података

Tun appointment_status

```
CREATE TYPE appointment_status AS ENUM ('pending','confirmed','canceled');
```

Дефинише могуће статусе заказивања:

Вредности:

1. pending - Термин је захтеван али још није потврђен
2. confirmed - Термин је потврђен и активан
3. canceled - Термин је отказан

Предности ENUM типа:

1. Гаранција валидних вредности на нивоу базе
2. Боље перформансе од VARCHAR са CHECK constraint-ом
3. Јасна документација дозвољених вредности

Tun staff_role

```
CREATE TYPE staff_role AS ENUM ('owner','barber');
```

Дефинише улоге корисника у систему:

Вредности:

1. owner - Власник салона, има права управљања салоном
2. barber - Фризер, пружа услуге клијентима

3.3.5. Шема базе података - Детаљан опис табела

Табела users (Корисници)

Чува податке о свим корисницима система (власници салона и фризери).

```
CREATE TABLE IF NOT EXISTS users (
  id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  email           CITEXT UNIQUE NOT NULL,
  password_hash   TEXT NOT NULL,
  full_name       TEXT NOT NULL,
  phone           TEXT,
  role            staff_role NOT NULL DEFAULT 'barber',
  created_at      TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

Детаљан опис колона:

- id - Примарни кључ, UUID тип, аутоматски генерисан
  - UUID обезбеђује јединственост чак и у дистрибуираним системима
  - gen_random_uuid() генерише криптографски безбедне UUID вредности
- email - CITEXT тип, јединствен, не може бити NULL
  - CITEXT тип омогућава case-insensitive упоређивање
  - UNIQUE constraint спречава дуплирање email адреса
  - Аутоматски преводи у мала слова при упоређивању
- password_hash - TEXT тип, не може бити NULL
  - Чува хеширану лозинку (никада plain text)
  - Користи се bcrypt алгоритам за хеширање
- full_name - TEXT тип, пуно име корисника
  - Обавезно поље за идентификацију
- phone - TEXT тип, опционо
  - Контакт телефон корисника
- role - staff_role ENUM, подразумевана вредност 'barber'
  - Одређује улогу корисника у систему
- created_at - TIMESTAMPTZ (timestamp with time zone)
  - Аутоматски постављен на тренутно време
  - Чува информацију о временској зони

Табела salons (Салони)

Чува информације о салонима за шишање.

```
CREATE TABLE IF NOT EXISTS salons (
  id              UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  name            TEXT NOT NULL,
  phone           TEXT NOT NULL,
  address         TEXT NOT NULL,
  timezone        TEXT NOT NULL DEFAULT 'Europe/Belgrade',
  currency        TEXT NOT NULL DEFAULT 'RSD',
  owner_id        UUID NOT NULL REFERENCES users(id) ON DELETE RESTRICT,
  created_at      TIMESTAMPTZ NOT NULL DEFAULT now()
);
```

Детаљан опис колона:

- id - Примарни кључ, UUID тип
    - name - Назив салона
    - Обавезно поље
- phone - Контакт телефон салона
    - Користи се за комуникацију са клијентима
- address - Физичка адреса салона
    - Обавезно за лоцирање салона
- timezone - Временска зона салона
    - Подразумевана вредност: 'Europe/Belgrade'
    - Важно за правилно рачунање времена термина
    - Омогућава подршку за салоне у различитим временским зонама
- currency - Валута у којој се исказују цене
    - Подразумевана вредност: 'RSD' (српски динар)
    - Омогућава међународну подршку
- owner_id - Спољни кључ ка табели users
    - REFERENCES users(id) - Дефинише релацију
    - ON DELETE RESTRICT - Не дозвољава брисање власника ако има салон
    - Везује салон за корисника са улогом 'owner'
- created_at - Датум креирања салона

Релације:

Један власник (user) може имати више салона (1:N), салон мора имати тачно једног власника.

Табела barbers (Фризери)

Представља везу између корисника и салона у улози фризера.

```
CREATE TABLE IF NOT EXISTS barbers (
  id                      UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  user_id                 UUID NOT NULL REFERENCES users(id) ON DELETE CASCADE,
  salon_id                UUID NOT NULL REFERENCES salons(id) ON DELETE CASCADE,
  display_name            TEXT NOT NULL,
  active                  BOOLEAN NOT NULL DEFAULT TRUE,
  slot_duration_minutes   INT NOT NULL CHECK (slot_duration_minutes > 0),
  created_at              TIMESTAMPTZ NOT NULL DEFAULT now(),
  UNIQUE (salon_id, user_id),
  UNIQUE (salon_id, display_name)
);
```

Детаљан опис колона:

- id - Примарни кључ фризера у салону
- user_id - Спољни кључ ка табели users
    - ON DELETE CASCADE - Ако се обрише корисник, бришу се и његови barber записи
- salon_id - Спољни кључ ка табели salons
    - ON DELETE CASCADE - Ако се обрише салон, бришу се и сви фризери
- display_name - Име под којим се фризер приказује клијентима
    - Може се разликовати од full_name у табели users
    - Корисно за надимке или скраћена имена
- active - Статус активности фризера
    - TRUE = фризер је активан и прима термине
    - FALSE = фризер је неактиван (нпр. привремено одсутан)
- slot_duration_minutes - Трајање временског слота у минутима
    - Дефинише минималну јединицу времена за заказивање
    - CHECK ограничење обезбеђује да је вредност > 0
    - Пример: ако је 15, термини се могу заказивати сваких 15 минута
- created_at - Датум када је фризер додат у салон

Ограничења:

`UNIQUE (salon_id, user_id)` – Један корисник може бити фризер у само једном салону-

`UNIQUE (salon_id, display_name)` – У једном салону не могу два фризера имати исто display_name.

Табела barber_services (Услуге фризера)

Дефинише услуге које сваки фризер нуди.

```
CREATE TABLE IF NOT EXISTS barber_services (
  id                  UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  barber_id           UUID NOT NULL REFERENCES barbers(id) ON DELETE CASCADE,
  name                TEXT NOT NULL,
  price               NUMERIC(10,2) NOT NULL CHECK (price >= 0),
  duration_min        INT NOT NULL CHECK (duration_min > 0),
  active              BOOLEAN NOT NULL DEFAULT TRUE,
  created_at          TIMESTAMPTZ NOT NULL DEFAULT now(),
  UNIQUE (barber_id, name)
);

CREATE INDEX IF NOT EXISTS idx_barber_services_active
  ON barber_services (barber_id, active);
```

Детаљан опис колона:

- id - Примарни кључ услуге
- barber_id - Спољни кључ ка табели barbers
    - ON DELETE CASCADE - Ако се обрише фризер, бришу се све његове услуге
- name - Назив услуге
    - Примери: "Шишање", "Брадобрија", "Шишање + брада"
- price - Цена услуге
    - NUMERIC(10,2) - До 8 цифара пре децимале, 2 после
    - CHECK (price >= 0) - Цена не може бити негативна
    - Пример: 1500.00 (динара)
- duration_min - Трајање услуге у минутима
    - CHECK (duration_min > 0) - Мора бити позитивно
    - Пример: 30 минута за шишање
- active - Да ли је услуга тренутно доступна
    - FALSE = услуга није доступна за заказивање
- created_at - Датум креирања услуге

Ограничења:

`UNIQUE (barber_id, name)` – Један фризер не може имати две услуге са истим називом.

Индекси:

```
CREATE INDEX idx_barber_services_active
ON barber_services (barber_id, active);
```

- Убрзава упите који траже активне услуге одређеног фризера
- Composite индекс на два поља

Табела barber_working_hours (Радно време фризера)

Дефинише радне сате фризера за сваки дан у недељи.

```
CREATE TABLE IF NOT EXISTS barber_working_hours (
  id            UUID PRIMARY KEY DEFAULT gen_random_uuid(),
  barber_id     UUID NOT NULL REFERENCES barbers(id) ON DELETE CASCADE,
  day_of_week   INT  NOT NULL CHECK (day_of_week BETWEEN 0 AND 6),
  start_time    TIME NOT NULL,
  end_time      TIME NOT NULL,
  CHECK (start_time < end_time),
  UNIQUE (barber_id, day_of_week, start_time, end_time)
);

CREATE INDEX IF NOT EXISTS idx_working_hours_lookup
  ON barber_working_hours (barber_id, day_of_week);
```

Детаљан опис колона:

- id - Примарни кључ
- barber_id - Спољни кључ ка табели barbers
- day_of_week - Дан у недељи

  - 0 = недеља
  - 1 = понедељак
  - 2 = уторак
  - 3 = среда
  - 4 = четвртак
  - 5 = петак
  - 6 = субота

  - CHECK (day_of_week BETWEEN 0 AND 6) - Валидација вредности

- start_time - Почетак радног времена

  - TIME тип (само време без датума)

  - Пример: '09:00:00'

- end_time - Крај радног времена

  - Пример: '17:00:00'

Ограничења:

`CHECK (start_time < end_time)` – Почетак мора бити пре краја радног времена.

`UNIQUE (barber_id, day_of_week, start_time, end_time)` – Спречава дупликате истог радног времена.

Индекси:

```
CREATE INDEX idx_working_hours_lookup
  ON barber_working_hours (barber_id, day_of_week);
```

Убрзава упите за проналажење радног времена за одређени дан.

Табела appointments (Заказивања)

Најважнија табела која чува све заказане термине.

```
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)
);
```

Детаљан опис колона:

- id - Примарни кључ термина
- salon_id - Спољни кључ ка табели salons
    - ON DELETE CASCADE - Ако се обрише салон, бришу се сви термини
- barber_id - Спољни кључ ка табели barbers
    - ON DELETE RESTRICT - Не дозвољава брисање фризера ако има заказане термине

- barber_service_id - Спољни кључ ка табели barber_services
    - Означава која услуга се пружа
- customer_name - Име клијента
    - Обавезно поље за идентификацију
- customer_phone - Телефон клијента (опционо)
    - За контакт и потврду термина
- customer_email - Email клијента (опционо)
    - За слање обавештења
- price - Цена услуге
    - Копира се из barber_services у тренутку заказивања
    - CHECK (price >= 0) - Не може бити негативна
- duration_min - Трајање у минутима
    - Копира се из barber_services
- start_at - Почетак термина
    - TIMESTAMPTZ са временском зоном
    - Пример: '2025-01-20 10:00:00+01'
- end_at - Крај термина
    - Аутоматски се израчунава тригером
    - end_at = start_at + duration_min
- status - Статус термина
    - 'pending' = чека потврду
    - 'confirmed' = потврђен
    - 'canceled' = отказан
    - notes - Додатне напомене (опционо)
    - Пример: "Клијент жели бријање машиницом број 2"
- created_at - Датум креирања термина

Ограничења:

`CHECK (end_at > start_at)` – Крај мора бити после почетка.

3.3.6. Напредне функционалности базе података

Аутоматски тригер за израчунавање end_at

Систем користи PostgreSQL тригер за аутоматско израчунавање времена завршетка термина.

```
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();
```

Објашњење:

- Функција set_appointment_end_at():

  - Написана у PL/pgSQL језику
  - Позива се пре INSERT или UPDATE операције
  - Израчунава end_at као start_at + duration_min
  - make_interval(mins => NEW.duration_min) креира PostgreSQL interval

- Тригер trg_set_appointment_end_at:

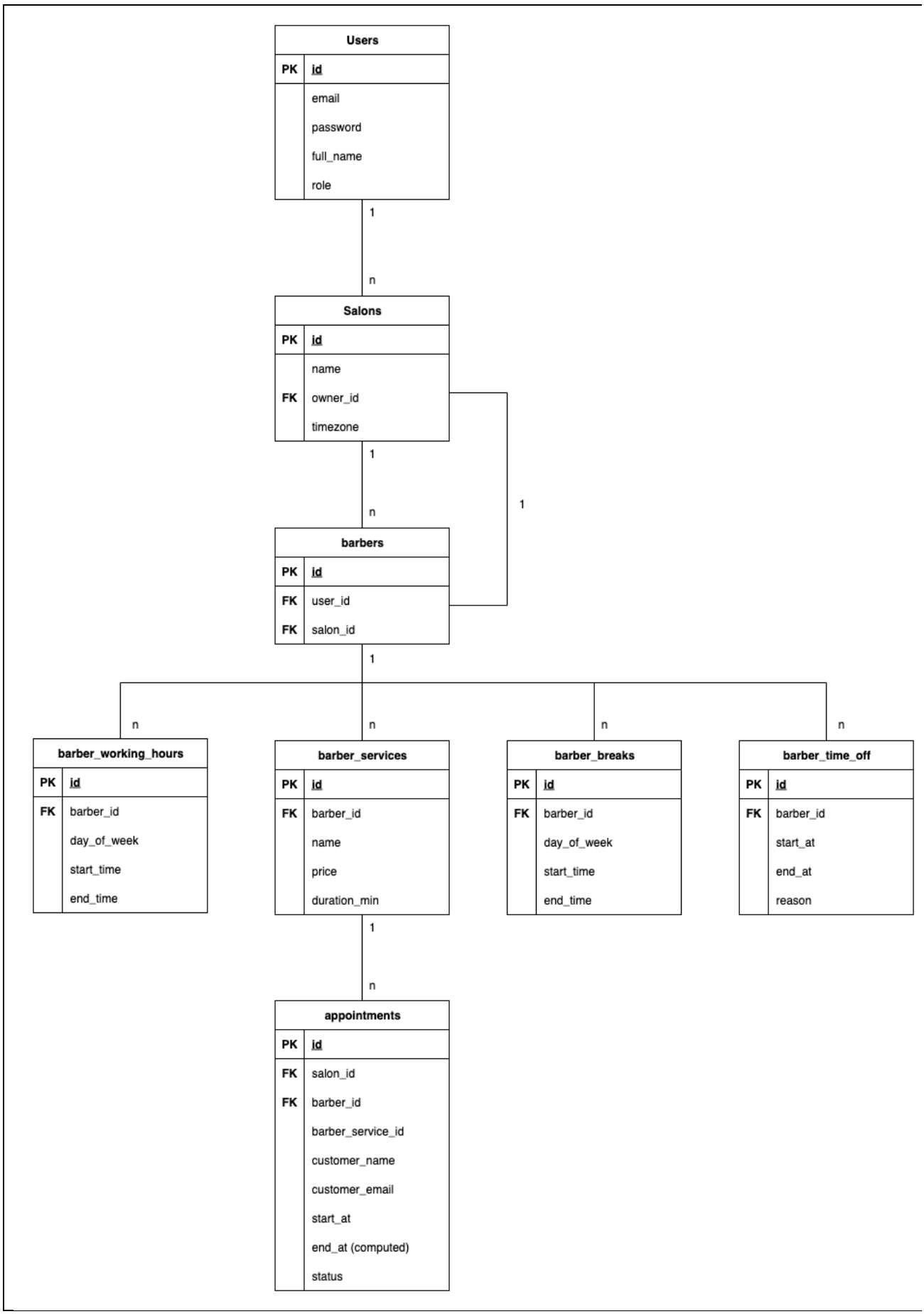
  - BEFORE INSERT OR UPDATE - Извршава се пре уписа у базу
  - OF start_at, duration_min - Активира се само када се мењају ова поља
  - FOR EACH ROW - Извршава се за сваки ред појединачно

Пример рада:

```
-- Клијент заказује термин који почиње у 10:00 и траје 30 минута
INSERT INTO appointments (start_at, duration_min, ...)
VALUES ('2025-01-20 10:00:00', 30, ...);

-- Тригер аутоматски поставља:
-- end_at = '2025-01-20 10:30:00'
```

3.3.7. Дијаграм релација (ER дијаграм)



3.4. Структура Frontend-а

3.4.1. Увод

Frontend BarberBook апликације представља модерну, респонзивну веб апликацију развијену помоћу React 19 библиотеке, Vite build tool-a и Tailwind CSS framework-a за стилизовање. Апликација пружа интуитивно корисничко искуство за три типа корисника: клијенте који заказују термине, власнике салона који управљају пословањем, и фризере који пружају услуге.

3.4.2. Технолошки стек

package.json преглед:

```
{
  "dependencies": {
    "react": "^19.1.1",
    "react-dom": "^19.1.1",
    "react-router-dom": "^6.26.2",
    "tailwindcss": "^4.1.13",
    "@tailwindcss/vite": "^4.1.13"
  },
  "devDependencies": {
    "vite": "^7.1.7",
    "@vitejs/plugin-react": "^5.0.3",
    "eslint": "^9.36.0"
  }
}
```

Објашњење технологија:

- React 19.1.1 - Најновија верзија React библиотеке
    - Component-based архитектура
    - Virtual DOM за брзе перформансе
    - Hooks API за state management
    - Унапређене перформансе и нове функционалности
- React Router DOM 6.26.2 - Рутирање апликације
    - Декларативно дефинисање рута
    - Nested routes подршка
    - Programmatic navigation
    - Protected routes за аутентификацију
- Vite 7.1.7 - Build tool и development server
    - Екстремно брз development server са HMR (Hot Module Replacement)
    - Оптимизован production build
    - Native ES modules подршка
    - Брже од традиционалних bundler-а (Webpack)

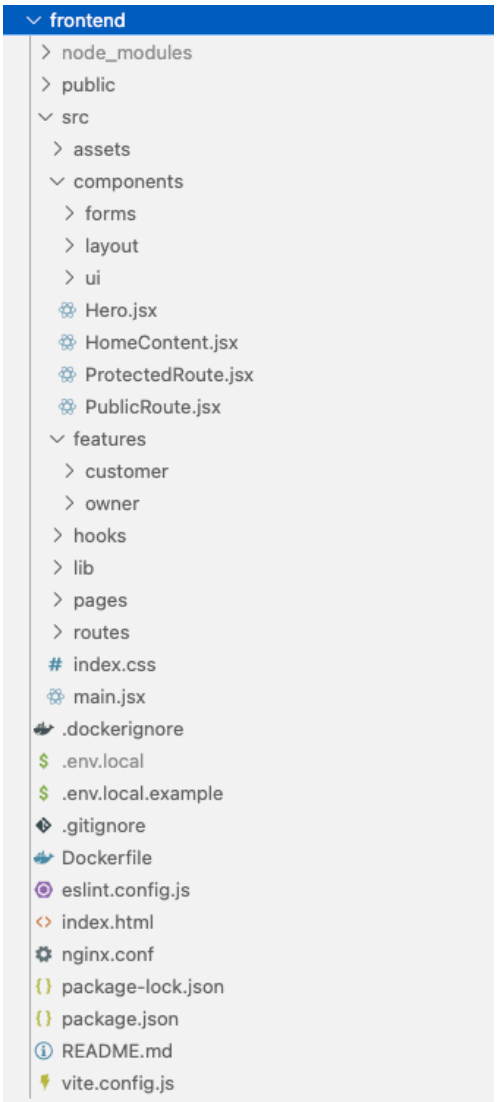
- Tailwind CSS 4.1.13 - Utility-first CSS framework
    - Responsive design out-of-the-box
    - Customizable design system
    - Минимална величина production CSS-a
    - Нема потребе за писањем custom CSS-a

3.4.3. Архитектура апликације

Принципи организације:

- Separation of Concerns - Јасна подела одговорности
- Feature-based структура - Груписање по функционалности
- Reusability - Поновно искоришћиве компоненте у ui/
- Scalability - Лако проширива архитектура

Структура фолдера:



3.4.4. Улазна тачка апликације

main.jsx:

```
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>,
)
```

Објашњење:

- StrictMode - React-ова развојна помоћ
    - Детектује потенцијалне проблеме
    - Упозорава на застареле API-је
    - Проверава нежељене side effects-е
    - Ради само у development режиму
- createRoot - React 18+ API за рендеровање
    - Concurrent rendering подршка
    - Боље перформансе
    - Аутоматски batching updates
- document.getElementById('root') - Монтира апликацију на DOM елемент

3.4.5. Рутирање и навигација

Објашњење рута:

- RootLayout компонента:
    - Омотач за све руте
    - Користи useScrollToTop hook за аутоматски scroll на врх при промени руте
    - <Outlet /> рендерује child руту
- Руте апликације:
    - / - Почетна страница (HomePage)
        - Јавна страница са Hero секцијом
        - Доступна свим корисницима

- /register-salon - Регистрација салона (PublicRoute)
  - Формулар за креирање новог салона
  - Ако је корисник већ пријављен, редиректује на dashboard

- /owner/login - Пријава власника (PublicRoute)
  - Login форма за власнике салона
  - Ако је корисник већ пријављен, редиректује на dashboard

- /owner/dashboard - Dashboard власника (ProtectedRoute)
  - Заштићена рута, захтева аутентификацију
  - Управљање салоном, фризерима, услугама

- /s/:salonId - Јавна страница салона
  - Приказ информација о салону
  - Booking widget за заказивање термина
  - :salonId је динамички параметар (UUID салона)

- - Catch-all рута
  - Редиректује све непостојеће руте на почетну

ProtectedRoute компонента

```
import { Navigate } from 'react-router-dom';
import { getToken } from '../lib/api.js';

export default function ProtectedRoute({ children }) {
  const token = getToken();

  if (!token) {
    return <Navigate to="/owner/login" replace />;
  }

  return children;
}
```

Објашњење:

Проверава да ли корисник има JWT токен у localStorage, ако нема токен редиректује на /owner/login. Ако има токен приказује заштићену страницу, replace prop спречава да се заштићена страница сачува у browser history

PublicRoute компонента

```
import { Navigate } from 'react-router-dom';
import { getToken } from '../lib/api.js';

export default function PublicRoute({ children }) {
  const token = getToken();

  if (token) {
    return <Navigate to="/owner/dashboard" replace />;
  }

  return children;
}
```

Објашњење:

- Проверава да ли корисник има JWT токен
- Ако има токен → редиректује на /owner/dashboard
- Ако нема токен → приказује јавну страницу (login/register)
- Спречава да пријављени корисници приступе login/register страницама

Custom Hook - useScrollToTop

```
import { useEffect } from 'react';
import { useLocation } from 'react-router-dom';

export function useScrollToTop() {
  const { pathname } = useLocation();

  useEffect(() => {
    window.scrollTo(0, 0);
  }, [pathname]);
}
```

Објашњење:

- Аутоматски скролује на врх странице при промени руте
- useLocation() детектује промену pathname-а
- useEffect се извршава при свакој промени pathname
- Побољшава UX - корисник увек види врх нове странице

3.4.6. API комуникација

API utility библиотека

lib/api.js:

```
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;
}
```

Детаљно објашњење:

- Environment Variable:

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

  - Vite користи import.meta.env за environment променљиве
  - Мора почети са VITE_ префиксом
  - Пример: VITE_API_BASE_URL=http://localhost:8080/api

- buildUrl функција:

  - Комбинује BASE_URL са path-ом
  - Аутоматски додаје / ако недостаје
  - Баца грешку ако BASE_URL није постављен

- api функција - параметри:

  - path - API endpoint (нпр. /salons/123)
  - method - HTTP метод (GET, POST, PUT, DELETE)
  - headers - Додатни HTTP headers
  - body - Тело захтева (објекат или FormData)
  - token - JWT токен за аутентификацију

- Аутоматско JSON serialization:

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

  - Ако је body објекат (али не FormData), аутоматски се конвертује у JSON
  - Аутоматски поставља Content-Type: application/json header

- Authorization header:

```
if (token) finalHeaders['Authorization'] = `Bearer ${token}`;
```

  - Додаје JWT токен у Authorization header
  - Format: Bearer <token>

- Error handling:

```
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;
}
```

  - Баца грешку ако статус није 2xx
  - Додаје додатне информације на Error објекат
  - Омогућава детаљан error handling у компонентама

- Примери коришћења:

```
// GET захтев без аутентификације
const salons = await api('/public/salons');

// POST захтев са body-јем
const result = await api('/auth/owner/login', {
  method: 'POST',
  body: { email: 'owner@salon.com', password: 'password123' }
});

// GET захтев са токеном
const mySalon = await api('/owner/me/salon', { token: getToken() });

// PUT захтев са токеном и body-јем
await api(`/salons/${salonId}`, {
  method: 'PUT',
  token: getToken(),
  body: { name: 'Нови назив', phone: '+381...' }
});
```

3.4.7. Странице апликације (Pages)

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 **BarberBook** и служи као први контакт корисника са системом. Компонента **HomePage** је једноставно структурисана 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 концепт и сврху система, **HomeContent** пружа детаљније информације о услугама и функционалностима, док **Footer** садржи основне контакт податке и додатне линкове.

Компонента **HomePage** представља визуелни оквир веб апликације и задужена је за пријатан први утисак корисника, уз једноставан и прегледан дизајн реализован помоћу **Tailwind CSS** класа.

OwnerLoginPage - Пријава власника

```
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-у
              </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>
  );
}
```

Функционалност пријаве власника салона реализована је у оквиру компоненте **OwnerLoginPage**, написане у React-у. Ова компонента представља почетну страницу за све власнике салона који имају свој налог у систему **BarberBook** и омогућава приступ њиховом административном панелу.

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 корисник притисне дугме *„Пријави се“*, активира се функција **onSubmit()**, која има неколико корака:

1. **Спречавање подразумеваног понашања форме:**
   Спречава се освежавање странице позивом `e.preventDefault()`, како би се пријава извршила асинхроно без прекида приказа.
2. **Слање захтева серверу:**
   Функција користи помоћну функцију `api()` да пошаље `POST` захтев на backend руту **`/auth/owner/login`**, при чему се шаљу унети подаци – имејл (претворен у мала слова и без празнина) и лозинка.
3. **Пријем и чување токена:**
   Ако сервер врати позитиван одговор, из JSON одговора се издваја **access_token**, који представља JWT токен за аутентикацију. Тај токен се чува локално помоћу функције `setToken()`, тако да се корисник не мора поново пријављивати током сесије.
4. **Преузимање података о салону:**
   Након успешне пријаве, апликација шаље додатни захтев на руту **`/owner/me/salon`** како би преузела информације о салону који припада пријављеном власнику. Ако је пронађен, `salon_id` се чува у **LocalStorage** како би се подаци могли користити у административном делу апликације.
5. **Навигација ка контролној табли:**
   Када су токен и подаци успешно добијени, корисник се преусмерава на страницу **`/owner/dashboard`**, где може управљати својим салоном, фризерима и услугама.
6. **Обрада грешака:**
   Уколико пријава није успешна (нпр. погрешна лозинка или непостојећи налог), систем хвата грешку и приказује поруку *„Неисправни подаци за пријаву“*.
7. **Визуелни изглед форме:**
   Форма је дизајнирана коришћењем **Tailwind CSS** класа, што обезбеђује модеран и одзиван дизајн.
   Садржи поља за имејл и лозинку, дугме за пријаву, као и обавештење о грешци које се приказује у оквиру црвеног блока када дође до неуспеле пријаве.

Функционалност ове компоненте омогућава једноставан и сигуран приступ систему, при чему свака пријава пролази кроз backend валидацију и користи JWT токен за потврду идентитета власника. На овај начин обезбеђена је потпуна интеграција између frontend-a и backend-a, као и стабилан и безбедан приступ административном делу апликације.

OwnerDashboardPage - Dashboard власника

```
export default function OwnerDashboardPage() {
  const [salon, setSalon] = useState(null);
  const [loading, setLoading] = useState(true);
  const [error, setError] = useState('');
  const [barberCount, setBarberCount] = useState(0);

  useEffect(() => {
    const token = getToken();
    if (!token) {
      setError('Нисте пријављени');
      setLoading(false);
      return;
    }

    let mounted = true;
    setLoading(true);
    setError('');

    api('/owner/me/salon', { token })
      .then((s) => {
        if (!mounted) return;
        setSalon(s);
        if (s?.id) localStorage.setItem('salon_id', s.id);
      })
      .catch((e) => {
        if (!mounted) return;
        setError(e.message || 'Грешка при учитавању салона');
      })
      .finally(() => mounted && setLoading(false));

    return () => { mounted = false; };
  }, []);

  const highlightCards = useMemo(() => {
    const teamLabel = (() => {
      if (!barberCount) return 'Нема фризера';
      if (barberCount === 1) return '1 фризер';
      if (barberCount === 2 || barberCount === 3 || barberCount === 4)
        return `${barberCount} фризера`;
      return `${barberCount} фризера`;
    })();

    return [
      {
        label: 'Адреса',
        value: salon?.address || 'Није унето',
        helper: 'Локација вашег салона',
        icon: '📍',
      },
      {
        label: 'Контакт',
        value: salon?.phone || 'Није унет',
        helper: 'Телефон за клијенте',
        icon: '☎️',
      },
      {
        label: 'Тим',
        value: teamLabel,
        helper: 'Активни фризери у систему',
        icon: '💈',
      },
    ];
  }, [salon?.address, salon?.phone, barberCount]);

  // ... Loading and Error states ...
```

```
  return (
    <Layout>
      <div className="min-h-screen bg-white">
        <div className="mx-auto max-w-6xl px-4 sm:px-6 lg:px-8 py-12">
          <div className="space-y-12">
            {/* Header Section */}
            <section className="overflow-hidden rounded-3xl border border-zinc-200 bg-
white shadow-sm">
              <div className="px-6 py-8 sm:px-10 sm:py-9">
                <div className="flex flex-col gap-6 sm:flex-row sm:items-center
sm:justify-between">
                  <div className="space-y-3">
                    <p className="text-xs font-medium uppercase tracking-[0.35em] text-
zinc-400">
                      Контролна табла
                    </p>
                    <h1 className="text-3xl sm:text-4xl font-semibold tracking-tight
text-zinc-900">
                      {salon?.name}
                    </h1>
                  </div>
                  <Button as="a" href={`/s/${salon.id}`} target="_blank">
                    Отвори јавну страницу
                  </Button>
                </div>

                {/* Highlight Cards */}
                <div className="mt-10 grid gap-4 sm:grid-cols-3">
                  {highlightCards.map((card) => (
                    <div key={card.label} className="rounded-2xl border border-zinc-100
bg-white px-5 py-5">
                      <div className="flex items-start gap-3">
                        <span className="flex h-9 w-9 items-center justify-center
rounded-full bg-zinc-100 text-lg">
                          {card.icon}
                        </span>
                        <div>
                          <p className="text-xs uppercase tracking-[0.35em] text-zinc-
400">{card.label}</p>
                          <p className="mt-2 text-base font-semibold text-zinc-
900">{card.value}</p>
                          <p className="mt-1 text-xs text-zinc-400">{card.helper}</p>
                        </div>
                      </div>
                    </div>
                  ))}
                </div>
              </div>
            </section>

            {/* Salon Details Section */}
            <section>
              <h2 className="text-xl font-semibold text-zinc-900">Подаци о салону</h2>
              <SalonDetailsSection salon={salon} />
            </section>

            {/* Barbers Section */}
            <section>
              <h2 className="text-xl font-semibold text-zinc-900">Тим фризера</h2>
              <BarbersSection salonId={salon.id} onCountChange={setBarberCount} />
            </section>
          </div>
        </div>
      </div>
    </Layout>
  );
}
```

Компонента **OwnerDashboardPage** представља централни део веб апликације **BarberBook** намењен власнику салона. Њена основна улога је да пружи преглед свих кључних информација о салону, члановима тима и подацима који су релевантни за свакодневно пословање.

По учитавању странице, апликација прво проверава да ли је корисник пријављен. Ово се врши помоћу функције **getToken()**, која из локалног складишта узима JWT токен добијен током процеса пријаве. Ако токен не постоји, систем приказује поруку о грешци *„Нисте пријављени“* и зауставља даље учитавање података.

Уколико је токен присутан, апликација шаље захтев ка backend API-ју на руту `/owner/me/salon` ради преузимања података о салону. Приликом успешног одговора, подаци о салону (назив, адреса, број телефона, итд.) се чувају у стању компоненте помоћу `useState`, а ID салона се снима у **LocalStorage**, како би био доступан и другим деловима система.

У случају грешке током учитавања, систем хвата изузетак и приказује поруку као што је *„Грешка при учитавању салона“*.
На овај начин обезбеђена је поуздана обрада свих могућих сценарија, како би се спречио погрешан приказ података.

Компонента користи React hook **useMemo** за динамичко приказивање *информативних картица (highlight cards)* које садрже најважније податке:

- **Адреса салона** – физичка локација салона.
- **Контакт телефон** – број телефона који клијенти користе за контакт.
- **Тим фризера** – приказ броја активних фризера који раде у салону.

Ове картице се аутоматски ажурирају када се промене подаци о салону или када се промени број фризера у систему.

SalonPublicPage - Јавна страница салона

Компонента **SalonPublicPage** представља страницу коју виде клијенти када отворе јавни линк фризерског салона. Њена основна сврха је да прикаже све основне податке о салону (назив, адресу, контакт информације) и омогући клијенту да изврши заказивање термина путем уграђеног *BookingWidget*-а.

- Увоз и иницијализација података

На почетку компоненте користи се React hook `useParams()` који омогућава приступ параметрима из URL адресе:

```
const { salonId } = useParams();
```

Овај параметар `salonId` представља јединствени идентификатор салона и користи се за динамичко преузимање података са backend-а.

Затим се дефинишу основна стања компоненте:

```
const [salon, setSalon] = useState(null);
const [loading, setLoading] = useState(true);
const [error, setError] = useState('');
```

- salon чува објекат са подацима о салону који је добијен са сервера.
- loading контролише приказ индикатора током учитавања.
- error чува текст поруке у случају грешке.

- Преузимање података о салону

Коришћењем React hook-а `useEffect()`, апликација аутоматски шаље захтев ка backend API-ју чим се страница учита:

```
  useEffect(() => {
    let mounted = true;
    setLoading(true);
    setError('');

    api(`/public/salons/${salonId}`)
      .then((data) => { if (mounted) setSalon(data); })
      .catch((e) => { if (mounted) setError(e.message || 'Грешка'); })
      .finally(() => { if (mounted) setLoading(false); });

    return () => { mounted = false; };
  }, [salonId]);
```

- Користи се прилагођена функција `api()` за комуникацију са сервером.
- На основу идентификатора `salonId`, шаље се захтев на backend руту **`/public/salons/:id`**.
- При успешном одговору, подаци се чувају у променљиви `salon`.
- У случају грешке (нпр. салон не постоји или сервер није доступан), исписује се порука „Грешка“.
- Променљива `mounted` спречава покушај ажурирања стања након размонтирања компоненте (добра пракса у React-у).

Овим се обезбеђује да се подаци о сваком салону учитавају динамички, у зависности од тога који линк је клијент отворио.

- Приказ података о салону

Након што се подаци успешно учитају, они се приказују у визуелном делу компоненте:

```
<div className="mb-12 text-center">
            <h1 className="text-4xl sm:text-5xl font-bold text-gray-900 mb-4">
              {salon.name}
            </h1>
            <div className="space-y-2 text-gray-600">
              <div>{salon.address}</div>
              <div>{salon.phone}</div>
              <div className="text-sm">
                {salon.timezone} • {salon.currency}
              </div>
            </div>
          </div>
```

Овде се приказују следеће информације:

- **Назив салона** у великом и упадљивом фонту.
- **Адреса** и **број телефона**, како би клијент лако ступио у контакт.
- **Временска зона** и **валута**, што може бити корисно за међународне клијенте.

Сви елементи су визуелно центрирани и стилизовани помоћу **Tailwind CSS** класа како би изглед био модеран и прегледан.

- Уграђен систем за заказивање (Booking Widget)

На крају компоненте налази се део који омогућава клијентима да директно закажу термин:

```
    {/* Booking Widget */}
          <div className="max-w-4xl mx-auto">
            <BookingWidget salonId={salonId} />
          </div>
```

Компонента **BookingWidget** прима као параметар `salonId` и на основу тога приказује све доступне услуге, датуме и временске термине које је фризер подесио у свом профилу. Корисник може да одабере услугу, датум, време, унесе своје податке (име, е-пошту, број телефона) и потврди резервацију.

Овај механизам је директно повезан са backend API-јем, који обрађује податке и чува резервацију у PostgreSQL бази.

3.4.8. Feature компоненте

BookingWidget - Widget за заказивање

Ово је најкомплекснија компонента у апликацији. Implementира multi-step booking процес.

```
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' },
  ];

  // ... Implementation ...
}
```

Детаљно објашњење:

- Multi-step wizard pattern:
  - 5 корака у booking процесу
  - Секвенцијални flow
  - Validation на сваком кораку

- Cascade updates:

```
function handleSelectBarber(id) {
  setSelectedBarber(id);
  setSelectedService(null);  // Reset зависних поља
  setSlots([]);
  setSelectedSlot(null);
  setStep(2);
}
```

  - Промена фризера resetује услуге, датум, термине
  - Спречава inconsistent state

- Dynamic service loading:

```
useEffect(() => {
  if (!selectedBarber) {
    setServices([]);
    return;
  }
  api(`/public/barbers/${selectedBarber}/services`)
    .then((data) => setServices(data))
    .catch((e) => setError(e.message));
}, [selectedBarber]);
```

  - Учитава услуге када се изабере фризер
  - Празни array ако нема фризера

- Availability checking:

```
useEffect(() => {
  if (!selectedBarber || !selectedService || !date) {
    setSlots([]);
    return;
  }
  setLoadingSlots(true);

api(`/public/barbers/${selectedBarber}/services/${selectedService}/availability?date=${
date}`)
    .then((data) => {
      setSlots(Array.isArray(data.slots) ? data.slots : []);
      if (data.is_time_off && data.time_off_info) {
        setTimeOffInfo(data.time_off_info);
      }
    })
    .finally(() => setLoadingSlots(false));
}, [selectedBarber, selectedService, date]);
```

  - Проверава доступне термине
  - Loading state за UX
  - Обрађује одсуства фризера

- Booking submission:

```
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);
  }
}
```

  - Валидација података
  - Error handling
  - Success state за confirmation screen

3.5. Структура андроид апликације

Android апликација BarberBook представља нативну мобилну апликацију развијену у React Native оквиру, намењену искључиво за фризере који управљају својим услугама, радним временом, одсуствима и терминима. Апликација обезбеђује брз и интуитиван приступ свим кључним функционалностима директно са мобилног уређаја.

3.5.1. Технолошки стек

Основни framework и библиотеке

```
{
  "react": "19.1.0",
  "react-native": "0.81.4",
  "typescript": "^5.8.3"
}
```

- React Native 0.81.4 - Најновија стабилна верзија која омогућава развој нативних апликација користећи JavaScript/TypeScript
- React 19.1 - Најновија верзија React библиотеке са побољшаним перформансама
- TypeScript 5.8 - Типизација која обезбеђује сигурност кода и бољи developer experience

3.5.2. Кључне зависности

```
"dependencies": {
  "@react-native-async-storage/async-storage": "^2.1.1",
  "@react-native-community/datetimepicker": "^8.4.5",
  "react-native-dotenv": "^3.4.11",
  "react-native-linear-gradient": "^2.8.3",
  "react-native-safe-area-context": "^5.5.2",
  "react-native-vector-icons": "^10.3.0"
}
```

Детаљан опис библиотека:

- AsyncStorage

Persistent storage за чување JWT токена и корисничких података:

```
await AsyncStorage.multiSet([
  ['auth_token', token],
  ['auth_token_type', tokenType],
]);
```

- DateTimePicker

Нативни date picker за избор датума:

```
<DateTimePicker
  value={selectedDate}
  mode="date"
  display="default"
  onChange={handleDateChange}
/>
```

- react-native-dotenv

Environment променљиве за конфигурацију:

```
import { API_BASE_URL } from '@env';
```

- Linear Gradient

Модерни gradient ефекти за позадине:

```
<LinearGradient
  colors={['#000000', '#1C1C1E']}
  start={{ x: 0, y: 0 }}
  end={{ x: 1, y: 1 }}
>
  {children}
</LinearGradient>
```

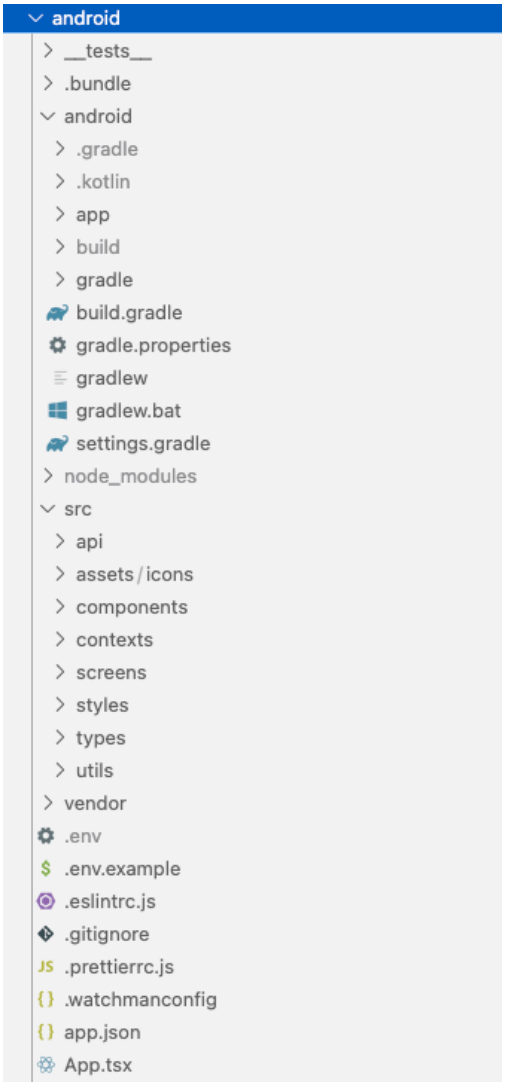
- Safe Area Context

Аутоматско управљање notch-ем и safe areas:

```
<SafeAreaView edges={['bottom']}>
  {content}
</SafeAreaView>
```

3.5.3. Архитектура апликације

Структура пројекта:



3.5.4. Аутентификација

AuthContext имплементација

AuthContext обезбеђује глобално стање аутентификације:

```
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:

```
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 });
}, []);
```

Објашњење:

- Email нормализација - Претварање у мала слова и уклањање whitespace-а
- API захтев - POST на /auth/barber/login endpoint
- Persistent storage - Чување токена у AsyncStorage за поновну употребу
- State update - Ажурирање React стања за UI реакцију

Аутоматска пријава при покретању

```
useEffect(() => {
  let isMounted = true;

  // Читање токена из AsyncStorage
  AsyncStorage.multiGet([STORAGE_TOKEN_KEY, STORAGE_TOKEN_TYPE_KEY])
    .then(entries => {
      if (!isMounted) return;

      const storedToken = entries.find(
        ([key]) => key === STORAGE_TOKEN_KEY
      )?.[1] ?? null;

      const storedType = entries.find(
        ([key]) => key === STORAGE_TOKEN_TYPE_KEY
      )?.[1] ?? undefined;

      setState({
        token: storedToken,
```

```
        tokenType: storedType || 'Bearer',
        isLoading: false,
      });
    })
    .catch(() => {
      if (!isMounted) return;
      setState(prev => ({ ...prev, isLoading: false }));
    });

  return () => { isMounted = false; };
}, []);
```

Објашњење:

- Mounted flag - Спречава memory leak ако се компонента unmount-yje
- multiGet - Ефикасно читање више вредности одједном
- Error handling - Graceful fallback ако читање не успе

3.5.5. API слој

HTTP клијент

```
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-а
  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;
}
```

Кључне особине:

- Generic типизација - request<T> за type-safe одговоре
- Аутоматска JSON серијализација - Претварање body-ја у JSON
- Authorization header - Аутоматско додавање JWT токена
- Error handling - Структурисане грешке са ApiError класом
- AbortSignal подршка - Могућност отказивања захтева

3.5.6. Главни екрани апликације

LoginScreen - Екран за пријаву

Функционалности:

- Унос email адресе и лозинке
- Валидација форме на клијенту
- Loading индикатор током пријаве
- Приказ грешака и успешне пријаве
- KeyboardAvoidingView за iOS

Имплементација валидације:

```
const validateForm = useCallback(() => {
  const trimmedEmail = email.trim().toLowerCase();
  const emailRegex = /[^\s@]+@[^\s@]+\.[^\s@]+/;

  if (!trimmedEmail || !emailRegex.test(trimmedEmail)) {
    setError('Unesite ispravnu email adresu.');
    return false;
  }

  if (!password || password.length < 6) {
    setError('Lozinka mora imati najmanje 6 karaktera.');
    return false;
  }

  return true;
}, [email, password]);
```

Submit handler:

```
const handleSubmit = useCallback(async () => {
  if (loading || authLoading) return;

  setError(null);
  setInfo(null);

  if (!validateForm()) return;

  setLoading(true);

  try {
    await login(email, password);
    setInfo('Uspešno ste prijavljeni.');
    setEmail('');
    setPassword('');
  } catch (e) {
    if (e instanceof ApiError) {
      setError(e.message || 'Neuspešna prijava. Proverite podatke.');
    } else {
      setError('Došlo je do greške prilikom prijave.');
    }
  } finally {
    setLoading(false);
  }
}, [authLoading, email, loading, login, password, validateForm]);
```

ServicesScreen - Управљање услугама

Функционалности:

- Листа свих услуга фризера
- Додавање нове услуге
- Измена постојеће услуге
- Брисање услуге
- Toggle активности услуге
- Pull-to-refresh
- Празно стање када нема услуга

Структура података:

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

CRUD операције:

```
// 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]);
```

Овај део кода представља три основне функције које омогућавају фризеру да управља услугама у систему **BarberBook**. Реализоване су помоћу React hook-a `useCallback()`, који обезбеђује оптимизацију и спречава непотребно поновно креирање функција при сваком ререндеру компоненте.

Функција **handleCreateService** служи за додавање нове услуге у систем. Када фризер унесе податке о услузи и потврди, апликација шаље захтев серверу, а након успешног креирања, нова услуга се додаје у постојећу листу услуга. Истовремено се затвара модални прозор за додавање и ресетују се евентуалне грешке.

Функција **handleUpdateService** омогућава измену података постојеће услуге, као што су назив, цена, трајање или статус (активна/неактивна). Након измене, апликација ажурира приказ тако што проналази услугу по идентификатору и замењује је новом верзијом добијеном са сервера.

Функција **handleDeleteService** служи за брисање услуге из система. Након што се позове, шаље се захтев серверу за брисање услуге по њеном идентификатору, а затим се услуга уклања и из локалне листе без потребе за поновним учитавањем странице.

Овим функционалностима обезбеђено је потпуно управљање услугама у оквиру фризерског салона — од додавања нових, преко измене постојећих, до брисања оних које више нису актуелне.

ScheduleScreen - Радно време

Функционалности:

- Приказ радног времена за сваки дан у недељи
- Измена почетка и краја радног времена
- Toggle радног статуса (ради/не ради)
- Додавање пауза током дана
- Брисање пауза

Структура података:

```
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;
}
```

Дневни преглед:

```
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}
    />
  );
})}
```

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 (**dayHours**) и паузе (**dayBreaks**), који се затим приказују у одговарајућој картици.
Компонента **DayCard** омогућава фризеру да ажурира радно време, дода или обрише паузу.

TimeOffScreen - Одсуства

Функционалности:

- Листа свих планираних одсуства
- Додавање новог одсуства
- Брисање одсуства
- Приказ разлога одсуства
- Сортирање по датуму

Структура података:

```
interface BarberTimeOff {
  id: string;
  barberId: string;
  salonId: string;
  startAt: string; // ISO timestamp
  endAt: string;   // ISO timestamp
  reason?: string;
  createdAt: string;
  updatedAt: string;
}
```

Форма за додавање одсуства:

```
<Modal visible={showAddModal} transparent>
  <View style={styles.modalContent}>
    <Text style={styles.modalTitle}>Dodaj odsustvo</Text>

    {/* Датум почетка */}
    <TouchableOpacity onPress={() => setShowStartPicker(true)}>
      <Text>Od: {formatDate(startDate)}</Text>
    </TouchableOpacity>

    {showStartPicker && (
      <DateTimePicker
        value={startDate}
        mode="date"
        onChange={(event, date) => {
          setShowStartPicker(false);
          if (date) setStartDate(date);
        }}
      />
    )}

    {/* Датум краја */}
    <TouchableOpacity onPress={() => setShowEndPicker(true)}>
      <Text>Do: {formatDate(endDate)}</Text>
    </TouchableOpacity>

    {/* Разлог */}
    <TextInput
      placeholder="Razlog (opciono)"
      value={reason}
      onChangeText={setReason}
```

```
      multiline
    />

    <Button title="Sačuvaj" onPress={handleSubmit} />
  </View>
</Modal>
```

Овај део кода приказује модални прозор (**Modal**) у мобилној апликацији који служи за додавање периода одсуства фризера.

Корисник може да изабере **датум почетка** и **датум краја** одсуства помоћу `DateTimePicker` компоненти, као и да унесе **разлог** (опционо).
Након уноса података, кликом на дугме **„Сачувај“** позива се функција `handleSubmit`, која чува унето одсуство у систему.

Овим се фризеру омогућава да лако подеси нерадне дане, што се касније одражава на календар заказивања.

AppointmentsScreen - Термини

Функционалности:

- Приказ термина за изабрани датум
- Календар са date picker-ом
- Timeline view са временским слотовима
- Креирање новог термина
- Потврђивање термина (pending → confirmed)
- Отказивање термина
- Брисање термина
- Филтрирање само активних термина

Структура података:

```
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;
}
```

Timeline генерисање:

```
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);
```

Овај део кода служи за **генерисање временских слотова** у календару заказивања, односно за креирање листе доступних и заузетих термина током дана.

На основу радног времена фризера (**todayWorkingHours**) и подешеног трајања термина (**slotDuration**), код израчунава сваки појединачни временски интервал (нпр. сваких 15 минута). За сваки слот проверава се да ли се преклапа са неким постојећим заказаним термином (**appointments**).

Ако се слот поклапа са постојећим заказивањем, у објекат се додаје информација о тој резервацији, у супротном остаје слободан.

На крају се сви генерисани слотови чувају у стању компоненте (**setTimeSlots(slots)**), како би се могли приказати у интерфејсу за избор термина.

Rendering timeline-a:

```
<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>
```

Овај део кода приказује **временску линију термина** у мобилној апликацији.

Помоћу `ScrollView` компоненте приказује се листа свих временских слотова (**timeSlots**) за изабрани дан.
За сваки слот приказује се време, а затим:

- ако постоји заказан термин (**slot.appointment**), приказује се картица са подацима о клијенту, услузи и трајању,
- ако нема заказивања, приказује се дугме „Слободно“ које омогућава фризеру да директно креира нови термин.

Стилизација омогућава визуелну разлику између **потврђених**, **чекајућих** и **слободних** термина, што фризеру пружа јасан и прегледан приказ распореда у току дана.

Акције над терминима:

```
// 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 otkažete 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);
          }
        },
      },
    ]
  );
};
```

Овај део кода управља акцијама над заказаним терминима у мобилној апликацији.

Функција **handleConfirm** служи за потврду термина – шаље захтев серверу да промени статус заказивања на „потврђен“, освежава листу термина и приказује поруку о успеху.

Функција **handleCancel** омогућава отказивање термина. Пре извршења, приказује се дијалог за потврду. Ако корисник потврди, термин се отказује и уклања из активне листе.

Функција **handleDelete** брише термин из система. Такође приказује дијалог упозорења, а након потврде шаље захтев серверу за брисање и ажурира приказ.

Све функције користе **Alert** за приказ обавештења и **setActionLoading** за контролу стања током извршавања акције.

3.5.7. Навигација и UI компоненте

TopNavBar - Hamburger мени

Функционалности:

- Hamburger икона у горњем десном углу
- Sliding drawer анимација
- Листа навигационих ставки
- Logout дугме
- Backdrop overlay

Анимирани drawer:

```
const slideAnim = React.useRef(new Animated.Value(screenWidth)).current;
const opacityAnim = React.useRef(new Animated.Value(0)).current;

const openMenu = () => {
  setMenuVisible(true);
  Animated.parallel([
    Animated.timing(slideAnim, {
      toValue: screenWidth - menuWidth,
      duration: 300,
      useNativeDriver: false,
    }),
    Animated.timing(opacityAnim, {
      toValue: 1,
      duration: 300,
      useNativeDriver: false,
    }),
  ]).start();
};

const closeMenu = () => {
  Animated.parallel([
    Animated.timing(slideAnim, {
      toValue: screenWidth,
      duration: 250,
      useNativeDriver: false,
    }),
    Animated.timing(opacityAnim, {
      toValue: 0,
      duration: 250,
      useNativeDriver: false,
    }),
  ]).start(() => {
    setMenuVisible(false);
  });
};
```

Овај део кода реализује **анимацију бочног менија (sidebar)** у мобилној апликацији.

Користе се две **Animated** вредности —

- `slideAnim` контролише **померање менија по хоризонтали**,

- `opacityAnim` контролише **провидност позадине** током отварања и затварања.

Функција **openMenu** покреће две анимације истовремено: мени клизи са десне стране екрана ка видљивом делу, а позадина постепено постаје видљивија.

Функција **closeMenu** ради супротно — враћа мени ван екрана и смањује провидност, након чега сакрива мени постављањем `setMenuVisible(false)`.

Овим се постиже глатка и природна анимација при отварању и затварању навигационог менија.

Dashboard - Главни контејнер

```
function Dashboard() {
  const [activeTab, setActiveTab] = useState<TabKey>('appointments');
  const { logout } = useAuth();

  return (
    <LinearGradient
      colors={[palette.systemBackground, palette.secondarySystemBackground]}
      style={styles.dashboardGradient}
    >
      <SafeAreaView style={styles.dashboardSafeArea} edges={['bottom']}>
        <TopNavBar
          navItems={NAV_ITEMS}
          activeItem={activeTab}
          onItemPress={(key) => setActiveTab(key as TabKey)}
          onLogout={logout}
        />
        <View style={styles.screenWrapper}>
          {renderScreen(activeTab)}
        </View>
      </SafeAreaView>
    </LinearGradient>
  );
}

function renderScreen(tab: TabKey) {
  switch (tab) {
    case 'services':
      return <ServicesScreen />;
    case 'working-hours':
      return <ScheduleScreen />;
    case 'time-off':
      return <TimeOffScreen />;
    case 'appointments':
      return <AppointmentsScreen />;
    default:
      return <AppointmentsScreen />;
  }
}
```

Овај код представља **главни екран (Dashboard)** мобилне апликације за фризере.

Компонента садржи **навигациони бар (TopNavBar)** са више табова — *appointments*, *services*, *working-hours* и *time-off*.
Кроз промењиву **activeTab** прати се који је таб тренутно активан, а функција **renderScreen()** приказује одговарајући екран у зависности од избора.

Позадина екрана је стилизована помоћу **LinearGradient**, а целокупни садржај је смештен у **SafeAreaView** ради бољег приказа на различитим уређајима.

На овај начин, фризер једноставно прелази између секција за термине, услуге, распоред и одсуства, све унутар јединственог интерфејса.

3.5.8. TypeScript типови

```
export type AppointmentStatus = 'pending' | 'confirmed' | 'canceled';

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

export interface BarberWorkingHour {
  id: string;
  barberId: string;
  salonId: string;
  dayOfWeek: number;
  startTime: string;
  endTime: string;
  createdAt: string;
  updatedAt: string;
}

export interface BarberTimeOff {
  id: string;
  barberId: string;
  salonId: string;
  startAt: string;
  endAt: string;
  reason?: string;
  createdAt: string;
  updatedAt: string;
}

export interface Appointment {
```

```
  id: string;
  salonId: string;
  barberId: string;
  barberServiceId: string;
  customerName: string;
  customerPhone?: string;
  customerEmail?: string;
  price: number;
  durationMin: number;
  startAt: string;
  endAt: string;
  status: AppointmentStatus;
  notes?: string;
  createdAt: string;
  updatedAt: string;
}

export interface BarberProfile {
  id: string;
  userId: string;
  salonId: string;
  displayName: string;
  active: boolean;
  slotDurationMinutes: number;
  createdAt: string;
  updatedAt: string;
}
```

Овај део кода представља **TypeScript моделе (интерфејсе)** који дефинишу структуру података у систему **BarberBook**. Они служе за типизацију и јасно описују како изгледају објекти који се користе у апликацији.

Ево кратког објашњења сваке структуре:

- **AppointmentStatus** — дефинише три могућа статуса термина: *pending* (на чекању), *confirmed* (потврђен), и *canceled* (отказан).
- **BarberService** — описује услугу коју фризер нуди (назив, цена, валута, трајање, активност, време креирања и измене).
- **BarberWorkingHour** — представља радно време фризера по данима у недељи, са временом почетка и завршетка рада.
- **BarberTimeOff** — чува податке о одсуству фризера: период трајања (од – до) и разлог одсуства.
- **Appointment** — описује термин који је клијент закажао: податке о фризеру, салону, изабраној услузи, клијенту, времену термина и статусу.
- **BarberProfile** — представља профил фризера у систему (име, статус активности, подешено трајање временског слота, и основне идентификаторе).

Ови интерфејси обезбеђују јасну структуру података и лакше одржавање кода, као и сигурност приликом размене података између frontend-а и backend-а.

3.5.9. Закључак

Android апликација BarberBook представља професионално решење за управљање фризерским салоном са мобилног уређаја. Кључне карактеристике:

- Технолошке предности
  - React Native 0.81 - Нативне перформансе са JavaScript развојем
  - TypeScript - Type-safety и бољи developer experience
  - Modern React patterns - Hooks, Context API, functional components
  - Apple-inspired UI - Професионалан и минималистички дизајн
- Функционалне могућности за фризере:
  - Управљање услугама (CRUD)
  - Постављање радног времена
  - Дефинисање одсуства
  - Преглед и управљање терминима
  - Timeline приказ дневних термина
- Корисничко искуство
  - Интуитивна навигација - Hamburger мени са анимацијама
  - Брзо учитавање - Pull-to-refresh на свим екранима
  - Offline-first - AsyncStorage за persistent state
  - Error handling - Јасне поруке о грешкама
  - Loading states - Индикатори прогреса
- Безбедност
  - JWT аутентификација - Сигуран token-based приступ
  - AsyncStorage - Енкриптовано чување осетљивих података
  - HTTPS only - Сва комуникација преко сигурне конекције
  - Type validation - TypeScript спречава runtime грешке
- Будућа побољшања
  - iOS верзија - Проширење на iOS платформу
  - Push notifications - Реал-тајм обавештења о новим терминима
  - Analytics - Статистика и извештаји
  - Multi-language - Подршка за више језика
  - Dark mode - Тамна тема за ноћну употребу

3.6. Структура Docker-а

3.6.1. Увод

BarberBook систем користи Docker и Docker Compose за једноставно deployовање и покретање свих компоненти апликације (Backend, Frontend, PostgreSQL база) у изолованим контејнерима. То је веома битно због тога што комплетан пројекат вероватно не би радио када би хтели да га покренемо на другом рачунару.

3.6.2. Docker Compose архитектура

```yaml
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
```

Објашњење:

- services - Дефинише 3 сервиса: база, backend, frontend
- volumes - Persistent storage за PostgreSQL податке
- networks - Bridge network за комуникацију између контејнера
- healthcheck - Проверава да ли је база спремна пре покретања backend-а
- depends_on - Дефинише редослед покретања (база → backend → frontend)

3.6.3. Backend Dockerfile

```
# 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"]
```

Објашњење:

- Stage 1 (Builder) - Компајлира Go апликацију
    - Користи golang:1.21-alpine (мала слика)
    - go mod download - Преузима зависности
    - CGO_ENABLED=0 - Статички build без C библиотека
    - Резултат: извршна main датотека
- Stage 2 (Production) - Финални контејнер
    - Користи alpine:latest (само ~5MB)
    - Копира само извршну датотеку (не цео Go toolchain)
    - Копира миграције и .env конфигурацију
    - Предност: Мали image (~20MB уместо ~300MB)
- entrypoint.sh - Опционални startup script:

```
#!/bin/sh
set -e

# Чека да база буде спремна
until pg_isready -h db -p 5432 -U barberbook; do
  echo "Waiting for database..."
  sleep 2
done

# Покреће миграције
./migrate

# Покреће апликацију
exec ./main
```

3.6.4. Frontend Dockerfile

```
# 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;"]
```

Објашњење:

- Stage 1 (Builder) - Build React апликације
    - npm ci - Clean install (брже од npm install)
    - npm run build - Vite генерише оптимизоване static фајлове у dist/
- Stage 2 (Nginx) - Служи static фајлове
    - nginx:alpine (само ~40MB)
    - Копира dist/ у nginx root
    - Резултат: Брзо служење static садржаја

nginx.conf

```
server {
    listen 80;
    server_name localhost;
    root /usr/share/nginx/html;
    index index.html;

    # Gzip компресија
    gzip on;
```

```
    gzip_types text/plain text/css application/json application/javascript text/xml application/xml;
    gzip_min_length 1000;

    # SPA рутирање - све усмери на index.html
    location / {
        try_files $uri $uri/ /index.html;
    }

    # Cache за static assets (1 година)
    location ~* \.(js|css|png|jpg|jpeg|gif|ico|svg|woff|woff2)$ {
        expires 1y;
        add_header Cache-Control "public, immutable";
    }

    # API proxy (опционо)
    location /api {
        proxy_pass http://backend:8080;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
    }
}
```

- try_files - React Router подршка (све руте иду на index.html)
- gzip - Компресија одговора (~70% смањење величине)
- expires 1y - Browser cache за static фајлове
- proxy_pass - Опционо проксирање API захтева

3.6.5. Покретање система

Скрипте за управљање

start-docker.sh:

```
#!/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"
```

Овај **Bash** скрипт служи за аутоматско покретање целог **BarberBook** система у Docker окружењу.

Прво исписује поруку о покретању, затим помоћу команде
`docker-compose up -d --build`
покреће све контејнере (frontend, backend и базу података) и гради их ако је потребно.

Након тога чека неколико секунди да се сви сервиси покрену, проверава њихов статус командом `docker-compose ps`, и на крају исписује поруке са линковима ка фронтенду, backend API-ју и PostgreSQL бази.

Овим се омогућава једноставно и брзо покретање целе апликације једном командом.

stop-docker.sh:

```
#!/bin/bash

echo "🛑 Заустављање BarberBook система..."

docker-compose down

echo "✅ Систем је заустављен!"
```

logs-docker.sh:

```
#!/bin/bash

# Приказује логове свих сервиса
docker-compose logs -f
```

Команде:

```
# Покретање (први пут)
docker-compose up -d --build

# Заустављање
docker-compose down

# Преглед логова
docker-compose logs -f

# Рестартовање једног сервиса
docker-compose restart backend

# Улазак у контејнер
docker exec -it barberbook-backend sh

# Брисање свега (укључујући volumes)
docker-compose down -v
```

3.6.6. Environment променљиве

Backend .env:

```
# Сервер
SERVER_PORT=8080

# База података
DATABASE_URL=postgresql://user:password@db:5432/barberbook?sslmode=disable

# JWT
JWT_SECRET=your-super-secret-key-change-in-production

# Email (опционо)
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USERNAME=your-email@gmail.com
SMTP_PASSWORD=your-app-password
```

Frontend .env.local:

```
VITE_API_BASE_URL=http://localhost:8080
```

3.6.7. Закључак

Docker контејнеризација система **BarberBook** представља кључни корак ка професионалној и стабилној имплементацији апликације.
Коришћењем Docker-а омогућено је да се читав систем — база података, backend и frontend — покрену јединственом командом, без потребе за ручним подешавањем сваког сервиса посебно.

Оваква архитектура обезбеђује потпуну **изолацију окружења**, што значи да сваки део система ради независно и без утицаја на остале.
Поред тога, **конзистентност** између развојног, тестног и продукционог окружења гарантује да апликација функционише исто на сваком серверу, без обзира на оперативни систем.

Docker омогућава лако **скалирање система**, јер се број инстанци сервиса може повећати у складу са оптерећењем.
Захваљујући **persistent storage-у**, подаци у PostgreSQL бази остају сачувани чак и након рестартовања контејнера.

Цео BarberBook систем је тиме постао **production-ready**, спреман за једноставно постављање на било који сервер који подржава Docker, уз поуздан рад, лако одржавање и брзо ажурирање компоненти.

4. Закључак

Овај завршни рад представља детаљан приказ процеса израде софтверског система **„BarberBook“**, 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. Остварењем тог циља, показана је примена савремених технологија, архитектонских принципа и добрих пракси у развоју реалних, продукционо спремних софтверских решења.

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 – покрећу се јединственом 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.

5. 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