УНИВЕРЗИТЕТ У НОВОМ САДУ
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У НОВОМ САДУ

Данило Цвијетић

Архитектура и имплементација система за управљање теретаном

ЗАВРШНИ РАД

Основне академске студије

Нови Сад, 2025

	УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА** 21000 НОВИ САД, Трг Доситеја Обрадовића 6	Број:
	ЗАДАТАК ЗА ЗАВРШНИ РАД	Датум:

(Податке уноси предметни наставник - ментор)

Студијски програм:	Софтверско инжењерство и информационе технологије		
Студент:	Данило Цвијетић	Број индекса:	SV25/2021
Степен и врста студија:	Основне академске студије		
Област:	Електротехничко и рачунарско инжењерство		
Ментор:	Игор Дејановић		

НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:

- проблем – тема рада;
- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;

НАСЛОВ ЗАВРШНОГ РАДА:

Архитектура и имплементација система за управљање теретаном

ТЕКСТ ЗАДАТКА:

Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magnam aliquam quaerat voluptatem. Ut enim aeque doleamus animo, cum corpore dolemus, fieri tamen permagna accessio potest, si aliquod aeternum et infinitum impendere malum nobis opinemur. Quod idem licet transferre in voluptatem, ut postea variari voluptas distinguique possit, augeri amplificarique non possit. At etiam Athenis, ut e patre audiebam facete et urbane Stoicos irridente, statua est in quo a nobis philosophia defensa et collaudata est, cum id, quod maxime placeat, facere possimus, omnis voluptas assumenda est, omnis dolor repellendus. Temporibus autem quibusdam et.

Руководилац студијског програма:	Ментор рада:

Примерак за: □ - Студента; □ - Ментора

УНИВЕРЗИТЕТ У НОВОМ САДУ ● **ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА**
21000 НОВИ САД, Трг Доситеја Обрадовића 6

КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, **РБР**:	
Идентификациони број, **ИБР**:	
Тип документације, **ТД**:	Монографска документација
Тип записа, **ТЗ**:	Текстуални штампани материјал
Врста рада, **ВР**:	Дипломски - бечелор рад
Аутор, **АУ**:	Данило Цвијетић
Ментор, **МН**:	Др Игор Дејановић, редовни професор
Наслов рада, **НР**:	Архитектура и имплементација система за управљање теретаном
Језик публикације, **ЈП**:	српски/ћирилица
Језик извода, **ЈИ**:	српски/енглески
Земља публиковања, **ЗП**:	Република Србија
Уже географско подручје, **УГП**:	Војводина
Година, **ГО**:	2025
Издавач, **ИЗ**:	Ауторски репринт
Место и адреса, **МА**:	Нови сад, трг Доситеја Обрадовића 6
Физички опис рада, **ФО**: (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/52/18/2/23/0/23
Научна област, **НО**:	Електротехничко и рачунарско инжењерство
Научна дисциплина, **НД**:	Примењене рачунарске науке и информатика
Предметна одредница/Кључне речи, **ПО**:	Шаблон, завршни рад, упутство
УДК	
Чува се, **ЧУ**:	У библиотеци Факултета техничких наука, Нови Сад
Важна напомена, **ВН**:	
Извод, **ИЗ**:	Овај документ представља упутство за писање завршних радова на Факултету техничких наука Универзитета у Новом Саду. У исто време је и шаблон за Typst.
Датум прихватања теме, **ДП**:	
Датум одбране, **ДО**:	01.01.2025
Чланови комисије, **КО**: Председник:	Др Петар Петровић, ванредни професор
Члан:	Др Марко Марковић, доцент
Члан, ментор:	Др Игор Дејановић, редовни професор

Потпис ментора

Образац **Q2.НА.04-05** - Издање 1

UNIVERSITY OF NOVI SAD ● **FACULTY OF TECHNICAL SCIENCES**
21000 NOVI SAD, Trg Dositeja Obradovića 6

KEY WORDS DOCUMENTATION

Accession number, **ANO**:	
Identification number, **INO**:	
Document type, **DT**:	Monographic publication
Type of record, **TR**:	Textual printed material
Contents code, **CC**:	
Author, **AU**:	Danilo Cvijetić
Mentor, **MN**:	Igor Dejanović, Phd., full professor
Title, **TI**:	Architecture and Implementation of a Gym Management System
Language of text, **LT**:	Serbian
Language of abstract, **LA**:	Serbian
Country of publication, **CP**:	Republic of Serbia
Locality of publication, **LP**:	Vojvodina
Publication year, **PY**:	2025
Publisher, **PB**:	Author's reprint
Publication place, **PP**:	Novi Sad, Dositeja Obradovica sq. 6
Physical description, **PD**: (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/52/18/2/23/0/23
Scientific field, **SF**:	Electrical and Computer Engineering
Scientific discipline, **SD**:	Applied computer science and informatics
Subject/Key words, **S/KW**:	Template, thesis, tutorial
UC	
Holding data, **HD**:	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, **N**:	
Abstract, **AB**:	This document provides guidelines for writing final theses at the Faculty of Technical Sciences, University of Novi Sad. At the same time, it serves as a Typst template.

Accepted by the Scientific Board on, **ASB**:			
Defended on, **DE**:		01.01.2025	
Defended Board, **DB**:	President:	Petar Petrović, Phd., assoc. professor	
	Member:	Marko Marković, Phd., asist. professor	Menthor's sign
	Member, Mentor:	Igor Dejanović, Phd., full professor	

ИЗЈАВА О НЕПОСТОЈАЊУ СУКОБА ИНТЕРЕСА

Изјављујем да нисам у сукобу интереса у односу ментор – кандидат и да нисам члан породице (супружник или ванбрачни партнер, родитељ или усвојитељ, дете или усвојеник), повезано лице (крвни сродник ментора/кандидата у правој линији, односно у побочној линији закључно са другим степеном сродства, као ни физичко лице које се према другим основама и околностима може оправдано сматрати интересно повезаним са ментором или кандидатом), односно да нисам зависан/на од ментора/кандидата, да не постоје околности које би могле да утичу на моју непристрасност, нити да стичем било какве користи или погодности за себе или друго лице било позитивним или негативним исходом, као и да немам приватни интерес који утиче, може да утиче или изгледа као да утиче на однос ментор-кандидат.

У Новом Саду, дана _____

Ментор

_____

Кандидат

_____

Садржај

1 Увод . . . 1
1.1 Мотивација . . . 1
1.2 Циљ рада . . . 2
1.3 Структура рада . . . 2
2 Спецификација и пројектовање техничког решења . . . 3
2.1 Функционални захтеви . . . 3
2.2 Нефункционални захтеви . . . 4
2.3 Архитектура система . . . 6
3 Имплементација система . . . 13
3.1 Имплементација десктоп апликације . . . 13
3.2 Имплементација позадинске логике (Rust) . . . 13
3.3 Имплементација корисничког интерфејса (SvelteKit) . . . 17
3.4 Имплементација инфраструктуре у облаку . . . 20
3.5 Имплементација мобилне апликације . . . 23
3.6 Имплементација веб апликације за управљање лозинком . . . 26
4 Приказ имплементираног система . . . 29
4.1 Десктоп апликација: Интерфејс за управљање . . . 29
4.2 Ажурирање десктоп апликације . . . 36
4.3 Процес активације и коришћења мобилне апликације . . . 37
5 Закључак . . . 39
5.1 Остварени резултати и решени проблеми . . . 39
5.2 Предности и мане имплементираног решења . . . 40
5.3 Правци даљег развоја . . . 41
6 Додатак - Списак коришћених скраћеница и појмова . . . 43
Биографија . . . 49
Литература . . . 51

Глава 1

Увод

1.1 Мотивација

Последњих деценија сведоци смо интензивне дигитализације у готово свим областима друштвеног и привредног живота. Савремене организације и предузећа настоје да своје пословање учине ефикаснијим применом информационих система. Ови системи аутоматизују радне процесе и нуде лакши приступ подацима [1]. Једна од области у којој се потреба за дигиталним решењима посебно истиче јесте фитнес индустрија која у последње време доживљава снажан раст и популаризацију [2].

Традиционални начини вођења евиденције у фитнес центрима, као што су папирне картице и ручно вођење списка чланова, постају све мање одрживи услед повећаног броја корисника и сложености управљања. Поред саме евиденције чланова, неопходно је водити рачуна о различитим типовима чланарина, праћењу њиховог важења и контроли приступа објекту. Осим тога, постоји потреба за анализом података како би се побољшале услуге и креирале ефективније стратегије пословања [3]. Без јединственог система, ови задаци могу бити изузетно захтевни, дуготрајни и подложни грешкама [2].

Поред потребе за ефикасношћу, посебан изазов представља и питање доступности и поузданости података. Многе теретане функционишу у окружењима где интернет конекција није увек стабилна, док неке настоје да своје податке задрже искључиво у локалном окружењу, без ослањања на спољне сервисе. У таквим условима коришћење система који у потпуности зависе од инфраструктуре у облаку (енгл. *cloud*) може бити непрактично [4]. Свака прекинута конекција може довести до немогућности приступа подацима, успоравања рада или чак губитка информација.

На тржишту данас постоји више готових софтверских решења за управљање фитнес центрима, која углавном раде по *cloud* моделу. Неки од најпознатијих су *Mindbody*[1], *Glofox*[2] и *Zen Planner*[3]. Ови системи нуде широк спектар функционалности, али су често превише комплексни и не могу се прилагодити једноставнијим моделима пословања. За мање теретане, које често раде у ограниченим условима, трошкови лиценци и претплата могу бити превисоки. Зависност од интернет конекције представља додатни

[1] https://www.mindbodyonline.com/
[2] https://www.glofox.com/
[3] https://www.zenplanner.com/

услов који не може увек бити испуњен. Супротан приступ представљају локални системи који се инсталирају на рачунаре унутар објекта (*GymOne*[4]). Они обезбеђују већу конторолу над подацима али увелико смањују поузданост и доступност услуге, јер су подаци везани за једну локацију и један уређај [5]. Ова чињеница такође онемогућава коришћење мобилне апликације за чланове, што је данас стандардна понуда већине фитнес центара.

1.2 Циљ рада

Због свега наведеног, постоји јасна потреба за развојем новог система који ће бити прилагођен специфичним потребама мањих фитнес центара. Циљ овог рада је пројектовање и имплементација софтверског решења које би комбиновало предности *cloud* и локалних система, обезбеђујући поузданост, доступност и једноставност коришћења. Овај систем би требао да омогући ефикасно управљање члановима, праћење чланарина, контролу приступа и анализу података. Архитектура система би била првенствено заснована на локалном раду. Поузданост података би се успоставила имплементацијом система за резервне копије уз коришћење привремене интернет конекције. Међутим, систем би требао да подржи и синхронизацију са *cloud* сервисом и коришћење мобилне апликације када је конекција стално доступна и стабилна.

1.3 Структура рада

Рад је организован тако да читаоца постепено води од теоријских основа до практичне реализације и визуелног приказа система. Након овог уводног поглавља, у другом поглављу се детаљно разрађује спецификација и пројектовање техничког решења, где се кроз функционалне захтеве и UML дијаграме поставља темељ за развој. Треће поглавље, које је уједно и централни део рада, посвећено је имплементацији, и у њему се детаљно описује техничка израда сваке компоненте система, од десктоп апликације до инфраструктуре у облаку. Четврто поглавље визуелно представља готово решење кроз приказ корисничког интерфејса и кључних функционалности. На крају, у петом поглављу, дат је закључак који сумира остварене резултате, анализира предности и мане решења и предлаже могућности за будућа унапређења.

[4] https://gymoneglobal.com/

Глава 2

Спецификација и пројектовање техничког решења

Развој сложеног софтверског решења, као што је систем за управљање фитнес центром, захтева да се јасно дефинишу сви захтеви, ограничења и архитектонски принципи пре него што се ступи у фазу имплементације. Потребно је извршити детаљну анализу потреба корисника и идентификовати пословне процесе које треба обухватити системом. Тиме се дефинишу основне функционалности које систем мора да пружи. Циљ је да се прикаже на који начин су дефинисани процеси управљања члановима, чланаринама и контролом приступа, као и како су ови процеси интегрисани у јединствену платформу која подржава локални рад и рад у облаку.

У наредним потпоглављима су најпре дефинисани функционални и нефункционални захтеви, након чега следи приказ предложене архитектуре. Архитекура система је описана одговарајућим UML дијаграмима који визуелно представљају структуру и токове података у систему.

2.1 Функционални захтеви

Функционални захтеви дефинишу конкретно понашање система, односно функционалности које треба да буду омогућене крајњим корисницима. Систем GymMan треба да подржи све процесе неопходне за ефикасно управљање теретаном. Захтеви су груписани по главним модулима система.

- **Управљање члановима**
  - Регистрација нових чланова са свим потребним подацима (име, презиме, контакт информације, датум рођења и јединствени идентификатор - број картице)
  - Ажурирање података постојећих чланова од стране радника
  - Преглед листе чланова са могућношћу претраге
  - Брисање чланова из система
  - Позив за креирање налога за мобилну апликацију путем е-поште
  - Захтев за ресетовање лозинке за мобилну апликацију путем е-поште
- **Управљање чланаринама**
  - Дефинисање различитих типова чланарина на основу трајања, броја посета, цене и ограничења приступа

- Додела чланарина члановима - члан може имати једну активну и више предстојећих чланарина
- Праћење статуса чланарине и ажурирање права приступа на основу тог статуса
- Обнављање чланарине након истека
- Евиденција историје чланарина и плаћања за сваког члана

- **Контрола приступа**
  - Скенер чланске картице при уласку у теретану и идентификација члана
  - Верификација статуса чланарине у реалном времену
  - Омогућавање или одбијање приступа на основу валидности чланарине
  - Ажурирање броја преосталих улазака након сваког уласка
  - Евиденција сваког приступа са временском ознаком и идентификатором члана
  - Подршка за пробне и дневне тренинге уз евиденцију уласка

- **Аналитика и извештаји**
  - Преглед статистике посета уз истицање трендова
  - Приказ броја активних чланова кроз време
  - Приказ прихода од различитих типова чланарина и њихова дистрибуција
  - Приступ историјским подацима евиденције улазака и њихово филтрирање

- **Прављење резервних копија и синхронизација**
  - Локално чување података на уређају за рад у офлајн (енгл. *offline*) режиму и брз приступ
  - Ручно или периодично аутоматско прављење резервних копија података и слање на облак
  - Враћање података из резервних копија на истом или новом уређају уз потпуни интегритет података
  - Аутоматска синхронизација података у реалном времену између локалне базе и удаљене базе за мобилну апликацију, намењене само за читање података (енгл. *read-only*)

- **Кориснички налози и аутентификација**
  - Креирање налога за раднике са различитим улогама
  - Креирање налога за чланове теретане за приступ мобилној апликацији
  - Сервис за аутентификацију чланова при приступу мобилној апликацији

2.2 Нефункционални захтеви

Нефункционални захтеви дефинишу квалитативне особине система и одређују начин на који он треба да функционише, без директног утицаја на конкретну функционалност.

- **Перформансе**
  - Систем мора обезбедити брз одзив при свакој операцији

  - Сихронизација података са облаком не сме утицати на перформансе локалног рада
  - Процес резервне копије мора бити извршен у позадини без утицаја на рад корисника
  - Систем мора бити оптимизован за рад на уређајима са ограниченим ресурсима

- **Безбедност**
  - Све лозинке морају бити сигурно похрањене и хеширане
  - Комуникација између система и облака мора бити шифрована коришћењем TLS протокола
  - Свака грешка у аутентификацији мора бити евидентирана
  - Приступ осетљивим подацима мора бити ограничен на овлашћене кориснике

- **Доступност, конзистентност и интегритет података**
  - Систем мора бити доступан и у случају привремене нестанка интернет конекције
  - Локални подаци морају бити конзистентни и интегритет података мора бити очуван током свих операција
  - У случају кашњења у синхронизацији са мобилном апликацијом због прекида интернет конекције локални рад треба да остане конзистентан

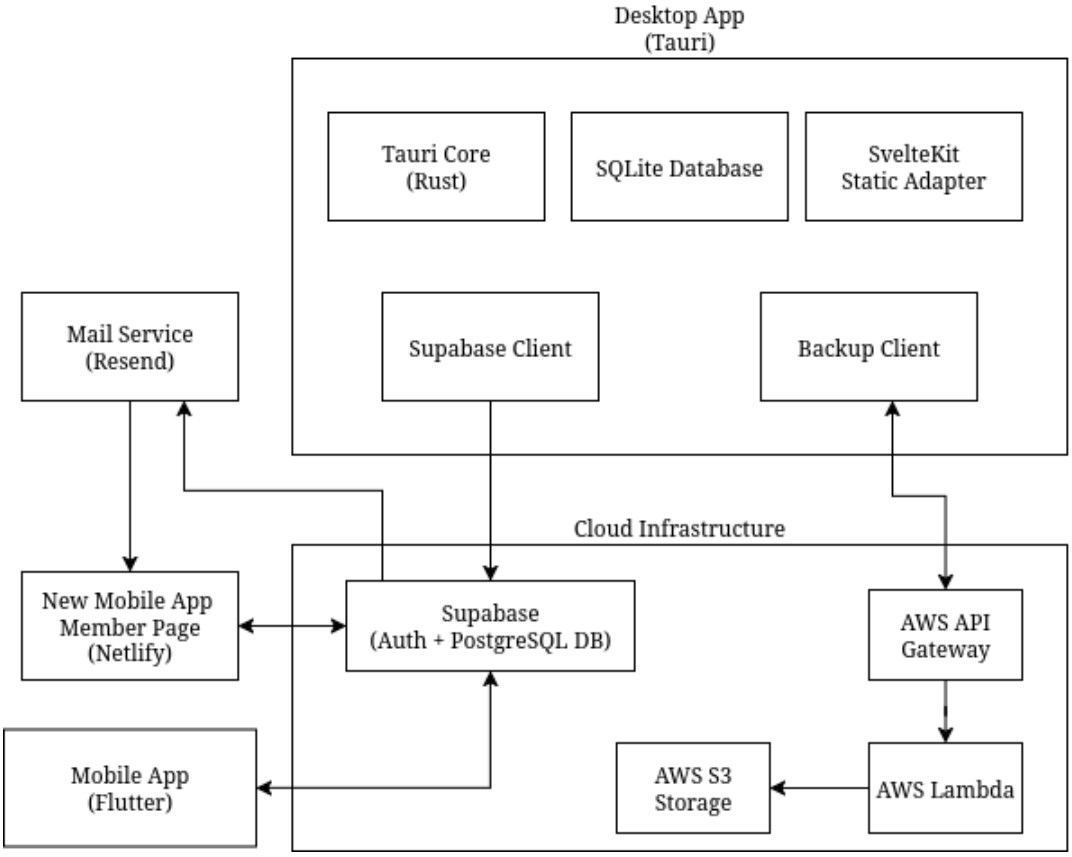
- **Корисничко искуство**
  - Кориснички интерфејс мора бити интуитиван и једноставан

- **Преносивост и независност од платформе**
  - Систем мора бити преносив и радити на различитим оперативним системима (Windows, macOS, Linux)
  - Локална база података мора бити независна од специфичне платформе
  - Систем мора бити лако преносив на нови уређај уз минималне напоре
  - Мобилна апликација мора да ради на најпопуларнијим мобилним платформама (iOS и Android)

- **Одрживост и проширивост**
  - Архитектура система мора бити модуларна како би се омогућило лако одржавање и надоградња
  - Систем мора бити дизајниран тако да се нове функционалности могу лако додати у будућности

- **Локализација**
  - Систем мора подржавати више језика и омогућити лако прилагођавање корисничког интерфејса

2.3 Архитектура система



Слика 1: Архитектура система.

Архитектура система GymMan је осмишљена као хибридно решење које комбинује локалну апликацију са *cloud* инфраструктуром. На слици 1 приказана је основна архитектура система која се састоји из следећих компоненти:

- **Desktop апликација** - представља централни део система, инсталира се на рачунар унутар теретане и користи за управљање свим пословним процесима у теретани. Изграђена је на основу клијент-сервер архитектуре унутар једног пакета. Ова компонента је одговорна за управљање локалном базом података, обраду пословне логике, аутентификацију радника, креирање резеврних копија, синхронизацију са облаком и комуникацију са скенером чланских картица. Уз то, она пружа интуитиван кориснички интерфејс за раднике и администраторе.
- **Cloud инфраструктура** - састоји се из два дела: сервиси за складиштење и добављање резервних копија и база података која представља поједностављену *read-only* реплику локалне базе. Ова компонента омогућава чување резервних копија

на сигурном месту ван теретане и пружа приступ подацима који су неопходни за коришћење мобилне апликације.

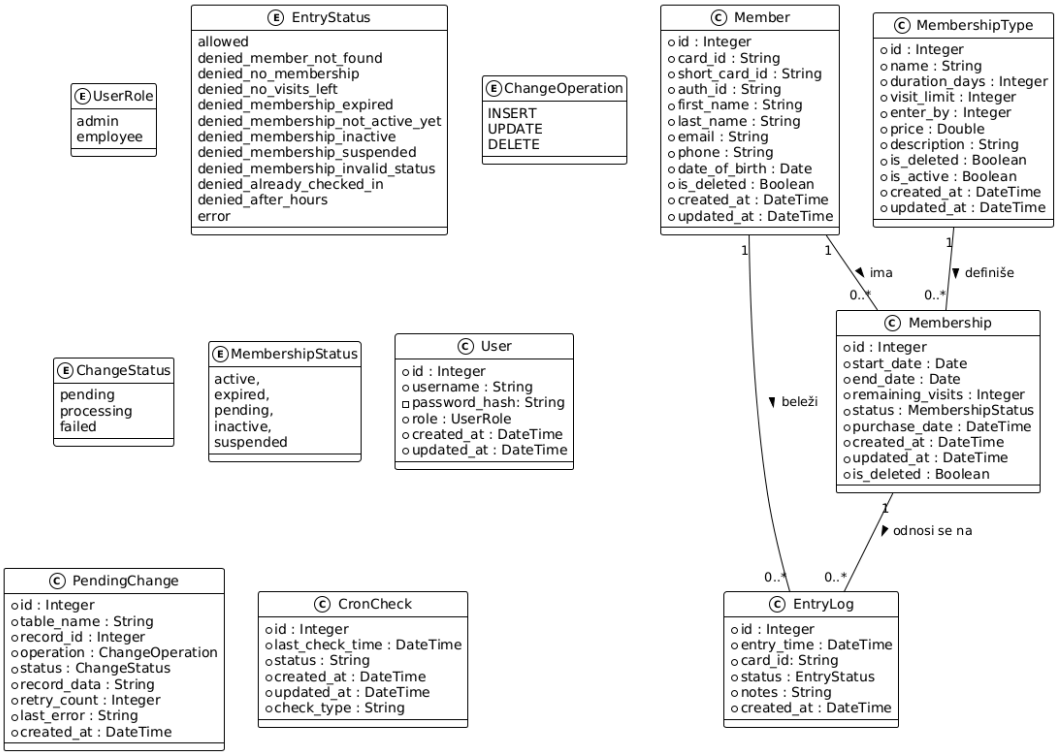
- **Мобилна апликација** - намењена је члановима теретане и омогућава им приступ информацијама о статусу чланарине и може да се користи уместо физичке чланске картице јер приказује бар-код за скенирање при уласку у теретану. Апликација комуницира са *cloud* базом података како би добила најновије информације.
- **Mail сервис** - користи се за слање е-порука члановима теретане, као што су позиви за креирање налога мобилне апликације и захтеви за ресетовање лозинке. Овај сервис такође комуницира сервисима за аутентификацију у *cloud* инфраструктури.
- **Веб апликација за креирање налога** - намењена је за креирање налога за мобилну апликацију. Корисници приступају овој апликацији путем сигурног линка из е-поруке. Веб апликација комуницира са сервисом за аутентификацију у *cloud* инфраструктури како би креирала налоге и омогућила приступ мобилној апликацији.

Комуникација између свих компоненти је шифрована коришћењем TLS протокола како би се обезбедила безбедност података током преноса.

Избор архитектуре заснован је на потреби да се обезбеди комбинација локалне поузданости и *cloud* интеграције. Традиционална веб (енг. *web*) решења често зависе од стабилне интернет конекције, што може бити проблематично у реалним условима рада теретана. GymMan пружа пуну функционалност чак и у одсуству мреже (енгл. *offline-first*), уз каснију синхронизацију података са *cloud*-ом. Овакав приступ омогућава најбољи баланс између аутономије, безбедности и једноставности коришћења и инсталације.

2.3.1 Дијаграм класа

На слици 2 приказан је UML дијаграм класа који представља структуру система GymMan. Систем није имплементиран коришћењем објектно оријентисаног програмског језика, али прилагођени дијаграм класа ипак може пружити јасан преглед главних ентитета у систему и њихових међусобних односа.



Слика 2: Дијаграм класа система.

Овај дијаграм приказује следеће главне класе:

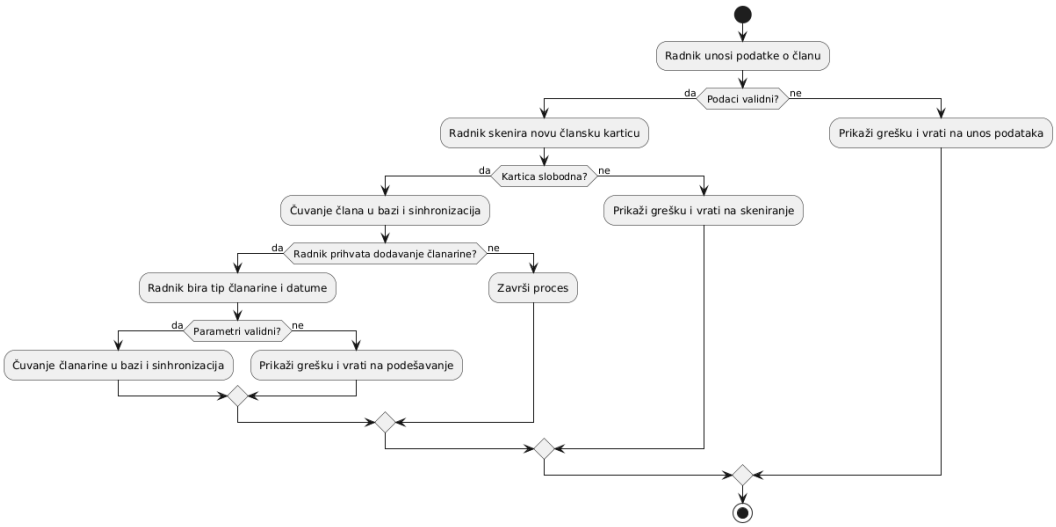
- **Member** - представља члана теретане са својим основним подацима као што су име, презиме, контакт информације и јединствени број чланске картице. Ова класа садржи поље *auth_id* које представља везу са налогом у моблиној апликацији.
- **Membership** - представља чланарину која је додељена члану. Један члан може да буде власник више чланарина, од којих само једна може да буде тренутно активна. Садржи информације о типу чланарине, датуму почетка и истека, датуму куповине, броју преосталих улазака и статусу чланарине.
- **MembershipType** - дефинише различите типове чланарина које теретана нуди. Садржи информације као што су назив типа, трајање, цена, број улазака и ограничења приступа. Поље *is_active* указује да ли је тип чланарине тренутно доступан за продају (временски ограничене промотивне понуде). Једна чланарина може бити повезана са једним типом чланарине.
- **EntryLog** - евидентира сваки улазак члана у теретану. Садржи временску ознаку уласка, идентификатор и број картице члана који је приступио објекту. Не постоји ограничење на број уноса за једног члана. Такође садржи информације о статусу уласка и покрива све случајеве (валидна чланарина, пробни тренинг, истекла чланарина, прекорачен број улазака, итд.). У пољу *notes* чувају се детаљи одбијања уласка.

- **User** - представља радника у теретани који користи десктоп апликацију. Садржи основне податке као што су корисничко име, хеширана лозинка и улога (администратор или радник). Омогућава аутентификацију и контролу приступа унутар апликације.
- **CronCheck** - садржи информације о периодичним проверама. Те провере су везане за резервне копије и за аутоматско дневно ажурирање статуса чланарина на основу датума почетка или истека. Ова класа чува информације о последњем извршавању провере и успешности последње операције.
- **PendingChange** - представља промене које су направљене у систему, али још увек нису синхронизоване са *cloud* сервисом. Ова класа омогућава праћење промена које треба послати на облак када је интернет конекција доступна. Садржи информације о типу промене, самој промени, повезаном ентитету и временској ознаци када је промена направљена.

Осим наведених класа, дијаграм приказује и везе између њих као и додатне енумерације које дефинишу могуће вредности за поља као што су статус чланарине, статус уласка и улога корисника.

2.3.2 Дијаграми активности

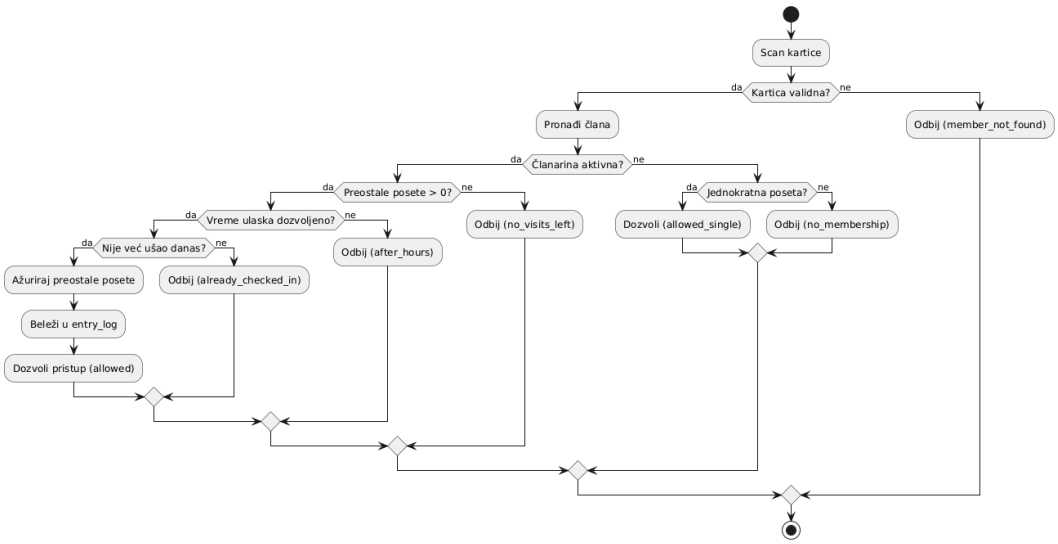
Да би се боље разумели пословни процеси у систему, на сликама 3, 4 и 5 приказани су UML дијаграми активности за неке од кључних операција.



Слика 3: Дијаграм активности за регистрацију новог члана.

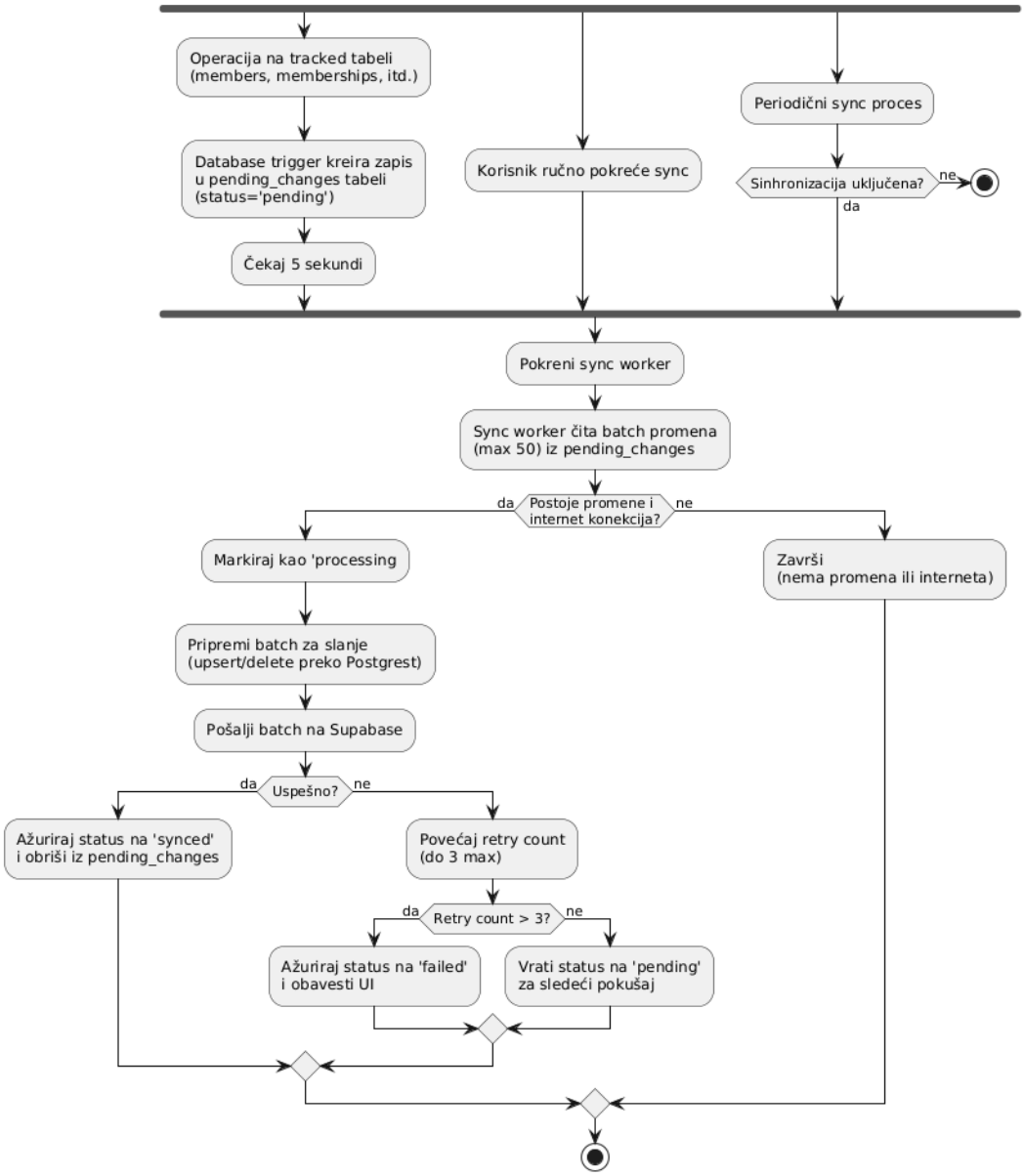
Слика 3 приказује кораке укључене у процес регистрације новог члана теретане. Процес почиње када радник у апликацији изабере опцију за додавање новог члана. Затим се уносе и проверавају основни подаци о члану. Радник користи бар-код скенер

или ручно уноси број нове чланске картице. Чланска картица се валидира да не припада ниједном постојећем члану, након чега се креира нови запис у локалној бази података. Систем нуди раднику опцију да креира чланарину за новог члана одмах након регистрације. Ако се одабере ова опција, радник бира тип чланарине и уноси потребне детаље пре него што се чланарина додели члану.



Слика 4: Дијаграм активности за евиденцију уласка члана.

Слика 4 приказује кораке укључене у процес евиденције уласка члана у теретану. Процес почиње када члан скенира своју чланску картицу на улазу. Систем чита број картице и тражи одговарајући запис у локалној бази података. Ако члан постоји, проверава се статус његове чланарине. У зависности од статуса, систем одлучује да ли ће дозволити или одбити приступ. У случају валидне чланарине, приступ је дозвољен, број преосталих улазака се ажурира, и евидентира се улазак са временском ознаком. Ако је чланарина истекла или нема довољно улазака, приступ се одбија и евидентира се неуспешан покушај уласка са одговарајућим разлогом.



Слика 5: Дијаграм активности за синхронизацију података са облаком.

Слика 5 приказује поједностављене кораке укључене у процес синхронизације података између локалне базе и *cloud* сервиса. Процес синхронизације може бити иницран ручно од стране радника или аутоматски у позадини. Аутоматско иницирање дешава се у случају неке битне промене над подацима која захтева тренутну синхронизацију и периодично у подешеном интервалу. Синхронизација почиње проверавањем доступности интернет конекције. Ако је конекција доступна, систем почиње са слањем

локалних промена које још нису синхронизоване. За сваку промену, систем покушава да је пошаље на *cloud* сервис и чека потврду о успешном пријему. У случају грешке, бележи се неуспешна синхронизација и наставља се са следећом променом. Неуспешне промене остају у локалној листи за касније покушаје слања. Синхронизација се врши само у једном смеру - са локалне базе на *cloud* сервис, јер мобилна апликација користи поједностављену *read-only* верзију података.

Глава 3

Имплементација система

Имплементација система GymMan представља кључни део овог рада, током којег су теоријски концепти и архитектонски принципи преточени у функционално софтверско решење. Процес развоја је вођен модуларним приступом, са циљем да се обезбеди јасна раздвојеност компоненти, лакше одржавање и будућа проширивост. Свака компонента система — десктоп апликација, мобилна апликација и пратећа инфраструктура у облаку — развијена је са специфичним технологијама које најбоље одговарају њеној намени, али су све повезане кроз заједнички модел података и пажљиво дефинисане комуникационе протоколе.

У овом поглављу биће детаљно описан процес имплементације сваке од наведених компоненти. Акценат ће бити на техничким одлукама, изабраним библиотекама и програмским језицима, као и на примерима кода који илуструју битне механизме система.

3.1 Имплементација десктоп апликације

Десктоп апликација представља централну компоненту система GymMan, осмишљену као главни алат за администраторе и радници у фитнес центру. За њену израду изабран је радни оквир Tauri, који омогућава развој вишеплатформских (енгл. *cross-platform*) апликација комбиновањем перформанси и сигурности програмског језика Rust са флексибилношћу модерних веб технологија за израду корисничког интерфејса. У поређењу са алтернативама као што је Electron, Tauri нуди значајне предности у виду мање потрошње меморије, мање величине финалне апликације и директнијег приступа системским ресурсима, што га је учинило идеалним избором за овај пројекат [6], [7].

Архитектура десктоп апликације подељена је на две основне целине: позадински део (енгл. *backend*) реализован у Rust-у и кориснички интерфејс (енгл. *frontend*) реализован помоћу Svelte оквира.

3.2 Имплементација позадинске логике (Rust)

Позадински део апликације, смештен у директоријуму `src-tauri`, представља њену основу и одговоран је за сву пословну логику, управљање подацима и комуникацију са спољним системима. Као програмски језик изабран је Rust због својих гаранција сигурности меморије, високих перформанси и одличног екосистема за конкурентно

програмирање, што је од великог значаја за асинхроне задатке попут синхронизације и прављења резервних копија.

3.2.1 Структура и управљање стањем

Основна структура позадинског дела организована је у модуле који одговарају различитим доменима апликације: `commands` (командни интерфејс), `db` (управљање базом), `sync` (логика синхронизације), `backup` (резервне копије), `models` (модели података), `dto` (типови за пренос података према корисничком интерфејсу).

Централно стање апликације (енгл. *state*) дефинисано је у структури `AppState`, која се иницијализује приликом покретања и дели између свих компоненти путем Tauri механизма за управљање стањем. Ова структура садржи групу отворених конекција према бази података (енгл. *connection pool*), као и конфигурациона подешавања апликације.

```
pub struct AppState {
    pub db_pool: SqlitePool,
    pub settings: RwLock<AppSettings>,
    pub last_membership_check: RwLock<Option<chrono::NaiveDateTime>>,
    pub last_backup: RwLock<Option<chrono::NaiveDateTime>>,
    pub sync_mutex: Mutex<()>,
    pub supabase_client: RwLock<Option<SupabaseClient>>,
}
```

3.2.2 Командни интерфејс

Комуникација између фронтенда и бекенда реализована је преко Tauri командног интерфејса. Свака функција у Rust-у која треба да буде доступна из JavaScript-а означена је са `#[tauri::command]`. Ове функције су асинхроне и враћају `AppResult` тип, што омогућава елегантно руковање грешкама које се затим прослеђују фронтенду. `AppResult` је дефинисан као `pub type Result<T> = std::result::Result<T, AppError>`, где се у случају грешке враћа прилагођени `AppError` тип који садржи детаље о специфичном проблему.

Пример команде за обраду скенирања картице:

```
#[tauri::command]
pub async fn process_scan(
    payload: ScanPayload,
    app_state: tauri::State<'_, AppState>
) -> Result<ScanProcessingResult, String> {
    // 1. Проналажење члана у локалној бази
    // 2. Валидација чланарине
    // 3. Ажурирање броја посета
    // 4. Упис у евиденцију улазака (entry_log)
    // 5. Враћање резултата фронтенду
    // ...
```

```
    Ok(result)
}
```

Овај приступ елиминише потребу за интерним HTTP (енгл. Hyper-Text-Transfer-Protocol) сервером и омогућава брзу и сигурну комуникацију између два дела апликације.

3.2.3 Управљање базом података и миграције

За интеракцију са SQLite [8] базом података користи се библиотека SQLx, која пружа типски безбедне (енгл. *type-safe*) SQL упите верификоване током компајлирања. Ово значајно смањује могућност грешака у раду са базом.

Структура базе података дефинисана је кроз скуп SQL миграционих фајлова, којима управља `sqlx-cli`. Приликом првог покретања апликације, Rust код аутоматски извршава све потребне миграције како би креирао или ажурирао локалну базу на најновију верзију, што осигурава конзистентност шеме.

Подаци се уписују и читају коришћењем асинхроних метода које пружа SQLx, чиме се избегава блокирање главног програмског тока и корисничког интерфејса током операција са базом.

3.2.4 Асинхрони задаци у позадини

Систем користи асинхрони Tokio *runtime* за извршавање дуготрајних задатака у позадини, независно од главног UI тока. Кључни позадински процеси су:

- *Sync Worker*: Периодично (нпр. на сваких 30 секунди) проверава pending_changes табелу и шаље нове промене на Supabase.
- *Backup Scheduler*: Покреће процес прављења резервне копије у конфигурисаним временским интервалима.
- *Membership Status Checker*: Једном дневно проверава и ажурира статусе чланарина (нпр. из “pending” у “active” или из “active” у “expired”) како би систем проактивно увек имао ажурне статусе чланарина.

Ови задаци се покрећу приликом стартовања апликације као засебни асинхрони задаци, користећи `tokio::spawn` [9]. Сви процеси се логују у структурисаном формату користећи библиотеку `tracing`.

3.2.5 Конфигурација и чување подешавања

Подешавања апликације као што су *Supabase* API кључ, сервер резервних копија, интервал синхронизације и избор језика поставља администратор и она се чувају се у посебном JSON фајлу (`app_settings.json`) у апликационом директоријуму добијеном преко `tauri::api::path::app_data_dir()` [10]. Овим се обезбеђује да подаци остану перзистентни и након рестартовања апликације.

3.2.6 Дистрибуција и аутоматско ажурирање апликације

Како би се корисницима омогућило једноставно преузимање и редовно ажурирање апликације, систем GymMan користи аутоматизоване процесе за израду, објављивање и дистрибуцију нових верзија. Овај приступ осигурава да сваки корисник увек има приступ најновијим функционалностима и сигурносним исправкама уз минималан напор.

3.2.6.1 Аутоматизована израда на GitHub платформи

Процес објављивања нових верзија је у потпуности аутоматизован помоћу GitHub Actions [11]. Сваки пут када се на репозиторијум постави нови "tag" (нпр. `v1.0.1`), аутоматски се покреће процес који:

1. Компајлира апликацију: За сваку циљну платформу (Windows, Linux) креира се одговарајући инсталациони пакет (`.msi`, `.deb`, `.AppImage`).
2. Дигитално потписује пакет: Користећи сигурносни кључ сачуван у GitHub Secrets [11], сваки инсталациони пакет добија дигитални потпис (`.sig` фајл). Овај потпис гарантује да ажурирање долази од аутентичног извора и да није мењано.
3. Генерише `latest.json` манифест: Кључни корак у овом процесу је аутоматско генерисање `latest.json` датотеке. Акција `tauri-action` унутар радног тока (енгл. *workflow*) прикупља информације о новој верзији, белешке о издању (енгл. `release notes`), и што је најважније, URL-ове до свих инсталационих пакета и садржај свих `.sig` датотека. Све ове информације се пакују у `latest.json` датотеку, која служи као *update manifest*.
4. Објављује на GitHub Releases: На крају, сви пакети, њихови потписи, и генерисани `latest.json` фајл се аутоматски објављују као артефакти у оквиру новог издања (енгл. *release*) на GitHub-у.

Ова континуирана интеграција и испорука (CI/CD, енгл. *Continuous Integration and Continuous Delivery*) елиминише ручни рад, смањује могућност грешке и убрзава испоруку нових верзија.

3.2.6.2 Механизам за аутоматско ажурирање

Десктоп апликација има уграђен механизам за аутоматско ажурирање, реализован помоћу `tauri-plugin-updater` плагина. Конфигурација овог плагина у `tauri.conf.json` фајлу је подешена да директно комуницира са провајдером издања на GitHub-у.

```
"plugins": {
  "updater": {
    "active": true,
    "endpoints": [
      "https://github.com/c-danil0o/gym_manager/releases/latest/download/
latest.json"
    ],
```

```
    "pubkey": "...",
    "dialog": true
  }
}
```

Овај механизам омогућава апликацији да провери `latest.json` фајл на GitHub-у и провери да ли постоје нове верзије. Када се открије нова верзија, апликација преузима одговарајући инсталациони пакет и његов дигитални потпис. Инсталација нове верзије се изводи само након успешне верификације дигиталног потписа, чиме се обезбеђује сигурност и интегритет ажурирања.

3.3 Имплементација корисничког интерфејса (SvelteKit)

Кориснички интерфејс (фронтенд) десктоп апликације представља компоненту са којом корисници (администратори и радници) директно интерагују. За његову реализацију изабран је SvelteKit, савремени радни оквир за израду веб апликација базиран на другом радном оквиру под именом Svelte. Кључна предност Svelte-а лежи у његовом компајлерском приступу: уместо да се ослања на виртуелни објектни модел документа (DOM, енгл. *Document Object Model*), Svelte компајлира компоненте у оптималан, императивни JavaScript код који директно манипулише DOM-ом [12]. Ово резултира изузетно брзим апликацијама са минималним додатним трошком у смислу перформанси (енгл. *overhead*), што га чини идеалним за ресурсно ефикасне десктоп апликације попут оних које се праве са Tauri радним оквиром. Интерфејс је имплементиран да буде потпуно прилагодљив величини прозора, омогућавајући рад на различитим уређајима и резолуцијама.

3.3.1 Архитектура и организација кода

Пројектна структура фронтенда прати конвенције SvelteKit-а и организована је на начин који подржава модуларност и лаку навигацију:

- `src/routes/`: Овај директоријум садржи структуру свих страница и рута у апликацији. SvelteKit користи *file-system based routing*, где структура фајлова директно одговара URL путањама. На пример, фајл `src/routes/members/[id]/+page.svelte` дефинише страницу за приказ детаља појединачног члана.
- `src/lib/`: Централни директоријум за сав дељени код.
  - `src/lib/components/`: Садржи Svelte компоненте као што су форме, табеле, дијалози, графикони и дугмад.
  - `src/lib/stores/`: Дефинише глобално стање апликације коришћењем Svelte *stores*. Овде се чувају подаци о улогованом кориснику, статусу синхронизације, изабраној теми (светла/тамна) и језику.
  - `src/lib/models/`: Садржи TypeScript типове који су аутоматски који одговарају типовима за пренос података (DTO, енгл. Data-Transfer-Object) дефинисаним у

Rust позадинском делу. Ово обезбеђује типску безбедност приликом комуникације између фронтенда и бекенда.

- `src/lib/paraglide/`: Директоријум који садржи превоце за интернационализацију.

3.3.2 Комуникација са позадинском логиком

Интеракција између фронтенда (JavaScript/TypeScript) и бекенда (Rust) је кључни аспект Tauri апликације. Ова комуникација се одвија асинхроно преко Tauri командног интерфејса, коришћењем `invoke` функције из `@tauri-apps/api` пакета. Свака `#[tauri::command]` функција у Rust-у је директно доступна за позивање, чиме се елиминише потреба за традиционалним REST или GraphQL API-јем.

Пример имплементације позива команде за додавање новог члана:

```
import { invoke } from '@tauri-apps/api/core';
import type { MemberPayload } from '$lib/models'; // Типови из Rust-а

async function saveNewMember(memberData: MemberPayload): Promise<void> {
  try {
    // Позив Tauri команде
    await invoke('save_member', { payload: memberData });
    // Потврда успешног чувања
  } catch (error) {
    // Приказ грешке
    console.error('Failed to save member:', error);
  }
```

3.3.3 Управљање стањем и реактивност

За управљање стањем апликације користи се уграђени систем Svelte *stores*. Овај приступ је једноставан, али изузетно моћан. На пример, *writable store* се користи за чување информација о тренутно улогованом кориснику, што омогућава да се делови интерфејса динамички приказују или сакривају у зависности од његове улоге (администратор или радник).

Svelte реактивност осигурава да се свака промена у *store*-у аутоматски пропагира кроз све компоненте које га користе, што резултира тренутним ажурирањем корисничког интерфејса без потребе за ручном манипулацијом. Пример једноставног *store*-а за тренутног корисника:

```
import { writable } from 'svelte/store';

const { subscribe, set, update } = writable({
    isAuthenticated: false,
    username: null as string | null,
    error: null as string | null,
```

```
    role: null as string | null
  });
```

Поред тога, апликација користи *Tauri events* за реактивност на промене иницране из бекенда. На пример, када позадински процес прими резултат асинхроне синхронизације података, он емитује догађај `sync_status_changed`. Фронтенд слуша овај догађај и аутоматски освежава приказ статуса.

```
import { listen } from '@tauri-apps/api/event';

// Слушање догађаја из бекенда
const unlisten = await listen('sync_status_changed', (event) => {
  console.log('Sync status changed. Refreshing data...', event.payload);
  refreshStatus();
});
```

3.3.4 Стилизовање и кориснички интерфејс

Стилизовање апликације урађено је помоћу Tailwind CSS. За сложеније UI елементе као што су модални дијалози, табеле са пагинацијом и сортирањем, и падајуће листе, коришћена је библиотека shadcn-svelte. Ова библиотека представља регистар прилагодљивих и приступачних (енгл. accessible) компоненти које се лако интегришу и модификују. Компоненте се инталирају појединачно, што омогућава минималну величину пакета и боље перформансе.

Апликација такође подржава светлу и тамну тему. Ова функционалност је имплементирана коришћењем CSS варијабли, а промена теме се врши преко `mode-watcher` компоненте која детектује системска подешавања или омогућава кориснику ручни избор.

3.3.5 Валидација форми и обрада грешака

За валидацију корисничког уноса у формама користи се библиотека Zod за дефинисање шема података и правила валидације, у комбинацији са Superforms библиотеком за SvelteKit. Ова комбинација омогућава:

- Типски сигурну дефиницију форми.
- Валидацију у реалном времену, док корисник уноси податке.
- Јасно приказивање грешака испод одговарајућих поља.
- Спречавање слања неисправних података на бекенд.

Грешке које стигну са бекенда (нпр. грешка при упису у базу) се хватају у *try-catch* блоковима, преводе се и затим приказују кориснику путем *toast* нотификација, пружајући јасну повратну информацију о исходу операције.

3.3.6 Интернационализација (i18n)

Подршка за више језика (српски, енглески) реализована је помоћу ParaglideJS. Овај алат компајлира преводе из JSON фајлова у типски сигурне TypeScript модуле, чиме се елиминише могућност грешака услед недостајућих превода или погрешних кључева.

Пример коришћења превода у Svelte компонентама:

```
import * as m from '$lib/paraglide/messages';
// ...
<button>{m.loginButtonLabel()}</button>
```

3.4 Имплементација инфраструктуре у облаку

Иако је GymMan систем дизајниран са *offline-first* архитектуром као основном премисом, инфраструктура у облаку (енгл. *cloud*) игра важну, двоструку улогу у његовом функционисању. Прва улога је обезбеђивање сигурног и поузданог механизма за прављење резервних копија (енгл. *backup*) комплетне локалне базе података, што је битно за очување података у случају хардверског квара или других непредвиђених догађаја. Друга улога је подршка за опциони онлајн (енгл. *online*) режим рада, где се подаци асинхроно синхронизују са централном базом како би се омогућила функционалност мобилне апликације.

За реализацију ове две различите потребе, изабране су две одвојене *cloud* платформе, свака оптимизована за свој задатак. За систем резервних копија коришћени су сервиси Amazon Web Services (AWS) , док је за синхронизацију података и подршку за мобилну апликацију изабрана Supabase платформа. У наставку ће бити детаљно описана архитектура и имплементација оба система.

3.4.1 Систем за резервне копије на AWS платформи

Циљ система за резервне копије је да омогући кориснику да, на захтев или аутоматски, сачува комплетну SQLite базу података на сигурну и удаљену локацију. Овај процес мора бити безбедан, ефикасан и једноставан за конфигурацију са корисничке стране. Изабрана је *serverless* архитектура на AWS-у, која елиминише потребу за одржавањем сервера и нуди високу скалабилност и поузданост уз плаћање по потрошњи (енгл. *pay-as-you-go*).

3.4.1.1 Архитектура AWS сервиса

Систем се састоји од четири AWS сервиса који раде заједно:

1. **Amazon S3 (Simple Storage Service)**: Користи се као складиште за чување компресованих `.sqlite` фајлова. Сваки корисник (теретана) добија јединствени сигурни токен, који се користи као префикс (“фолдер”) унутар S3 *bucket*-а, чиме се подаци различитих корисника строго изолују. За *bucket* је конфигурисано верзионисање и

правила која аутоматски бришу старе верзије резервних копија (нпр. чувају се само последње три), оптимизујући трошкове складиштења.

2. **AWS Lambda**: Представља *serverless* рачунарску платформу. Имплементиране су три одвојене Lambda функције написане у програмском језику Python, свака са специфичном наменом:
   - `upload-backup`: Прихвата `POST` захтев са компримованим фајлом, верификује аутентикациони токен и смешта фајл у одговарајући S3 префикс.
   - `download-backup`: Прихвата `GET` захтев, верификује токен и омогућава преузимање одређене верзије резервне копије.
   - `list-versions`: Прихвата `GET` захтев, верификује токен и враћа листу доступних верзија резервних копија са метаподацима (датум, величина).

3. **Amazon API Gateway**: Служи као сигуран и управљив улазни *endpoint* за Lambda функције. Он претвара HTTP захтеве (нпр. `POST /backup/{token}`) у догађаје који покрећу одговарајућу Lambda функцију. Конфигурисан је да рукује бинарним подацима (`application/octet-stream`), примењује CORS правила и прослеђује параметре путање (као што је токен) Lambda функцијама.

4. **AWS IAM (Identity and Access Management)**: Користи се за дефинисање прецизних дозвола. Креирана је посебна IAM улога за Lambda функције која им даје минималне потребне привилегије: дозволу за упис и читање објеката унутар S3 *bucket*-а и дозволу за упис логова у Amazon CloudWatch.

3.4.1.2 Процес слања и враћања резервне копије

Процес са стране десктоп апликације (Rust бекенд) је следећи:

1. Креирање резервне копије: На захтев корисника, апликација користи `VACUUM INTO` SQL команду да креира конзистентну слику (енгл. *snapshot*) локалне SQLite базе у привремени фајл. Овај приступ је сигуран за коришћење док апликација ради, јер исправно рукује WAL (*Write-Ahead Logging*) режимом.

2. Слање на API: Привремени фајл се чита у меморију, а затим се шаље као бинарни податак путем `POST` HTTP захтева на API Gateway *endpoint*. Захтев у заглављу (енгл. *header*) садржи и сигурносни токен који аутентификује клијента.

3. Враћање података (Restore): Процес враћања је вишефазни како би се осигурао интегритет података. Прво се преузима листа доступних верзија. Након што корисник одабере верзију, апликација преузима компримовани фајл, проверава његов интегритет, затвара конекцију ка локалној бази, брише старе `.sqlite-wal` и `.sqlite-shm` фајлове, и тек онда замењује главни `.sqlite` фајл са преузетим.

3.4.1.3 Конфигурација са корисничке стране

Да би систем био флексибилан, корисник (власник теретане) у подешавањима апликације уноси јединствени URL за резервну копију. Овај URL садржи *endpoint* и сигурносни токен, форматиран на следећи начин:

```
https://<api_gateway_id>.execute-api.<region>.amazonaws.com/
prod#<backup_token>
```

Апликација парсира овај URL како би добила базни URL за API позиве и токен за аутентификацију. Токен се генерише приликом првог покретања апликације као јединствени UUID и приказује се кориснику. Уколико корисник жели да користи исту резервнз копију на другом рачунару (пренесе систем на други рачунар), потребно је само да унесе исти URL.

3.4.2 *Online* режим и синхронизација са Supabase платформом

Online режим представља сложену архитектуру која омогућава континуирану синхронизацију података и функционалност мобилне апликације. За ову сврху, изабрана је Supabase платформа због своје интегрисане понуде која укључује PostgreSQL базу, систем за аутентификацију и аутоматски генерисан PostgREST API [13], [14].

3.4.2.1 Архитектура Supabase сервиса

1. **PostgreSQL база података**: Централна база података чија шема одражава поједностављену структуру локалне SQLite базе. Иако је могуће, формалне *foreign key* релације нису дефинисане, како би се поједноставила логика *upsert* (комбинација *update* и *insert*) операција из *Sync Worker*-а. Превод између SQLite и PostgreSQL дијалеката се обавља у самом синхронизационом процесу.

2. **Supabase Auth**: Користи се као систем за управљање идентитетима за кориснике мобилне апликације. Он рукује регистрацијом (путем позивног линка), пријавом (користећи email и лозинку) и издавањем JWT (*JSON Web Tokens*) токена који служе за аутентификацију API захтева.

3. **PostgREST API**: Supabase аутоматски излаже RESTful API преко свих табела у `public` шеми. Овај API је главни канал комуникације за *Sync Worker* из десктоп апликације. API подржава сложене упите, филтрирање и *upsert* операције.

4. **Функције базе података (RPC)**: За сложеније упите, као што је проналажење "најрелевантније" чланарине за корисника, креирана је PostgreSQL функција (`get_my_relevant_membership`). Ова функција енкапсулира комплексну SQL логику на серверу и изложена је као један RPC *endpoint*, што значајно поједностављује код на мобилном клијенту.

5. **Row-Level Security (RLS)**: Ово је кључни сигурносни механизам. На свим табелама којима приступа мобилна апликација активиран је RLS. Дефинисане су полисе

које користе `auth.uid()` функцију како би осигурале да сваки улоговани корисник може да чита **искључиво** податке који су повезани са његовим налогом. Ово ефикасно спречава неовлашћен приступ подацима других корисника.

3.4.2.2 Процес синхронизације

Процес је дизајниран да буде отпоран на грешке и прекиде у конекцији.

1. Хватање промена: Свака `INSERT`, `UPDATE` или `DELETE` операција у локалној SQLite бази над праћеним табелама, путем SQL тригера, аутоматски креира запис у `pending_changes` табели. Овај запис садржи све информације потребне за репликацију операције (назив табеле, тип операције, ID записа и JSON пакет (енгл. *payload*) са новим подацима).
2. Слање промена: *Sync Worker* у Rust бекенду периодично преузима групу (енгл. *batch*) записа из `pending_changes` са статусом *pending*.
3. "Claim-then-process" механизам: Да би се спречило да више *worker*-а обрађује исти запис, *worker* прво атомски ажурира статус записа у *processing*. Тек након тога шаље податке на Supabase PostgREST API.
4. Руковање грешкама: Уколико слање не успе, *worker* ажурира запис у `pending_changes`, повећава бројач покушаја (`attempts`) и уписује поруку о грешци. Након одређеног броја неуспешних покушаја, запис се означава као *failed* и изузима из даље обраде, како не би блокирао остатак реда. Уколико је слање успешно, запис се брише из `pending_changes` табеле.

Овај шаблон осигурава да се ниједна промена не изгуби и да систем може наставити са радом чак и након дужих периода без интернет конекције.

3.5 Имплементација мобилне апликације

Мобилна апликација представља битну тачку интеракције између фитнес центра и његових чланова, пружајући им дигитални алат који замењује физичку чланску картицу и нуди тренутни увид у статус њиховог чланства. За развој је изабран Flutter радни оквир компаније Google, који омогућава креирање нативних апликација за Android и iOS платформе у програмском језику Dart. Овај избор доноси значајне предности у виду бржег развоја, конзистентног корисничког искуства на обе платформе и приступа богатом екосистему готових компоненти [15].

Апликација је свесно дизајнирана да буде једноставна, брза и фокусирана на две основне функције: приказ дигиталне чланске картице у виду бар-кода и преглед детаља о тренутној или предстојећој чланарини. Сва комуникација са позадинским системом одвија се директно са Supabase платформом.

3.5.1 Архитектура и организација кода

Архитектура мобилне апликације је постављена тако да је код организован у директоријуме који одговарају функционалним целинама. Овакав приступ олакшава навигацију и одржавање кода.

```
lib/
├── api/
│ └── supabase_service.dart # Сва комуникација са Supabase API-јем
├── auth/
│ ├── auth_repository.dart # Логика за пријаву и одјаву
│ └── login_page.dart # UI за страницу за пријаву
├── core/
│ ├── models/ # Модели података (Member, Membership)
│ └── widgets/ # UI компоненте
├── home/
│ ├── home_page.dart # Главни екран са табовима
│ ├── barcode_tab.dart # UI за приказ бар-кода
│ └── membership_tab.dart # UI за приказ детаља чланарине
├── main.dart # Улазна тачка апликације
└── splash_page.dart # Екран за проверу аутентификације
```

Ова структура јасно одваја слој за приступ подацима (`api`), слој за аутентификацију (`auth`), дељене ресурсе (`core`) и главне функционалности апликације (`home`).

3.5.2 Управљање аутентификацијом и сесијом

Аутентификација корисника се у потпуности ослања на Supabase Auth. Процес је дизајниран да буде сигуран и да пружи беспрекорно корисничко искуство.

1. **Пријава**: Корисник уноси своју е-пошту и лозинку (коју је претходно поставио преко веб странице). Апликација позива `supabase.auth.signInWithPassword()` метод.
2. **Управљање сесијом**: Након успешне пријаве, `supabase-flutter` библиотека аутоматски безбедно складишти добијени JWT токен (сесију) на уређају (`SharedPreferences` на Android-у, `NSUserDefaults` на iOS-у).
3. **Аутоматско освежавање токена**: Библиотека такође аутоматски рукује освежавањем краткотрајних *access* токена користећи дуготрајни *refresh* токен. Ово значи да ће корисник остати улогован недељама или месецима, све док не изабере опцију за одјаву, што је очекивано понашање за модерне мобилне апликације.
4. **Реактивност на стање сесије**: Улазна тачка апликације (`main.dart`) користи `StreamBuilder` који слуша `supabase.auth.onAuthStateChange` ток. Овај механизам омогућава да се кориснички интерфејс аутоматски мења у зависности од тога да ли је корисник улогован или не, елегантно пребацујући између `LoginPage` и `HomePage` без икакве ручне логике.

Пример `StreamBuilder`-а у `main.dart`:

```
home: StreamBuilder<AuthState>(
  stream: supabase.auth.onAuthStateChange,
  builder: (context, snapshot) {
    if (snapshot.hasData && snapshot.data?.session != null) {
      return const HomePage(); // Корисник је улогован
    }
    return const LoginPage(); // Корисник није улогован
  },
),
```

3.5.3 Приступ подацима и сигурност

Како мобилна апликација ради у несигурном окружењу (на уређају корисника), приступ подацима је ригорозно контролисан.

- Коришћење *anon* кључа: Апликација се повезује на Supabase користећи јавни, *anonymous* кључ. Овај кључ сам по себи не даје никакве дозволе за читање података.
- Row-Level Security (RLS): Приступ подацима је омогућен искључиво кроз RLS полисе дефинисане на PostgreSQL бази. Свака полиса користи `auth.uid()` функцију да идентификује улогованог корисника и дозволи му приступ само оним редовима у табелама који су директно повезани са његовим налогом. Ово је темељ сигурности апликације.
- Позивање RPC функције: Уместо да апликација прави сложене упите, она позива једну, претходно дефинисану базу података функцију `get_my_relevant_membership`. Ова функција на серверу извршава сву потребну логику (укључујући JOIN са другим табелама) и враћа само минимални скуп података потребан за приказ. Овај приступ је ефикасан и додатно побољшава сигурност јер је логика енкапсулирана на серверу.

Пример позива RPC функције из *SupabaseService*:

```
final membershipResponse = await _supabase
    .rpc('get_my_relevant_membership')
    .single();

final membership = Membership.fromJson(membershipResponse);
```

3.5.4 Кориснички интерфејс и искуство

Кориснички интерфејс је изграђен коришћењем Flutter декларативног модела и скупа *Material Design* компоненти. Главни екран (*HomePage*) користи `BottomNavigationBar` за лаку навигацију између два главна таба: “Картица” и “Чланарина”.

- Таб “Картица”: Овај екран има централну улогу. Након што се подаци учитају, приказује јасан и велики бар-код генерисан помоћу `barcode_widget` пакета.

- Таб “Чланарина”: Овај екран приказује детаље о чланарини на структуриран начин, користећи низ “картица” за сваку информацију (статус, датум почетка, датум истека, преостале посете)

Асинхроно учитавање података: Оба таба користе `FutureBuilder` виџет. Ово омогућава да се приликом учитавања података са мреже прикаже индикатор учитавања (енгл. *loading spinner*), а у случају грешке, јасна порука кориснику. Ово спречава “замрзавање” интерфејса и побољшава корисничко искуство.

3.5.5 Интернационализација (i18n)

Апликација је од почетка дизајнирана да подржава више језика. Коришћен је уграђени `flutter_localizations` систем. Сви текстови видљиви кориснику су издвојени у .arb (*Application Resource Bundle*) фајлове за сваки подржани језик (енглески, српски). Flutter аутоматски бира одговарајући језик на основу подешавања на телефону корисника, чинећи апликацију доступном широј публици.

3.5.6 Дистрибуција

Апликација се гради у оптимизовани, потписани бинарни фајл спреман за дистрибуцију.

- За Android, генерише се *Android App Bundle* (.aab) фајл, који је стандард за објављивање на Google Play продавници.
- За iOS, процес израде се врши кроз Xcode, што је предуслов за дистрибуцију на Apple App Store.

3.6 Имплементација веб апликације за управљање лозинком

Веб апликација за управљање лозинком још једну једноставну компоненту GymMan система. Њена основна улога је да послужи као сигуран повезник између позива који члан добије путем е-поште и његовог успешног активирања налога за мобилну апликацију. Ова компонента решава два кључна сценарија:

1. **Постављање иницијалне лозинке**: Када администратор у десктоп апликацији позове новог члана, Supabase Auth шаље е-поруку са јединственим *magic link*-ом. Кликом на тај линк, корисник долази на ову веб страницу како би дефинисао своју лозинку по први пут.
2. **Ресетовање заборављене лозинке**: Уколико члан заборави своју лозинку, од радника у теретани може затражити процес ресетовања лозинке. Ова опција покреће слање е-поруке за ресетовање, која га такође усмерава на ову исту веб страницу.

Избор да се ова функционалност реализује као веб апликација, а не унутар саме мобилне апликације, је стратешки. Веб линкови су универзални и функционишу на свим уређајима, без обзира да ли корисник има већ инсталирану мобилну апликацију или не. Ово значајно поједностављује процес иницијалне регистрације. Веб апликација је

потпуно прилагођена мобилним уређајима, осигуравајући несметано функционисање када се приступа преко телефона.

3.6.1 Технологије и хостинг

За развој веб апликације изабран је SvelteKit, исти радни оквир који се користи и за фронтенд десктоп апликације. За хостинг је изабрана Netlify платформа.

Netlify је изабран због своје једноставности, бесплатног *tier*-а за мање пројекте и одличне интеграције са Git-ом, што омогућава континуирану испоруку (енгл. *Continuous Deployment*) [16]. Свака промена на главној грани репозиторијума аутоматски покреће процес израде и објављивања нове верзије сајта.

3.6.2 Процес постављања и ресетовања лозинке

Процес је дизајниран да буде што једноставнији за корисника, док се у позадини одвија сигуран протокол за аутентификацију који диктира Supabase Auth.

1. Покретање процеса регистрације или ресетовања лозинке: Радник у теретани иницира позив новог члана или процес ресетовања лозинке преко десктоп апликације. Supabase Auth шаље е-поруку са јединственим линком који садржи `access_token`.

2. Долазак на страницу: Корисник кликне на линк у својој е-поруци. Тај линк садржи *URL fragment* (део URL-а након #) у којем се налази `access_token` који је генерисао Supabase. Пример URL-а: `https://gymman-set-password.netlify.app/#access_token=...&token_type=bearer&...`

3. Аутоматска обрада токена: Након што се страница учита у прегледачу, `supabase-js` библиотека (клијент за Supabase на фронтенду) аутоматски детектује овај *URL fragment*. Она користи добијени `access_token` да покрене привремену, аутентификовану сесију за корисника. У овом тренутку, иако корисник још није унео лозинку, он је за Supabase систем "улогован".

4. Унос нове лозинке: Кориснички интерфејс приказује једноставну форму са пољем за унос нове лозинке и дугметом за потврду.

5. Ажурирање корисничког налога: Када корисник унесе и потврди лозинку, апликација позива `supabase.auth.updateUser()` метод, прослеђујући нову лозинку.

```
async function handleSetPassword(newPassword) {
  const { data, error } = await supabase.auth.updateUser({
    password: newPassword
  });
}
```

Пошто је корисник у привременој сесији (из корака 2), Supabase Auth зна за који налог треба да ажурира лозинку. Овај позив је сигуран јер је ауторизован `access_token`-ом из URL-а.

6. Завршетак процеса: Након успешног ажурирања, сесија се уништава, а кориснику се приказује порука која га упућује да преузме моблину апликацију и да се пријави користећи своју е-пошту и нову лозинку.

3.6.3 Интеграција са сервисом за слање е-поште (Resend)

Иако Supabase има уграђен систем за слање е-поште, он има одређена ограничења у погледу броја послатих порука и могућности прилагођавања. Да би се обезбедила већа поузданост и бољи *deliverability* (способност да мејлови не заврше у *spam* фолдеру), систем је конфигурисан да користи спољни SMTP сервис. За ову сврху изабран је Resend сервис за е-поруке.

Глава 4

Приказ имплементираног система

Након детаљног описа архитектуре и техничке имплементације, у овом поглављу биће представљен визуелни и функционални приказ готовог софтверског решења GymMan. Циљ је да се кроз серију снимака екрана (енгл. *screenshots*) и пратећих описа демонстрирају кључне функционалности система и прикаже корисничко искуство из перспективе различитих улога: администратора, радника и члана теретане.

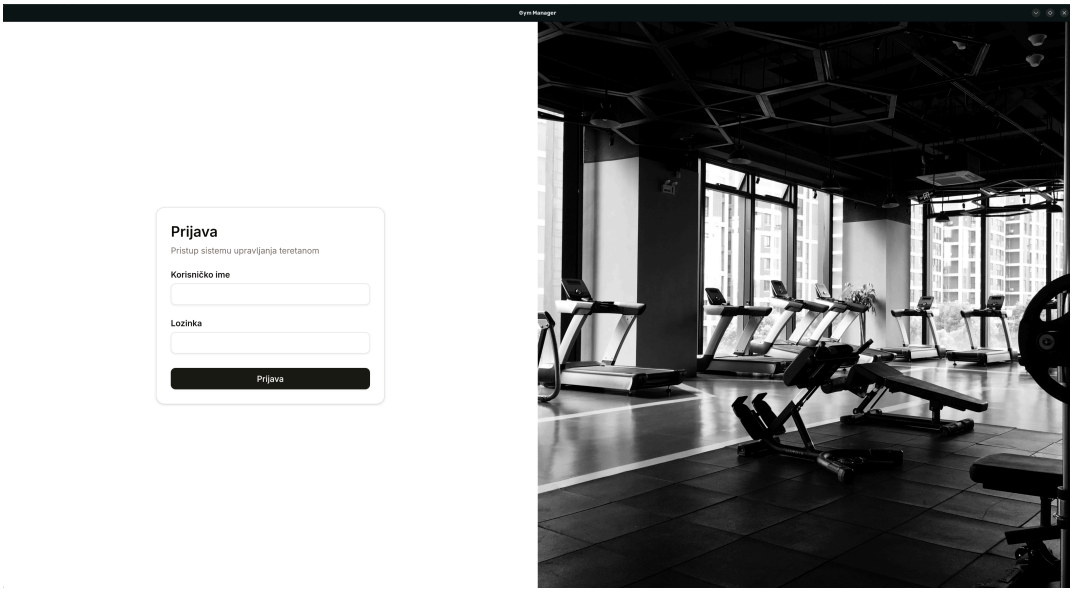
Поглавље је структурирано тако да прати логичан ток коришћења апликације, почевши од аутентификације, преко основних операција управљања, па све до приказа мобилне апликације и пратећих веб сервиса. Овај визуелни преглед служи као доказ успешне реализације претходно дефинисаних функционалних захтева и као илустрација финалног производа. Снимци екрана биће приказани са светлом темом апликације и одговарајућим нивоом увеличања због боље видљивости детаља на штампаном издању рада.

4.1 Десктоп апликација: Интерфејс за управљање

Десктоп апликација представља централни алат за управљање фитнес центром. Њен кориснички интерфејс је дизајниран да буде интуитиван, брз и прилагођен свакодневним операцијама.

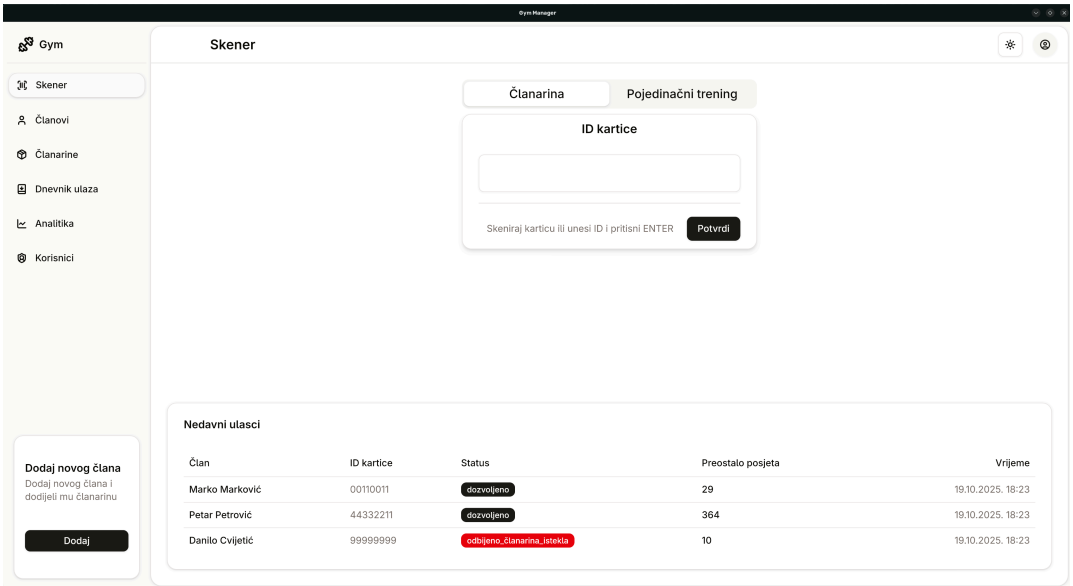
4.1.1 Пријава и главни екран

Приликом покретања апликације, корисника дочекује екран за пријаву (слика 6). Радници уносе своје корисничко име и лозинку, који се локално верификују коришћењем сигурносног Argon2 хеш алгоритма.



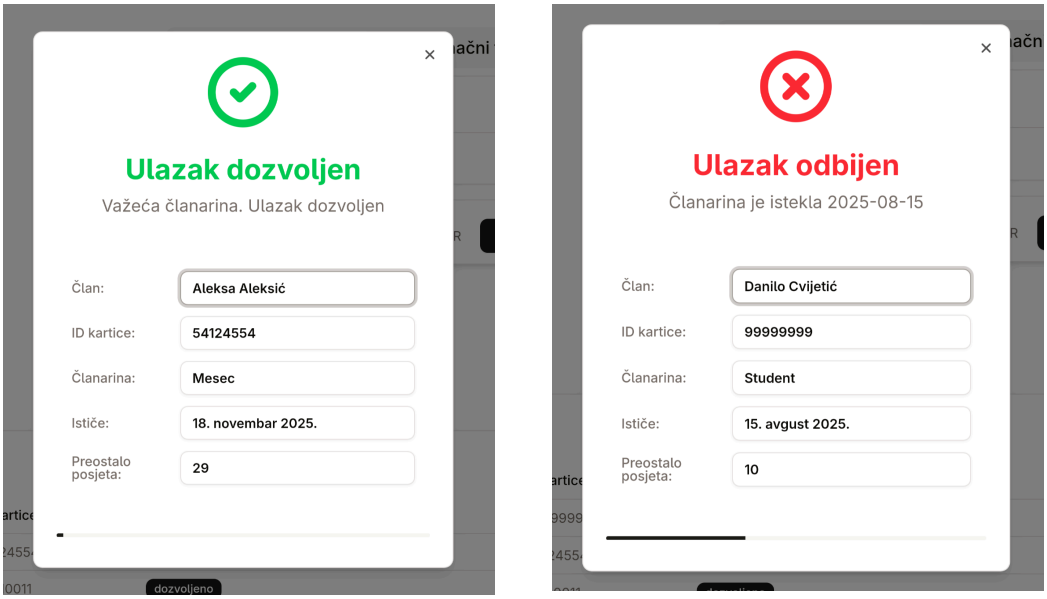
Слика 6: Екран за пријаву у десктоп апликацију

Након успешне пријаве, отвара се главни екран апликације, који је уједно и контролна табла скенера (слика 7). Овај екран је оптимизован за брзу обраду долазака чланова. У горњем делу се налази поље за унос броја картице, док се у доњем делу приказује листа последњих евидентираних улазака, са јасним визуелним индикаторима за одобрен (зелено) или одбијен (црвено) приступ. Са леве стране се налази главни навигациони мени који омогућава приступ осталим модулима система. Интерфејс је подешен да аутоматски фокусира унос на поље за скенирање, омогућавајући радницима да брзо и ефикасно обрађују доласке чланова без додатних корака при употреби уређаја за скенирање бар-кода.



Слика 7: Главни екран за скенирање чланских картица

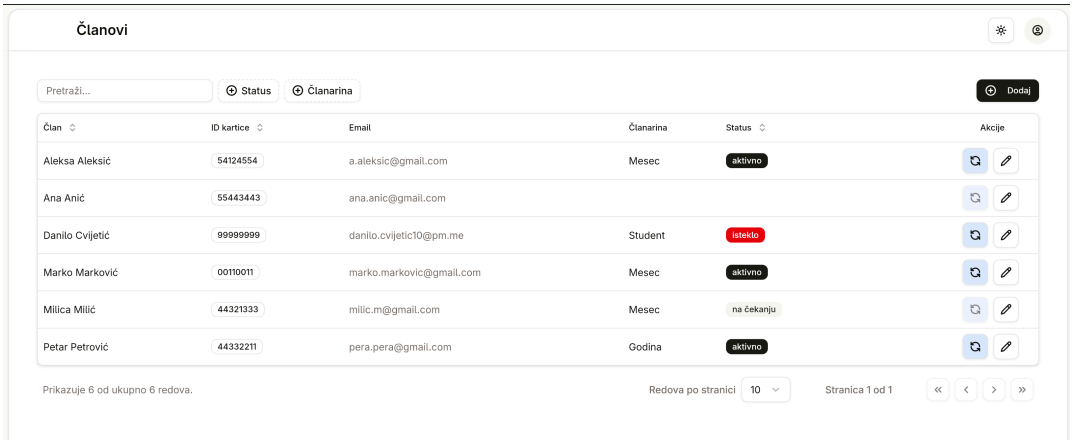
При скенирању бар-кода, систем тренутно проверава валидност чланарине и аутоматски евидентира улазак у базу података. Уколико је чланарина активна, приступ је одобрен и на екрану се приказује порука са именом члана (слика 8). У супротном, приступ се одбија уз одговарајућу обавештење.



Слика 8: Примери екрана скенера за одобрен и одбијен приступ члану

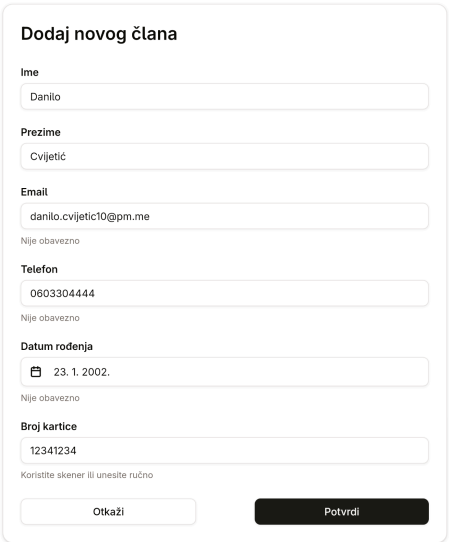
4.1.2 Управљање члановима

Модул за управљање члановима омогућава комплетну евиденцију свих клијената. Главни приказ је табела са листом свих чланова, која подржава претрагу, сортирање и пагинацију (слика 9). Ово омогућава радницима да брзо пронађу одређеног члана.

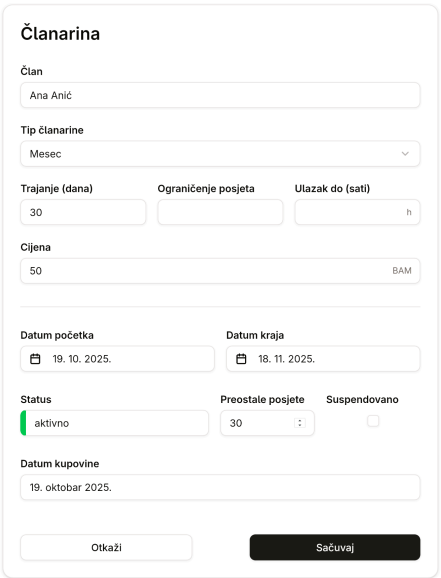


Слика 9: Листа чланова са опцијама за претрагу и филтрирање

Кликом на дугме “Додај” отвара се форма за регистрацију новог члана (слика 10), где се уносе сви потребни подаци. Форма укључује валидацију у реалном времену како би се спречио унос неисправних података. Након уписа у базу, систем нуди опцију за додавање чланарине новокреираном члану (слика 11).

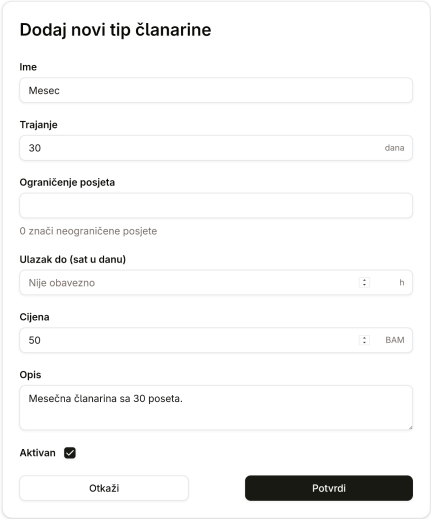


Слика 10: Форма за додавање новог члана у систем

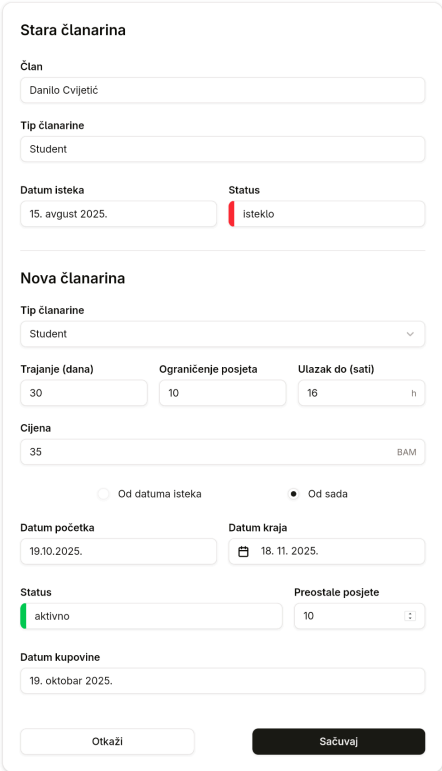


Слика 11: Форма за додавање чланарине новом члану

Такође, систем нуди могућност продужења постојећих чланарина (слика 13).



Слика 12: Форма за додавање новог типа чланарине

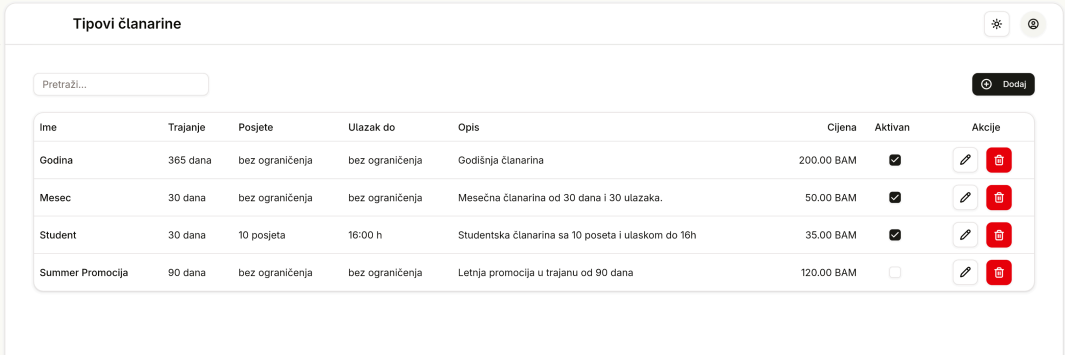


Слика 13: Форма за продужење постојеће чланарине

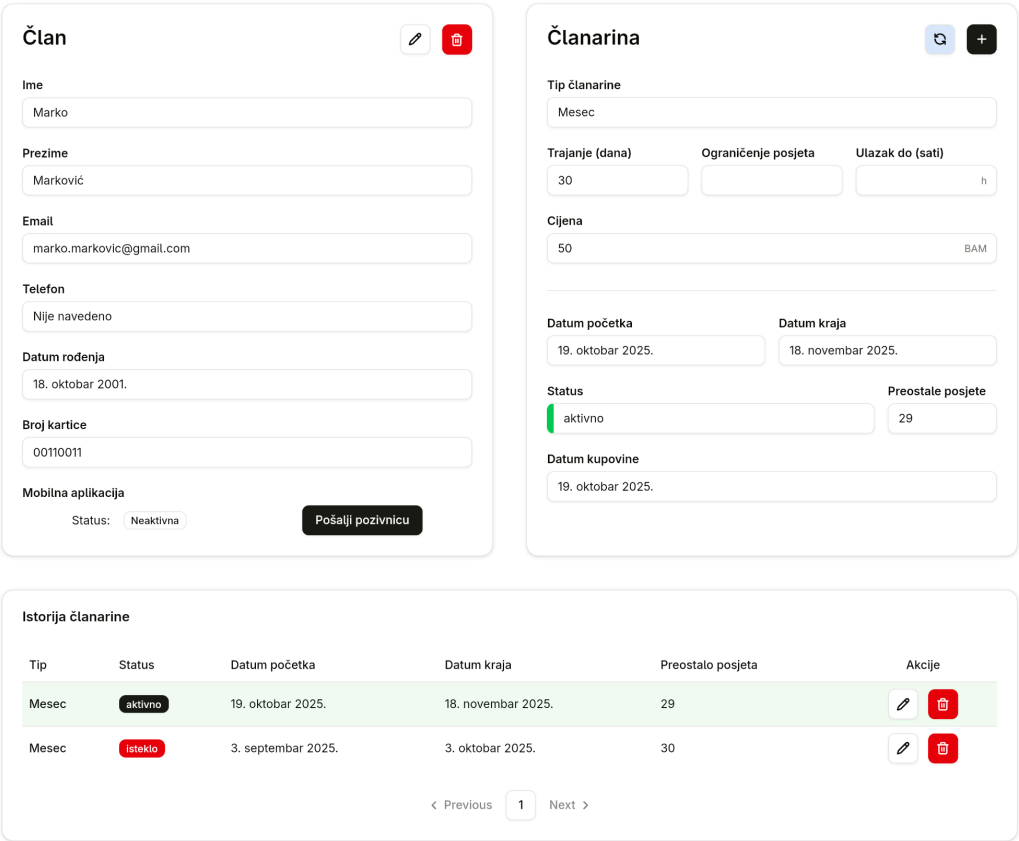
4.1.3 Управљање чланаринама

Систем омогућава креирање различитих типова чланарина (слика 12). Администратор може дефинисати назив, цену, трајање, број посета и временска ограничења за сваки тип чланарине.

На слици 14 приказана је табела са листом свих дефинисаних типова чланарина. Табела подржава претрагу што олакшава проналажење одређеног типа чланарине.



Слика 14: Листа типова чланарина са опцијама за претрагу



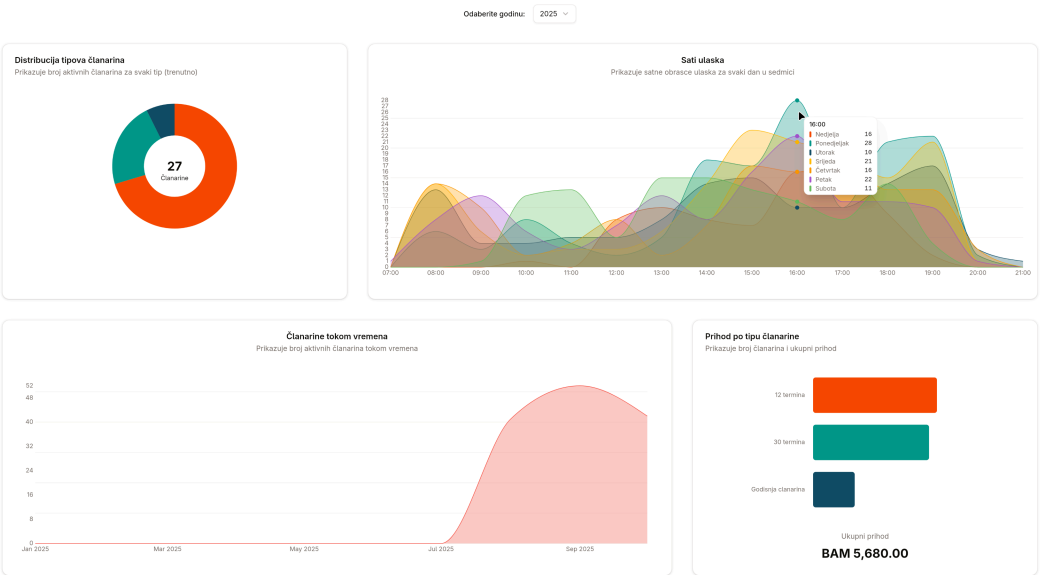
Слика 15: Приказ детаља о члану и управљање његовим чланаринама.

Додела и преглед чланарина врши се на страници са детаљима о члану (слика 15). Систем јасно визуелно одваја активну чланарину од оних које су истекле или су на чекању. Овде радник може додати нову чланарину, прегледати историју претходних

чланарина и видети тренутно активан статус. Такође, може продужити или суспендовати тренутну чланарину. Овај поглед пружа опцију за слање позива за креирање налога мобилне апликације.

4.1.4 Аналитика и извештавање

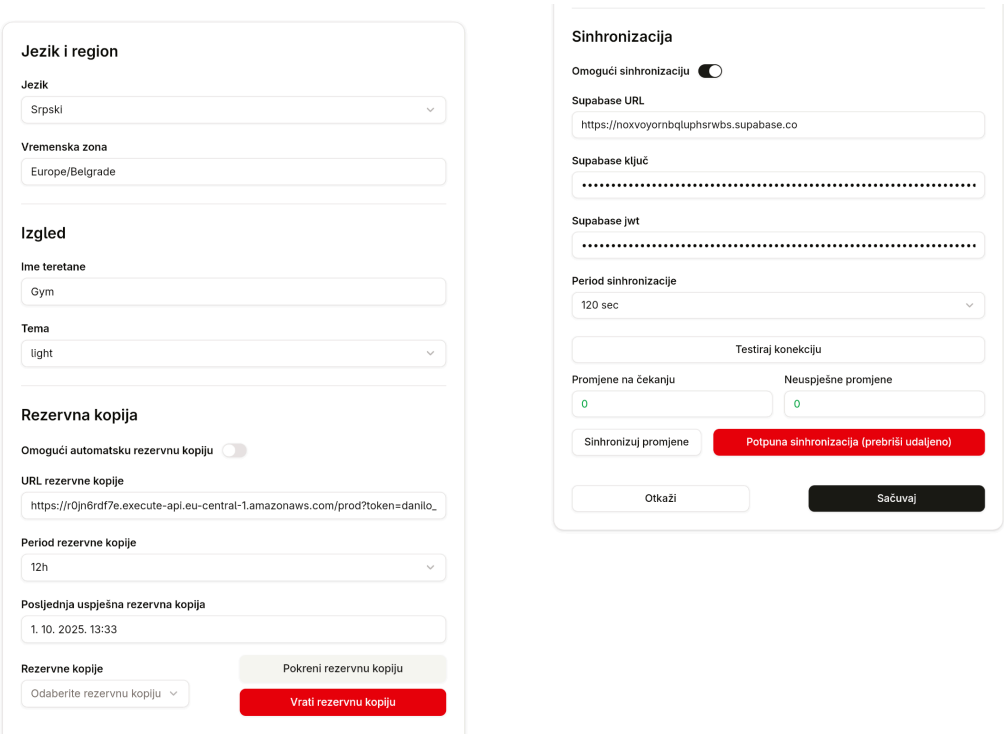
Модул за аналитику пружа визуелни увид у кључне метрике пословања. Коришћењем интерактивних графикона, администратори могу пратити трендове и доносити информисане одлуке. На слици 16 приказана је контролна табла са четири кључна графикона: дистрибуција прихода по типу чланарине, укупне посете по данима и сатима, број активних чланова током времена и дистрибуција типова чланарина.



Слика 16: Контролна табла за аналитику са кључним метрикама пословања

4.1.5 Подешавања и синхронизација

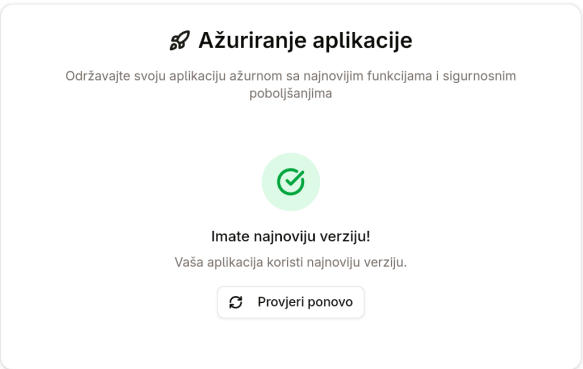
У делу за подешавања (слика 17), корисник може да конфигурише параметре апликације, као што су језик, тема (светла/тамна) и подешавања за *online* режим. Овде се уноси URL за резервне копије, као и URL и кључ за Supabase, чиме се активира синхронизација података. Такође, омогућено је и ручно покретање процеса прављења резервне копије, као и преписивање свих података на удаљену базу.



Слика 17: Екран за подешавање апликације

4.2 Ажурирање десктоп апликације

Интерфејс за ажурирање апликације је интегрисан у саму десктоп апликацију (слика 18). Када је доступна нова верзија, корисник може врло лако да покрене процес надоградње.



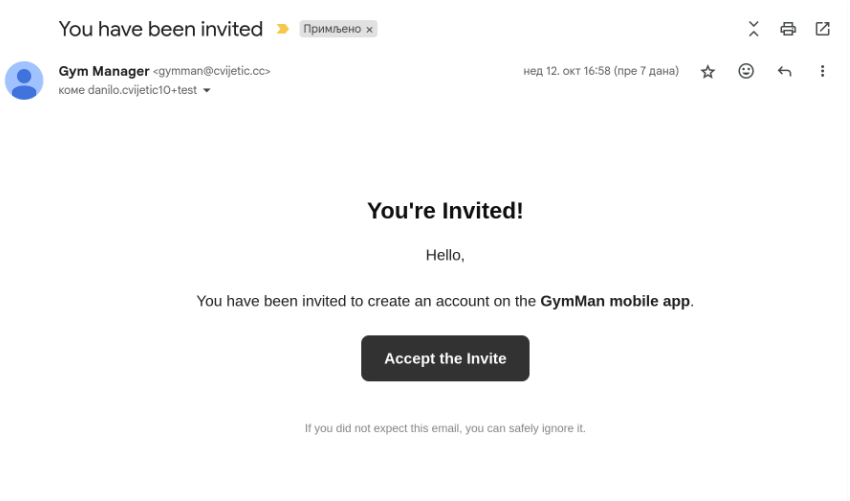
Слика 18: Екран за ажурирање десктоп апликације

4.3 Процес активације и коришћења мобилне апликације

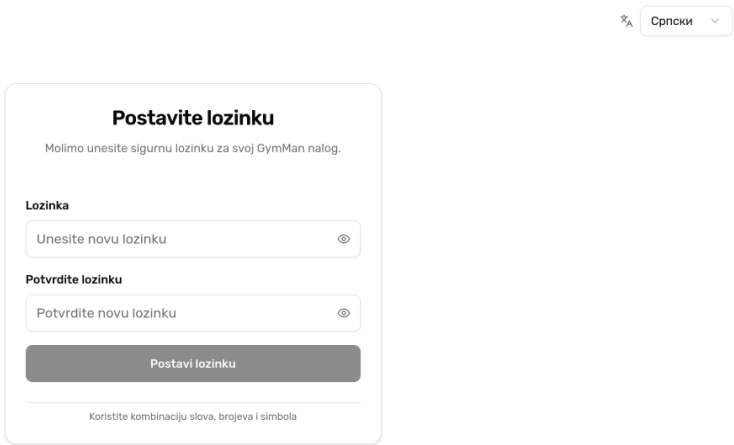
Процес активације за чланове је дизајниран да буде једноставан и сигуран.

4.3.1 Позив и постављање лозинке

Након што администратор пошаље позив, члан добија е-поруку са линком (слика 19). Кликом на линк, отвара се веб страница за постављање лозинке (слика 20). Ова страница омогућава кориснику да безбедно креира лозинку за свој налог. Након успешног постављања, кориснику се нуди линк за преузимање мобилне апликације.



Слика 19: Позивни мејл за новог члана



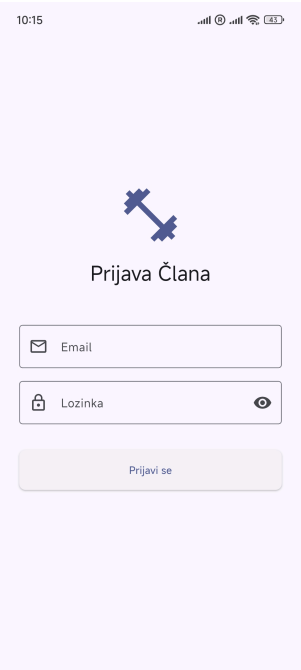
Слика 20: Веб страница за постављање лозинке новог члана

4.3.2 Мобилна апликација: Пријава и главни екран

Након преузимања апликације, члан се пријављује користећи своју е-пошту и лозинку коју је управо поставио (слика 21).

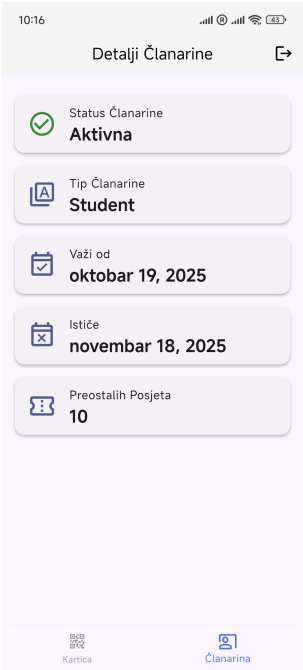
Главни екран мобилне апликације је подељен на два таба. Први таб, “Картица”, приказује персонализовану поруку добродошлице и велики, јасно видљив бар-код (слика 22). Овај бар-код члан може скенирати на улазу у теретану уместо физичке картице.

Други таб, “Чланарина”, пружа детаљан преглед тренутног статуса чланарине, укључујући датум почетка и истека, као и број преосталих посета (слика 23). Подаци су представљени на јасан и прегледан начин, омогућавајући члану да у сваком тренутку има увид у свој статус.



Слика 21: Екран за пријаву у мобилну апликацију

Слика 22: Екран са бар-кодом у мобилној апликацији



Слика 23: Екран са детаљима чланарине у мобилној апликацији

Глава 5

Закључак

Кроз овај рад представљен је процес пројектовања, развоја и имплементације техничког решења GymMan, намењеног управљању фитнес центрима. Рад је започет анализом изазова са којима се сусрећу мањи и средњи фитнес центри, где су као кључни проблеми идентификовани недостатак приступачних и поузданих софтверских решења. Постојећи системи на тржишту често намећу ригидан “cloud-only” модел који је неприкладан за окружења са нестабилном интернет конекцијом, или су у потпуности локални, чиме се губи могућност модерне дигиталне интеракције са клијентима.

Као одговор на ове проблеме, пројектована је и реализована хибридна архитектура која комбинује најбоље од оба света. Срж система чини *cross-platform* десктоп апликација са *offline-first* приступом, која омогућава потпуну функционалност и брз одзив без обзира на стање мреже. Ова централна компонента је допуњена опционом инфраструктуром у облаку која омогућава сигурносне резервне копије, асинхрону синхронизацију података и подршку за пратећу мобилну апликацију намењену члановима.

5.1 Остварени резултати и решени проблеми

Реализовани систем GymMan успешно решава све иницијално постављене функционалне и нефункционалне захтеве. Главни допринос овог рада огледа се у успешној интеграцији различитих технологија у кохерентан и робустан екосистем.

Кључни остварени резултати су:

1. **Развијена је потпуно функционална десктоп апликација** за управљање члановима, чланаринама, типовима чланарина и контролу приступа. Апликација, изграђена помоћу Tauri и SvelteKit оквира, пружа модеран и интуитиван кориснички интерфејс, доступан на Windows, macOS и Linux оперативним системима.

2. **Имплементирана је “offline-first” архитектура** као темељ система. Коришћењем локалне SQLite базе података као примарног извора података, постигнута је велика поузданост и перформансе. Све операције се извршавају тренутно, елиминишући кашњења узрокована мрежном комуникацијом и омогућавајући непрекидан рад чак и у потпуно офлајн окружењу.

3. **Дизајниран је и имплементиран робустан механизам за асинхрону синхронизацију**. Коришћењем SQL тригера и *outbox* обрасца, свака промена у локалној бази се аутоматски бележи. Позадински процес у Rust-у, користећи “claim-then-process”

логику, поуздано шаље ове промене на централну PostgreSQL базу на Supabase платформи, уз механизме за поновне покушаје и руковање грешкама. Овим је имплементиран образац евентуалне конзистентности, омогућавајући поуздану репликацију података уз минималан утицај на корисничко искуство.

4. **Креиран је сигуран и скалабилан систем за резервне копије**. Интеграцијом са AWS сервисима (S3, Lambda, API Gateway), корисницима је омогућено да на једноставан начин праве верзионисане резервне копије својих података на удаљеној локацији, чиме се штите од губитка података услед хардверских кварова.

5. **Развијена је мобилна апликација за чланове** коришћењем Flutter оквира. Апликација омогућава члановима да прате статус свог чланства и користе дигитални бар-код за приступ теретани. Сигурност података је обезбеђена кроз Supabase Auth систем за аутентификацију и Row-Level Security (RLS) полисе које гарантују да сваки корисник може приступити искључиво својим подацима.

6. **Успостављен је комплетан CI/CD процес** за дистрибуцију и аутоматско ажурирање десктоп апликације. Коришћењем *GitHub Actions* и *Tauri Updater* плагина, процес објављивања нових верзија је у потпуности аутоматизован, од компајлирања и дигиталног потписивања, до обавештавања корисника и инсталације ажурирања.

5.2 Предности и мане имплементираног решења

Свако инжењерско решење представља компромис, па тако и систем GymMan има своје предности и недостатке.

Предности:

- Поузданост и перформансе: “Offline-first” приступ гарантује да је апликација увек доступна и брза, што је од кључног значаја за пословне операције које се одвијају у реалном времену, као што је контрола приступа.
- Флексибилност: Систем може да функционише у три различита режима: потпуно офлајн, офлајн са резервном копијом у облаку, или потпуно онлајн са синхронизацијом. Ово омогућава да се прилагоди потребама и техничким могућностима различитих фитнес центара.
- Сигурност и контрола података: Корисници задржавају власништво над својим подацима на локалном уређају. “Cloud” компоненте су опционе и користе модерне сигурносне праксе (RLS, JWT, хеширање лозинки, HTTPS).
- Ниски трошкови одржавања: Ослањање на *serverless* архитектуру (AWS Lambda, Supabase) и *open-source* технологије значајно смањује оперативне трошкове у поређењу са традиционалним серверским решењима.
- Модерно корисничко искуство: Комбинација интуитивне десктоп апликације и пратеће мобилне апликације подиже квалитет услуге фитнес центра.

Мане и ограничења:

- Сложеност синхронизације: Иако је имплементирани *outbox* образац робустан, он подразумева једносмерну синхронизацију (са локалне на удаљену базу). Увођење двосмерне синхронизације би увело значајну комплексност у виду решавања конфликата (енгл. *conflict resolution*), што је изван оквира овог рада.
- Потенцијално кашњење података на мобилној апликацији: Због асинхроне природе синхронизације, подаци на мобилној апликацији могу каснити неколико секунди или минута у односу на стање у локалној бази, у зависности од интервала синхронизације и стања мреже. За тренутне потребе (преглед статуса), ово кашњење је прихватљиво.
- Захтеви за иницијалну конфигурацију: Иако је коришћење апликације једноставно, иницијално подешавање *online* режима (унос URL-ова и API кључева) захтева одређени ниво техничког знања администратора система и комуникацију са инжењерима за подршку.

5.3 Правци даљег развоја

Имплементирани систем представља чврсту основу на коју се могу надоградити бројне нове функционалности. Неки од потенцијалних праваца за даљи развој су:

- Интеграција система за плаћање: Додавање могућности за обраду онлајн плаћања чланарина путем “payment gateway” сервиса (нпр. Stripe) [17], што би омогућило члановима да самостално продужавају чланарину преко мобилне апликације.
- Напредни модул за извештавање: Развој комплекснијих аналитичких извештаја и функционалности извоза података у различитим форматима (CSV, PDF) за потребе пословне анализе.
- Систем за заказивање групних тренинга: Додавање модула који би омогућио члановима да преко мобилне апликације виде распоред тренинга и резервишу своје место.
- Подршка за више објеката: Проширење архитектуре како би један власник могао да управља са више фитнес центара унутар једне инстанце апликације, уз раздвајање података по локацији.
- Двосмерна синхронизација: Истраживање и имплементација механизма за двосмерну синхронизацију, што би омогућило, на пример, да се измене направљене на *cloud*-у (као што је онлајн обнова чланарине) рефлектују назад у локалној бази. У овом случају, прелазак на другу локалну базу података (нпр. Turso [18]) која нативно подржава ову функционалност могао би бити разматран.

У закључку, овај рад је успешно демонстрирао како се комбинацијом модерних *cross-platform* технологија и промишљеном архитектуром може креирати свеобухватно, поуздано и скалабилно софтверско решење. Пројекат GymMan не само да решава конкретне проблеме у пословању фитнес центара, већ служи и као практичан пример

примене савремених инжењерских принципа у развоју комплексних информационих система.

Глава 6

Додатак - Списак коришћених скраћеница и појмова

Табела 1: Списак коришћених скраћеница

Скраћеница	**Опис**
API	Application Programming Interface (апликациони програмски интерфејс)
AWS	Amazon Web Services (Амазон веб сервиси)
CI/CD	Continuous Integration / Continuous Delivery (континуирана интеграција / континуирана испорука)
CORS	Cross-Origin Resource Sharing (размена ресурса између извора и дестинације различитог порекла)
CSS	Cascading Style Sheets (језик за описивање стилова)
DOM	Document Object Model (објектни модел документа)
DTO	Data Transfer Object (објекат за пренос података)
HTTP	HyperText Transfer Protocol (протокол за пренос хипертекста)
JSON	JavaScript Object Notation (формат за размену података)
JWT	JSON Web Token (сигурносни токен заснован на JSON формату)
RLS	Row-Level Security (сигурност на нивоу реда)
REST	Representational State Transfer (скуп правила за комуникацију између клијента и сервера)
RPC	Remote Procedure Call (позив удаљене процедуре)
SQL	Structured Query Language (структурирани упитни језик)
TLS	Transport Layer Security (безбедност транспортног слоја)
UML	Unified Modeling Language (језик за моделовање дијаграма)
URL	Uniform Resource Locator (јединствени идентификатор и локатор ресурса)
UI	User Interface (кориснички интерфејс)

Скраћеница	Опис
UUID	Universally Unique Identifier (универзално јединствени идентификатор)
WAL	Write-Ahead Logging (записивање операција унапред)

Табела 2: Списак коришћених појмова

Појам	Објашњење
Асихнрони рад	Рад који се изводи независно од главног тока извршавања, омогућавајући наставак других операција без чекања на његов завршетак.
Bucket (S3)	Логичка јединица за складиштење у AWS S3 сервису, која организује фајлове у облаку.
Read-Only	Режим рада у коме су подаци само за читање, без могућности измене.
Cross-platform	Способност софтвера да се извршава на више различитих оперативних система из истог кода.
Connection pool	Механизам за управљање и поновну употребу веза са базом података како би се побољшале перформансе апликације.
Desktop апликација	Апликација која се инсталира и покреће на рачунарима, за разлику од веб или мобилних апликација.
Endpoint	URL адреса на коју клијент шаље захтеве за приступ одређеним ресурсима или услугама на серверу.
Offline-first	Архитектонски приступ где апликација примарно функционише на локалним подацима, без зависности од интернет конекције.
Outbox образац	Техника за поуздану асинхрону комуникацију где се одлазне поруке (промене) прво уписују у локалну табелу пре слања на спољни систем.
Serverless	Модел “cloud” рачунарства где провајдер динамички алоцира ресурсе, а корисник не управља серверима.
Sync Worker	Позадински процес који се бави синхронизацијом података између локалне и удаљене базе.
Type safety	Својство програмског језика које спречава грешке у типу података током компајлирања.
Update manifest	Фајл који садржи информације о доступним ажурирањима софтвера, укључујући верзије и URL-ове за преузимање.

Списак слика

Слика 1 Архитектура система. 6
Слика 2 Дијаграм класа система. 8
Слика 3 Дијаграм активности за регистрацију новог члана. 9
Слика 4 Дијаграм активности за евиденцију уласка члана. 10
Слика 5 Дијаграм активности за синхронизацију података са облаком. 11
Слика 6 Екран за пријаву у десктоп апликацију 30
Слика 7 Главни екран за скенирање чланских картица 31
Слика 8 Примери екрана скенера за одобрен и одбијен приступ члану 31
Слика 9 Листа чланова са опцијама за претрагу и филтрирање 32
Слика 10 Форма за додавање новог члана у систем 32
Слика 11 Форма за додавање чланарине новом члану 32
Слика 12 Форма за додавање новог типа чланарине 33
Слика 13 Форма за продужење постојеће чланарине 33
Слика 14 Листа типова чланарина са опцијама за претрагу 34
Слика 15 Приказ детаља о члану и управљање његовим чланаринама. 34
Слика 16 Контролна табла за аналитику са кључним метрикама пословања 35
Слика 17 Екран за подешавање апликације 36
Слика 18 Екран за ажурирање десктоп апликације 36
Слика 19 Позивни мејл за новог члана 37
Слика 20 Веб страница за постављање лозинке новог члана 37
Слика 21 Екран за пријаву у мобилну апликацију 38
Слика 22 Екран са бар-кодом у мобилној апликацији 38
Слика 23 Екран са детаљима чланарине у мобилној апликацији 38

Списак табела

Табела 1 Списак коришћених скраћеница 43
Табела 2 Списак коришћених појмова 44

Биографија

Данило Цвијетић рођен је 23. јануара 2002. године у Невесињу, Босна и Херцеговина. Завршио је општи смер Гимназије “Алекса Шантић” у Невесињу 2019. године са титулом ученика генерације. Године 2021. уписао је основне академске студије на Факултету техничких наука у Новом Саду, смер Софтверско инжењерство и информационе технологије. Положио је све испите предвиђене планом и програмом са просечном оценом 9.73.

Литература

[1] O. Kravchenko, M. Leshchenko, D. Marushchak, Y. Vdovychenko, и S. Boguslavska, „The digitalization as a global trend and growth factor of the modern economy“, *SHS Web of Conferences*, том 65, стр. 7004, 2019, doi: 10.1051/shsconf/20196507004.

[2] J. M. Rössel и B. M. Wasalatantri, „How IT Systems create value in fitness facilities“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://urn.kb.se/resolve?urn=urn:nbn:se:umu:diva-227079

[3] N. Y. Kay, „APGym: Enhancing Fitness for APU Gym Users through Gym Resource Monitoring, Reservations, and Fitness Analytics“, *Fusion 2024*, стр. 64, 2024.

[4] A. Sen и S. Madria, „Off-line risk assessment of cloud service provider“, у *2014 IEEE World Congress on Services*, 2014, стр. 58–65. doi: 10.1109/SERVICES.2014.20.

[5] K. Keeton и A. Merchant, „A framework for evaluating storage system dependability“, у *International Conference on Dependable Systems and Networks, 2004*, 2004, стр. 877–886. doi: 10.1109/DSN.2004.1311958.

[6] „Tauri documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://v2.tauri.app/start/

[7] „Goodbye Electron, Hello Tauri“. Приступљено: 20. Октобар 2025. [На Интернету]. Available at: https://dev.to/dedsyn4ps3/goodbye-electron-hello-tauri-26d5

[8] „SQLite documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://sqlite.org/docs.html

[9] „Tokio crate documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://docs.rs/tokio/latest/tokio/

[10] „Tauri filesystem path documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://v2.tauri.app/plugin/file-system/

[11] „GitHub Actions Quickstart documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://docs.github.com/en/actions/get-started/quickstart

[12] „Svelte documentation and quickstart guide“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://svelte.dev/tutorial/svelte/welcome-to-svelte

[13] „PostgREST documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://docs.postgrest.org/en/v13/

[14] „Supabase documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://supabase.com/docs

[15] „Flutter Material Component Widgets“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://docs.flutter.dev/ui/widgets/material

[16] „Netlify documentation and pricing“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://www.netlify.com/pricing/

[17] „Stripe financial services documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://docs.stripe.com/

[18] „Turso documentation“. Приступљено: 10. Октобар 2025. [На Интернету]. Available at: https://docs.turso.tech/introduction