



УНИВЕРЗИТЕТ У НОВОМ САДУ  
ФАКУЛТЕТ ТЕХНИЧКИХ НАУКА У  
НОВОМ САДУ

---



Ана Домоњи

# Развој Андроид апликације за друштвено умрежавање засноване на RESTful архитектури

ЗАВРШНИ РАД

---

Основне струковне студије

Нови Сад, 2024



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

## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Редни број, <b>РБР</b> :	
Идентификациони број, <b>ИБР</b> :	
Тип документације, <b>ТД</b> :	Монографска документација
Тип записа, <b>ТЗ</b> :	Текстуални штампани рад
Врста рада, <b>ВР</b> :	Завршни (Bachelor) рад
Аутор, <b>АУ</b> :	Ана Домоњи
Ментор, <b>МН</b> :	Марко Марковић
Наслов рада, <b>НР</b> :	Развој Андроид апликације за друштвено умрежавање засноване на RESTful архитектури
Језик публикације, <b>ЈП</b> :	Српски
Језик извода, <b>ЈИ</b> :	Српски
Земља публикаовања, <b>ЗП</b> :	Република Србија
Уже географско подручје, <b>УГП</b> :	Војводина
Година, <b>ГО</b> :	2024
Издавач, <b>ИЗ</b> :	Ауторски репринт
Место и адреса, <b>МА</b> :	Нови Сад, трг Доситеја Обрадовића 6
Физички опис рада, <b>ФО</b> : (поглавља/страна/ цитата/табела/слика/графика/прилога)	8/55/0/19/21/0/0
Научна област, <b>НО</b> :	електротехничко и рачунарско инжењерство
Научна дисциплина, <b>НД</b> :	мобилне апликације
Предметна одредница/Кључне речи, <b>ПО</b> :	мобилна апликација, друштвено умрежавање, RESTful архитектура, Retrofit
<b>УДК</b>	
Чува се, <b>ЧУ</b> :	Библиотека Факултета техничких наука, Трг Д. Обрадовића 6, Нови Сад
Важна напомена, <b>ВН</b> :	
Извод, <b>ИЗ</b> :	У овом раду је описан развој Андроид апликације за друштвено умрежавање засноване на RESTful архитектури. За реализацију је коришћена Retrofit библиотека за интеграцију са RESTful API-јем, омогућавајући комуникацију са сервером и размену података у JSON формату. Применом предложеног решења се обезбеђује размена података између корисника, што омогућава корисницима апликације приступ информацијама и интеракцију у реалном времену. Развијен је прототип апликације који демонстрира основне функционалности као што су регистрација корисника, креирање и преглед објава, и повезивање корисника.
Датум прихватања теме, <b>ДП</b> :	



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

## КЉУЧНА ДОКУМЕНТАЦИЈСКА ИНФОРМАЦИЈА

Датум одбране, **ДО**:

Чланови комисије, **КО**: Председник: др Жељко Вуковић, доцент

Члан: др Синиша Николић, доцент

Члан, ментор: др Марко Марковић, ванредни професор

Потпис ментора



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

## KEY WORDS DOCUMENTATION

Accession number, <b>ANO</b> :	
Identification number, <b>INO</b> :	
Document type, <b>DT</b> :	Monographic publication
Type of record, <b>TR</b> :	Textual printed material
Contents code, <b>CC</b> :	Bachelor Thesis
Author, <b>AU</b> :	Ana Domonji
Mentor, <b>MN</b> :	Marko Marković
Title, <b>TI</b> :	Development of Android application for social networking based on RESTful architecture
Language of text, <b>LT</b> :	Serbian
Language of abstract, <b>LA</b> :	Serbian
Country of publication, <b>CP</b> :	Republic of Serbia
Locality of publication, <b>LP</b> :	Vojvodina
Publication year, <b>PY</b> :	2024
Publisher, <b>PB</b> :	Author's reprint
Publication place, <b>PP</b> :	Faculty of Technical Sciences, Trg Dositeja Obradovića 6, Novi Sad
Physical description, <b>PD</b> : (chapters/pages/ref./tables/pictures/graphs/appendixes)	8/55/0/19/21/0/0
Scientific field, <b>SF</b> :	electrical and computer engineering
Scientific discipline, <b>SD</b> :	mobile applications
Subject/Key words, <b>S/KW</b> :	mobile application, social networking, RESTful architecture, Retrofit
<b>UC</b>	
Holding data, <b>HD</b> :	The Library of Faculty of Technical Sciences, Novi Sad, Serbia
Note, <b>N</b> :	
Abstract, <b>AB</b> :	This paper describes the development of an Android application for social networking based on a RESTful architecture. For the implementation, the Retrofit library is used for integration with the RESTful API, enabling communication with the server and data exchange in JSON format. The application of the proposed solution ensures the exchange of data between users, which enables the users of the application to access information and interact in real time. An application prototype is developed that demonstrates basic functionalities such as user registration, creation and review of posts, and user connection.
Accepted by the Scientific Board on, <b>ASB</b> :	



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

## KEY WORDS DOCUMENTATION

Defended on, **DE**:

Defended Board, **DB**:

President:

Željko Vuković, assist. prof.

Member:

Siniša Nikolić, assist. prof.

Menthor's sign

Member,

Marko Marković, assoc. prof.

Mentor:



УНИВЕРЗИТЕТ У НОВОМ САДУ • ФАКУЛТЕТ ТЕХНИЧКИХ  
НАУКА

21000 НОВИ САД, Трг Доситеја Обрадовића 6

## ЗАДАТАК ЗА ЗАВРШНИ РАД

Број:

Датум:

(Податке уноси предметни наставник - ментор)

Студијски програм:	Софтверске и информационе технологије		
Студент:	Ана Домоњи	Број индекса:	SR 46/2021
Степен и врста студија:	Основне струковне студије		
Област:	Електротехничко и рачунарско инжењерство		
Ментор:	Марко Марковић		
НА ОСНОВУ ПОДНЕТЕ ПРИЈАВЕ, ПРИЛОЖЕНЕ ДОКУМЕНТАЦИЈЕ И ОДРЕДБИ СТАТУТА ФАКУЛТЕТА ИЗДАЈЕ СЕ ЗАДАТАК ЗА ЗАВРШНИ РАД, СА СЛЕДЕЋИМ ЕЛЕМЕНТИМА:			
<div><div>- проблем – тема рада;</div><div>- начин решавања проблема и начин практичне провере резултата рада, ако је таква провера неопходна;</div></div>			

### НАСЛОВ ЗАВРШНОГ РАДА:

Развој Андроид апликације за друштвено умрежавање засноване на RESTful архитектури

### ТЕКСТ ЗАДАТКА:

1. Анализирати стање у области друштвених мрежа и одабрати технологије погодне за израду софтверског решења за друштвено умрежавање.
2. Израдити спецификацију захтева овог софтверског решења.
3. Израдити спецификацију дизајна софтверског решења.
4. Имплементирати софтверско решење према израђеној спецификацији.
5. Тестирати имплементирано софтверско решење.
6. Документовати (1), (2), (3), (4) и (5).

Руководилац студијског програма:	Ментор рада:

Примерак за:  - Студента;  - Ментора

# Садржај

<b>1. Увод</b>	<b>1</b>
<b>2. Сродна решења</b>	<b>2</b>
<b>3. Коришћене технологије</b>	<b>4</b>
<b>4. Спецификација захтева</b>	<b>8</b>
4.1. Спецификација функционалних захтева	8
4.2. Спецификација нефункционалних захтева	17
<b>5. Спецификација дизајна</b>	<b>18</b>
5.1. Архитектура система	18
5.2. Модел података	20
<b>6. Имплементација система</b>	<b>23</b>
6.1. Клијентски део апликације	23
6.2. Серверски део апликације	37
<b>7. Демонстрација</b>	<b>43</b>
<b>8. Закључак</b>	<b>53</b>
<b>Литература</b>	<b>54</b>

## 1. Увод

Развој савремених технологија и интернета омогућио је људима широм света да комуницирају и деле информације на начине који су некад били незамисливи. Друштвене мреже су постале кључни облик свакодневне приватне и пословне комуникације, омогућавајући људима да остану у контакту са породицом и пријатељима, упознају нове људе, изражавају своју креативност, уче нове ствари и промовишу своје производе и пословање.

Без друштвених мрежа, било би тешко остварити глобално повезивање и комуникацију међу људима, брзо дељење информација и садржаја, пословима би било много теже да досегну своје циљне групе, представе производе и услуге и умреже се са потенцијалним купцима. Без ових платформи, креативност и забава биле би ограничене, а изградња и организација друштвених заједница захтевала би више времена и ресурса. Проблем социјалне интеракције на интернету решавао се путем првих форума на интернету и соба за ћаскање крајем 20. века. Током година, друштвене мреже су еволуирале, прелазећи од једноставних платформи за размену порука до сложених система за дељење мултимедијалних садржаја.

Мотивација за креирање мобилне апликације друштвене мреже јесте да све ове аспекте учини доступнијим и увек при руци кориснику. Мобилне апликације омогућавају корисницима да користе друштвене мреже без обзира где се налазе, чиме се повећава приступачност и практичност. У будућности, решавање ових проблема постаје још значајније. Како се дигитална интеракција све више интегрише у наше животе, потреба за сигурним, једноставним и ефикасним друштвеним мрежама ће само расти.

Остатак овог рада је организован као што је објашњено у наставку. У другом поглављу су представљена сродна истраживања у изradi мобилне апликације друштвене мреже, док су у трећем поглављу описане коришћене технологије. У четвртном поглављу је дата спецификација захтева за систем представљен у овом раду. Пето поглавље обухвата спецификацију дизајна система. Имплементација решења је представљена у шестом поглављу. Седмо поглавље приказује демонстрацију функционалности система. Закључак је дат у осмом поглављу са смерницама за даљи развој овог система.



## 2. Сродна решења

У овом поглављу је дат преглед постојећих софтверских решења за друштвено умрежавање. Обухваћене су следеће друштве мреже: Facebook [1], Instagram [2], Twitter [3], LinkedIn [4] и TikTok [5].

### Facebook

Facebook је интернет мрежа коју је 2004. године основао Марк Закерберг, у почетку намењена само студентима Харварда како би могли комуницирати и размењивати информације. Међутим, убрзо се проширила у глобалну мрежу са бројном заједницом корисника. Постао је једна од водећих светских друштвених мрежа која данас броји готово две милијарде корисника. Facebook омогућава корисницима да се социјализују, деле фотографије, постављају статусе, играју игре, воде послове и користе га као извор информација.

Упркос многим предностима, Facebook и друге друштвене мреже често смањују културу дружења уживо, замењујући је комуникацијом путем тастатуре. Такође, приватност на друштвеним мрежама може бити нарушена на различите начине, а деца су посебно рањива на преваре, злостављање и непримерене садржаје због недовољно развијене свести о ризицима.

### Instagram

Instagram је једна од најпопуларнијих друштвених мрежа данашњице. Основали су је Кевин Систром и Мајк Кригер, 2010. године. Ова апликација, намењена првенствено за паметне телефоне, омогућава корисницима да деле и објављују фотографије и видео записе. Назив "Instagram" потиче од израза "instant camera", рефлектујући идеју брзог фотографисања и објављивања слика. Поред објављивања фотографија, корисници могу пратити друге људе, користити "hashtag" за лакше проналажење садржаја, и означавати локације на својим објавама.

Instagram је постао озбиљан извор посла за многе људе, познате као инфлуенсери, који користе своје атрактивне профиле и велики број пратилаца да би рекламирали производе и услуге различитих компанија. Предности Instagram-а укључују једноставну комуникацију, могућност зараде путем рекламирања, и богат избор филтера за уређивање слика. Недостаци укључују проблеме које ова мрежа изазива код корисника, укључујући зависност, online насиље, губитак социјализације у стварном животу, и присуство лажних профила.

### Twitter

Twitter је платформа за тзв. микроблогинг која омогућава корисницима да објављују кратке поруке познате као „твитови“, са максимално 280 карактера. Основан 2006. године, од стране Џека Дорсија, Евана Вилијамса и Биз Стоуна, Twitter је стартовао као

SMS сервис за обавештавање пријатеља о тренутним активностима. Временом је постао кључан алат за актуелне информације и дискусије. Брза дистрибуција информација и вести чине Twitter изузетно релевантним за новинаре, политичаре и јавне личности. Платформа омогућава директну комуникацију са јавним личностима и брендovima, коришћење хештагова за повезивање тема и догађаја. Ипак, ограничење дужине твитова може бити фрустрирајуће за детаљније изражавање, а изложеност негативним коментарима представља изазов за многе кориснике.

Twitter се данас користи за брзу комуникацију и размену информација, што га чини идеалним за маркетиншке и пословне сврхе. Кроз време, додате су многе функционалности попут скраћивања линкова и дељења фотографија, што је додатно повећало његову популарност.

## **LinkedIn**

LinkedIn, чији су оснивачи Рид Хофман и Ерик Ли, је присутан у јавности од 2003. године и представља друштвену мрежу намењену професионалцима и пословној заједници. Без обзира на то да ли је корисник директор у великој компанији, власник мале радње, или студент који истражује могућности за каријеру, LinkedIn им помаже да се повежу с другим професионалцима и пронађу нове пословне прилике. Омогућава корисницима да "упознају" друге професионалце, разговарају о својој професији и размењују контакте. Ова мрежа функционише путем "конекција", сличних пријатељствима на Facebook-у, омогућавајући приватне поруке, попуњавање профила и дељење достигнућа.

LinkedIn је погодан за проналажење посла, повезивање са колегама и проширивање професионалне мреже.

## **TikTok**

TikTok је релативно нова друштвена мрежа која је настала 2016. године, али је тек у последњих неколико година доживела енормну популарност. Највише је користе особе између 16 и 24 године, иако је присутан велики број корисника млађих од 16 година, као и старијих. TikTok омогућава корисницима да креирају и објављују кратке видео записе које могу уређивати директно у апликацији, користећи разне песме и аудио исечке. Користи софистицирани алгоритам који корисницима приказује садржај базиран на њиховим интересовањима. Може се користити као плаћени огласни простор са различитим начинима оглашавања.

Иако TikTok пружа креативне могућности, често је критикован због садржаја који се на њему објављују. На овој мрежи може се наићи на присуство говора мржње, национализма, негативних коментара и сукоба између корисника. Истраживања су показала да је значајан проценат садржаја неприкладан, али мрежа ипак привлачи кориснике могућношћу зараде путем преноса уживо и спонзорисаних садржаја.

### 3. Коришћене технологије

Ово поглавље пружа преглед и објашњења технологија које омогућавају израду друштвене мреже каква је приказана у овом раду.

#### Android

Android [6] је софтверска платформа за мобилне уређаје која обухвата многе аспекте управљања овим уређајима, од хардверских компоненти до корисничког интерфејса. Доступан је од 2008. године као пројекат отвореног кода. Android се темељи на Linux кернелу и користи Java [7] виртуелну машину за извршавање апликација, што омогућава широку функционалност и приступ разним хардверским ресурсима уређаја. Једна од кључних карактеристика Android платформе је отвореност која програмерима омогућава слободу у развоју апликација. Апликације имају могућности приступа ресурсима уређаја, што омогућава израду разноврсних типова апликација прилагођених потребама корисника. Аутоматско управљање животним циклусом апликација оптимизује коришћење меморије и других ресурса уређаја, чиме се смањује потреба за ручним затварањем апликација. Поред тога, Android подржава висок квалитет графике и звука, као и рад са разним хардверским компонентама као што су камере и сензори.

Android такође пружа богату инфраструктуру за развој апликација, укључујући библиотеке корисних алата и API-је, чиме се олакшава процес креирања нових апликација. Осим тога, омогућава повезивање са различитим мрежним технологијама и пружа подршку за напредне функционалности као што су гласовно претраживање и видео позиви. Све то чини Android једним од најпопуларнијих оперативних система за мобилне уређаје у свету.

#### Retrofit

Retrofit [8] је софтверска библиотека која олакшава имплементацију HTTP [9] клијената за Android у Java програмском језику, како би се омогућила комуникација са HTTP API-јима. Помаже програмерима да своје API-је претворе у Java интерфејсе, чиме се поједностављује процес слања HTTP захтева и обраде одговора.

Помоћу Retrofit-а, може се дефинисати интерфејс који садржи методе које одговарају различитим HTTP операцијама, као што су `HTTP`, `GET`, `POST`, `PUT`, `PATCH`, `DELETE`, `OPTIONS` и `HEAD`, које се могу користити да се опише тип захтева који метода врши. Retrofit генерише имплементацију интерфејса, омогућавајући да се шаљу захтеви ка серверу. Када се креира Retrofit инстанца, може се одредити основна URL адреса API-ја и додати конвертори за различите формате података, као што су JSON [10] или XML [11], користећи библиотеке као што су Gson [12] или Moshi [13]. Retrofit подржава и асинхроне и синхроне позиве, уз једноставно коришћење повратних позива или Kotlin [14] “suspend” функција за рад са асинхроним захтевима. Такође, омогућава динамичко постављање заглавља и параметара захтева, као и манипулацију URL-овима користећи заменљиве блокове.

## REST API

REST API [15] представља интерфејс за програмирање апликација који се ослања на принципима архитектуралног стила REST, што означава репрезентативни пренос стања. REST се не сматра протоколом или стандардом, већ скупом смерница које програмери могу применити на различите начине. Када клијент затражи податке путем RESTful API-ја, добија репрезентацију стања ресурса у једном од више формата, најчешће JSON, који је читљив како људима, тако и машинама.

Важно је напоменути да су заглавља и параметри такође битни у HTTP методама RESTful API-ја, јер садрже важне идентификационе информације везане за метаподатке захтева, ауторизацију и друге аспекте. Да би се за неки API сматрало да је RESTful, мора да се придржава одређених критеријума, као што су архитектура клијент-сервер, stateless комуникација, кеширање података и униформни интерфејс између компоненти. Овај приступ омогућава брже и лакше коришћење API-ја у поређењу са сложенијим протоколима, што га чини идеалним решењем за развој апликација.

## Spring Boot

Spring Boot [16] је софтверски пакет који значајно олакшава креирање апликација базираних на Spring [17] радном оквиру. Он омогућава програмерима да брзо започну рад са минималним потребним конфигурацијама и укључује алат за командну линију који олакшава покретање и управљање апликацијама.

Главни циљеви Spring Boot-а су да обезбеди брз и једноставан почетак рада са Spring технологијом, да омогуће лако прилагођавање апликација када захтеви корисника одступе од подразумеваних поставки и да умање потребну количину програмског кода. Такође, Spring Boot укључује подршку за низ функционалности које су уобичајене за веће пројекте, као што су уграђени сервери, безбедносне мере, метрике, провере стања система и екстернализована конфигурација.

## Spring Framework

Spring Framework [18] је радни оквир за развој апликација у Java програмском језику који омогућава програмерима да лакше креирају сложене, скалабилне и поуздане апликације. Његова модулarna архитектура омогућава корисницима да одаберу само оне компоненте које су им потребне, чиме се поједностављује процес развоја и одржавања.

Основне карактеристике Spring Framework-а укључују инверзију контроле, која омогућава управљање зависностима и поједностављује развој и тестирање апликација, и аспектно оријентисано програмирање, које омогућава издвајање и управљање заједничким функционалностима као што су безбедност, логовање и управљање трансакцијама. Овај оквир подржава Model-View-Controller (MVC) архитектуру, која раздваја податке апликације (Model) од корисничког интерфејса (View) и логику која управља подацима и корисничким интерфејсом (Controller), чиме се олакшава развој веб апликација и интеграција са различитим технологијама.

## **Maven**

Maven [19] је алат који поједностављује процесе израде и управљања Java пројектима. Користећи модел објекта пројекта (POM) и комплет модула (енг. plugins), он стандардизује процес изградње. То значи да, једном усвојен концепт коришћења алата Maven, може врло лако да се примени и на друге софтверске пројекте који користе овај алат.

Такође, Maven пружа и корисне информације о пројекту, које се могу генерисати на основу POM-а и изворног кода. Ове информације укључују логове промена, изворе са унакрсним референцама и извештаје о тестовима. Иако нуди велику флексибилност у прилагођавању различитим потребама, важно је напоменути да може бити изазовно користити га за пројекте који не могу бити реорганизовани у складу са његовим принципима.

## **Spring security**

Spring Security [20] је оквир за заштиту апликација заснованих на Spring-у и који пружа подршку за аутентификацију и ауторизацију апликација. Олакшава имплементацију механизма за заштиту од разних врста напада, као што су преузимање контроле над активном сесијом корисника, техника које омогућавају нападачу да превари корисника да кликне на нешто што није намеравао, и манипулација веб прегледача ради слања захтева на сервер без знања корисника. Такође може да се интегрише с другим компонентама, као што су Servlet API [21] и Spring Web MVC [22], што програмерима олакшава додавање безбедносних функција у веб апликације.

## **Hibernate**

Hibernate [23] је оквир за мапирање објеката у релационе базе података, што олакшава развој апликација у којима подаци остају сачувани и после завршетка рада апликације. Hibernate се фокусира на очување података који се односе на релационе базе података путем JDBC [24] (Java Database Connectivity). Такође, имплементира Java Persistence API (JPA) [25] спецификацију, што значи да се може релативно лако применити у различитим окружењима која подржавају JPA.

Једна од кључних карактеристика Hibernate-а је способност да перзистира објекте моделоване помоћу класа које користе објектно-оријентисане концепте као што су наслеђивање, полиморфизам, асоцијације, композиција и Java колекције.

## **JWT Token**

JSON Web Token [26] (JWT) је стандард (RFC 7519 [27]) који дефинише начин за сигурно преношење информација међу учесницима у облику JSON објекта. Ове информације могу бити верификоване и поуздане јер су дигитално потписане, помоћу тајног податка или јавног/приватног кључа. Корисни су у различитим сценаријима, као што су ауторизација и размена информација. У оквиру ауторизације, када се корисник

пријави, сваки следећи захтев укључује JWT, што кориснику омогућава приступ ресурсима и услугама које су дозвољене тим токеном.

Када говоримо о структури JWT-а, она се састоји од три дела одвојена тачкама: заглавља (енг. Header), садржаја (енг. Payload) и потписа (енг. Signature).

- заглавље - последњи део токена, у коме је наведен криптографски алгоритам преко кога настаје потпис
- садржај - главни део токена, који носи информације за ауторизацију (корисничко име, ниво привилегија и друге информације које су неопходне за ауторизацију, али, нису поверљиве)
- потпис - контролни део који настаје енкрипцијом комбинације претходна два дела, уз коришћење шифара

## MySQL

MySQL [28] је систем за управљање релационим базама података, намењен складиштењу и управљању подацима. Он подржава широк спектар апликација, од малих личних пројеката до обимних пословних система, а познат је по својој поузданости, перформансама, скалабилности и једноставности коришћења.

Осмишљен је за управљање структурираним подацима, а SQL (Structured Query Language) служи као језик за манипулацију подацима, укључујући преузимање, ажурирање и брисање. MySQL такође подржава ACID трансакције (атомичност, конзистентност, изолованост и издржљивост), што осигурава да су сви захтеви за обраду података поуздани, као и способност да паралелно обрађује већи број истовремених захтева.

## Firebase

Firebase [29] је платформа за развој мобилних апликација коју је развио Google [30], и она пружа услуге као што су аналитика, аутентикација, базе података, складиштење датотека, слање обавештења и друге.

Једна од кључних предности Firebase-а је то што су све услуге хостоване у облаку, чиме се олакшава скалирање апликација. Поред тога, backend компонентама управља Google, што омогућава клијентским апликацијама да директно комуницирају са овим сервисима без потребе за посредницима. Ова конфигурација поједностављује развој и побољшава перформансе и сигурност апликација.

## **4. Спецификација захтева**

У овом поглављу су објашњени функционални и нефункционални захтеви софтверског решења представљеног у овом раду.

### **4.1. Спецификација функционалних захтева**

У овом одељку су описани функционални захтеви које испуњава софтверско решење за друштвену мрежу. Функционални захтеви овог софтверског решења су представљени UML дијаграмом случајева коришћења, као што је приказано на слици 1.



Слика 1 – UML дијаграм случаја коришћења



Табела 1 приказује опис случаја коришћења **“Регистрација на систем”**.

Назив	Регистрација на систем
Учесници	Корисник
Предуслови	-
Кораци	1. Корисник бира опцију за регистрацију 2. Корисник уноси личне податке 3. Корисник потврђује унос
Резултат	Креиран је кориснички налог за приступ апликацији
Изузеци	Унети подаци нису валидни

Табела 1 - Опис случаја коришћења “Регистрација на систем”

Табела 2 приказује опис случаја коришћења **“Пријава на систем”**.

Назив	Пријава на систем
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора имати налог
Кораци	1. Корисник бира опцију за пријаву 2. Корисник уноси своје креденцијале 3. Корисник потврђује унос
Резултат	Корисник приступа апликацији
Изузеци	Унети подаци нису валидни

Табела 2 - Опис случаја коришћења “Пријава на систем”

Табела 3 приказује опис случаја коришћења **“Одјава са система”**.

Назив	Одјава са система
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1. Корисник бира опцију за одјаву
Резултат	Корисник је одјављен
Изузеци	-

Табела 3 - Опис случаја коришћења “Одјава са система”

Табела 4 приказује опис случаја коришћења **“Руковање објавама”**.

Назив	Руковање објавама
-------	-------------------

Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1а.1. Корисник бира опцију за креирање нове објаве 1а.2. Корисник уноси садржај објаве или мења постојећи 1а.3. (опционо) Корисник додаје слике уз објаву 1а.4. Корисник потврђује унос 1б.1. Корисник бира опцију за измену постојеће објаве 1б.2. Корисник мења садржај објаве 1б.3. (опционо) Корисник мења слике уз објаву 1б.4. Корисник потврђује измену
Резултат	Корисник је креирао нову објаву или извршио измену постојеће објаве
Изузеци	Унети подаци нису валидни (нпр. празан садржај објаве)

Табела 4 - Опис случаја коришћења “Руковање објавама”

Табела 5 приказује опис случаја коришћења “**Руковање коментарима**”.

Назив	Руковање коментарима
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1а.1. Корисник бира опцију за креирање новог коментара 1а.2. Корисник уноси текст коментара 1а.3. Корисник потврђује унос 1б.1. Корисник бира опцију за измену постојећег коментара 1б.2. Корисник мења постојећи текст коментара 1б.3. Корисник потврђује измену
Резултат	Корисник је креирао нови коментар или извршио измену постојећег
Изузеци	Унети подаци нису валидни (нпр. празан текст коментара)

Табела 5 - Опис случаја коришћења “Руковање коментарима”

Табела 6 приказује опис случаја коришћења “**Реаговање на коментаре и објаве**”.

Назив	Реаговање на коментаре и објаве
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1. Корисник прегледа објаву или коментар 2. Корисник бира једну од опција за реакцију: лајк, дислајк, срце
Резултат	Реакција корисника је евидентирана и број реакција на објави или коментару је ажуриран
Изузеци	-

Табела 6 - Опис случаја коришћења “Реаговање на коментаре и објаве”

Табела 7 приказује опис случаја коришћења “Сортирање коментара”.

Назив	Сортирање коментара
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1. Корисник прегледа коментаре 2. Корисник бира једну од опција за сортирање: по броју лајкова, по броју дислајкова, по броју срца, по датуму објављивања (у растућем или опадајућем поретку)
Резултат	Коментари су сортирани према изабраном критеријуму
Изузеци	-

Табела 7 - Опис случаја коришћења “Сортирање коментара”

Табела 8 приказује опис случаја коришћења “Сортирање објава”.

Назив	Сортирање објава
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1. Корисник прегледа објаве 2. Корисник бира једну од опција за сортирање: по датуму објављивања (у растућем или опадајућем поретку)
Резултат	Објаве су сортиране према изабраном критеријуму
Изузеци	-

Табела 8 - Опис случаја коришћења “Сортирање објава”

Табела 9 приказује опис случаја коришћења “Руковање групама”.

Назив	Руковање групама
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1а.1. Корисник бира опцију за креирање нове групе 1а.2. Корисник уноси назив групе 1а.3. Корисник потврђује унос 1б.1. Корисник бира опцију за измену постојеће групе 1б.2. Корисник мења постојећи назив групе 1б.3. Корисник потврђује измену
Резултат	Корисник је креирао нову групу или извршио измену постојеће
Изузеци	Унети подаци нису валидни (нпр. празан назив групе)

Табела 9 - Опис случаја коришћења “Руковање групама”

Табела 10 приказује опис случаја коришћења “**Пријављивање неприкладног коментара, објаве или корисника**”.

Назив	Пријављивање неприкладног коментара, објаве или корисника
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	- Корисник мора бити пријављен - Корисник прегледа објаву, коментар или профил који жели да пријави као неприкладан
Кораци	1. Корисник бира опцију за пријаву неприкладног садржаја 3. Корисник уноси разлог пријаве 4. Корисник потврђује пријаву
Резултат	Корисник је успешно пријавио објаву, коментар или профил
Изузеци	Унети подаци нису валидни (нпр. празан разлог пријаве)

Табела 10 - Опис случаја коришћења “Пријављивање неприкладног коментара, објаве или корисника”

Табела 11 приказује опис случаја коришћења “**Промена лозинке**”.

Назив	Промена лозинке
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1. Корисник приступа опцији за промену лозинке у подешавањима налога 2. Корисник уноси важећу лозинку 3. Корисник уноси нову лозинку и поново уноси нову лозинку за потврду 4. Корисник потврђује унос
Резултат	Корисник је успешно променио лозинку
Изузеци	Унети подаци нису валидни (нпр. тренутна лозинка је нетачна или се унос и поновни унос нове лозинке не подударају).

Табела 11 - Опис случаја коришћења “Промена лозинке”

Табела 12 приказује опис случаја коришћења “**Промена додатних података на профилу**”.

Назив	Промена додатних података на профилу
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен Корисник прегледа свој кориснички профил у подешавањима налога
Кораци	1. Корисник приступа опцији за измену корисничког профила 2. Корисник врши измену постојећих података, као што су име које се приказује на профилу, опис, слика 3. Корисник потврђује измене
Резултат	Корисник је успешно променио податке на профилу
Изузеци	Унети подаци нису валидни

Табела 12 - Опис случаја коришћења “Промена додатних података на профилу”

Табела 13 приказује опис случаја коришћења “Претрага корисника на систему”.

Назив	Претрага корисника на систему
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	Корисник мора бити пријављен
Кораци	1. Корисник приступа опцији за претрагу корисника 2. Корисник уноси име или презиме корисника у поље за претрагу
Резултат	Кориснику се приказује списак резултата претраге који одговарају унетим критеријумима
Изузеци	-

Табела 13 - Опис случаја коришћења “Претрага корисника на систему”

Табела 14 приказује опис случаја коришћења “Руковање захтевима за пријатељство”.

Назив	Руковање захтевима за пријатељство
Учесници	Корисник, Администратор групе, Администратор система
Предуслови	- Корисник мора бити пријављен - Корисник прегледа листу захтева за пријатељство
Кораци	1а.1. Корисник бира опцију за одобрење захтева за пријатељство 1а.2. У случају одобрења, захтев постаје пријатељство 1б.1. Корисник бира опцију за одбијање захтева за пријатељство 1б.2. У случају одбијања, захтев се уклања
Резултат	Корисник који је послао захтев постаје пријатељ или се захтев одбацује
Изузеци	-

Табела 14 - Опис случаја коришћења “Руковање захтевима за пријатељство”

Табела 15 приказује опис случаја коришћења “**Блокирање и одблокирање корисника групе**”.

Назив	Блокирање и одблокирање корисника
Учесници	Администратор групе
Предуслови	Администратор групе мора бити пријављен
Кораци	1. Администратор групе приступа листи корисника који припадају групи 2. Администратор групе бира опцију за блокирање или одблокирање корисника
Резултат	Кориснички налог је блокиран (чиме му се спречава приступ групи) или одблокиран (чиме му се враћа могућност приступа групи)
Изузеци	-

Табела 15 - Опис случаја коришћења “Блокирање и одблокирање корисника”

Табела 16 приказује опис случаја коришћења “**Руковање захтевима за придружење групи**”.

Назив	Руковање захтевима за придружење групи
Учесници	Администратор групе
Предуслови	Администратор групе мора бити пријављен
Кораци	1. Администратор групе приступа листи захтева за придружење. 2. Администратор групе бира опцију за одобравање или одбијање захтева
Резултат	Кориснику је одобрен односно одбијен захтев за придружење групи
Изузеци	-

Табела 16 - Опис случаја коришћења “Руковање захтевима за придружење групи”

Табела 17 приказује опис случаја коришћења “**Руковање пријавама садржаја**”.

Назив	Руковање пријавама садржаја
Учесници	Администратор групе, Администратор система
Предуслови	- Администратор групе или система мора бити пријављен - Администратор групе или система прегледа листу са пријављеним садржајем

Кораци	1а.1. Администратор групе или система бира опцију за одобрење пријаве садржаја 1а.2. У случају одобрења, садржај се суспендује или корисник блокира 1б.1. Администратор групе или система бира опцију за одбијање пријаве садржаја 1б.2. У случају одбијања, захтев се уклања
Резултат	Пријављени садржај се суспендује (више се не приказује на систему), а корисник се блокира и не може да се пријави на систем или се пријава одбацује
Изузеци	-

Табела 17 - Опис случаја коришћења “Руковање пријавама садржаја”

Табела 18 приказује опис случаја коришћења “Уклањање администратора група”.

Назив	Уклањање администратора група
Учесници	Администратор система
Предуслови	Администратор система мора бити пријављен
Кораци	1. Администратор система бира администратора групе за уклањање 2. Администратор система потврђује уклањање
Резултат	Уклоњени администратор групе постаје редован корисник, а његова администраторска права су укинута
Изузеци	-

Табела 18 - Опис случаја коришћења “Уклањање администратора група”

Табела 19 приказује опис случаја коришћења “Суспендовање групе”.

Назив	Суспендовање групе
Учесници	Администратор система
Предуслови	Администратор система мора бити пријављен
Кораци	1. Администратор система проналази групу за суспендовање 2. Администратор система бира опцију за суспендовање 3. Администратор система уноси разлог за суспендовање 4. Администратор система потврђује суспендовање
Резултат	Група постаје суспендована и сви администратори групе су уклоњени
Изузеци	Унети подаци нису валидни (нпр. празан разлог суспендовања)

Табела 19 - Опис случаја коришћења “Суспендовање групе”

## 4.2. Спецификација нефункционалних захтева

У овом одељку је дата спецификација нефункционалних захтева, којима су дефинисана својства софтверског решења, али не представљају његове функционалности. Ови захтеви се тичу аутентификације корисника и бележења значајних догађаја у вези са радом система.

**Аутентификација корисника:** Апликација подржава аутентификацију корисника коришћењем јединственог корисничког имена и лозинке. Поред тога, ауторизација се врши употребом механизма токена, што омогућава додатни ниво сигурности приликом приступа систему.

**Бележење важних догађаја:** Апликација имплементира систем за бележење порука о важним догађајима који се дешавају током њеног извршавања. Ове информације помажу у праћењу рада система, дијагностици проблема и унапређењу корисничког искуства.

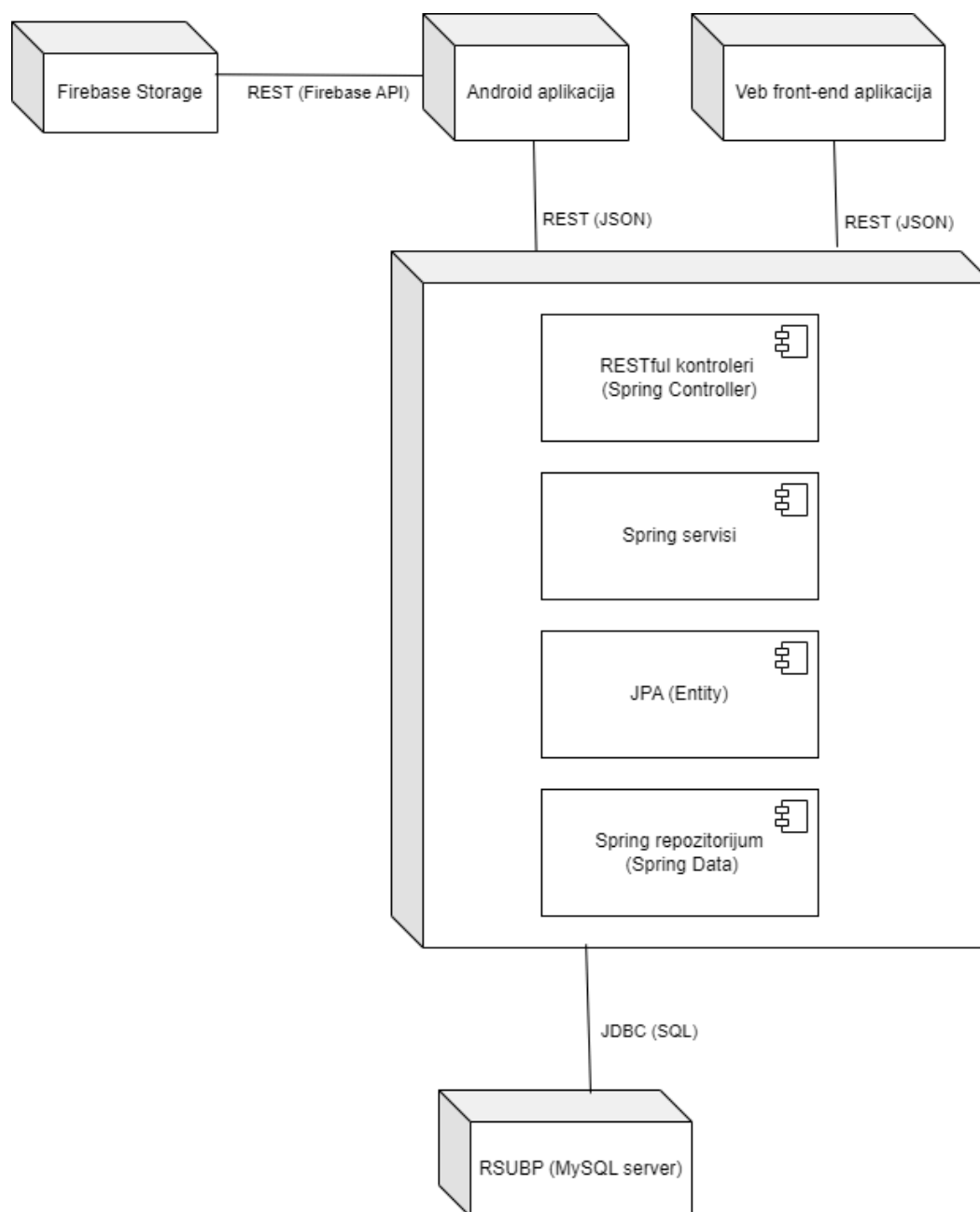


## 5. Спецификација дизајна

Ово поглавље објашњава дизајн софтверског решења за друштвену мрежу развијену за Android платформу и засновану на RESTful архитектури.

### 5.1. Архитектура система

Нодове система чине Android апликација намењена мобилним уређајима, front-end апликација за приступ систему из веб прегледача, Spring контејнер, систем за управљање базом података (SUBP) и Firebase. Дијаграм распореда приказан је на слици 2.



Слика 2 - Дијаграм распореда

Android апликација имплементирана у овом пројекту користи REST веб сервис за комуникацију са серверским делом система и на тај начин омогућава корисницима да интерагују са апликацијом путем мобилних уређаја. Android апликација комуницира са серверским делом тако што шаље захтеве и прима одговоре у JSON формату.

Front-end апликација за веб окружење није посебно анализирана у овом раду, с обзиром на то да није у фокусу теме коју рад обрађује.

REST веб сервис се фокусира на ресурсе и омогућава приступ тим ресурсима. Ресурси могу бити различити објекти, датотеке на диску, подаци из апликације, базе података итд. Приликом дизајнирања система, прво је потребно идентификовати ресурсе и њихове међусобне везе, што је слично моделовању базе података. Након идентификације ресурса, следећи корак је њихова репрезентација у систему, где се најчешће користи JSON формат.

JSON је једноставан формат за размену података. Подаци се записују као парови кључ-вредност, а помоћу витичастих заграда су обједињени у објекте. На пример, JSON објекат може изгледати овако: `{"firstName": "Ana", "lastName": "Domonji"}`. Кључеви су текстуални подаци записани унутар дуплих наводника, док вредности могу бити број, string, boolean, низ, објекат или null.

Слање захтева неком веб сервису захтева креирање HTTP захтева, који се састоји од неколико елемената: акције коју желимо да извршимо, као што су GET, PUT, POST, DELETE, OPTIONS и друго, путање до ресурса, и тела захтева (за POST и PUT захтеве). HTTP одговор генерисан на серверу укључује садржај и статусни код, којим се саопштава да ли је операција успешно извршена, да ли је потребна редирекција, да ли операција није извршена услед грешке у клијентском захтеву или је наступила серверска грешка.

Приликом слања HTTP захтева неопходно је познавати адресу на који се тај захтев шаље. На пример, да би се приступило подацима свих постојећих објава, то би била адреса: <http://localhost:8080/api/posts/all>.

Firebase платформа се користи за комуникацију Android апликације са складиштем података, што је у овом случају Firebase Storage. Овај слој омогућава чување слика у облаку, што значи да апликација може директно да учита и складишти слике, као и да приступа и преузима те слике по потреби.

Серверски део апликације обезбеђује REST сервисе помоћу којих се клијентској апликацији омогућава извршавање неопходних функционалности. Овај део апликације је развијен уз помоћ Spring радног оквира односно пакета Spring Boot.

Spring Boot апликација поседује REST контролере који омогућавају комуникацију са клијентом, односно мобилном апликацијом. Ови контролери обрађују долазне HTTP

захтеве и позивају методе сервиса за обраду логике апликације, те враћају клијенту у одговору резултате ове обраде.

Spring сервиси садрже пословну логику апликације. Сервиси обрађују податке, врше валидације, трансформације и при томе се ослањају на слој за управљање подацима како би се све потребне промене сачувале у бази података.

Java Persistence API (JPA) се користи за мапирање објеката у апликацији на табеле у бази података. Ентитети представљају структуру података која се чува у бази. Да би се подржало управљање подацима у апликацији неопходно је најпре дефинисати основне ентитете и њихове односе. Ове класе представљају модел података апликације.

Spring репозиторијум (Spring Data) је слој који омогућава интеракцију са базом података користећи репозиторијумски образац. Spring Data аутоматски генерише основне CRUD операције и омогућава сложеније упите.

JDBC (Java Database Connectivity) се користи за директну комуникацију између Java апликације (Spring) и базе података, што је у овом случају MySQL сервер. Овај слој омогућава извршавање SQL упита и враћање резултата апликацији, чиме се омогућава приступ и управљање подацима у бази.

## 5.2. Модел података

Модел података система је приказан помоћу дијаграма класа датог на слици 3. У наставку је објашњено шта ове класе репрезентују и какви су њихови међусобни односи.

Класа **User** представља кориснике апликације и игра кључну улогу у процесу аутентификације и ауторизације. Овај ентитет садржи основне информације о кориснику, као што су корисничко име, лозинка, e-mail адреса и други релевантни подаци. С обзиром на то да у систему постоје различите улоге корисника, он тако може бити нерегистрован, администратор система или администратор групе. Нерегистровани корисници имају могућност регистрације у апликацији, чиме постају регистровани корисници. Администратор система има све привилегије, укључујући могућност управљања групама и корисницима, као и уклањање група које више нису релевантне. Администратор групе је одговоран за одржавање одређене групе, управљање њеним члановима и садржајем унутар те групе.

Класа **Post** описује објаве које корисници креирају унутар апликације. Ове објаве су претежно текстуалне, али могу садржати и слике. Свака објава може бити повезана са одређеним корисником који је креира, као и са коментарима и реакцијама које она може добити. Објава може садржати једну или више слика.

Класа **Comment** представља коментаре које корисници могу оставити на објавама. Коментари такође могу бити повезани једни са другима, омогућавајући корисницима да одговарају на друге коментаре.

Класа **Reaction** служи као механизам за изражавање ставова корисника према одређеним објавама или коментарима. Корисници могу реаговати на објаве путем лајкова, дислајкова или “срца”.

Случајеви у којима неки садржај или корисник крши правила репрезентовани су помоћу класе **Report**. Овај ентитет се односи на одређену објаву, коментар или корисника који је прекршио правила. Администратор групе или система може прегледати пријаве и донети одговарајуће одлуке о суспендовању или уклањању неприкладног садржаја.

Класа **Group** представља групу унутар апликације која може садржати објаве, коментаре и кориснике. Свака група има свог администратора који управља њеним садржајем.

Класа **Image** представља слике које могу бити додате уз објаве или као профилне слике корисника. Свака слика је повезана са одређеним објавама или корисницима.

Класа **GroupRequest** представља захтев корисника за придруживање одређеној групи. Када корисник жели да постане члан групе, он шаље захтев који администратор групе може одобрити или одбити.

Класа **Banned** представља стање корисника који су блокирани у оквиру система или одређене групе. Блокирани корисници немају приступ групи или систему докле год су у блокираном стању. Администратор система може прегледати блокиране кориснике и одлучити о њиховом одблокирању.

Класа **FriendRequest** представља захтев за пријатељство који један корисник може послати другом кориснику. Пријатељство се успоставља када друга страна прихвати захтев. Сваки захтев може бити и одбијен.



## 6. Имплементација система

Ово поглавље приказује начин на који су имплементиране неке од функција система за друштвену мрежу. У наставку су засебно приказане имплементације клијентског дела апликације и серверског дела апликације.

### 6.1. Клијентски део апликације

У овом одељку је објашњена имплементација кључних компоненти клијентске Android апликације, укључујући имплементацију додавања објава, имплементацију креирања објава, имплементацију пријаве на систем и имплементацију адаптера.

За комуникацију са серверским делом система, неопходно је у клијентску Android апликацију уградити подршку за рад са REST веб сервисима. За то је у овом пројекту изабрана библиотека Retrofit. Да би се користила Retrofit библиотека у Android апликацији, потребно је у `build.gradle` датотеку пројекта додати зависности приказане на листингу 1.

```
implementation("com.squareup.retrofit2:retrofit:2.9.0")
implementation("com.squareup.retrofit2:converter-gson:2.9.0")
implementation("com.squareup.okhttp3:okhttp:4.9.3")
implementation("com.squareup.okhttp3:logging-interceptor:4.9.3")
```

Листинг 1 - Додавање библиотека

Ове зависности омогућавају коришћење Retrofit-а за HTTP захтеве, конверзију JSON података користећи Gson пакет, и логовање HTTP захтева.

Након укључивања библиотека, потребно је дефинисати Retrofit инстанцу која ће обављати HTTP захтеве ка REST сервису. Ово се може постићи креирањем класе `ClientUtils` која ће садржати конфигурацију Retrofit-а (листинг 2).

```
object ClientUtils {

    private const val SERVICE_API_PATH =
"http://${BuildConfig.IP_ADDR}:8080/api/"

    private fun createOkHttpClient(token: String? = null): OkHttpClient {
        val interceptor = HttpLoggingInterceptor().apply {
            level = HttpLoggingInterceptor.Level.BODY
        }

        val builder = OkHttpClient.Builder()
            .connectTimeout(120, TimeUnit.SECONDS)
            .readTimeout(120, TimeUnit.SECONDS)
            .writeTimeout(120, TimeUnit.SECONDS)
            .addInterceptor(interceptor)

        if (token != null) {
            builder.addInterceptor { chain ->
```

```

        val original = chain.request()
        val requestBuilder = original.newBuilder()
            .header("Authorization", "Bearer $token")
        val request = requestBuilder.build()
        chain.proceed(request)
    }
}

return builder.build()
}

private val gson: Gson by lazy {
    GsonBuilder()
        .setLenient()
        .registerTypeAdapter(LocalDate::class.java,
LocalDateDeserializer())
        .registerTypeAdapter(LocalDate::class.java,
LocalDateSerializer())
        .registerTypeAdapter(LocalDateTime::class.java,
LocalDateTimeDeserializer())
        .registerTypeAdapter(LocalDateTime::class.java,
LocalDateTimeSerializer())
        .create()
}

private val retrofit: Retrofit by lazy {
    Retrofit.Builder()
        .baseUrl(SERVICE_API_PATH)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .client(createOkHttpClient())
        .build()
}

private fun createRetrofit(token: String? = null): Retrofit {
    return Retrofit.Builder()
        .baseUrl(SERVICE_API_PATH)
        .addConverterFactory(GsonConverterFactory.create(gson))
        .client(createOkHttpClient(token))
        .build()
}

```

Листинг 2 - Класа ClientUtils

Најпре је помоћу варијабле `SERVICE_API_PATH` дефинисана основна путања за API, што омогућава одређивање адресе сервиса са којим апликација комуницира. Метода `createOkHttpClient` конфигурише `OkHttpClient` тако да подржава логовање захтева, поставља временске лимите за конекцију, читање и писање, и опционално додаје токен за ауторизацију уколико је он доступан. Инстанца `gson` је подешена за конверзију локалних датума и времена користећи Gson, ради лакше обраде временских података. На крају, методе `retrofit` и `createRetrofit` креирају и враћају Retrofit инстанце са конфигурацијом која укључује базни URL, конвертер за JSON и HTTP клијент. Ова конфигурација омогућава апликацији да ефикасно комуницира са сервером путем RESTful API-ја, шаљући захтеве и примајући одговоре у JSON формату.

Да би омогућила комуникација клијента, односно мобилне апликације, са сервером, односно Spring Boot апликацијом, потребно је да им рачунарска мрежа то омогућује и да клијентска информација поседује информацију о IP адреси сервера. На пример, IP адреса сервера може бити нешто попут 192.168.0.20. Да би се ова IP адреса сачувала у клијентској апликацији, искоришћена је `local.properties` датотека (листинг 3).

```
## This file is automatically generated by Android Studio.
# Do not modify this file -- YOUR CHANGES WILL BE ERASED!
#
# This file should *NOT* be checked into Version Control Systems,
# as it contains information specific to your local configuration.
#
# Location of the SDK. This is only used by Gradle.
# For customization when using a Version Control System, please read the
# header note.
sdk.dir=C:\\Users\\domon\\AppData\\Local\\Android\\Sdk
ip_addr=192.168.0.20
```

Листинг 3 - Датотека `local.properties`

Да би се у коду могла користити ову `ip_addr` варијабла, потребно је извршити конфигурацију у `build.gradle` датотеци као што је приказано на листингу 4.

```
plugins {
    id("com.android.application")
    id("org.jetbrains.kotlin.android")
}

fun getIpAddress(): String? {
    val properties = Properties()
    val localPropertiesFile = rootProject.file("local.properties")

    FileInputStream(localPropertiesFile).use {
        properties.load(it)
    }

    return properties.getProperty("ip_addr")
}

android {
    namespace = "com.example.socialnetwork"
    compileSdk = 34

    defaultConfig {
        applicationId = "com.example.socialnetwork"
        minSdk = 34
        targetSdk = 34
        versionCode = 1
        versionName = "1.0"
        buildConfigField("String", "IP_ADDR", "\"${getIpAddress()}\"")
        testInstrumentationRunner = "androidx.test.runner.AndroidJUnitRunner"
    }
}
```



#### Листинг 4 - build.gradle датотека

Након тога, могуће је приступити параметру IP адресе преко **BuildConfig** класе на следећи начин: **BuildConfig.IP\_ADDR**. У класи **ClientUtils** је тако дефинисана путања до сервера (листинг 5).

```
private const val SERVICE_API_PATH =  
"http://${BuildConfig.IP_ADDR}:8080/api/"
```

#### Листинг 5 - Дефинисање путање до сервера

Након дефинисања путање до сервера, креирања Retrofit инстанце и дефинисања REST сервиса, потребно је извршити и инстанцирање REST сервиса у **ClientUtils** класи. Функција **getPostService** узима опциони токен као параметар и враћа инстанцу **PostService** користећи претходно креирану Retrofit инстанцу (листинг 6).

```
fun getPostService(token: String? = null): PostService {  
    return createRetrofit(token).create(PostService::class.java)  
}
```

#### Листинг 6 - Инстанцирање PostService објекта

За сваки сервис који је потребно позвати, неопходно је дефинисати методе за одговарајуће HTTP операције (GET, POST, PUT, DELETE), као и додатне параметре упита, путање, и садржај (у случају POST или PUT захтева). Пример **PostService** интерфејса садржи методе као што су **getAll**, **getId**, **create**, **createGroupPost**, **update**, **delete**, **getByGroup**, и друге. Ове методе дефинишу која се HTTP операција користи и шта је њен резултат. Такође, за сервисе који захтевају додатне елементе унутар HTTP заглавља, користи се анотација **@Headers**.

Имплементација **PostService** интерфејса, намењеног извршавању операција над објавама, приказана је на листингу 7.

```
interface PostService {  
  
    @Headers(  
        "User-Agent: Mobile-Android",  
        "Content-Type: application/json"  
    )  
    @GET("posts/all")  
    fun getAll(): Call<ArrayList<Post>>  
  
    @Headers(  
        "User-Agent: Mobile-Android",  
        "Content-Type: application/json"  
    )  
    @GET("posts/find/{id}")  
    fun getId(@Path("id") id: Long): Call<Post>
```

```

@Multipart
@POST("posts/create")
fun create(
    @Part("content") content: RequestBody,
    @Part images: List<MultipartBody.Part>
): Call<Post>

@Multipart
@POST("posts/create/{id}")
fun createGroupPost(
    @Path("id") groupId: Long,
    @Part("content") content: RequestBody,
    @Part images: List<MultipartBody.Part>? = null
): Call<Post>

@Multipart
@PUT("posts/update/{id}")
fun update(
    @Path("id") id: Long,
    @Part("content") content: RequestBody,
    @Part images: List<MultipartBody.Part>? = null
): Call<Post>

@Headers(
    "User-Agent: Mobile-Android",
    "Content-Type: application/json"
)
@PUT("posts/delete/{id}")
fun delete(@Path("id") id: Long): Call<Post>

@Headers(
    "User-Agent: Mobile-Android",
    "Content-Type: application/json"
)
@GET("posts/all/{groupId}")
fun getByGroup(@Path("groupId") groupId: Long): Call<List<Post>>
}

```

Листинг 7 - Интерфејс PostService

Инстанцирање потребних сервиса је имплементирано у методи `initializeServices` (листинг 8).

```

private fun initializeServices() {
    val token = PreferencesManager.getToken(this) ?: return
    postService = ClientUtils.getPostService(token)
    reactionService = ClientUtils.getReactionService(token)
    userService = ClientUtils.getUserService(token)
    reportService = ClientUtils.getReportService(token)
    friendRequestService = ClientUtils.getFriendRequestService(token)
    groupRequestService = ClientUtils.getGroupRequestService(token)
    bannedService = ClientUtils.getBannedService(token)
}

```

Листинг 8 - Иницијализација сервиса

Функција `fetchPostsFromServer()` која је приказана на листингу 9, служи за асинхронно добављање објава са сервера користећи Retrofit библиотеку. У зависности од редоследа сортирања (узлазно или силазно), функција прави одговарајући позив над сервисом (`getAllAscending` или `getAllDescending`). Када сервер одговори, метода `onResponse` проверава да ли је одговор успешан, а затим позива функцију `fetchFriendsAndGroups(posts)` ради учитавања пријатеља и групе за тренутног корисника. У случају неуспеха са статусним кодом 401, позива се функција `handleTokenExpired()` за обраду истека токена, а за друге грешке кориснику се приказује порука о неуспешном преузимању објава помоћу `showToast()`. Ако дође до грешке приликом позива, кориснику се такође приказује порука са детаљима о грешци.

```
private fun fetchPostsFromServer() {
    val call = if (sortingOrder == "ascending") postService.getAllAscending()
    else postService.getAllDescending()

    call.enqueue(object : Callback<List<Post>> {
        override fun onResponse(call: Call<List<Post>>, response:
Response<List<Post>>) {
            if (response.isSuccessful) {
                val posts = response.body() ?: arrayListOf()
                fetchFriendsAndGroups(posts)
            } else if (response.code() == 401) {
                handleTokenExpired()
            } else {
                showToast("Failed to load posts")
            }
        }

        override fun onFailure(call: Call<List<Post>>, t: Throwable) {
            showToast("Error: ${t.message}")
        }
    })
}
```

#### Листинг 9 - Добављање објава

Функција `fetchFriendsAndGroups(posts: List<Post>)` на листингу 10, служи за преузимање одобрених пријатеља и група тренутног корисника. Прво се покрећу два API позива: један за добијање одобрених пријатеља корисника (`getApprovedFriendsForUser`) и други за добијање одобрених група (`getApprovedGroupsForUser`). Након што се позив за пријатеље успешно заврши, пристигли подаци се чувају, а затим се покреће други позив за групе. Ако су оба позива успешна, користи се `CoroutineScope` за покретање функције `filterPosts(posts, friends, groups)` на главној нити, како би се филтрирале објаве према пријатељима и групама. У случају грешке приликом било ког од позива, кориснику се приказује порука о грешци помоћу `showToast()`.

```
private fun fetchFriendsAndGroups(posts: List<Post>) {
    val friendsCall =
friendRequestService.getApprovedFriendsForUser(currentUser?.id ?: return)
```

```

        val groupsCall =
groupRequestService.getApprovedGroupsForUser(currentUser?.id ?: return)

        friendsCall.enqueue(object : Callback<Set<User>> {
            override fun onResponse(call: Call<Set<User>>, response:
Response<Set<User>>) {
                if (response.isSuccessful) {
                    val friends = response.body() ?: emptyList()
                    groupsCall.enqueue(object : Callback<Set<Group>> {
                        override fun onResponse(call: Call<Set<Group>>, response:
Response<Set<Group>>) {
                            if (response.isSuccessful) {
                                val groups = response.body() ?: emptyList()
                                CoroutineScope(Dispatchers.Main).launch {
                                    filterPosts(posts, friends, groups)
                                }
                            }
                        }
                    })

                    override fun onFailure(call: Call<Set<Group>>, t:
Throwable) {
                        showToast("Error: ${t.message}")
                    }
                })
            }

            override fun onFailure(call: Call<Set<User>>, t: Throwable) {
                showToast("Error: ${t.message}")
            }
        })
    }
}

```

#### Листинг 10 - Добављање одобрених пријатеља и група тренутног корисника

Функција `filterPosts` приказана на листингу 11, филтрира објаве на основу корисничких интеракција и припадности групама. У оквиру ове функције, за сваку објаву се проверава да ли ју је креирао текући корисник (`isUserPost`), да ли је објава неког од пријатеља (`isFriendPost`), или се ради о објави из групе којој корисник припада (`isGroupPost`). Уколико објава припада групи, такође се проверава да ли је корисник блокиран у тој групи користећи `checkIfUserIsBlocked`. Резултантна листа филтрираних објава, која садржи само релевантне објаве, се затим прослеђује функцији `updateListView`, која је одговорна за ажурирање приказа објава у корисничком интерфејсу. Ова логика омогућава корисницима да виде само објаве које су за њих релевантне.

```

private suspend fun filterPosts(posts: List<Post>, friends: Collection<User>,
groups: Collection<Group>) {
    val filteredPosts = posts.filter { post ->
        val isUserPost = post.user?.username == currentUser?.username
        val isFriendPost = friends.any { it.username == post.user?.username }
        val isGroupPost = post.group?.let { group ->
            groups.any { it.id == group.id } &&
            !checkIfUserIsBlocked(group.id!!)
        }
    }
}

```

```

        } ?: false

        isUserPost || isFriendPost || isGroupPost
    }
    updateListView(filteredPosts)
}

```

Листинг 11 - Филтрирање објава

Функција `checkIfUserIsBlocked` служи за проверу да ли је тренутни корисник блокиран унутар одређене групе. Ова функција се извршава у IO контексту користећи `withContext(Dispatchers.IO)`, што омогућава да се мрежни позив изврши ван ток главне нити, чиме се избегава блокирање корисничког интерфејса. Функција шаље захтев сервису `bannedService` за добијање листе блокираних корисника у групи са прослеђеним идентификатором `groupId`.

Након слања захтева, резултати се проверавају, те ако је одговор успешан (`response.isSuccessful`), преузима се листа блокираних корисника, а затим се проверава да ли тренутни корисник (`currentUser?.id`) постоји међу блокираним корисницима. Ако се корисник налази на тој листи, функција враћа резултат `true`, указујући да је корисник блокиран, док у супротном враћа вредност `false`. Ова функција је приказана на листингу 12.

```

private suspend fun checkIfUserIsBlocked(groupId: Long): Boolean {
    return withContext(Dispatchers.IO) {
        val call = bannedService.getAllBlockedGroupUsers(groupId)
        val response = call.execute()
        if (response.isSuccessful) {
            val blockedUsers = response.body()
            blockedUsers?.any { it.bannedUser?.id == currentUser?.id } ?:
false
        } else {
            true
        }
    }
}

```

Листинг 12 - Провера да ли је корисник блокиран

Функција `updateListView` се користи за ажурирање приказа објава унутар `ListView` компоненте у апликацији (листинг 13). Најпре функција лоцира `ListView` користећи идентификатор ресурса `R.id.postsListView`. Затим креира инстанцу `PostAdapter`, прослеђујући јој контекст и листу објава које треба приказати.

У овој функцији се такође постављају обрађивачи догађаја за интеракцију са сваком објавом, омогућавајући кориснику да коментарише, пријављује, брише, ажурира, лајкује, дислајкује и изражава емоције путем срца. Ови обрађивачи су повезани са инстанцом активности, што омогућава да се одговарајуће методе позову када корисник изврши неку од ових акција. На крају, погледу типа `ListView` се додељује адаптер, чиме се приказује ажурирана листа објава.

```
private fun updateListView(posts: List<Post>) {
    val listView: ListView = findViewById(R.id.postsListView)
    adapter = PostAdapter(this, posts)
    adapter.commentButtonClickListener = this
    adapter.reportButtonClickListener = this
    adapter.deleteButtonClickListener = this
    adapter.editButtonClickListener = this
    adapter.likeButtonClickListener = this
    adapter.dislikeButtonClickListener = this
    adapter.heartButtonClickListener = this
    listView.adapter = adapter
}
```

Листинг 13 - Ажурирање приказа објава

Функција `handleCreatePost()` се користи за креирање нове објаве (листинг 14). Прво, функција узима текстуални садржај објаве из поља за унос и уклања евентуалне празнине са почетка и краја. Затим, ако је садржај празан, кориснику се приказује порука о грешци како би се обавестио да садржај не може бити празан. Након тога, садржај објаве се конвертује у `RequestBody` објекат како би се могао послати као део HTTP захтева. Изабране слике, које су представљене URI идентификаторима, припремају се за слање и прогресивна трака се поставља на видљиво стање како би се приказало да је процес додавања објаве започет.

Функција затим креира асинхрони HTTP захтев ка серверу помоћу `postService.create` методе, која шаље садржај објаве и слике. Када сервер одговори, функција проверава да ли је одговор успешан. Ако је одговор успешан, функција проверава да ли постоје слике за објављивање, и ако да, позива одговарајућу функцију за објављивање слика. Ако слика нема или је објављивање завршено, прогресивна трака се сакрива, кориснику се приказује порука о успеху, поново се учитавају објаве са сервера, и дијалог за унос нове објаве се затвара.

У случају неуспеха било којег дела процеса, прогресивна трака се сакрива, а кориснику се приказује одговарајућа порука о грешци.

```
private fun handleCreatePost() {
    val postContent = popupPostEditText.text.toString().trim()

    if (postContent.isEmpty()) {
        showValidationError("Post content cannot be empty")
        return
    }

    val contentPart =
postContent.toRequestBody("text/plain".toMediaTypeOrNull())

    val imageFiles = getFilesFromUris(selectedImages)
    val images = prepareImages(imageFiles)

    progressBar.visibility = View.VISIBLE
```

```

val call = postService.create(contentPart, images)

call.enqueue(object : Callback<Post> {
    override fun onResponse(call: Call<Post>, response: Response<Post>) {
        if (response.isSuccessful) {
            val post = response.body()
            post?.let {
                if (imageFiles.isNotEmpty() && post.images.isNotEmpty())
                    uploadImagesToFirebase(it.id, post.images, imageFiles,
progressBar)
            } else {
                progressBar.visibility = View.GONE
                showToast("Post created successfully")
                token?.let { it1 -> fetchPostsFromServer() }
                dismissPostPopup()
            }
        } else {
            progressBar.visibility = View.GONE
            showValidationError("Failed to create post")
        }
    }

    override fun onFailure(call: Call<Post>, t: Throwable) {
        progressBar.visibility = View.GONE
        showValidationError("Error: ${t.message}")
    }
})
}

```

#### Листинг 14 - Креирање објава

Функција `performLogin` у овој апликацији рукује процесом пријаве корисника (листинг 15). Прво узима корисничко име и лозинку унете у одговарајућа поља за унос и уклања све празнине са почетка и краја. Затим, креира објекат `JwtAuthenticationRequest` који садржи ове податке и користи `userService` објекат да креира захтев за аутентификацију.

Позив се извршава асинхроно користећи `enqueue` методу, која дефинише две `callback` методе: `onResponse` и `onFailure`. Ако је одговор успешан (`onResponse`), функција добија токен за приступ из одговора и чува га користећи `PreferencesManager.saveToken`. Затим приказује поруку о успешној пријави и покреће активност `PostsActivity` са токеном, затварајући при томе тренутну активност. Ако пријава није успешна, позива се `handleError` метода да обради грешку. У случају неуспешног позива серверу (`onFailure`), функција приказује поруку о грешци у погледу `errorMessageTextView`.

```

private fun performLogin() {
    val username = usernameEditText.text.toString().trim()
    val password = passwordEditText.text.toString().trim()

```

```

val userService = ClientUtils.userService
val loginRequest = JwtAuthenticationRequest(username, password)
val call = userService.createAuthenticationToken(loginRequest)

call.enqueue(object : Callback<UserTokenState> {
    override fun onResponse(call: Call<UserTokenState>, response:
Response<UserTokenState>) {
        if (response.isSuccessful) {
            val tokenState = response.body()
            val token = tokenState?.accessToken

            if (token != null) {
                PreferencesManager.saveToken(this@LoginActivity, token)
            }

            Toast.makeText(this@LoginActivity, "Login successful",
Toast.LENGTH_SHORT).show()

            val intent = Intent(this@LoginActivity,
PostsActivity::class.java)
            intent.putExtra("TOKEN", token)
            startActivity(intent)
            finish()

        } else {
            handleError(response)
        }
    }

    override fun onFailure(call: Call<UserTokenState>, t: Throwable) {
        errorMessageTextView.text = "Login error: ${t.message}"
        errorMessageTextView.visibility = View.VISIBLE
    }
})
}

```

#### Листинг 15 - Пријава на систем

Објект `PreferencesManager` служи за управљање аутентификационим токеном у апликацији. Токен се користи за аутентификацију корисника након пријаве. Овај објект садржи три функције (`saveToken`, `getToken` и `clearToken`), а њихова имплементација је приказана на листингу 16.

Функција `saveToken` узима контекст и токен као параметре, отвара или креира фајл са подешавањима под називом `user_prefs` у приватном моду (`Context.MODE_PRIVATE`), што значи да му само ова апликација може приступити. Затим користи `edit()` метод за уређивање подешавања и спрема токен под кључем `auth_token`, користећи `apply()` за асинхронно чување промена.

Функција `getToken` узима контекст као параметар, отвара фајл са подешавањима `user_prefs` и враћа вредност токена са кључем `auth_token`, уколико он постоји. Ако токен није пронађен, функција враћа резултат `null`.



Функција `clearToken` такође узима контекст као параметар, отвара фајл са подешавањима `user_prefs`, користи `edit()` метод за уређивање подешавања, уклања вредност са кључем `auth_token` и користи `apply()` за асинхроно чување промена.

```
object PreferencesManager {

    private const val PREFS_NAME = "user_prefs"
    private const val KEY_TOKEN = "auth_token"

    fun saveToken(context: Context, token: String) {
        val prefs = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE)
        with(prefs.edit()) {
            putString(KEY_TOKEN, token)
            apply()
        }
    }

    fun getToken(context: Context): String? {
        val prefs = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE)
        return prefs.getString(KEY_TOKEN, null)
    }

    fun clearToken(context: Context) {
        val prefs = context.getSharedPreferences(PREFS_NAME,
Context.MODE_PRIVATE)
        with(prefs.edit()) {
            remove(KEY_TOKEN)
            apply()
        }
    }
}
```

#### Листинг 16 - Управљање аутентификационим токеном

Адаптери се у Android апликацијама користе како би се подаци приказали у корисничком интерфејсу, односно погледима као што су `ListView`, `RecyclerView` и `Spinner`. Адаптер делује као мост између података и компоненти корисничког интерфејса, омогућавајући прилагођавање приказа података у складу са потребама апликације. Адаптери преузимају податке из извора података и мапирају их на одговарајући приказ у компоненти корисничког интерфејса.

На листингу 17, је приказан један од адаптера. Овај адаптер (`SearchAdapter`) се користи за приказ резултата претраге корисника и имплементиран је наслеђивањем и прилагођавањем уграђеног адаптера `ArrayAdapter`. На тај начин се омогућава приказ резултата података о корисницима у погледу `ListView`, у оквиру активности `SearchActivity`. Такође, дефинише се обрађивач клика корисника на дугме за додавање пријатеља односно за упућивање захтева за пријатељство.

Интерфејс `AddFriendButtonClickListener` садржи једну методу `onAddFriendButtonClick`, која ће се позивати када се дугме за додавање пријатеља кликне. Адаптер има променљиву `addFriendButtonClickListener` која представља референцу на имплементацију интерфејса, омогућавајући активности `SearchActivity` да одговори на кликове.

Метод `getView` се користи за управљање приказом за сваки елемент у листи. Ако поглед `convertView` није доступан, креира се нови приказ помоћу `LayoutInflater`. Приказ се попуњава са подацима о кориснику (име, презиме, слика профила), а дугме за додавање пријатеља добија обрађивач клика на дугме.

```
class SearchAdapter(context: Context, users: ArrayList<User>) :
    ArrayAdapter<User>(context, R.layout.fragment_search, users) {

    interface AddFriendButtonClickListener {
        fun onAddFriendButtonClick(user: User)
    }

    var addFriendButtonClickListener: AddFriendButtonClickListener? = null

    override fun getView(position: Int, convertView: View?, parent:
    ViewGroup): View {
        var view = convertView
        val user: User? = getItem(position)

        if (view == null) {
            view =
            LayoutInflater.from(context).inflate(R.layout.fragment_search, parent,
            false)
        }

        val firstNameTextView =
        view!!.findViewById<TextView>(R.id.firstNameTextView)
        val lastNameTextView =
        view.findViewById<TextView>(R.id.lastNameTextView)
        val addFriendButton = view.findViewById<Button>(R.id.addFriendButton)
        val profileImageView =
        view.findViewById<ImageView>(R.id.searchProfileImage)

        user?.let {
            firstNameTextView.text = it.firstName
            lastNameTextView.text = it.lastName
            loadProfileImage(it, profileImageView)
        }

        addFriendButton.setOnClickListener {
            if (user != null) {
                addFriendButtonClickListener?.onAddFriendButtonClick(user)
            }
        }

        return view
    }
}
```

## Листинг 17 - Адаптер за претрагу корисника

Класа `SearchActivity` имплементира интерфејс `SearchAdapter.AddFriendButtonClickListener`, што значи да мора имплементирати методу `onAddFriendButtonClick` (листинг 18). Активност такође иницијализује `ListView` и `SearchAdapter`, постављајући адаптер за `ListView` и постављајући обрађивач за клик на дугме `addFriendButton` на референцу `this` односно на инстанцу активности `SearchActivity`.

Метод `initializeSearch` поставља поглед типа `ListView`, креира инстанцу адаптера `SearchAdapter` са листом корисника и поставља адаптер за `ListView`. Такође поставља `TextWatcher` за претрагу и ослушкивач за додир на поље за претрагу.

```
class SearchActivity: AppCompatActivity(),
    SearchAdapter.AddFriendButtonClickListener {

    private lateinit var searchEditText: EditText
    private lateinit var listView: ListView
    private lateinit var adapter: SearchAdapter
    private val users = ArrayList<User>()

    @SuppressWarnings("ClickableViewAccessibility")
    private fun initializeSearch() {
        listView = findViewById(R.id.searchListView)
        adapter = SearchAdapter(this, users)
        adapter.addFriendButtonClickListener = this
        listView.adapter = adapter

        searchEditText = findViewById(R.id.searchEditText)
        searchEditText.addTextChangedListener(createTextWatcher())
        searchEditText.setOnTouchListener(createOnTouchListener())
    }
}
```

## Листинг 18 - Део имплементације активности `SearchActivity`

## 6.2. Серверски део апликације

Овај одељак објашњава имплементацију контролера за руковање објавама, сервиса који садржи основне методе за креирање, ажурирање, брисање и претраживање, као и имплементацију репозиторијума. На крају, објашњена је и имплементација пријаве на систем са креирањем аутентификационог токена.

На листингу 19 је приказана имплементација једног REST контролера, који управља подацима о објавама. Контролер поседује имплементације за основне CRUD операције коришћењем GET, POST, PUT и DELETE метода. Контролер је обележен са `@RestController` и `@RequestMapping("api/posts")` анотацијама, које дефинишу да је ово REST контролер и постављају основни URL путању за све методе унутар контролера. Контролер користи неколико сервиса (`PostService`, `UserService`, `GroupService`, `ImageService`) за руковање различитим аспектима података о објавама, корисницима, групама и сликама.

Метода `createPost()` омогућава креирање нове објаве. Она прихвата садржај објаве и листу слика као параметре, као и информације о тренутном кориснику (`Principal`). Након што пронађе корисника по корисничком имену, креира нову објаву, поставља информацију о кориснику, садржај, датум креирања и иницијалну вредност да објава није обрисана. Затим позива приватну методу `getPostResponseEntity` која чува објаву и обрађује слике, ако су присутне. Сlike се чувају помоћу сервиса `ImageService` и повезују се са објавом.

Метода `delete()` служи за логичко брисање објаве, односно означавање да је та објава обрисана уместо трајног уклањања из базе података. Метода прима идентификатор објаве у путањи `/delete/{id}` и проналази дату објаву у бази података помоћу позива сервису `postService.findOneById(id)`. Ако објава није пронађена, враћа се одговор са статусом 404 (NOT FOUND), а ако јесте, атрибут `isDeleted` се поставља на `true`, а промена се чува у бази помоћу позива `postService.updatePost(existing)`. На крају, метода враћа ажурирану објаву са статусом 200 (OK).

Метода `getPostById()` служи за проналажење објаве по идентификатору тако што прима идентификатор у путањи `/find/{id}` и користи га да пронађе објаву путем позива сервису `postService.findOneById(id)`. Ако је објава пронађена, биће враћена клијенту са статусом 200 (OK).

Метода `getAllPosts()` проналази све објаве које нису обрисане. Користи позив сервису `postService.findAllByIsDeleted(false)` да пронађе све необрисане објаве, сортира их по датуму креирања у опадајућем редоследу, и враћа листу објава са статусом 200 (OK).

```
@RestController
```

```

@RequestMapping("api/posts")
public class PostController {
    @Autowired
    private PostService postService;

    @Autowired
    private UserService userService;

    @Autowired
    private GroupService groupService;

    @Autowired
    private ImageService imageService;

    @PostMapping("/create")
    public ResponseEntity<Post> createPost(@RequestPart("content") String
content,
                                           @RequestPart(name = "images",
required = false) List<MultipartFile> imageFiles,
                                           Principal principal) {
        String username = principal.getName();
        User user = userService.findByUsername(username);

        Post post = new Post();
        post.setUser(user);
        post.setContent(content);
        post.setCreationDate(LocalDateDateTime.now());
        post.setIsDeleted(false);

        return getPostResponseEntity(imageFiles, post);
    }

    private ResponseEntity<Post> getPostResponseEntity(@RequestPart(name =
"images", required = false) List<MultipartFile> imageFiles, Post post) {
        Post createdPost = postService.createPost(post);
        if (imageFiles != null && !imageFiles.isEmpty()) {
            List<Image> images = new ArrayList<>();
            for (MultipartFile imageFile : imageFiles) {
                String imagePath = imageService.saveImage(imageFile);
                Image image = new Image();
                image.setPath(imagePath);
                imageService.save(image);

                image.setPost(createdPost);
                images.add(image);
            }
            createdPost.setImages(images);
        }
        Post updatedPost = postService.updatePost(createdPost);

        return ResponseEntity.ok(updatedPost);
    }

    @PutMapping("/delete/{id}")
    public ResponseEntity<Post> delete(@PathVariable Long id) throws
ChangeSetPersister.NotFoundException {

```

```

        Post existing = postService.findOneById(id);

        if (existing == null) {
            return new ResponseEntity<>(null, HttpStatus.NOT_FOUND);
        }

        existing.setIsDeleted(true);
        Post updated = postService.updatePost(existing);

        return new ResponseEntity<>(updated, HttpStatus.OK);
    }

    @GetMapping("/find/{id}")
    public ResponseEntity<Post> getPostById(@PathVariable Long id) throws
    ChangeSetPersister.NotFoundException {
        Post post = postService.findOneById(id);
        return ResponseEntity.ok(post);
    }

    @GetMapping("/all")
    public ResponseEntity<List<Post>> getAllPosts() {
        List<Post> posts = postService.findAllByIsDeleted(false);
        Collections.sort(posts, (post1, post2) ->
        post2.getCreationDate().compareTo(post1.getCreationDate()));

        return ResponseEntity.ok(posts);
    }

```

#### Листинг 19 - REST контролер за управљање подацима о објавама

Интерфејс `PostService` садржи основне методе за креирање, ажурирање, брисање и претраживање објава, док класа `PostServiceImpl`, приказана на листингу 20, имплементира те методе.

Метода `savePost()` прима објекат `Post` као параметар и чува га у бази података користећи позив репозиторијуму `postRepository.save(post)`. Ова метода се користи и за креирање и за ажурирање објава. Након чувања, метода враћа сачувану или ажурирану објаву.

Метода `deletePost()` прима идентификатор објаве типа `Long` и брише одговарајућу објаву из базе података коришћењем `postRepository.deleteById(id)`. Ова метода враћа једино `null` вредност.

Метода `findOneById()` прима идентификатор објаве типа `Long` и враћа објаву са тим идентификатором уколико она постоји у бази података. Метода користи позив репозиторијуму `postRepository.findById(id)`, те ако објава није пронађена генерише се изузетак `ChangeSetPersister.NotFoundException()`.

Метода `findAll()` враћа листу свих објава из базе података помоћу позива репозиторијуму `postRepository.findAll()`.

```

@Service
public class PostServiceImpl implements PostService {

    @Autowired
    private PostRepository postRepository;

    @Override
    public Post savePost(Post post) {
        return postRepository.save(post);
    }

    @Override
    public Post deletePost(Long id) {
        postRepository.deleteById(id);
        return null;
    }

    @Override
    public Post findOneById(Long id) throws
    ChangeSetPersister.NotFoundException {
        return postRepository.findById(id)
            .orElseThrow(() -> new
    ChangeSetPersister.NotFoundException());
    }

    @Override
    public List<Post> findAll() {
        return postRepository.findAll();
    }
}

```

Листинг 20 - Имплементација сервиса PostService

Репозиторијум PostRepository, који је приказан на листингу 21, проширује JpaRepository, што омогућава приступ основним функцијама за рад са ентитетом Post. У овом репозиторијуму је дефинисана метода `findAllByGroupIdAndIsDeleted` која омогућава претрагу свих објава на основу тога да ли припадају одређеној групи и на основу тога да ли су обрисани.

```

@Repository
public interface PostRepository extends JpaRepository<Post, Long> {
    List<Post> findAllByGroupIdAndIsDeleted(Long groupId, boolean isDeleted);
}

```

Листинг 21 - Имплементација репозиторијума PostRepository

У оквиру Spring Boot апликације, конфигурација за приступ бази података је извршена путем `application.properties` датотеке. На листингу 22 је дат изглед ових подешавања за MySQL базу података.

```

spring.datasource.url=jdbc:mysql://localhost:3306/socialNetwork?serverTimezone=Europe/Belgrade&useSSL=false
spring.datasource.username=root

```

```
spring.datasource.password=root
spring.datasource.driver-class-name=com.mysql.cj.jdbc.Driver
```

Листинг 22 - Датотека application.properties

У овој датотеци поље `spring.datasource.url` дефинише URL за конекцију са базом података. Поља `spring.datasource.username` и `spring.datasource.password` одређују корисничко име и лозинку за приступ бази. Поље `spring.datasource.driver-class-name` подешава драјвер који ће се користити за повезивање са MySQL базом.

Да би се омогућило коришћење поменутог JDBC драјвера у Spring Boot апликацији, потребне су и зависности које се додају у датотеку `pom.xml` (листинг 23). Ове зависности омогућавају коришћење одговарајућег JDBC драјвера за MySQL базу података.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-data-jdbc</artifactId>
</dependency>

<dependency>
<groupId>mysql</groupId>
<artifactId>mysql-connector-java</artifactId>
<version>8.0.33</version>
<scope>runtime</scope>
</dependency>
```

Листинг 23 - Датотека pom.xml

Имплементација пријаве на систем је приказана на листингу 24. Ова метода прихвата `JwtAuthenticationRequest` објекат као параметар, који садржи корисничко име и лозинку и `HttpServletResponse` објекат за управљање одговором.

Метода најпре аутентификује корисника користећи позив методе `authenticationManager.authenticate()`, која користи корисничко име и лозинку за креирање `UsernamePasswordAuthenticationToken` објекта. Ако је аутентификација успешна, `SecurityContextHolder` се ажурира са новим `Authentication` објектом, а `UserDetails` из аутентификације се користи за генерисање JWT токена помоћу `tokenUtils.generateToken()` методе. Затим, метода ажурира статус пријаве корисника и враћа `User` објекат на основу корисничког имена. Ако је корисник блокиран, враћа се одговор са статусом 403 (Forbidden). Ако није, ажурира се време последње пријаве корисника и информације се чувају у бази података.

Ако аутентификација није успешна, метод враћа одговор са статусом 401 (Unauthorized) и поруком `"Invalid username or password."`. У случају других грешака, враћа



се одговор са статусом 500 (Internal Server Error) и поруком "An unexpected error occurred."

```
@PostMapping("/login")
public ResponseEntity<?> createAuthenticationToken(
    @RequestBody JwtAuthenticationRequest authenticationRequest,
    HttpServletResponse response) {

    try {
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(
                authenticationRequest.getUsername(),
                authenticationRequest.getPassword()
            )
        );

        SecurityContextHolder.getContext().setAuthentication(authentication);
        UserDetails userDetails = (UserDetails) authentication.getPrincipal();
        String jwt = tokenUtils.generateToken(userDetails);
        int expiresIn = tokenUtils.getExpiredIn();

        updateUserLoginStatus(authenticationRequest.getUsername(), true);
        User user =
            userService.findByUsername(authenticationRequest.getUsername());

        Banned existingBanned = bannedService.findExistingBanned(user);
        if (existingBanned != null && existingBanned.isBlocked()) {
            return ResponseEntity.status(HttpStatus.FORBIDDEN)
                .body("Your account is blocked.");
        }

        user.setLastLogin(LocalDateDateTime.now());
        userService.updateUser(user);

        return ResponseEntity.ok(new UserTokenState(jwt, expiresIn));

    } catch (BadCredentialsException e) {
        return ResponseEntity.status(HttpStatus.UNAUTHORIZED)
            .body("Invalid username or password.");
    } catch (Exception e) {
        return ResponseEntity.status(HttpStatus.INTERNAL_SERVER_ERROR)
            .body("An unexpected error occurred.");
    }
}
```

Листинг 24 - Имплементација пријаве на систем

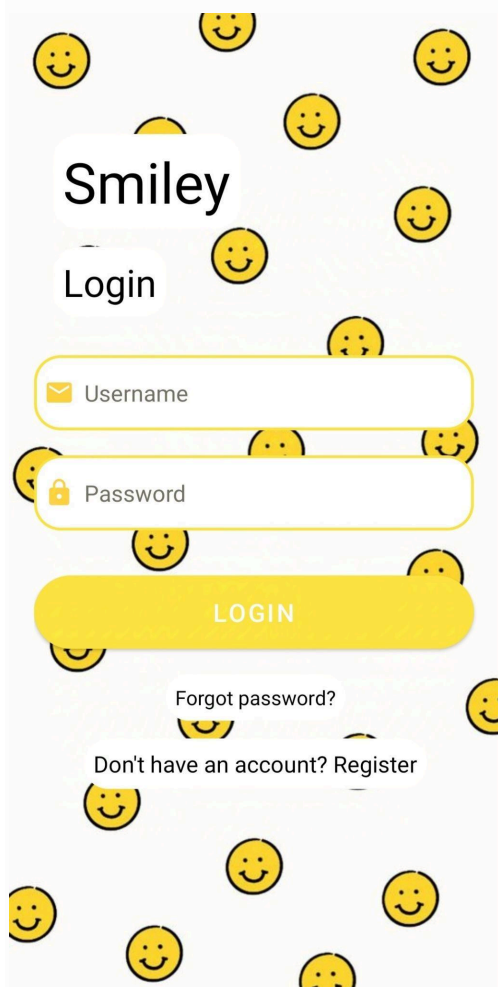
## 7. Демонстрација

Ово поглавље приказује начин коришћења система путем мобилне апликације.

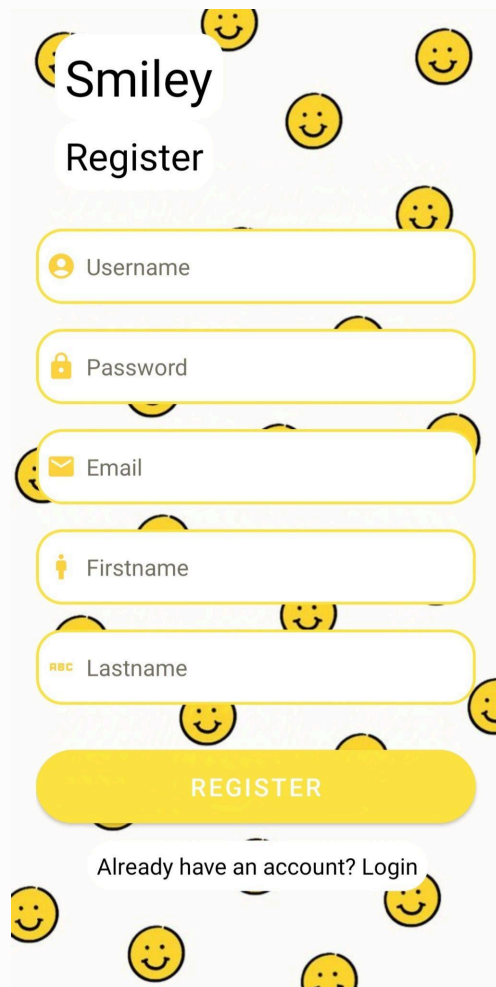
Приликом покретања апликације, кориснику се приказује екран за пријаву, као што је илустровано на слици 4. Корисник уноси своје креденцијале како би се пријавио у систем. Када корисник унесе своје креденцијале и кликне на дугме "LOGIN", систем проверава да ли су корисничко име и лозинка исправни. Ако су креденцијали тачни, корисник добија приступ свом налогу. У супротном, приказује се порука о грешци са обавештењем да су корисничко име или лозинка неисправни.

Ако корисник нема налог, може се регистровати путем форме за регистрацију приказане на слици 5. Процес регистрације захтева следеће податке: корисничко име, лозинку, e-mail адресу, име и презиме.

Када корисник унесе све потребне податке и кликне на дугме "REGISTER", систем проверава да ли су сви подаци валидни и да ли корисничко име и e-mail адреса нису већ у употреби. Ако су сви подаци валидни, корисников налог се креира и он добија могућност да се пријави у систем користећи своје креденцијале.



Слика 4 - Пријава на систем



Слика 5 - Регистрација на систем

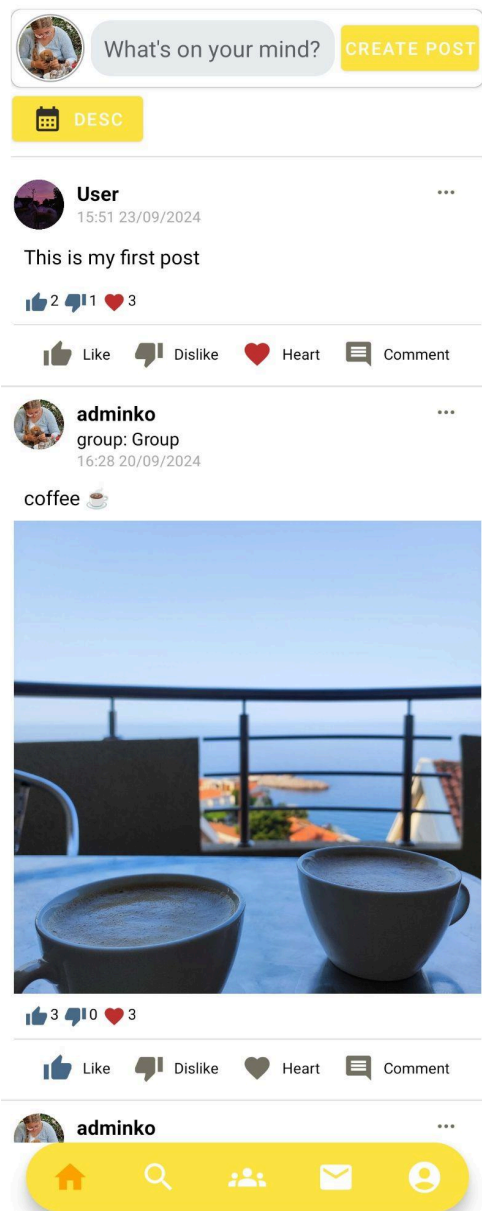
Након успешне пријаве, кориснику се приказује почетни екран апликације, као што је илустровано на слици 6. На почетном екрану, регистровани корисници виде објаве које су јавне (сопствене објаве и објаве својих пријатеља) или објаве из група којима припадају. Поред сваке објаве која потиче из неке од група, налази се линк који води ка тој групи.

На дну екрана налази се навигациони мени који омогућава корисницима да изаберу неку од опција, као што су почетни екран, претрага група, захтева и профила. Такође, корисници могу реаговати на приказане објаве помоћу дугмади за лајковање, дислајковање и "срце", као и прегледати коментаре на те објаве. У горњем десном углу екрана налази се дугме за сортирање објава. Објаве је могуће сортирати по датуму објављивања, у растућем или опадајућем поретку.

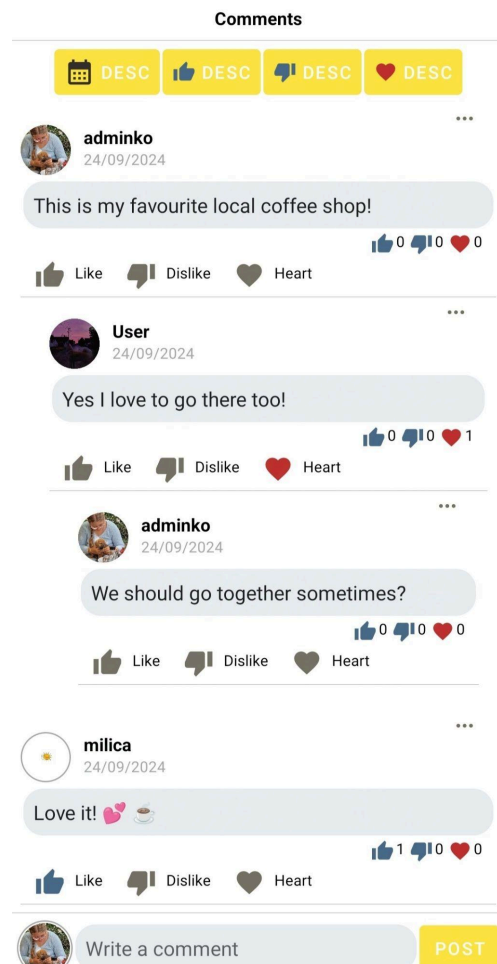
Када корисник кликне на дугме за коментар на објави, приказаће му се сви постојећи коментари на ту објаву (слика 7). На врху екрана са коментарима налазе се четири дугмета за сортирање коментара: према броју лајкова, према броју дислајкова, према броју "срца", у опадајућем или растућем поретку, као и према датуму објављивања од најскоријег ка најстаријем или од најстаријег ка најскоријем.

Испод ових дугмади приказују се сви коментари, а на дну екрана налази се форма за креирање нових коментара, где корисник може унети текст коментара и кликнути на дугме "POST" за додавање тог коментара.

Кликом на три тачке поред коментара, кориснику се пружа могућност да одговори на коментар. Могућ је произвољан број одговора на коментар, што омогућава стварање низа одговора. Такође, корисници могу реаговати на коментаре помоћу дугмади за лајковање, дислајковање и "срце".



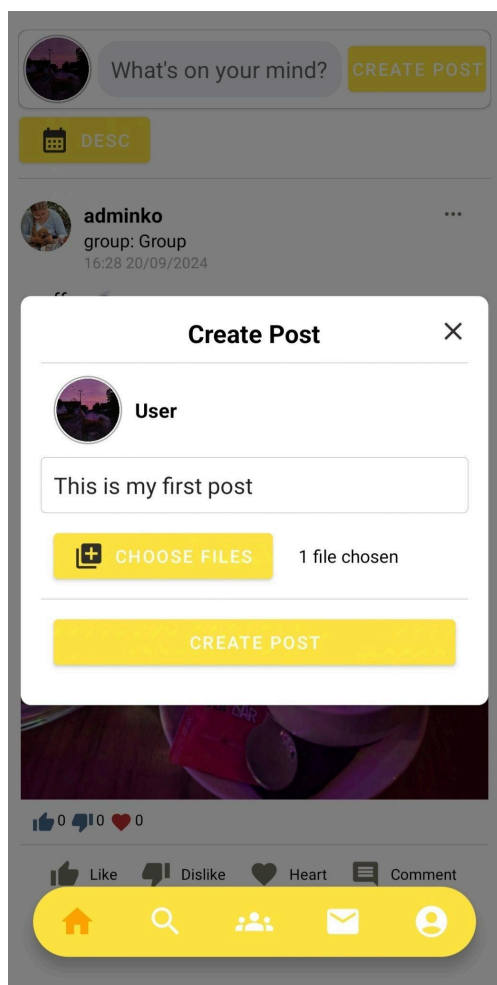
Слика 6 - Почетни екран



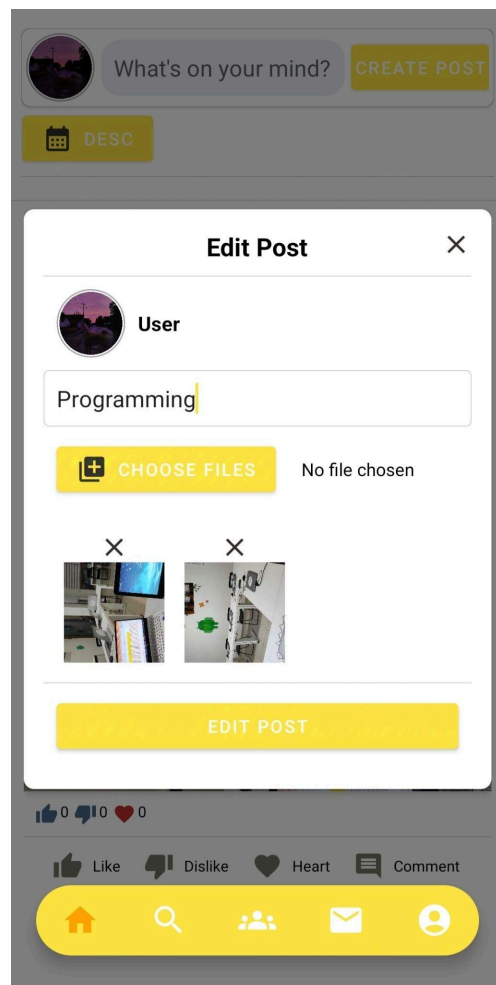
Слика 7 - Коментари

На врху почетног екрана налази се дугме "CREATE POST" које отвара форму за креирање нове објаве, као што је приказано на слици 8. Објава може садржати текст и опционо слике. Сlike се додају кликом на дугме "CHOOSE FILES". Уколико их објава садржи, може имати једну или више слика. Обавезно поље за креирање објаве је садржај, односно текст објаве. Кликот на дугме "CREATE POST" објава се креира. У горњем десном углу форме налази се "X" дугме за одустајање од креирања објаве.

На почетном екрану, корисници могу кликнути на три тачке приказане уз њихове објаве, чиме им се омогућава измена или брисање објаве. Кликот на дугме "EDIT POST" отвара се форма (слика 9) слична оној за креирање објаве, с тим да се у њој већ налазе попуњени подаци који се могу изменити. Кликот на дугме "EDIT POST" направљене измене се чувају.



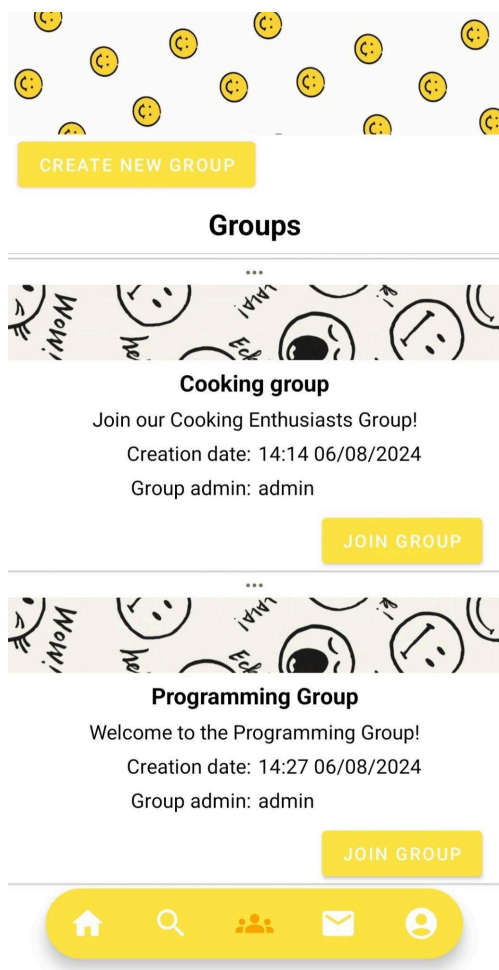
Слика 8 - Креирање објаве



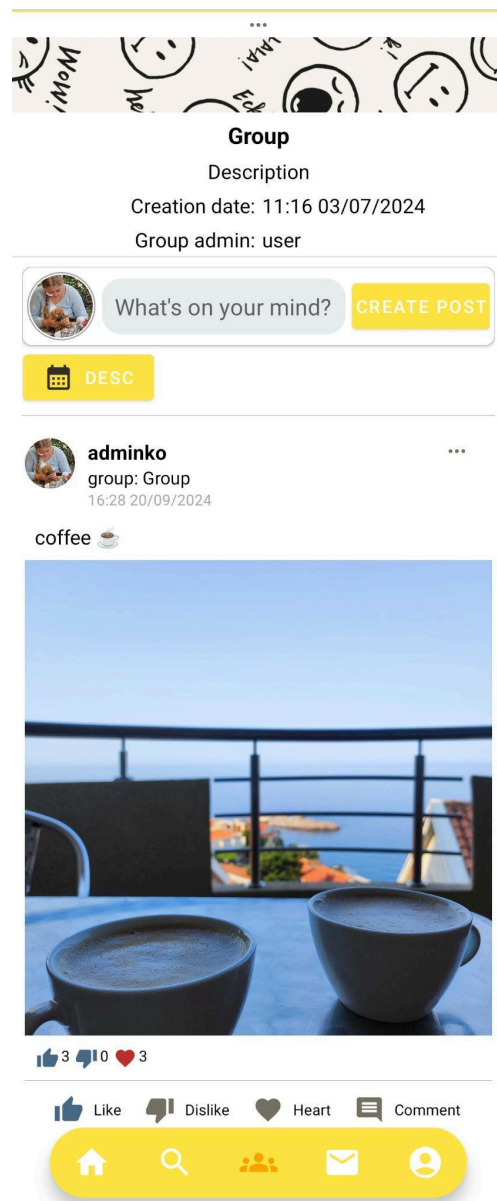
Слика 9 - Измена објаве

Преглед свих група је приказан на слици 10. Њему се приступа преко навигационог менија, кликом на трећу иконицу с леве стране. На овом екрану су приказане све групе са њиховим називом, описом, датумом креирања и администратором групе, као и дугметом "JOIN GROUP" помоћу којег се шаље захтев за учлањење у групу.

Кликом на неку од група приказаше се почетна страница групе (слика 11), а уколико је корисник истовремено и члан те групе, приказаше му се и њен садржај што подразумева објаве те групе, као и могућност за креирање групне објаве помоћу дугмета "CREATE POST".



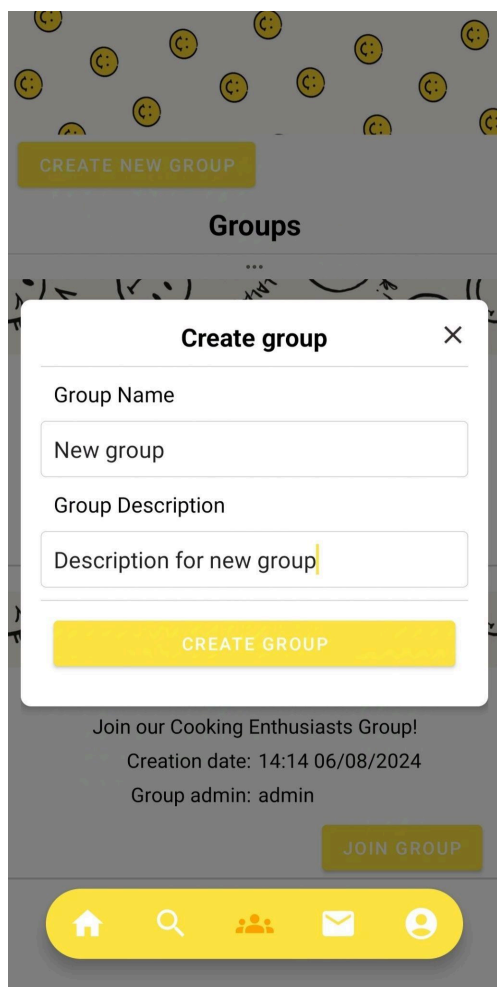
Слика 10 - Преглед свих група



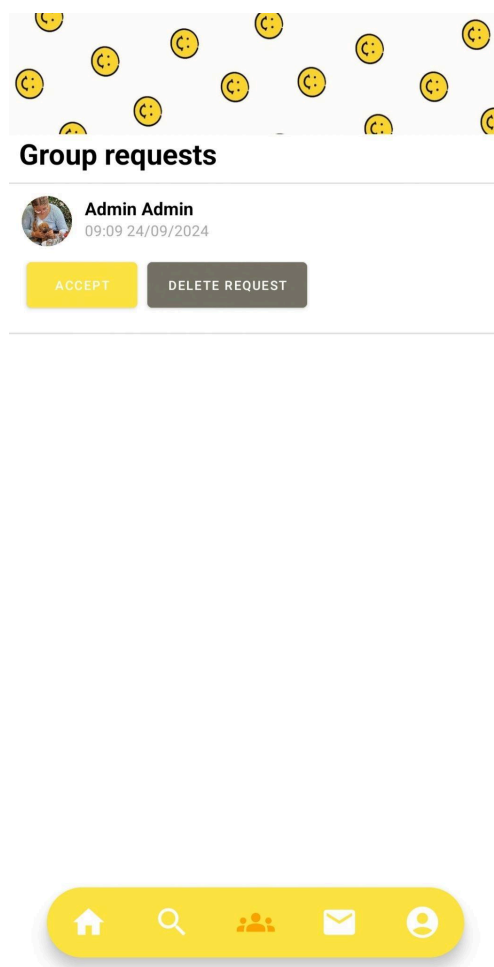
Слика 11 - Почетни екран групе

На екрану са прегледом свих група, у горњем левом углу налази се дугме за креирање групе, на чији клик се отвара форма приказана на слици 12. Било који регистровани корисник може да креира групу и аутоматски постаје администратор те групе. Након уноса назива и описа групе, кликом на дугме "CREATE GROUP" нова група ће бити креирана.

Корисник на почетном екрану групе коју је сам креирао, након клика на три тачкице, може прегледати захтеве других корисника за приступ тој групи. Преглед захтева за улазак у групу је приказан на слици 13. Тек након што одобри захтев неког корисника, тај корисник постаје део групе. Такође, пристигли захтеви се могу и одбити.



Слика 12 - Креирање групе

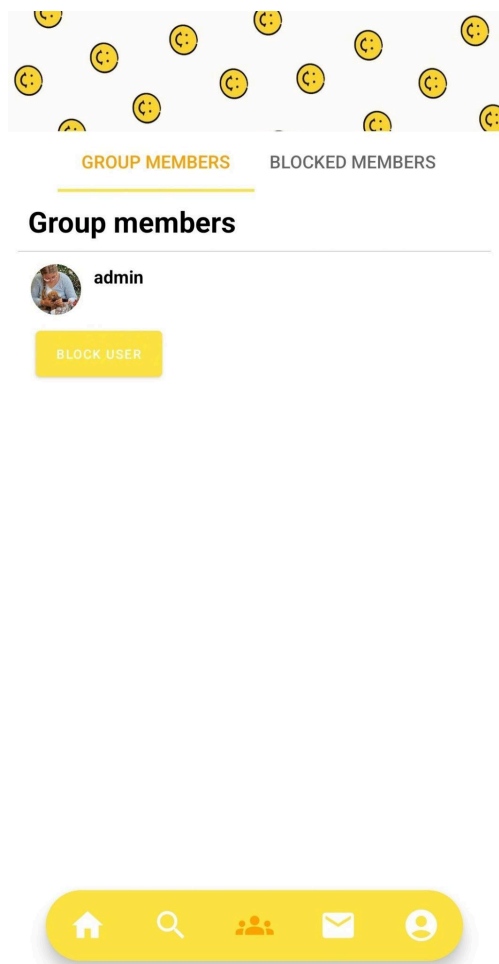


Слика 13 - Преглед захтева за улазак у групу

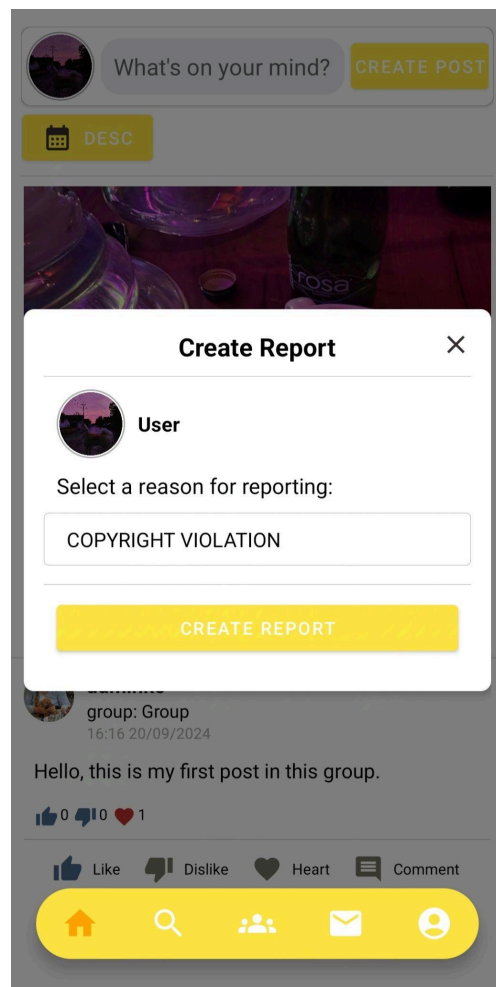
На почетном екрану групе, корисник који ју је креирао, односно њен администратор, кликом на три тачкице може отворити списак чланова групе и списак блокираних чланова групе (слика 14). Администратор групе има могућност да блокира било ког члана групе чиме спречава тог корисника да приступа групи. Осим што се том кориснику онемогућава да додаје објаве у групи, неће му се ни приказивати објаве из те групе на његовом почетном екрану. Такође, администратор групе на списку блокираних чланова може и да одблокира неког од корисника.

За пријаву неприкладног садржаја, корисници могу отворити форму тако што ће кликнути три тачкице на објави, коментару или профилу корисника, а затим одабрати опцију "REPORT." Ова функционалност омогућава сваком кориснику да пријави садржај или корисника као неприкладног. Форма за пријаву је приказана на слици 15 и у њој корисник може изабрати разлог пријаве и притиснути дугме "CREATE REPORT" како би комплетирао процес пријављивања.





Слика 14 - Преглед чланова групе и блокираних чланова

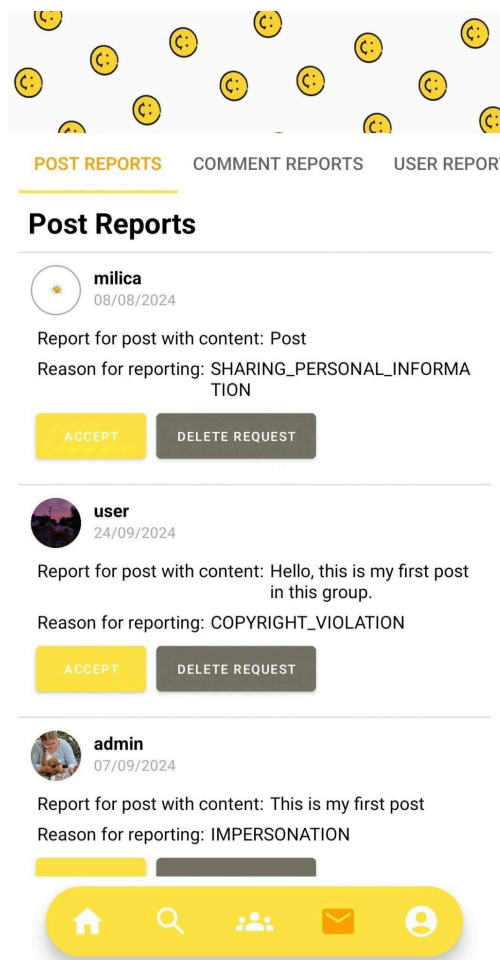


Слика 15 - Пријава садржаја

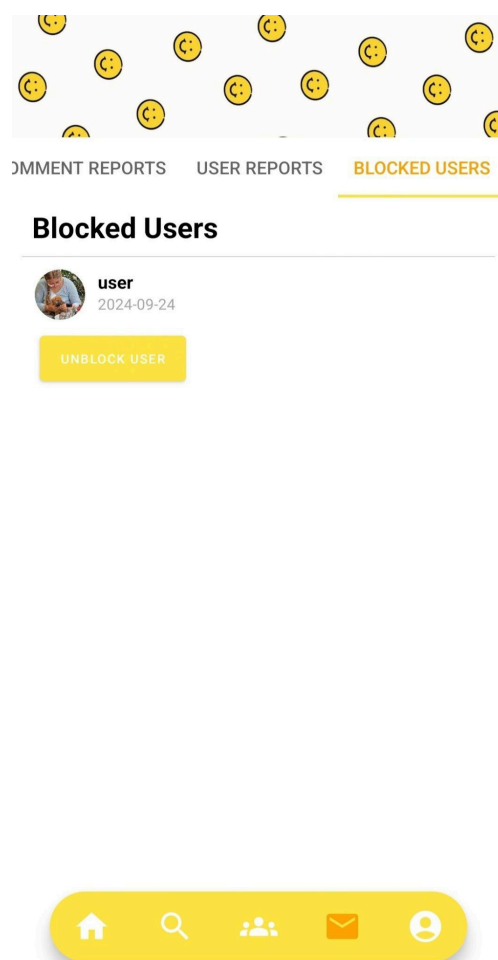
Прегледу пријављеног садржаја се приступа преко четврте иконице у навигацији, након чега се приказује екран са пријављеним објавама, коментарима, корисницима, као и блокираним корисницима (слика 16 и слика 17). Пријављени садржај се прегледа од стране администратора групе ако садржај припада групи, или од администратора система уколико је садржај јаван. Пријављени садржај (коментар или објава) може бити суспендован од стране администратора групе или администратора система. Само администратор система може да блокира кориснике.

Блокирани корисник не може да се пријави на систем, док се суспендовани садржај даље не приказује на систему. Администратор система такође има преглед свих блокираних корисника које може одблокирати у било којем тренутку.





Слика 16 - Преглед пријављеног садржаја

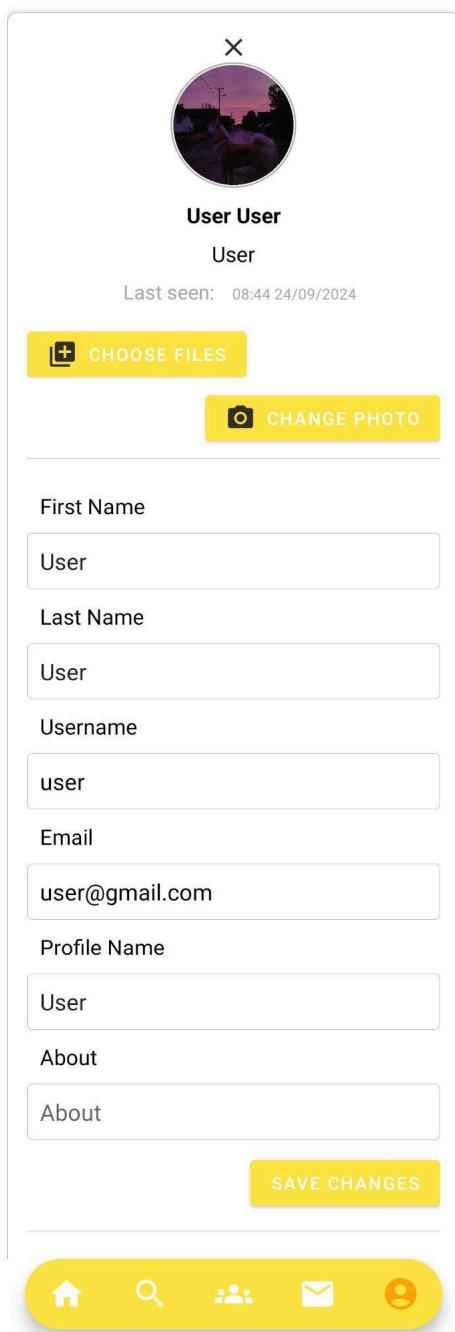


Слика 17 - Преглед блокираних корисника


На екрану са профилем корисника, који је приказан на слици 18 и слици 19, корисници могу прегледати своје податке, укључујући последње време логовања, као и информације о профилима других корисника. Такође, корисници имају могућност да измене своје податке.

Промене додатних података обухватају могућност подешавања имена које ће се приказивати уместо корисничког имена, као и додавање описа профила и слике. Корисници могу додати или заменити слику путем дугмета "CHOOSE FILES", док се брисање врши притиском на дугме "x". Након уноса свих измена, корисници треба да притисну дугме "SAVE CHANGES" како би сачували промене.

У наставку екрана са профилем корисника, такође је доступна форма за промену лозинке. Процес укључује унос тренутне лозинке и два пута уношење нове лозинке за потврду. На профилу се такође може видети списак свих група којима корисник припада, као и листа његових пријатеља. На крају, корисници имају и опцију за одјаву из система.




✕




**User User**

User

Last seen: 08:44 24/09/2024

 CHOOSE FILES

 CHANGE PHOTO

---

First Name

User

Last Name

User

Username

user

Email

user@gmail.com

Profile Name

User

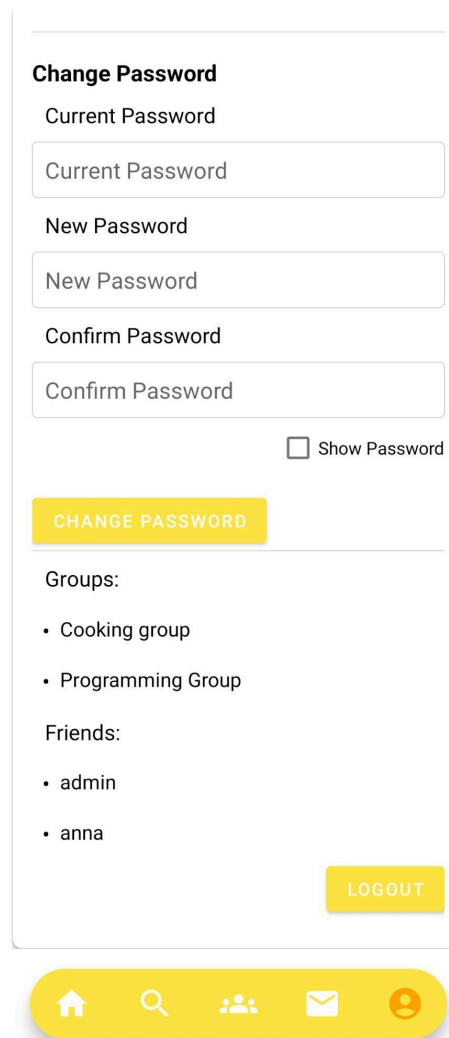
About

About

SAVE CHANGES

Home Search Friends Messages Profile

Слика 18 - Профил корисника



**Change Password**

Current Password

Current Password

New Password

New Password

Confirm Password

Confirm Password

☐ Show Password

CHANGE PASSWORD

---

Groups:

- Cooking group
- Programming Group

Friends:

- admin
- anna

LOGOUT

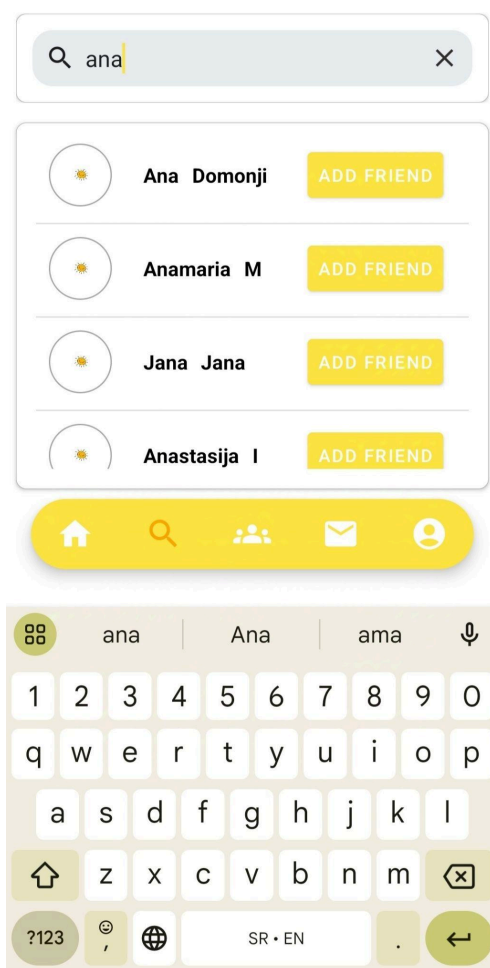
Home Search Friends Messages Profile

Слика 19 - Профил корисника (наставкак)

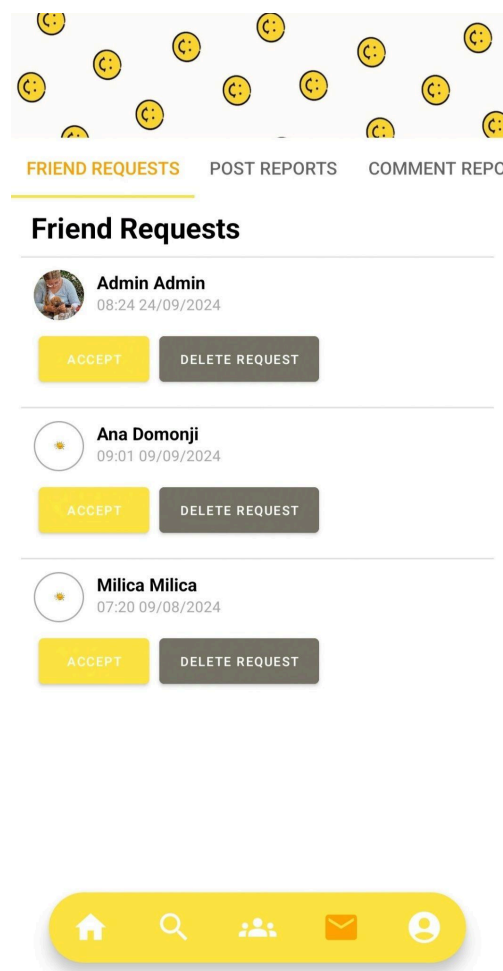
Претрага корисника у систему, с циљем проналаска нових пријатеља, доступна је на другој иконици навигационог менија, као што је приказано на слици 20. Претрага се врши на основу имена и презимена корисника који се уносе у форму, а процес претраге се аутоматски започиње. Сваки корисник може да пошаље захтев за пријатељство кликом на дугме поред имена и презимена приказаних у резултатима претраге корисника.

Преглед захтева за пријатељство отвара се помоћу четврте иконце навигационог менија (слика 21). Корисници имају увид у све пристигле захтеве за пријатељство, уз

могућност прихватања или одбијања сваког захтева путем два доступна дугмета. Тек по прихватању захтева, корисник постаје пријатељ са другим корисником.



Слика 20 - Претрага корисника



Слика 21 - Преглед захтева за пријатељство

## 8. Закључак

У овом раду је представљена апликација друштвене мреже која је развијена за Android платформу користећи RESTful архитектуру и Retrofit библиотеку. Ова апликација омогућава корисницима да креирају своје профиле, повезују се са пријатељима, деле објаве и коментаришу садржај. Комуникација између клијентске апликације и серверског дела се остварује путем RESTful API интерфејса.

Прототипском апликацијом је демонстрирано функционисање свих задатих функционалности уз једноставност коришћења. Употребљена Retrofit библиотека је над постојећим подацима омогућила одзив система у реалном времену, односно ажурност података приказаних кориснику. Дизајн апликације је интуитиван, што олакшава корисницима креирање и дељење садржаја. Ипак, постоје и изазови, као што је зависност система од поуздане интернет конекције, што у подручјима са слабијим сигналом мобилне односно WiFi мреже може отежати коришћење апликације.

Иако је апликација у потпуности одговорила на постављене захтеве, постоје могућности за њено побољшање. У будућим верзијама, било би корисно размотрити имплементацију функционалности која омогућава коришћење апликације у одсуству интернет конекције на тај начин што би информације о операцијама над подацима привремено чувале у меморији уређаја, а по опоравку интернет конекције би се коначно реализовале и на серверском делу система. Осим тога, подршка размени приватних порука између корисника би значајно унапредила могућности система. Такође, било би корисно уколико би се даљим развојем система обухватила изградња клијентске апликације и за друге мобилне платформе, што се може остварити коришћењем радних оквира као што су Flutter [31] или React Native [32].

## Литература

- [1] Facebook. 2024. About Facebook. Преузето са <https://about.facebook.com> (посећено 14.10.2024.)
- [2] Instagram. 2024. About Us. Преузето са <https://about.instagram.com> (посећено 14.10.2024.)
- [3] Twitter. 2024. About Twitter. Преузето са <https://about.twitter.com> (посећено 14.10.2024.)
- [4] LinkedIn. 2024. About LinkedIn. Преузето са <https://about.linkedin.com> (посећено 14.10.2024.)
- [5] TikTok. 2024. About TikTok. Преузето са <https://www.tiktok.com/about> (посећено 14.10.2024.)
- [6] Android. Преузето са <https://developer.android.com/> (посећено 14.10.2024.)
- [7] Java. Преузето са <https://www.java.com/en/> (посећено 14.10.2024.)
- [8] Retrofit. Преузето са <https://square.github.io/retrofit/> (посећено 14.10.2024.)
- [9] HTTP. Преузето са [https://www.w3schools.com/whatis/whatis\\_http.asp](https://www.w3schools.com/whatis/whatis_http.asp) (посећено 14.10.2024.)
- [10] JSON. Преузето са <https://www.json.org/json-en.html> (посећено 14.10.2024.)
- [11] XML. Преузето са <https://mailchimp.com/marketing-glossary/xml/> (посећено 14.10.2024.)
- [12] Gson. Преузето са <https://github.com/google/gson> (посећено 14.10.2024.)
- [13] Moshi. Преузето са <https://github.com/square/moshi> (посећено 14.10.2024.)
- [14] Kotlin. Преузето са <https://kotlinlang.org/> (посећено 14.10.2024.)
- [15] REST API. Преузето са <https://www.redhat.com/en/topics/api/what-is-a-rest-api> (посећено 14.10.2024.)
- [16] Spring Boot. Преузето са <https://projects.spring.io/spring-boot/> (посећено 14.10.2024.)
- [17] Spring. Преузето са <https://spring.io/> (посећено 14.10.2024.)
- [18] Spring Framework. Преузето са <https://spring.io/> (посећено 14.10.2024.)
- [19] Maven. Преузето са <https://maven.apache.org/> (посећено 14.10.2024.)
- [20] Spring security. Преузето са <https://spring.io/projects/spring-security> (посећено 14.10.2024.)
- [21] Servlet API. Преузето са <https://www.javatpoint.com/servlet-api> (посећено 14.10.2024.)
- [22] Spring Web MVC. Преузето са <https://docs.spring.io/spring-framework/reference/web/webmvc.html> (посећено 14.10.2024.)
- [23] Hibernate. Преузето са <https://hibernate.org/> (посећено 14.10.2024.)
- [24] Java JDBC API. Преузето са <https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/> (посећено 14.10.2024.)
- [25] Java Persistence API. Преузето са <https://www.oracle.com/java/technologies/persistence-jsp.html> (посећено 14.10.2024.)
- [26] JWT Token. Преузето са <https://jwt.io/> (посећено 14.10.2024.)
- [27] RFC 7519. Преузето са <https://datatracker.ietf.org/doc/html/rfc7519>
- [28] MySQL. Преузето са <https://www.oracle.com/mysql/what-is-mysql/> (посећено 14.10.2024.)
- [29] Firebase. Преузето са <https://firebase.google.com/> (посећено 14.10.2024.)
- [30] Google. Преузето са <https://about.google/> (посећено 14.10.2024.)

[31] Flutter. Build apps for any screen. Преузето са <https://flutter.dev/> (посећено 14.10.2024.)

[32] React Native. Learn once, write anywhere. Преузето са <https://reactnative.dev/> (посећено 14.10.2024.)

## Биографија

Ана Домоњи је рођена 02.07.2002. у Сремској Митровици. У Старој Пазови је стекла своје основно и средње образовање. Школске 2021/22 године се уписала на Факултет техничких наука на студијски програм Софтверске и информационе технологије. Положила је све испите предвиђене планом и програмом и стекла услов за одбрану завршног рада.