# Razvoj aplikacije za podršku učenju matematike na tehničkim fakultetima

**Vulama, Filip**

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 339

# RAZVOJ APLIKACIJE ZA PODRŠKU UČENJU MATEMATIKE NA TEHNIČKIM FAKULTETIMA

Filip Vulama

Zagreb, lipanj 2024.

SVEUČILIŠTE U ZAGREBU
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

DIPLOMSKI RAD br. 339

# RAZVOJ APLIKACIJE ZA PODRŠKU UČENJU MATEMATIKE NA TEHNIČKIM FAKULTETIMA

Filip Vulama

Zagreb, lipanj 2024.

**SVEUČILIŠTE U ZAGREBU**
**FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA**

Zagreb, 4. ožujka 2024.

# DIPLOMSKI ZADATAK br. 339

| | |
|---|---|
| Pristupnik: | **Filip Vulama (0036523812)** |
| Studij: | Računarstvo |
| Profil: | Programsko inženjerstvo i informacijski sustavi |
| Mentorica: | izv. prof. dr. sc. Josipa Pina Milišić |

Zadatak: **Razvoj aplikacije za podršku učenju matematike na tehničkim fakultetima**

Opis zadatka:

Na Fakultetu elektrotehnike i računarstva prepoznata je potreba dodatne podrške studentima tijekom procesa svladavanja matematičkih koncepata koji čine temelje inženjerske struke. Cilj ovog diplomskog rada je razvoj korisnički orijentiranog interaktivnog programskog rješenja za sintezu gradiva temeljnih matematičkih kolegija preddiplomskog studija. Uvodni dio rada bit će posvećen dizajnu aplikacija usmjerenih na korisnika. U središnjem dijelu rada obuhvatit će se oblikovanje i implementacija baze podataka i pripadajućeg REST aplikacijskog programskog sučelja. Konačno, napravljenu aplikaciju potrebno je testirati i evaluirati na nizu karakterističnih upita.

Rok za predaju rada: 28. lipnja 2024.

# Sadržaj

# Uvod

Cilj ovog diplomskog rada je pružiti detaljan uvid u procese i tehnologije koji su ključni u razvoju softvera, stvarajući temelje za razumijevanje kako se teorija primjenjuje u praksi.

Ovaj diplomski rad usredotočen je na razvoj aplikacije za podršku učenju matematike na tehničkim fakultetima, koristeći Kotlin kao primarni programski jezik i integrirajući brojne suvremene tehnologije i alate kao što su Android Studio, Git, Firebase i mnogi drugi. Međutim, važno je napomenuti da primarni fokus rada nije samo razvoj funkcionalne aplikacije, koja je također dio projekta, već detaljno istraživanje i dokumentiranje svakog koraka u procesu razvoja. Time se osigurava da rad služi kao praktični vodič kroz ključne aspekte softverskog inženjerstva koje će, prije ili kasnije, susresti svaki programer.

Osim što opisuje tehničke aspekte razvoja aplikacije, ovaj rad također razmatra i teoretske osnove koje stvaraju temelj za razumijevanje i primjenu navedenih tehnologija. Kroz praktični rad razvijena je aplikacija koja služi kao konkretan primjer kako teorija vodi do stvaranja stvarnih softverskih rješenja. Analizirajući procese kao što su inicijalizacija projekta, upravljanje verzijama koda, implementacija funkcionalnosti i testiranje, rad nudi duboki uvid u svakodnevne izazove i rješenja s kojima se susreću programeri.

Diplomski rad tako stavlja snažan naglasak na važnost procesa razvoja, educirajući buduće programere ne samo o tome kako pisati kod, već i kako efikasno upravljati kompleksnim softverskim projektima. Kroz ovo detaljno istraživanje, rad ne samo da pridonosi akademskoj zajednici, već pruža i praktične resurse koji će pomoći studentima i profesionalcima da unaprijede svoje tehničke vještine i profesionalni razvoj.

# 1. Matematika u obrazovanju

U razmatranju učinkovitosti metodologije učenja matematike i primjeni inovativnih tehnologija, posebice razvoju aplikacija koje podupiru proces učenja, možemo razmotriti kako tehnološki alati mogu obogatiti iskustvo učenja matematike na tehničkim fakultetima. Suvremeni pristupi, poput aplikacije ForMat, pružaju poticajno okruženje koje studentima omogućuje interaktivno matematičkih koncepta. Ova aplikacija koristi niz modernih tehnologija uključujući Kotlin, Android Studio, Git i Firebase, koje ne samo da olakšavaju razvoj aplikacije, već i doprinose njenoj funkcionalnosti, skalabilnosti i korisnosti.

Konkretno, ForMat aplikacija služi kao praktičan alat u kojem studenti mogu eksperimentirati s matematičkim formulama i konceptima kroz interaktivno sučelje. Ova platforma nudi mogućnosti dodavanja, uređivanja, objavljivanja i pregledavanja grupa formula, što studentima omogućuje lakše učenje kroz teoriju. Korištenje takvih alata može transformirati tradicionalne metode poučavanja matematike u dinamičnije i angažiranije procese, čime se povećava efikasnost učenja i studentima pruža realniji kontekst primjene naučenog.

Kroz primjenu ovakvih inovativnih tehnologija i metodologija, obrazovanje matematike na tehničkim fakultetima može postići znatno poboljšanje u smislu učinkovitosti učenja, studentove motivacije i konačne primjene naučenih vještina u profesionalnom okruženju. ForMat aplikacija i slični alati stoga predstavljaju značajan korak naprijed u modernizaciji i adaptaciji obrazovnih praksi koje mogu zadovoljiti zahtjeve suvremenog tehnološkog i inženjerskog okruženja.

Današnji napredak u obrazovanju sve više naglašava važnost integracije tehnologije u učionice, posebno u području tehničkih i matematičkih disciplina. Aplikacija ForMat primjer je kako moderni alati mogu značajno doprinijeti ne samo studentima već i predavačima u procesu učenja i poučavanja.

Jedna od glavnih prednosti koju ForMat nudi predavačima je mogućnost uvida u razumljivost matematičkih formula koje koriste studenti. Kroz interaktivnu platformu, predavači mogu pratiti koje formule i koncepti predstavljaju izazove studentima, te na temelju toga prilagoditi tempo i metodologiju nastave. Ova značajka omogućuje predavačima da izvrše pravovremene intervencije, osiguravajući da svi studenti mogu pratiti nastavni plan i program na efikasan način.

S druge strane, studenti danas većinu vremena provode s mobilnim uređajima u rukama, što ForMat koristi na svoju prednost. Aplikacija omogućuje studentima da pristupaju obrazovnim materijalima bilo kada i bilo gdje, potičući samostalno učenje i fleksibilnost. Ova stalna dostupnost pomaže studentima da učinkovito koriste svoje vrijeme i integriraju učenje u svoj dnevni raspored, što je ključno u današnjem ubrzanom načinu života.

Za daljnje poboljšanje efikasnosti stjecanja znanja preko aplikacije ForMat, bilo bi korisno uvesti sljedeće značajke:

- Personalizirano učenje: Razviti algoritme koji bi prepoznali individualne obrazovne potrebe i stilove učenja studenata, prilagođavajući sadržaj i tempo učenja pojedincu.
- Gamifikacija: Implementacija elemenata igre, poput bodovanja, medalja i ljestvica uspjeha, može povećati motivaciju i angažman studenata, čineći učenje zabavnijim i interaktivnijim.
- Interaktivni problemi i simulacije: Uključivanje složenijih simulacija i interaktivnih problema koji bi studentima omogućili da primijene teorijska znanja na praktične scenarije, što bi poboljšalo razumijevanje materijala.
- Odmah dostupna povratna informacija: Razvoj sustava za automatsko ocjenjivanje koji bi studentima odmah pružao povratne informacije na njihove odgovore, omogućujući im da brzo shvate i isprave svoje greške.
- Podrška za kolaborativno učenje: Mogućnost da studenti rade u virtualnim grupama na zajedničkim projektima ili diskutiraju probleme u realnom vremenu s drugim studentima i predavačima, potičući time suradnički pristup učenju.

Ove poboljšane funkcionalnosti mogu značajno unaprijediti kako individualno tako i grupno učenje, te omogućiti predavačima da još učinkovitije pristupe obrazovanju, posebno u tehničkim i matematičkim disciplinama. Integracija ForMat aplikacije u redoviti obrazovni sustav mogla bi transformirati kako pristupamo učenju matematike, čineći ga pristupačnijim, interaktivnijim i prilagođenijim potrebama suvremenog studenta.

# 2. Općenito o ForMat aplikaciji

ForMat korisniku nudi set funkcionalnosti vezane uz dodavanje, upravljanje, objavljivanje i pregled matematičkih formula.

Funkcionalnosti ForMat aplikacije su:

- Registracija / ulazak u korisnički račun
- Pregled, dodavanje i uređivanje grupe formula
- Objavljivanje grupe formula
- Preuzimanje objavljenih grupa formula
- Reakcije na preuzete formule
- Pregled reakcija na svoje objavljene grupe fromula

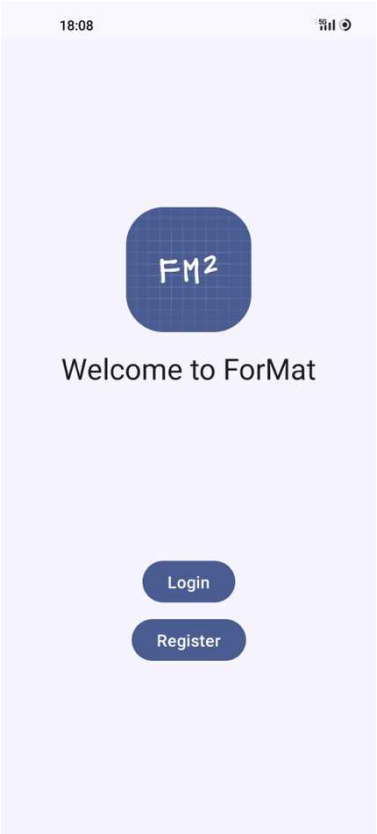
## 2.1. Dijelovi mobilne aplikacije

ForMat aplikacija sastoji se od nekolicine jednostavnih ekrana koji korisniku omogućuju korištenje prethodno navedenih funkcionalnosti

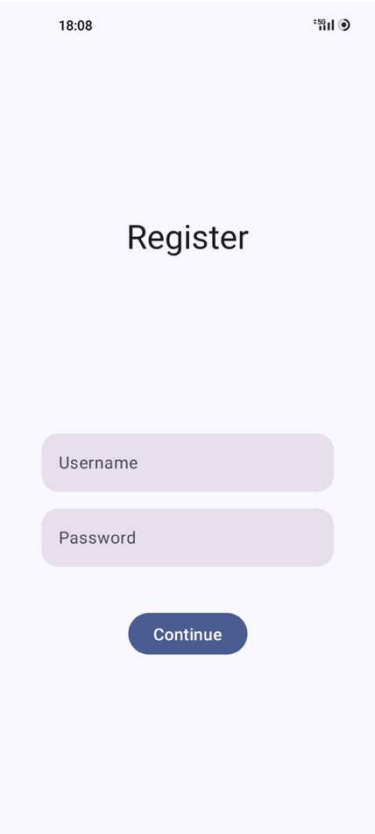
### 2.1.1. Ulazak u aplikaciju

Prilikom samog ulaska u aplikaciju dočekuje nas ekran dobrodošlice na kojem možemo birati opciju ulaska u već postojeći račun ili odabir kreacije novog računa.

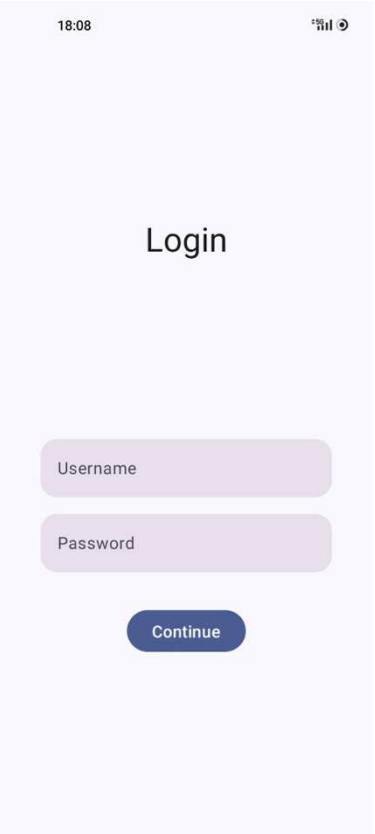
Ekran za ulazak u postojeći račun i ekran za kreaciju novog računa su gotovo jednaki uz male izmjene u samim tekstovima. Na njima se nalaze dva polja za unos korisničkog imena i lozinke, te gumb za nastavljanje u aplikaciju.




Slika 1. Ekran dobrodošlice




Slika 2. Ekran za registraciju



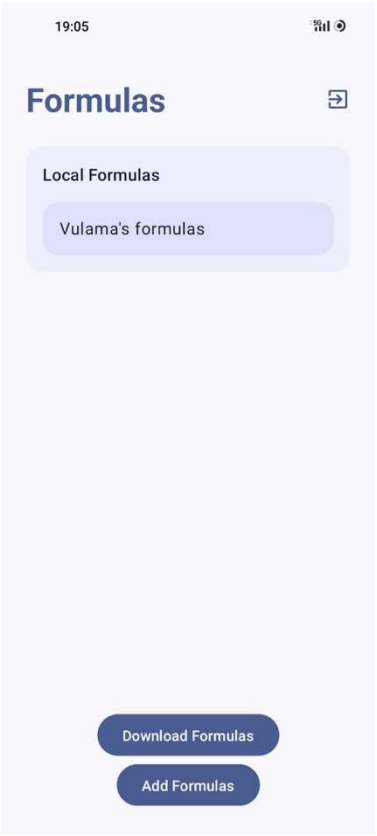

Slika 3. Ekran za ulaz u račun

### 2.1.2. Glavni ekran

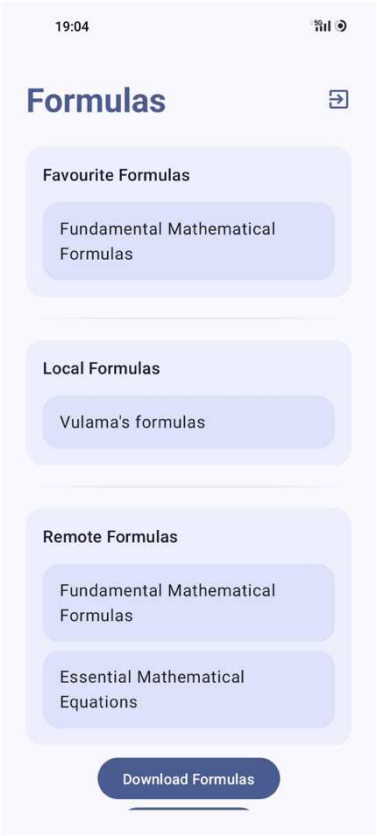
Na glavnom ekranu pri prvom ulazu dočekat će nas prazna ekran gdje će u budućnosti doći naše formule. Tako na tom praznom ekranu imamo opciju kreirati novu lokalnu grupu formula koja će biti pohranjena na našem uređaju, možemo preuzeti već prethodno objavljene grupe formula ili možemo izaći iz trenutno odabranog račun pritiskom na gornji desni gumb.

Nakon kreacije ili preuzimanja grupe formula na glavnom ekranu pokazuje nam se formula u jednoj od 3 kategorije.

Prva kategorija su favoriti, druga su lokalne grupe formula i treća su preuzete grupe formula. U favoritima se nalaze formule koje mogu biti i lokalne i preuzete s liste objavljenih grupa formula.



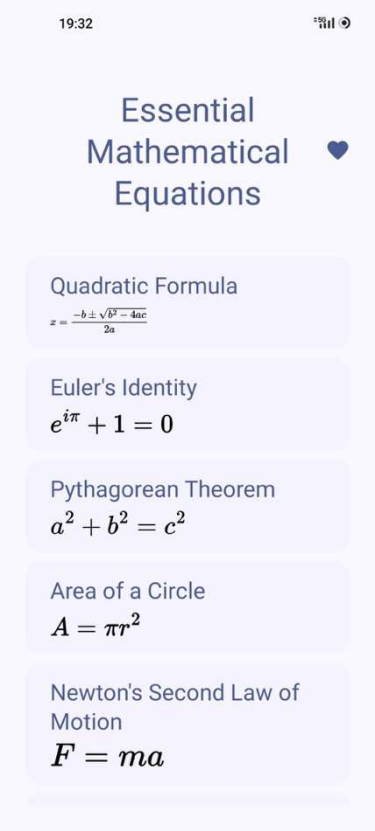

Slika 4. Glavni ekran s jednom grupom




Slika 5. Glavni ekran sa tri grupe

### 2.1.3. Ekran za pregled grupe i formule

Pritiskom na jednu od grupa formula na listi, ulazimo na ekran za pregled grupe formula. Na tom ekranu dobivamo opciju za uklanjanje grupe iz naše liste (koja god to lista bila), imamo opciju da označimo grupu kao favorit i imamo opciju ući u pregled pojedinačne formule.

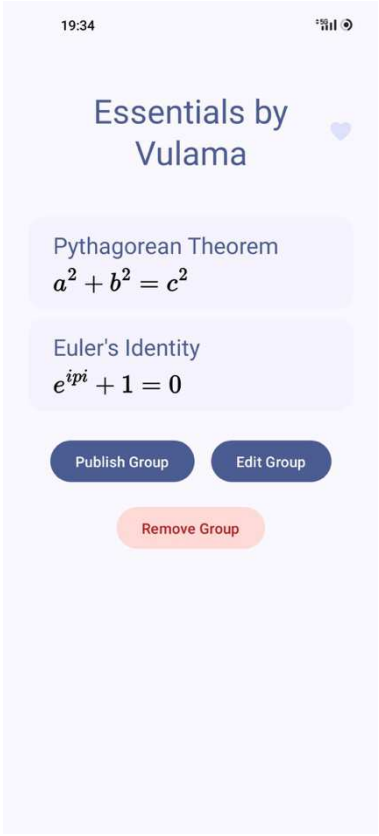



Slika 6. Standardan ekran pregleda preuzete grupe

Uz te opcije za svaku grupu, postoje i dvije dodatne opcije koje se pojavljuju samo na lokalnim grupama. Prva je objavljivanje trenutne grupe, a druga nam donosi opciju za uređivanje te iste grupe.

Ukoliko smo vlasnik objavljene formule na samom pregledu grupe imat ćemo i prikaz sume reakcija u grupi formula.

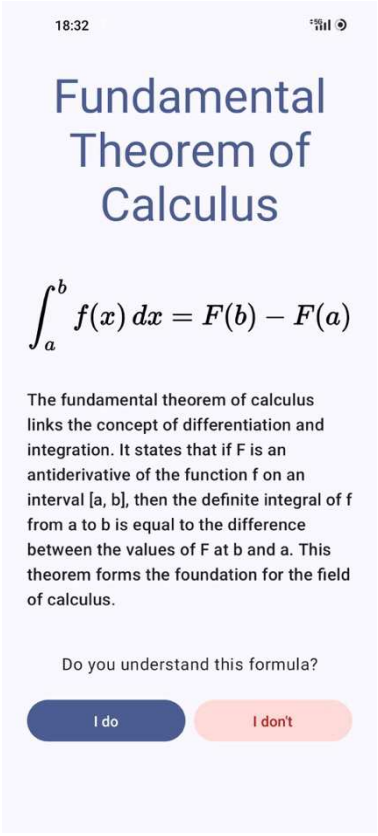

Slika 7. Ekran vlasnika grupe nakon objavljivanja



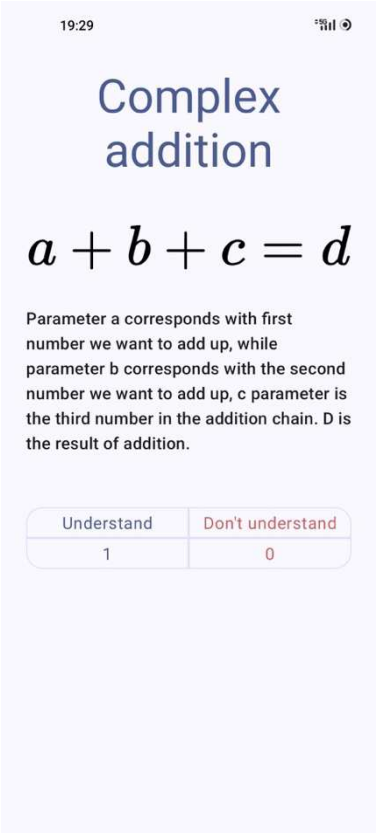

Slika 8. Ekran vlasnika grupe prije objavljivanja

Ulaskom u određenu formulu imamo prikazan naslov formule, matematički izraz formule i opis uz formulu.

Isto tako ukoliko smo vlasnik formule vidjet ćemo prikaz reakcija na samu formulu, a ukoliko nismo vlasnik formule dobit ćemo opcije da reagiramo na nju sa „Razumijem“ ili „Ne razumijem“ gumbima.




Slika 9. Standardni prikaz pregleda formule




Slika 10. Pregled formule vlasnika grupe

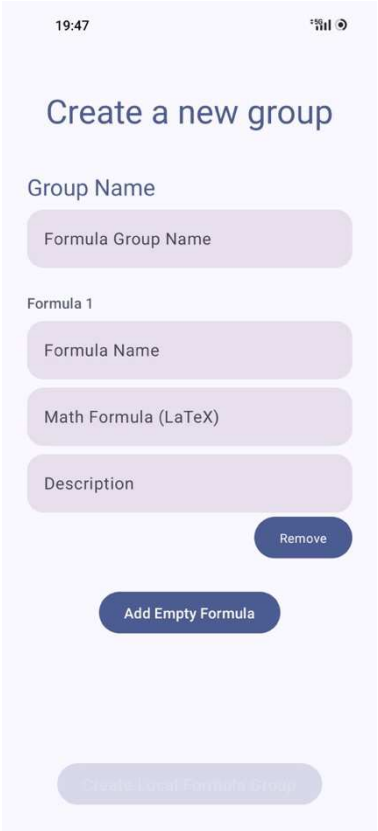
### 2.1.4. Ekran za uređivanje grupe formula

Ekran za uređivanje formula dijeljen je s ekranom za dodavanje novih formula samo mu je u tom slučaju inicijalno stanje prazno.

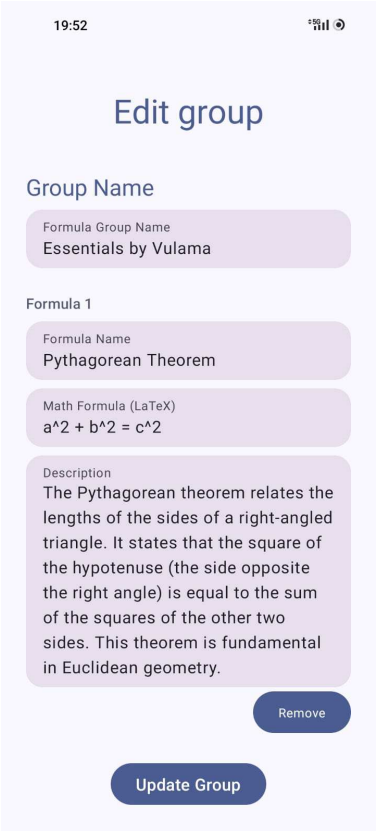
Pri samom vrhu ekrana nalazi se polje za upis imena grupe formula.

Ispod toga se nalaze 3 polja (za svaku formulu po 3 polja) za upis pojedine formule. To su ime formule, njena matematička formula te opis. Ispod svake formule u desnom kutu nalazi se gumb za uklanjanje određene formule. Dok je na kraju svih formula gumb za dodavanje jedne „prazne“ formule.

Na samom dnu ekrana nalazi se gumb za završetak uređivanja/kreacije grupe formula.



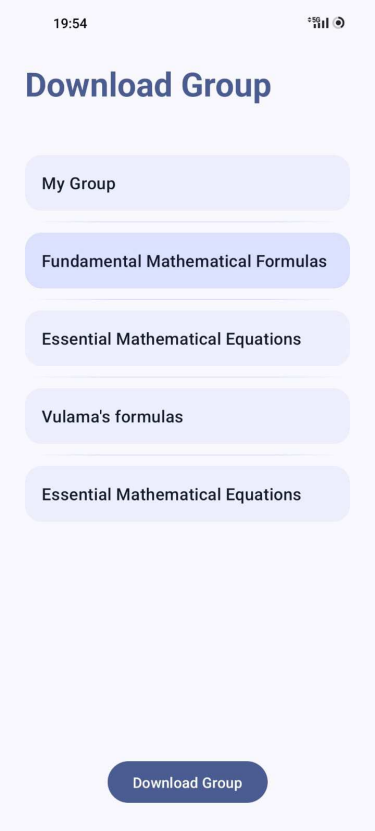

Slika 11. Ekran za dodavanje nove grupe




Slika 12. Ekran za izmjenu već postojeće grupe

### 2.1.5. Ekran za preuzimanje formula

Ekran za preuzimanje formula poprilično je jednostavan jer se na njemu samo nalazi lista grupa koje možemo preuzeti. Klikom na određenu grupu nju odabiremo te klikom na gumb na dnu ekrana potvrđujemo svoj odabir i dodajemo formulu na listu svojih preuzetih formula. Ta grupa je i registrirana na korisničkom računu te ukoliko korisnik odluči uzeti drugi uređaj i na njega se prijaviti, ta odabrana grupa će mu se preuzeti i na novom uređaju.




Slika 13. Ekran za preuzimanje objavljenih formula

## 2.2. Prednosti za korisnika

Iako aplikacije ne sadrži velik broj značajki kao neke od današnjih aplikacija, u svojoj domeni primjene i dalje nudi velik značaj za svoje korisnike. Za same studente/korisnike aplikacije, ona nudi brz pregled kroz odabrane liste formula koje studentima uvelike olakšavaju „brzo“ ponavljanje prije samog odlaska na pismenog ili usmenog ispita. Isto tako nudi im mogućnost da si sažmu i formuliraju formule u njima poželjnom i prepoznatljivom obliku što je nekima velik plus i to već rade samo u svojim bilježnicama ili virtualnim pločama. Isto tako te formule mogu dijeliti s ostalim korisnicima kojima bih možda takav jedan pojednostavljen pregled omogućio lakše pamćenje istih.

Korisnici koji bi u aplikaciji vidjeli primarnu funkcionalnost kao objavljivanje svojeg seta formula bi mogli biti profesori, asistenti, voditelji auditornih vježbi i sl. Oni bih na taj način lakše studentu pružali popis formula koje se od studenta očekuje da nauči a uz to bih dobili i reakcije na same formule koje bi onda mogli dodatno objasniti ili ponoviti na samim predavanjima ili auditornim vježbama.

# 3. Tehnologije

U današnjem digitalnom dobu, mobilne aplikacije su postale neizostavan dio svakodnevnog života milijardi ljudi širom svijeta. S obzirom na raznolike potrebe korisnika i brz tempo tehnoloških inovacija, postoji konstantna potreba za efikasnim alatima i tehnologijama koje omogućuju brz i pouzdan razvoj mobilnih aplikacija.

## 3.1. Kotlin – odabrani programski jezik

Kotlin je relativno novi programski jezik kojeg je razvio JetBrains, postao je sve popularniji izbor među programerima za razvoj Android aplikacija. Ovaj jezik kombinira izražajnost, sigurnost tipova i interoperabilnost sa postojećim Java kodom, čineći ga moćnim alatom za moderni razvoj mobilnih aplikacija.

Neke od prednosti Kotlina su:

- Interoperabilnost sa Javom: Kotlin se lako integrira sa postojećim Java kodom
- Sigurnost tipova: Kotlin je statički tipiziran jezik
- Kompaktna i izražajna sintaksa: Kotlin ima čistu i konciznu sintaksu
- Null safety: Kotlin eliminira greške vezane za null reference
- Jednostavna upotreba korutina: Kotlin olakšava asinkrono programiranje
- Lakše održavanje koda: Kotlin olakšava održavanje i restrukturiranje koda
- Brza kompilacija: Kotlin se brzo kompilira.
- Aktivna zajednica i podrška: Kotlin ima veliku i rastuću zajednicu programera.

### 3.1.1. JVM vs KMM

JVM (Java Virtual Machine) – **izabrano radno okruženje**:

- JVM je okruženje za izvršavanje Java i Kotlin koda.
- Kotlin se može kompilirati u bytecode koji se izvršava na JVM-u.
- JVM se najčešće koristi za izradu serverskih aplikacija, desktop aplikacija i Android aplikacija.

- Biblioteke i okruženja kao što su Spring i Android SDK podržavaju JVM.

KMM (Kotlin Multiplatform Mobile):

- KMM je platforma koja omogućava dijeljenje koda između različitih platformi poput Android-a, iOS-a, i web-a.
- KMM omogućava pisanje zajedničkog Kotlin koda koji se može koristiti na više platformi.
- KMM koristi specifične plugin-ove za kompilaciju koda za svaku ciljanu platformu.
- Za KMM se često koriste biblioteke poput Kotlinx.serialization za razmjenu podataka između različitih platformi.
- KMM se uglavnom koristi za razvoj mobilnih aplikacija koje trebaju dijeliti poslovnu logiku između Android i iOS platformi.

## 3.2. Android Studio – integrirano razvojno okruženje

Android Studio je integrirano razvojno okruženje (IDE) specijalizirano za razvoj Android aplikacija. Razvijen od strane Google-a, Android Studio pruža programerima sve alate potrebne za izradu visokokvalitetnih mobilnih aplikacija za Android platformu. Ovo napredno IDE omogućava programerima da lako upravljaju projektima, pišu kod, testiraju aplikacije, dizajniraju korisničko sučelje, i optimiziraju performanse svojih aplikacija. Sa integriranim emulatorom za Android uređaje, podrškom za različite programerske jezike kao što su Java i Kotlin, te bogatim setom alata za razvoj, Android Studio je ključni alat za svakog programera koji se bavi Android razvojem.

### 3.2.1. Prednosti

- Specijaliziran za Android: Android Studio je specifično dizajniran za razvoj Android aplikacija, pružajući bogat set alata i integracija koji olakšavaju izradu aplikacija za ovu platformu.
- Integrirani Emulator: Android Studio dolazi s integriranim Android Emulatorom koji omogućava programerima testiranje aplikacija na različitim uređajima i Android verzijama, što je ključno za osiguranje kompatibilnosti i ispravnosti aplikacija.

- Podrška za Kotlin: Android Studio ima punu podršku za Kotlin, što omogućava programerima da pišu aplikacije koristeći ovaj moderni programski jezik koji nudi mnoge prednosti u odnosu na tradicionalni Java jezik.
- Google Integracije: Budući da je razvijen od strane Google-a, Android Studio ima duboku integraciju s Google uslugama i alatima, što olakšava razvoj aplikacija koje koriste Google API-jeve i tehnologije poput Firebase-a.
- Bogat Ekosistem Pluginova: Android Studio podržava širok spektar pluginova koji dodaju dodatne funkcionalnosti i alate za razvoj, prilagođavajući se potrebama programera i olakšavajući rad na projektima.

### 3.2.2. Mane

- Resursno Intenzivan: Android Studio može zahtijevati značajne hardverske resurse na računalu, posebno za veće projekte, što može usporiti rad na starijim ili manje moćnim računalima.
- Krivulja učenja: Za programere koji nisu upoznati s IDE-om, učenje svih funkcionalnosti i alata Android Studija može biti izazovno, posebno za početnike u Android razvoju.
- Povremeni nedostatci: Kao i svaki softver, Android Studio nije imun na bugove i povremeno može doći do problema s performansama ili stabilnošću, što može uzrokovati frustracije programerima.

## 3.3. Express – REST API

Express.js je minimalisticki web razvojni okvir za Node.js koji omogućava izgradnju brzih i skalabilnih web aplikacija i API-ja. Jedna od glavnih prednosti Express-a je njegova jednostavnost korištenja i fleksibilnost, što ga čini popularnim izborom među programerima. Express pruža bogat set alata za usmjerivanje, srednjih slojeva, te podršku za obradu HTTP zahtjeva i odgovora. Ovaj razvojni okvir omogućava programerima da brzo razvijaju aplikacije koristeći koncepte kao što su usmjerivanje, dinamičke rute, upravljanje sesijama i kolačićima-ima, te integraciju sa različitim predlošcim. Osim toga, Express je veoma prilagodljiv i omogućava integraciju sa različitim modulima i bibliotekama kako bi se

zadovoljile specifične potrebe projekta. S obzirom na sve ove karakteristike, Express.js je postao standardni alat u svijetu Node.js-a za razvoj web aplikacija i API-ja. Razlog odabira Express.js-a kao razvojnog okvira za diplomski rad je točno to zbog čega je i stvoren, brza implementacija i iteracija, jednostavnost korištenja i lakoća postavljanja na produkcijska okruženja.

## 3.4. Git – praćenje koda

Git je distribuirani sistem za kontrolu verzija koji je postao nezaobilazan alat u svijetu razvoja softvera. Git omogućava programerima praćenje promjena u kodu, suradnju na projektima, upravljanje verzijama i praćenje povijesti izmjena. Jedna od ključnih prednosti Git-a je što je distribuirani sistem, što znači da svaki korisnik ima punu kopiju cjelokupne povijesti projekta, što olakšava rad u situacijama kada nema pristupa centralnom serveru. Git koristi koncepte kao što su commit-ovi, grane, merge-ovi i rebase-ovi kako bi omogućio fleksibilno i efikasno upravljanje kodom. Također, Git pruža podršku za razne servise kao što su GitHub, GitLab i Bitbucket, koji dodatno olakšavaju timsku suradnju, pregled kodova i upravljanje projektima. Sa svojom fleksibilnošću, brzinom i moćnim setom alata, Git je postao standardni alat za kontrolu verzija u industriji softverskog razvoja.

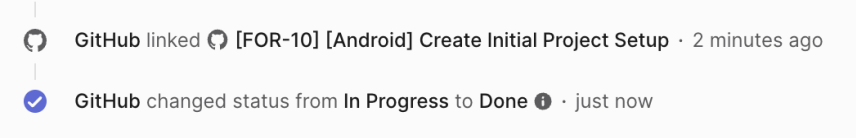
Slika 14. Prikaz Git grana – Android Studio

### 3.4.1. GitHub

GitHub je vodeća platforma za upravljanje softverskim projektima koristeći Git kao sistem za kontrolu verzija. Ona pruža web bazirane hosting usluge za repozitorije koda, alate za kolaboraciju, upravljanje projektima, pregled koda i više. Jedna od glavnih prednosti GitHuba je njegova ogromna zajednica korisnika i projekata, što olakšava pronalaženje

otvorenih izvornih kodova, učenje od drugih programera i doprinos globalnoj bazi znanja. Također, GitHub nudi bogat set alata za upravljanje projektima kao što su „issues“, „pull requests“, „actions“ i „projects“, što olakšava organizaciju i praćenje napretka projekata. Osim toga, GitHub je popularan među programerima zbog svoje integracije sa raznim servisima i alatima, kao što su CI/CD servisi, razvojno okruženje (IDE) i druge aplikacije za podršku softverskom razvoju. Ukratko, GitHub se ističe kao vodeća platforma za razvoj softvera zbog svoje moćne zajednice, naprednih alata za upravljanje projektima i širokog spektra integracija koje pruža.

GitHub je iz točno tih razloga izabran kao softver za upravljanje ForMat projektom, uz globalnu popularnost dolaze i lakoća implementacije sa drugim servisima. Tako da je GitHub jednostavno uklopiv u gotovo sve ostale korištene softvere i sisteme. Od korištenih u ovom projektu to su Linear, Android Studio i Slack (koji nije korišten ali je objašnjena njegova primjena u većim projektima)




Slika 15. Integracija GitHub-a s Linear-om

### 3.4.2. Git Procesi (Git Flows)

Git procesi su model upravljanja radnim tokovima koji se koristi prilikom razvoja softvera uz pomoć Git-a kao sistema za kontrolu verzija. Ovaj model definiše jasne smjernice o tome kako se organizuju grane u Git repozitoriju, kako se vrše merge-i između grana te kako se upravlja procesom razvoja softvera.

1. GitFlow: Ovaj model uključuje master granu za stabilne verzije, develop granu za aktivni razvoj, feature grane za nove funkcionalnosti, release grane za pripremu izdanja te hotfix grane za hitne popravke.
2. GitHub Flow (*odabran proces*): Ovaj model se fokusira na kontinuirano isporučivanje softvera. Uključuje samo jednu granu, obično master, i koristi pull request-ove za dodavanje novih funkcionalnosti, što olakšava kontinuiranu integraciju i isporuku.
3. GitLab Flow: Ovaj model je sličan GitFlow-u, ali se više usredotočuje na automatizaciju i integraciju kontinuirane isporuke. Uključuje master granu za stabilne verzije, a svaka

nova funkcionalnost razvija se u zasebnim granama i spoji u master putem pull request-ova.

4. Trunk Based Development: Ovaj model se temelji na održavanju jedne glavne (trunk) grane u Git repozitoriju, koja predstavlja "source of truth" za cjelokupan kod. U ovom modelu, sve promjene se direktno integriraju u glavnu granu, što potiče česte merge-eve i brzu iteraciju. Ovaj pristup promovira kontinuiranu integraciju i kontinuirano isporučivanje, te je pogodan za agilne timove i projekte s visokom stopom promjena.

## 3.5. Retrofit – REST API komunikacija

Retrofit je popularna biblioteka za rad s HTTP zahtjevima u Android razvoju, koja pruža jednostavan i efikasan način za komunikaciju s web servisima. Ova biblioteka omogućava programerima da definiraju API sučelje koristeći Kotlin sučelja, što olakšava kreiranje i održavanje HTTP zahtjeva. S Retrofit-om, programeri mogu lako izvršavati GET, POST, PUT, DELETE i druge vrste zahtjeva, manipulirati JSON podacima i rukovati odgovorima servera. Osim toga, Retrofit pruža podršku za razne napredne funkcionalnosti kao što su upravljanje sesijama, upravljanje prikazom grešaka, logging, cache-iranje i autentifikacija, čime olakšava razvoj kompleksnih aplikacija koje zahtijevaju komunikaciju s web servisima. Zahvaljujući svojoj jednostavnoj integraciji, visokoj pouzdanosti i performansama, Retrofit je postao ključni alat u Android razvoju za obradu mrežnih zahtjeva i integraciju s raznim API-ima.

Retrofit i OkHttp su često korišteni zajedno u Android razvoju kao komplementarne biblioteke za upravljanje HTTP zahtjevima. Dok Retrofit služi kao biblioteka visoke razine za definiranje API interfejsa i izvođenje HTTP zahtjeva, OkHttp je niži nivo biblioteke koja pruža podršku za izvršavanje samih zahtjeva. Jedna od ključnih prednosti povezivanja ove dvije biblioteke je što Retrofit koristi OkHttp kao svoj zadani HTTP klijent.

## 3.6. Postman

Postman je popularan alat za razvoj API-ja koji pruža pregledno i intuitivno korisničko sučelje za testiranje, dokumentiranje i upravljanje API-jima. Sa svojim jednostavnim

sučeljem, Postman omogućava programerima da jednostavno izvršavaju HTTP zahtjeve, pregledaju odgovore servera, organiziraju zahtjeve u kolekcije, dijele testne skripte s timom i automatski generiraju dokumentaciju API-ja. Također, Postman nudi napredne funkcionalnosti poput automatskog testiranja, varijabli za dinamičko generiranje podataka, mogućnosti izvoza i uvoza kolekcija, kao i integracije s drugim alatima za razvoj softvera. Zahvaljujući svojoj jednostavnosti korištenja i bogatim mogućnostima, Postman je postao nezamjenjiv alat u svakodnevnom radu programera za razvoj i testiranje API-ja.

## 3.7. Firebase – analtika, logiranje i konfiguracija

Firebase je platforma za razvoj mobilnih i web aplikacija koju je razvio Google, a koja pruža razne alate i usluge za olakšavanje razvoja, testiranja, analize i nadgledanja aplikacija. Njegove mogućnosti se protežu od autentifikacije korisnika i upravljanja bazama podataka do hostinga web stranica i upotrebe cloud funkcija. Firebase je posebno poznat po svom jednostavnom API-ju, skalabilnosti i brzini integracije s mobilnim aplikacijama.

Jedna od ključnih mogućnosti Firebase platforme je Firebase Authentication, koja omogućava autentifikaciju korisnika putem raznih metoda, uključujući e-mail i lozinku, Google, Facebook, Twitter i druge. Firebase Realtime Database i Firebase Firestore pružaju real-time baze podataka u oblaku, što omogućava programerima da lako sinkroniziraju podatke između korisnika i uređaja. Firebase Cloud Messaging (FCM) omogućava slanje push obavijesti korisnicima, dok Firebase Hosting pruža brz i siguran hosting za web stranice i web aplikacije.

Firebase također nudi alate za analizu performansi aplikacija, testiranje, praćenje grešaka (crash reporting) i praćenje korisničke analitike.

Ukratko, Firebase je sveobuhvatna platforma koja pruža širok spektar alata i usluga za razvoj modernih mobilnih i web aplikacija, olakšavajući programerima da se fokusiraju na razvoj funkcionalnosti aplikacije umjesto na infrastrukturu.

### 3.7.1. Analitika

Firebase Analytics je usluga za analizu korisničkog ponašanja i performansi aplikacija koja je dio Firebase platforme. Omogućava programerima da prate ključne metrike vezane uz korištenje njihove aplikacije, kao što su broj korisnika, trajanje sesija, konverzije, retencija korisnika i mnoge druge. Firebase Analytics koristi podatke o korisničkom ponašanju kako bi pružio uvid u to kako korisnici komuniciraju s aplikacijom, što pomaže programerima da bolje razumiju svoje korisnike i optimiziraju korisničko iskustvo. Osim toga, Firebase Analytics integrira se sa drugim alatima Firebase platforme, poput Firebase Remote Config-a i Firebase Predictions-a, što omogućava personalizaciju aplikacije i pružanje relevantnog sadržaja korisnicima na temelju njihovih interakcija s aplikacijom. Firebase Analytics pruža programerima dragocjene uvide koji im pomažu da donose informirane odluke o razvoju i marketinškim strategijama svojih aplikacija.

### 3.7.2. Udaljena konfiguracija

Firebase Remote Config je usluga koja omogućava programerima da dinamički konfiguriraju svoje aplikacije u stvarnom vremenu, bez potrebe za ažuriranjem aplikacije putem app store-a ili posluživanja novih verzija. Koristeći Firebase Remote Config, programeri mogu definirati različite varijable, vrijednosti i uvjete koji se primjenjuju na korisničke uređaje u stvarnom vremenu. To omogućava prilagodbu ponašanja aplikacije na temelju različitih faktora poput lokacije korisnika, jezika, uređaja i drugih korisničkih atributa. Na primjer, programer može dinamički promijeniti izgled aplikacije, prikazivati različite promotivne sadržaje ili prilagoditi funkcionalnosti na temelju promjenjivih uvjeta. Firebase Remote Config integrira se s Firebase Analytics-om i Firebase Predictions-om kako bi omogućio personalizirane prilagodbe i poboljšao korisničko iskustvo. Ova usluga pruža programerima moćan alat za optimizaciju i prilagodbu aplikacija bez potrebe za ponovnim distribuiranjem aplikacija ili izmjene koda.

### 3.7.3. Notifikacije

Firebase push notifikacije omogućavaju programerima da šalju obavijesti korisnicima njihovih mobilnih aplikacija, bez obzira na to jesu li aplikacija trenutno otvorena ili ne. Ova

funkcionalnost omogućava programerima da angažiraju korisnike, obavijeste ih o novim sadržajima, ažuriranjima ili događajima te ih ponovno uključe u korištenje aplikacije. Firebase pruža alate za slanje ciljanih push notifikacija na temelju različitih segmenata korisnika, kao što su lokacija, uređaj, korisničke aktivnosti i preferencije. Također, Firebase Cloud Messaging (FCM) omogućava programerima da automatiziraju slanje push obavijesti putem jednostavnog API-ja ili integracijom s drugim Firebase uslugama poput Firebase Analytics-a. Firebase push notifikacije su moćan alat za poboljšanje angažmana korisnika i povećanje zadržavanja korisnika u mobilnim aplikacijama.

## 3.8. Injekcija ovisnosti – dependency injection

Dependency injection (DI) je koncept u softverskom inženjeringu koji se koristi za poboljšanje modularnosti, fleksibilnosti i testiranja softverskih aplikacija. Osnovna ideja DI-a je da se objekti u aplikaciji ne trebaju sami brinuti o instanciranju svojih ovisnosti, već će im te ovisnosti biti dostavljene izvana. Umjesto da objekti direktno stvaraju instance svojih ovisnosti, one im se ubacuju ili 'ubrizgavaju' (injektiraju) izvanjskim sustavom, što omogućava lakše upravljanje ovisnostima i njihovu zamjenu ili izmjenu bez mijenjanja koda koji koristi te ovisnosti.

Dependency injection se često postiže korištenjem injekcije putem konstruktora, metoda ili svojstava. Ovo omogućava programerima da jasno definiraju ovisnosti u izvornom kodu, olakšavajući razumijevanje i održavanje aplikacije. Također, DI olakšava testiranje aplikacija jer omogućava programerima da zamijene stvarne implementacije ovisnosti sa lažnim (mock) ili ispravnim (stub) objektima za potrebe testiranja.

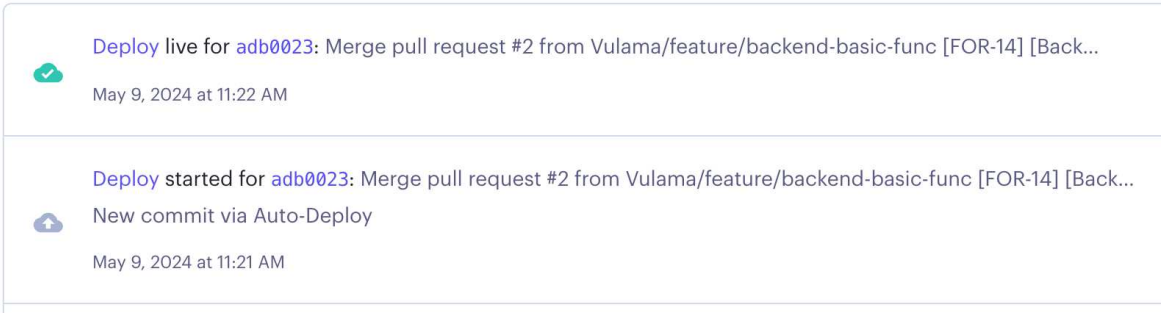
U konačnici, dependency injection promiče princip 'odvajanja brige' (separation of concerns) jer odvaja kôd koji stvara objekte od kôda koji ih koristi, čime se povećava modularnost, ponovna upotrebljivost i skalabilnost aplikacija. Ovaj koncept je ključan u modernom softverskom dizajnu i implementaciji, te se često koristi u kombinaciji s drugim principima dizajna poput obrnutog upravljanja (Inversion of Control) i jednostavne odgovornosti (Single Responsibility Principle).

## 3.9. CI/CD – kontinuirana integracija i kontinuirana dostava

CI/CD (Continuous Integration/Continuous Delivery ili Continuous Deployment) je praksa u softverskom inženjeringu koja omogućava automatsko integriranje, testiranje i isporuku kodnih promjena u aplikaciju ili sustav na brz, pouzdan i učinkovit način. Continuous Integration se odnosi na proces integracije novih kodnih promjena u glavnu (master) granu repozitorija softvera, gdje se automatski pokreću testovi kako bi se osigurala funkcionalna ispravnost aplikacije. Continuous Delivery/Delivery se odnosi na automatizirano testiranje i isporuku aplikacije u produkcijsko okruženje nakon uspješne integracije, čime se osigurava brza i pouzdana isporuka novih značajki ili popravaka grešaka. Continuous Deployment ide korak dalje, automatski isporučujući kodne promjene u produkcijsko okruženje bez ručnog odobrenja, čime se postiže još veća brzina isporuke i reakcije na promjene. CI/CD prakse omogućavaju programerima da brzo iteriraju, smanjuju vrijeme između kodne promjene i isporuke u produkcijsko okruženje, te povećavaju kvalitetu, stabilnost i pouzdanost softverskih aplikacija.

U sklopu diplomskog rada CD će biti implementiran pomoću servisa Render koji se koristi za posluživanje samog REST API-a i baze podataka. Render kao jednu od svojih mogućnosti nudi integraciju s GitHub-om. GitHub komunicira s Render-om kada se na odabranu granu dodaje novi Commit. U tom trenutku Render pokreće ponovno build fazu u kojoj se nalazi dohvaćanje potrebnih biblioteka, pokretanje migracije baze podataka a mogu biti uključeni i testovi.

CI se u sklopu diplomskog rada neće koristit jer se radi o projektu s jednim članom te ne postoji dovoljno velik broj promjena na korištenim repozitorijima da bi se opravdalo njegovo korištenje.




Slika 16. Prikaz automatskog posluživanja na servisu Render

## 3.10. Gradle

Gradle je moderni alat za automatizaciju izgradnje softvera koji se široko koristi u razvoju Android aplikacija, ali i drugih vrsta projekata. On omogućava programerima definiranje, konfiguriranje i izgradnju projekata na efikasan i fleksibilan način. Glavne značajke i koncepti Gradlea uključuju:

- **Deklarativna sintaksa**: Gradle koristi deklarativni pristup za definiranje izgradnje projekta pomoću Groovy ili Kotlin DSL (Domain Specific Language). Ova sintaksa omogućava jasno definiranje ovisnosti, taskova, i drugih elemenata izgradnje.
- **Fleksibilnost**: Gradle je izuzetno fleksibilan i omogućava prilagodbu izgradnje projekta prema specifičnim potrebama. Programeri mogu definirati vlastite taskove, pluginove i skripte kako bi automatizirali različite zadatke u procesu izgradnje.
- **Ovisnosti i upravljanje verzijama**: Gradle olakšava upravljanje ovisnostima i verzijama biblioteka pomoću Maven ili Ivy repozitorija. Programeri mogu jednostavno dodavati, ažurirati i uklanjati ovisnosti iz projekta putem jednostavne sintakse.
- **Paralelna izgradnja**: Gradle podržava paralelnu izgradnju projekata, što rezultira bržim vremenom izgradnje i povećava produktivnost razvojnog tima.
- **Pluginovi i ekstenzije**: Gradle dolazi s bogatim setom pluginova koji omogućavaju integraciju s različitim alatima i tehnologijama kao što su Android, Java, Kotlin, TestNG, i mnogi drugi. Osim toga, programeri mogu kreirati vlastite pluginove i ekstenzije kako bi proširili funkcionalnost Gradlea prema svojim potrebama.

U konačnici, Gradle je snažan alat koji olakšava automatizaciju i upravljanje izgradnjom projekata. Njegova fleksibilnost, performanse i podrška za različite tehnologije čine ga preferiranim alatom među programerima za razvoj raznovrsnih softverskih projekata.

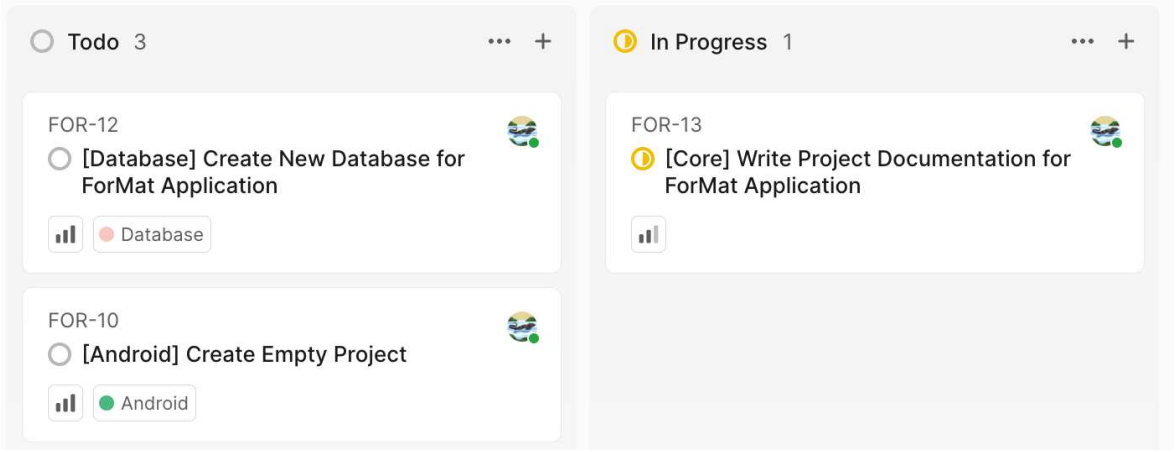
# 4. Tehnologije većih projekata – skalabilnost

Povećanje skalabilnosti u projektima često zahtijeva korištenje odgovarajućih tehnologija i arhitektonskih pristupa koji omogućavaju rast i prilagodljivost sustava kako bi se nosili s povećanim opterećenjem.

Slack i Docker iz nastavka nisu bili korišteni tijekom ovog diplomskog rada pošto za to nije bilo potrebe. Treba i naglasiti kako inženjerstvo zbog svrhe inženjerstva nije uvijek ispravna stvar za odabrati te je u dosta slučajeva ispravnije koristiti jednostavniju opciju.

## 4.1. Linear – upravljanje zadacima

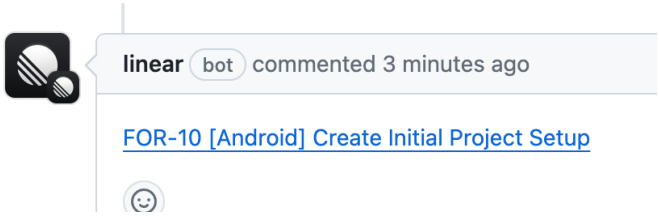
Linear je alat za upravljanje zadacima (taskovima) i projektima koji omogućava timovima da organiziraju, prate i surađuju na radu. Glavni naglasak Linear-a je na jednostavnosti korištenja, fleksibilnosti i efikasnosti. Korisnicima omogućava kreiranje zadataka, dodjeljivanje zadataka članovima tima, postavljanje rokova i praćenje napretka. Također, pruža mogućnost organiziranja zadataka u projekte i rad u okviru tih projekata.




Slika 17. Linear - upravljanje zadacima

Linear nudi pregledne tablice za praćenje radnih zadataka, mogućnost komentiranja i diskusije o zadacima, kao i integraciju s drugim alatima i servisima kao što su GitHub, Slack i druge. Ovaj alat je posebno popularan među softverskim timovima zbog svoje

jednostavnosti i efikasnosti u upravljanju projektima. Jednostavna integracija u samo nekoliko klikova veliki je benefit za sve veličine timova i sve vrste okruženja.




Slika 18. Integracija Linear-a s GitHub-om

## 4.2. Slack – komunikacije s kolegama

Slack je platforma za komunikaciju koja omogućava timovima da komuniciraju, surađuju i dijele informacije na jednom mjestu. Glavni fokus Slack-a je na organizaciji razgovora u kanale, što omogućava korisnicima da grupiraju razgovore prema temama, projektima ili timovima. Osim toga, Slack pruža mogućnost direktnih poruka, grupnih razgovora, dijeljenja datoteka, integracija s drugim alatima i servisima, kao i mogućnost vođenja video i audio poziva.

Slack olakšava komunikaciju unutar tima, ali i sa vanjskim suradnicima putem poziva u kanale ili dijeljenja posebnih kanala za suradnju. Ovaj alat je popularan među različitim organizacijama i timovima zbog svoje intuitivne upotrebe, mogućnosti organizacije i bogatih mogućnosti komunikacije.

## 4.3. Docker – kontejnerizacija

Docker je popularna platforma za kontejnerizaciju aplikacija koja omogućava programerima da pakiraju, distribuiraju i pokreću aplikacije s minimalnim resursima i bez obzira na okruženje u kojem se izvršavaju. Kontejneri stvaraju izolirano okruženje koje sadrži sve potrebne dijelove aplikacije, uključujući kod, biblioteke i druge zavisnosti, čime se osigurava dosljedno i pouzdano izvršavanje aplikacije bez obzira na razlike u okruženju.

Docker pruža jednostavan API i alate za upravljanje kontejnerima, omogućavajući programerima da brzo pokrenu, skaliraju i upravljaju aplikacijama u kontejneriziranom

okruženju. Ova tehnologija je postala ključni alat u modernom razvoju softvera zbog svoje fleksibilnosti, pouzdanosti i skalabilnosti. Jedna od korisnih primjena bi bila pakiranje REST API-a u kontejner koji se lako može distribuirati mobilnim i web razvojnim programerima u svrhu implementiranja svojih dijelova aplikacije lokalno.

# 5. Implementacijski mehanizmi

Lokalizacija, AB testiranje, razdvajanje razvojnih, i slični mehanizmi, ključni su u modernom softverskom razvoju. Lokalizacija omogućava prilagodbu aplikacija različitim jezicima i regionalnim postavkama. AB testiranje omogućava usporedbu performansi različitih verzija, dok razdvajanje okruženja omogućava testiranje aplikacije prije puštanja u produkciju, smanjujući rizik od grešaka i osiguravajući kvalitetu. Ovi mehanizmi pružaju programerima alate i pristupe za brzu, sigurnu i pouzdanu isporuku aplikacija.

## 5.1. Razdvajanje razvojnih okruženja

Razdvajanje razvojnih okruženja na staging, produkciju i potencijalno i develop okruženje je ključno u softverskom razvoju radi efikasnog upravljanja ciklusom razvoja, testiranja i puštanja aplikacija u produkciju.

- *Develop okruženje*: To je okruženje u kojem programeri rade na razvoju novih značajki ili popravcima grešaka. Ovdje se vrši integracija novog koda i testiranje promjena. Develop okruženje često simulira produkcijsko okruženje koliko je to moguće, ali može biti manje stabilno i imati manje resursa od produkcijskog okruženja.
- *Staging okruženje*: Staging okruženje je kopija produkcijskog okruženja u kojoj se testiraju promjene prije nego što se uvedu u produkcijsko okruženje. Ovdje se simulira stvarno korištenje aplikacije u stvarnom okruženju, ali bez izravnog utjecaja na korisnike. Staging okruženje omogućava programerima da provjere da li su promjene ispravno implementirane i da li aplikacija radi kako treba prije puštanja u produkcijsko okruženje.
- *Produkcijsko okruženje*: To je okruženje u kojem se krajnje korisnicima pruža pristup aplikaciji. Produkcijsko okruženje mora biti stabilno, pouzdano i visokih performansi. Ovdje se izvršavaju sve operacije u stvarnom vremenu, stoga je važno da aplikacija bude stabilna i da nema grešaka koje bi mogle utjecati na korisničko iskustvo.

## 5.2. Lokalizacije sadržaja

Lokalizacija sadržaja je ključna za prilagodbu aplikacija različitim jezicima, kulturama i regionalnim postavkama kako bi se omogućilo bolje korisničko iskustvo i širenje na globalno tržište. Kada aplikacija nije lokalizirana, korisnici iz različitih dijelova svijeta mogu naići na prepreke u razumijevanju i korištenju aplikacije, što može dovesti do gubitka korisnika i poslovnih prilika.

Postoje dva glavna pristupa lokalizaciji sadržaja:

1. *Lokalno lokaliziranje sadržaja*: Ovaj pristup uključuje ručno dodavanje lokaliziranih resursa (tekst, slike, audio datoteke, itd.) u izvorni kod aplikacije. Programeri obično koriste alate kao što su stringovi resursa ili posebne biblioteke za lokalizaciju kako bi implementirali lokalizirane verzije sadržaja za svaki podržani jezik. Ovaj pristup može biti prilično zamoran i zahtijeva promjene u izvornom kodu za svaku lokaliziranu verziju aplikacije.
2. Remote lokaliziranje sadržaja: Ovaj pristup koristi vanjsku uslugu, poput Lokalise-a, za upravljanje lokaliziranim sadržajem. Umjesto ručnog dodavanja lokaliziranih resursa u izvorni kod, programeri integriraju uslugu za lokalizaciju koja omogućava upravljanje lokaliziranim resursima putem sučelja za korisnike ili API-ja. To omogućava brže i fleksibilnije upravljanje lokaliziranim sadržajem, uključujući dodavanje novih jezika, ažuriranje prijevoda i koordinaciju s prevodiocima. Osim toga, ovo omogućava i testiranje lokaliziranih verzija aplikacije u stvarnom vremenu prije puštanja u produkciju.

## 5.3. AB testiranje i konfiguracijske zastavice

AB testiranje (A/B testing) je metodologija koja se koristi za usporedbu performansi dviju ili više varijacija nekog elementa u aplikaciji ili web stranici. Osnovna ideja je podijeliti korisnike na različite grupe, prikazati im različite varijacije (A, B, C, itd.) i zatim analizirati kako se korisnici ponašaju u svakoj varijaciji.

AB testiranje omogućava programerima da donose informirane odluke o dizajnu, funkcionalnostima ili marketinškim strategijama na temelju stvarnih podataka o korisničkom ponašanju.

Konfiguracijske zastavice su tehnika koja omogućava programerima da kontroliraju aktivaciju ili isključivanje određenih funkcionalnosti u aplikaciji. Umjesto da se nova funkcionalnost aktivira ili deaktivira direktno u kodu aplikacije, programeri koriste konfiguracijske zastavice koji se mogu kontrolirati izvan aplikacije, često putem korisničkog sučelja ili API-ja. Ovo omogućava postupno uvođenje novih značajki, testiranje u stvarnom okruženju, kao i brzu reakciju na promjene ili hitne situacije. Također, konfiguracijske zastavice omogućavaju programerima da prilagode iskustvo korisnika na temelju njihovih atributa ili ponašanja, što može biti korisno za personalizaciju aplikacije ili eksperimentiranje s novim idejama.

# 6. Arhitektura

Arhitektura aplikacije igra ključnu ulogu u organizaciji i razvoju Android aplikacija. Odabir odgovarajuće arhitekture može značajno utjecati na performanse, održivost i skalabilnost aplikacije. U ovom poglavlju istražit ćemo nekoliko klasičnih i modernih arhitektura koje se koriste za razvoj Android aplikacija.

1. Model-View-Controller (MVC)

   MVC arhitektura je jedna od najstarijih i najpoznatijih arhitektura u softverskom inženjeringu. U kontekstu Android aplikacija, aktivnosti (Activities) ili fragmenti (Fragments) djeluju kao kontroleri koji upravljaju poslovnim logikama i korisničkim interakcijama. Pogledi (Views) predstavljaju korisničko sučelje, dok modeli (Models) sadrže poslovne podatke i logiku. Iako MVC pruža osnovni način organizacije koda, često može dovesti do gustog i teško održivog koda.

2. Model-View-Presenter (MVP)

   MVP arhitektura je evolucija MVC-a koja pokušava riješiti neke od nedostataka prethodne arhitekture. U MVP arhitekturi, aktivnosti ili fragmenti djeluju kao pogledi koji su pasivni i odgovorni samo za prikazivanje podataka korisniku. Presenteri sadrže poslovnu logiku i upravljaju interakcijom između pogleda i modela. Ovo često rezultira boljom razdjelom odgovornosti i lakšim testiranjem.

3. **Model-View-ViewModel (MVVM)**

   MVVM arhitektura postala je popularna u svijetu Android razvoja, posebno od pojave Android Architecture Components-a. U MVVM arhitekturi, ViewModeli djeluju kao posrednici između aktivnosti ili fragmenata (View-a) i podataka iz izvora poput baze podataka ili mrežnih poziva (Model). ViewModeli čuvaju stanje aplikacije i omogućavaju jednostavno upravljanje prikazanim podacima. Ova arhitektura pruža dobru razdjelu odgovornosti i podržava jednostavno testiranje.

Uz već navedene arhitekture, vrijedi istražiti i nekoliko drugih pristupa koji su postali popularni u razvoju Android aplikacija. Među njima su Onion Architecture, Domain-Driven Design (DDD) i Clean Architecture.

**Onion Architecture** je arhitekturni pristup koji promiče jasnu razdjelu odgovornosti unutar aplikacije. Arhitektura je organizirana oko jezgre (core) koja sadrži poslovnu logiku i domensku logiku aplikacije. Vanjski slojevi (npr. sloj sučelja, sloj infrastrukture) okružuju jezgru i ovise o njoj. Ovaj pristup omogućava bolju kontrolu nad poslovnim procesima aplikacije i olakšava zamjenjivanje vanjskih komponenti bez utjecaja na jezgru sustava.

**Domain-Driven Design** (DDD) je metodologija koja se fokusira na razumijevanje i modeliranje poslovnih domena kako bi se razvile bolje organizirane i fleksibilne aplikacije. U DDD-u, poslovna domena se modelira pomoću entiteta, vrijednosnih objekata, agregata i repozitorija. Razdvajanje poslovnih funkcionalnosti u bogato modeliranu poslovnu domenu omogućava bolje razumijevanje i upravljanje poslovnim procesima, što rezultira višom kvalitetom i efikasnošću aplikacije.

**Clean Architecture** je arhitekturni pristup koji promiče jasnu razdjelu odgovornosti unutar aplikacije i neovisnost različitih slojeva. U Clean Architecture, aplikacija je podijeljena na nekoliko slojeva: domenski sloj (Domain Layer), sloj upotrebe (Use Case Layer), sloj sučelja (Interface Layer) i sloj infrastrukture (Infrastructure Layer). Ova razdjela omogućava lako zamjenjivanje ili nadopunjavanje pojedinih dijelova aplikacije bez utjecaja na ostatak sustava, što rezultira skalabilnim, testiranim i održivim aplikacijama.

## 6.1. Odabrana arhitektura

U ovom projektu, odabrali smo kombinaciju MVVM arhitekture, Clean Architecture i Domain-Driven Design (DDD) pristupa kako bismo osigurali visoku kvalitetu, skalabilnost i održivost Android aplikacije.

MVVM arhitektura pruža jasnu razdjelu odgovornosti unutar aplikacije, s naglaskom na odvajanju korisničkog sučelja (View) od poslovnih modela (Model). ViewModel sloj djeluje kao posrednik između pogleda i modela, omogućavajući jednostavno upravljanje prikazanim podacima i poslovnom logikom.

Clean Architecture osigurava dodatnu razinu modularnosti i neovisnosti unutar aplikacije. Podjela aplikacije na nekoliko slojeva (npr. Domain Layer, Use Case Layer, Interface Layer, Infrastructure Layer) omogućava lakše upravljanje složenim poslovnim procesima i zamjenjivanje vanjskih komponenti bez utjecaja na ostatak sustava.

DDD pristup dodatno poboljšava organizaciju i strukturu aplikacije fokusirajući se na razumijevanje i modeliranje poslovnih domena. Korištenje entiteta, vrijednosnih objekata, agregata i repozitorija omogućava bolju organizaciju poslovne logike i upravljanje složenim poslovnim procesima.

Kombinacija ovih arhitekturnih pristupa pruža temelj za izgradnju visokokvalitetne, skalabilne i održive Android aplikacije koja će zadovoljiti potrebe korisnika i poslovanja. Ključno je pažljivo proučiti zahtjeve projekta i odabrati arhitekturu koja najbolje odgovara specifičnim potrebama i ciljevima aplikacije.

## 6.2. Dijelovi odabrane arhitekture

*Model* - Reprezentira podatke i poslovnu logiku aplikacije

*View* - Prikazuje korisničko sučelje i reagira na korisničke interakcije

*ViewModel* - Posrednik između pogleda i modela、 obrađuje logiku prikaza i upravlja prikazanim podacima

*Vrijednosni objekti* - Objekti koji sadrže vrijednosti i definiraju poslovna pravila

*Use Case* - Definira upotrebu slučajeva aplikacije

*Repozitoriji* - Sučelja koja definiraju pristup podacima i omogućavaju manipulaciju entitetima

### 6.2.1. Model

Model u arhitekturi predstavlja podatke i poslovnu logiku aplikacije. To su objekti ili klase koji sadrže podatke i metode za manipulaciju tim podacima. Na primjer, ako razvijate aplikaciju za upravljanje zadacima, model bi mogao sadržavati razred Formula s atributima kao što su naziv formule, opis te matematički izraz

### 6.2.2. View

View je komponenta koja je odgovorna za prikaz korisničkog sučelja i reagiranje na korisničke interakcije. To može biti aktivnost (Activity), fragment (Fragment) ili neki drugi komponenta koja prikazuje korisnički sučelje. View ne bi trebao sadržavati poslovnu logiku, već samo prikazati podatke koje prima od ViewModela i reagirati na korisničke interakcije.

### 6.2.3. ViewModel

ViewModel je posrednik između pogleda (View) i modela. On obrađuje logiku prikaza i upravlja prikazanim podacima. ViewModel sadrži logiku koja je potrebna za pripremu podataka koje prikazuje View, kao i metode koje reagiraju na korisničke interakcije. ViewModel također često implementira mehanizme za komunikaciju s modelom, kao što su pozivi za dohvat podataka.

### 6.2.4. Vrijednosni objekti

Vrijednosni objekti su objekti koji sadrže vrijednosti i definiraju poslovna pravila. Oni predstavljaju koncepte ili entitete unutar poslovnog domena aplikacije. Na primjer, u aplikaciji za upravljanje formulama, vrijednosni objekt može biti Tip koji sadrži vrijednosti poput „obavezan“ (treba se naučiti) i „neobavezan“ (ne treba se naučiti)

### 6.2.5. Use Case

Use Case definira specifične scenarije ili slučajeve upotrebe aplikacije. To su interakcije koje korisnik može imati s aplikacijom kako bi postigao određeni cilj. Primjeri Use Case-ova u aplikaciji za upravljanje formulama mogu uključivati dodavanje nove formule, označavanje formule kao shvaćenu ili neshvaćenu.

### 6.2.6. Repozitoriji

Repozitoriji su sučelja koja definiraju pristup podacima i omogućavaju manipulaciju entitetima. Oni pružaju apstrakciju nad slojem podataka aplikacije i omogućavaju komunikaciju s vanjskim izvorima podataka, kao što su baze podataka, mrežni pozivi ili lokalno pohranjeni podaci. Repozitoriji omogućavaju jasnu razdjelu odgovornosti između slojeva aplikacije i olakšavaju testiranje i održavanje koda.

# 7. REST API – Express.js

Express.js je popularni web okvir za Node.js koji omogućava jednostavno i brzo kreiranje web aplikacija i API-ja. Express pruža robustan set alata i middleware-a koji olakšavaju obradu HTTP zahtjeva i odgovora, usmjerivanje, upravljanje sesijama, autentikaciju, i mnoge druge funkcionalnosti.

Kada se koristi za razvoj REST API-ja, Express nudi nekoliko prednosti. Prvo, njegova jednostavnost omogućava brz početak rada. Kreiranje novog API sučelja je jednostavno, jer se rute mogu definirati s nekoliko linija koda. Također, Express pruža fleksibilnost u definiranju ruta i obrade zahtjeva, što omogućava razvoj API-ja prema specifičnim zahtjevima projekta.

Express također podržava integraciju s različitim modulima i bibliotekama, što olakšava dodavanje dodatnih funkcionalnosti poput autentikacije, autorizacije, validacije podataka i sl. Također, moguće je jednostavno implementirati middleware za obradu zahtjeva i odgovora, što olakšava ponovno iskorištavanje koda i održavanje aplikacije.

Konačno, Express je izuzetno skalabilan i velikih performansi, što ga čini pogodnim za razvoj API-ja različitih veličina i složenosti. Bez obzira radi li se o malom projektu ili velikom sustavu s velikim prometom, Express može pružiti pouzdanu i efikasnu platformu za razvoj REST API-ja.

## 7.1. Baza podataka - PostgreSQL

PostgreSQL je moćni objektno-relacijski sustav upravljanja bazama podataka (ORDBMS) koji pruža robusno i pouzdano rješenje za pohranu, upravljanje i manipulaciju podacima. Ova besplatna, otvorena i izuzetno pouzdana baza podataka nudi širok raspon naprednih značajki, što je čini popularnim izborom za razne vrste aplikacija, od malih web aplikacija do velikih poslovnih sustava.

Jedna od ključnih karakteristika PostgreSQL-a je podrška za kompleksne SQL upite i napredne tipove podataka. PostgreSQL podržava puni skup SQL standarda, uključujući kompleksne upite, transakcije, podupite i prozorske funkcije, što omogućava razvoj složenih i sofisticiranih upita.

Osim toga, PostgreSQL podržava razne napredne značajke poput transakcijskog upravljanja, ACID (Atomicity, Consistency, Isolation, Durability) svojstava, indeksiranja, pune tekstualne pretrage, pohrane JSON i JSONB tipova podataka, te geografske informacijske sustave (GIS) za rad s prostornim podacima.

PostgreSQL također nudi visoku razinu sigurnosti i pouzdanosti. Ugrađene sigurnosne značajke uključuju autentikaciju, autorizaciju, enkripciju podataka, redovite sigurnosne zakrpe i druge mehanizme zaštite podataka.

Sposobnost skaliranja PostgreSQL-a također ga čini popularnim izborom za razvoj aplikacija koje zahtijevaju visoku dostupnost i performanse. PostgreSQL se može koristiti u jednostavnim jednokorisničkim aplikacijama, ali i u složenim višekorisničkim okruženjima s visokim opterećenjem.

Primjer jedne od tablica korištenih za izradu ovog projekta.

```
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    username VARCHAR(255) UNIQUE NOT NULL,
    passwordHash VARCHAR(255) NOT NULL,
    createdAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Kao svako polje tablice, sastoji se od imena atributa, tipa atributa te dodatnih svojstva poput UNIQUE, NOT NULL, DEFAULT…

Naravno svaki zapis u svim tablicama sadrži jedinstven identifikator/ključ. U tablicama to je označeno ključnom riječi PRIMARY KEY. To svojstvo atributa „id“ u gore navedenoj tablici nam osigurava da će svaki korisnik u tablici imat jedinstven identifikator prema kojem ćemo moći kasnije povezivati korisnika s pripadajućim grupama i matematičkim funkcijama.

Ako kao primjer uzmemo tablicu „formulaGroup“ možemo vidjeti kako svaka grupa formula ima definiranog svog vlasnika preko baš prije spomenutog ključa.

```
ownerId INTEGER REFERENCES users(id),
```

Ključna riječ REFERENCES nam govori kako će „ownerId“ biti strani ključ tablice te se preko te vrijednosti može povezati korisnik sa grupom formula koja mu pripada i obratno.

Jedan od primjera dohvaćanja podataka iz same baze podataka je najjednostavnija „select“ skripta, ova skripta ispisat će nam sve informacije za prvih 100 korisnika koji su upisani u tablicu „User“.

```
SELECT * FROM public."User"
ORDER BY id ASC LIMIT 100
```

Kao što možemo vidjeti i već za najjednostavniji oblik SQL upita potrebno je zadovoljiti dosta pravila i sintaksu samog SQL-a. Kako bismo to izbjegli na samom API-u koristili smo biblioteku koja nam omogućava jednostavniji rad sa samom bazom podataka. Biblioteka **Prisma** omogućava nam jednostavnu konstrukciju same baze podataka te nam uz to generira modele koje možemo veoma jednostavnom implementirati u ostatak JavaScript koda.

## 7.2. Prisma – JavaScript biblioteka

Prisma je moderna JavaScript biblioteka za upravljanje bazama podataka koja omogućava programerima jednostavno povezivanje s bazama podataka i manipulaciju podacima. Pruža intuitivan i deklarativan API za izradu upita, upravljanje modelima podataka i izvršavanje operacija baze podataka bez potrebe za pisanjem SQL-a.

Jedna od ključnih prednosti Prisme je njezina jednostavnost korištenja i integracije s modernim JavaScript okruženjima poput Node.js. Prisma podržava različite vrste baza podataka, uključujući PostgreSQL, MySQL i SQLite, pružajući programerima fleksibilnost u odabiru baze podataka prema njihovim potrebama.

Prisma također nudi napredne značajke poput migracija podataka, transakcija, automatskog generiranja koda i punog tipiziranja podataka pomoću TypeScripta. Ove značajke olakšavaju razvoj aplikacija i održavanje koda, dok istovremeno pružaju visoku razinu sigurnosti i pouzdanosti.

Tako umjesto prethodno spomenutog SQL koda za kreaciju baze imamo „schema.prisma“ datoteku u kojoj definiramo kako želimo da izgleda naša baza podataka te koji su odnosi među tablicama.

Tablica „User“ sad u .prisma datoteci izgleda ovako:

```
model User {
  id            Int           @id @default(autoincrement())
  username      String        @unique
  passwordHash  String
  createdAt     DateTime      @default(now())
  formulaGroups FormulaGroup[]
}
```

Možemo vidjeti kako postoje vidljive sličnosti sa SQL kodom, no postoji jedna bitna razlika a to je obrnuta referenca na „formulaGroups“. Prizma nam zbog velike korištenosti takvih referenca zahtjeva njihovu definiciju unaprijed radi generacije modela koji će se kasnije koristit u kodu.

## 7.3. Struktura Express servera

Express server podijeljen je kod nas u 4 glavna dijela. To su Prisma koja je objašnjena ranije, app.js koja služi kao glavna datoteka aplikacije, rute koje sadrže funkcionalnosti samih API sučelja i middleware-a koji služi kao posrednički sloj između samog API poziva i obrade informacija od strane servera.

### 7.3.1. Konfiguracijska datoteka - .env

Kao što je i ranije spomenuto, jedna od bitnih stvari kod rada na većim projektima je sama konfiguracija projekta. Za svrhe statične konfiguracije u manjim projektima koriste se uglavnom „.env“ takve datoteke zbog prefiksa „.“ nisu vidljivu u običnim preglednicima i terminalima bez specificiranja posebnih opcija. Pošto u našem projektu postoje razlike između produkcijske i razvojne aplikacije (poveznice nisu jednake, tajne se razlikuju…) ova datoteka nam bez promjene koda omogućuje različito ponašanje u različitim okruženjima. Recimo da na svojem razvojnom računalu imamo u konfiguracijskoj datoteci spremljenu adresu baze podataka koja pokazuje na našu lokalnu bazu podataka, na produkcijskom okruženju u konfiguracijskoj datoteci bi polje pod istim imenom imalo različitu vrijednost.

Testno okruženje:

```
PORT = 1234
USERNAME = vulama
```

Produkcijsko okruženje:

```
PORT = 5050
USERNAME = admin
```

Takav mehanizam nam omogućuje da se ne zanemarujemo uvjetima i posebnim okolnostima na različitim okruženjima, već ukoliko trebamo vrijednost koja se mijenja s okruženja na okruženje nju definiramo u konfiguracijskoj datoteci te ju kasnije iz nje koristimo bez razmišljanja što se u njoj nalazi.

### 7.3.2. Glavna datoteka – app.js

app.js je glavna datoteka koja definira i konfigurira Express aplikaciju. U ovoj datoteci se učitavaju potrebni moduli, definiraju se rute, middleware-i i postavljaju postavke servera poput porta na kojem će server slušati. Ova datoteka obično sadrži glavni kod za pokretanje i konfiguraciju Express aplikacije.

Glavna datoteka sastoji se od učitavanja modula potrebnih za pokretanje servera, definiranja skupine ruta i njihovo povezivanje, povezivanje posredničkog sloja te samo pokretanje server.

Na početku definiramo i učitavamo module koji su nam potrebni za pokretanje. To je svakako Express na kojem se bazira čitav API. Isto tako spremamo referencu na express pod nazivom app koji će se kasnije koristit za definiranje dodatnih stvari.

```
const express = require('express');
const app = express();
```

Nakon definiranja potrebnih modula potrebno je definirati iz kojih datoteka ćemo koristit rute samog API-a.

```
const authRoutes = require('./routes/auth');
const apiRoutes = require('./routes/api');
app.use('/auth', authRoutes);
app.use('/api', apiRoutes);
```

Vidimo kako nakon definicije lokacija rute navodimo express server ranije definiran da koristi te datoteke za određeni set ruta. Tako će sve rute iz skupine „authRoutes“ imati prefix „/auth“.

Nakon povezivanja ruta slijedi nam povezivanje posredničkog sloja.

```
app.use(express.json());
```

Za kraj nam dolazi pokretanje samog servera koji u tom trenutku postaje funkcionalan.

```
const PORT = process.env.PORT || 3000;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Vidimo kako nam je PORT varijabla dohvaćena iz konfiguracijske datoteke, no isto tako u slučaju da u konfiguracijskoj datoteci ime „PORT“ ne postoji odlučili smo se postaviti i backup opciju 3000.

### 7.3.3. Rute API-a – routes

Rute su dijelovi Express aplikacije koji definiraju kako aplikacija odgovara na određene HTTP zahtjeve. Svaka ruta može biti definirana za određenu stazu (endpoint) i HTTP metodu, poput GET, POST, PUT ili DELETE. Rute obično pozivaju odgovarajuće kontrolere koji obrađuju logiku poslovne aplikacije i generiraju odgovor. Na primjer, moguće je definirati rutu za registraciju korisnika na stazi '/register' koja će obrađivati POST zahtjeve.

Na početku svake datoteke koje sadrži rute nalazi se kod za povezivanje s potrebnim dijelovima aplikacije ili vanjskim bibliotekama poput Prisme

```
const express = require('express');
const router = express.Router();
const { PrismaClient } = require('@prisma/client');
const { authenticateToken } =
require('../middleware/authToken');
const { checkGroupOwnership } =
require('../middleware/checkOwnership');
```

Za demonstracijski primjer uzeli smo „/groups“ sučelje iz datoteke „api.js“ zbog njegove jednostavnosti, no isti principi vrijede i za ostale.

```
router.get('/groups', async (req, res) => {…});
```

Prva stvar koji trebamo definirati je tip poziva koji se obavlja nad ovom rutom, to je u ovom slu;aju GET poziv koji označava kako pozi neće mijenjati podatke na serveru/bazi podataka već samo služi za dohvaćanje, naon toga dolazi sama definicija rute sučelja, to je u ovom slučaju "/api/groups" (prefiks dolazi zbog definicije grupe ruta u app.js), nakon toga definiramo kako se radi o asinkronom pozivu (poziv preko interneta ne može biti instant obavljen), te da se u toj asinkronoj funkciji radi sa req (zahtjevom) i res (dogovorm).

Prva stvar unutar definirane lambda funkcije je try-catch blok, koji radi na principu da ukoliko se u try dijelu bloka dogodi pogreška izvršit će se catch blok koji će "uhvatiti" pogrešku te obaviti operaciju nad njom.

```
try {
  …
} catch (error) {
  console.error('Error retrieving groups:', error);
  res.status(500).json({ error: 'Internal server error' });
}
```

Kao što možemo vidjet u slučaju pogreške u try dijelu bloka klijentu će se vratit status kod 500 s porukom kako je došlo do pogreške.

```
const groups = await prisma.formulaGroup.findMany({
  include: {
    formulas: true,
  },
});
res.status(200).json(groups);
```

Try dio bloka sastoji se od korištenja prizme za dohvaćanje svih groupa koje se nalaze u bazi podataka, isto tako možemo vidjeti da osim samih grupa, dobivamo i formule u toj grupi. Ukoliko sve prođe kao što je planirano korisniku će se vratiti objekt "groups" u json obliku i status kod 200.

### 7.3.4. Posrednički sloj – middleware

Middleware su funkcije koje se izvršavaju između dolaznih HTTP zahtjeva i obrade tih zahtjeva. Mogu se koristiti za izvršavanje različitih funkcionalnosti, kao što su analiza tijela zahtjeva, provjera autentikacije, provjera dozvola pristupa, logiranje zahtjeva i sl. Middleware se obično definiraju pomoću app.use() metode ili se mogu specifično primijeniti na određene rute.

Za potrebe ForMat aplikacije posrednički sloj sastoji se od dvije funkcionalnosti. To su authenticateToken i checkOwnership, koje se koriste u kombinaciji.

Svrha autentikacijskog tokena je kako bih spriječila nasumičnog korisnika da mijenja ili briše postojeće grupe i/ili formule ukoliko on nije njihov vlasnik. Svaki korisnik dobije autentifikacijski token prilikom logiranja u aplikaciju koji ima svoje vrijeme isteka. Korisnik neke od operacija nad API-em može obaviti samo slanjem odgovarajućeg tokena u zaglavlju zahtjeva ukoliko želi da njegov zahtjev bude obrađen. Ukoliko token nije poslan, nevažeći je ili nije od ispravnog korisnika, korisniku će se vratiti status kod 403 koji znači zabrana pristupa.

authenticateToken – služi za provjeru samog tokena te njegove valjanosti i ukoliko je valjan kao rezultat vraća korisnika

checkGroupOwnership – na temelju rezultata iz authenticateToken posredničke funkcije provjerava vlasništvo nat traženom grupom u bazi podataka te u slučaju da je vlasništvo ispravno, propušta zahtjev na obradu.

```
router.post('/groups/addFormula', authenticateToken,
checkGroupOwnership, async (req, res) => {…});
```

Kao što možemo i vidjeti kod jednog tako definiranog sučelja, prije same lambda funkcije odrađuju se funkcije posredničkog sloja

## 7.4. Sučelja REST API-a

U nastavku će biti pobrojana sva sučelja koje bi korisnik aplikacije mogao koristit.

### 7.4.1. POST /register

Nalazi se na ruti „/auth/register“, služi za registraciju korisnika sa korisnikovim korisničkim imenom i sažetkom lozinke (sažetak računat na klijentskoj strani).

Parametri tijela zahtjeva:

```
{
  "username": "john_doe",
  "password": "0xHASH "
}
```

### 7.4.2. POST /login

Nalazi se na ruti „/auth/login“, služi za ulazak korisnika u aplikaciju sa već prije kreiranim računom. Kao rezultat uspješnog ulaska u aplikaciju korisnik će dobiti autentifikacijski token koji služi za korištenje osiguranih sučelja, te refresh token.

Parametri tijela zahtjeva:

```
{
  "username": "john_doe",
  "password": "0xHASH"
}
```

### 7.4.3. POST /refreshToken

Nalazi se na ruti „/auth/refreshToken“, služi za dobivanje novog autentifikacijskog tokena na temelju korisnikovog refreshTokena.

Parametri tijela zahtjeva:

```
{
  "refreshToken":
"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6MSwiaWF0IjoxNzE1MTgyN
jUwLCJleHAiOjE3MTc3NzQ2NTB9.QkcpodOaRD6-
qWxELGIETyUF5EJ_RWsxQPCzppaUwd4"
}
```

### 7.4.4. POST /addGroup

Nalazi se na ruti „/api/addGroup“, služi za kreiranje nove groue formula koja u sebi sadrži listu formula.

Parametri tijela zahtjeva:

```
{
  "ownerId": 1,
  "name": "My Group",
  "formulas": [
    {
      "title": "Formula 1",
      "mathFormula": "x + y",
      "description": "Example formula 1"
    },
    {
      "title": "Formula 2",
      "mathFormula": "x * y",
      "description": "Example formula 2"
    }
  ]
}
```

### 7.4.5. POST /user/groupDownload

Nalazi se na ruti „/api/user/groupDownload“, služi za dodavanje grupe formula u korisnikovu listu preuzetih formula. Tako ako se korisnik ulogira preko drugog uređaja na svoj račun možemo znati koje formule su prethodno preuzete.

Zahtijeva autorizacijsko zaglavlje za uspješno izvršavanje.

Parametri tijela zahtjeva:

```
{
  "formulaGroupId": 1
}
```

### 7.4.6. POST /user/groupDelete

Nalazi se na ruti „/api/user/groupDelete“, služi za brisanje grupe formula iz korisnikove liste preuzetih formula. Tako ako se korisnik ulogira preko drugog uređaja na svoj račun možemo znati koje formule su uklonjene s liste te ih nije potrebno naknadno preuzimati.

Zahtijeva autorizacijsko zaglavlje za uspješno izvršavanje.

Parametri tijela zahtjeva:

```
{
  "formulaGroupId": 1
}
```

### 7.4.7. GET /loadUserData

Nalazi se na ruti „/api/loadUserData“, služi za dohvaćanje korisnikovih preuzetih formula i njegovih reakcija na te formule.

Zahtijeva autorizacijsko zaglavlje za uspješno izvršavanje.

### 7.4.8. GET /groups

Nalazi se na ruti „/api/groups“, služi za dohvaćanje svih grupa sa pripadajućim formulama.

# 8. Android aplikacija ForMat

U današnjem svijetu mobilnih tehnologija, razvoj Android aplikacija predstavlja jedno od ključnih područja informatičke industrije. S obzirom na rastuću potražnju za intuitivnim, brzim i funkcionalnim mobilnim iskustvima, programeri se sve više okreću alatima koji olakšavaju izradu visokokvalitetnih aplikacija. Upravo u tom kontekstu Kotlin postaje neizbježan jezik za razvoj Android aplikacija, zahvaljujući svojoj jednostavnosti, interoperabilnosti s Java kodom i podršci za moderni razvoj softvera.

Kotlin donosi niz prednosti u razvoju Android aplikacija. Osim što omogućava čišći kod u usporedbi s Javom, Kotlin donosi i napredne značajke poput nullable tipova, ekstenzija, funkcionalnog programiranja i još mnogo toga. Ovaj jezik olakšava razvoj aplikacija te smanjuje mogućnost grešaka i nejasnoća u kodu.

No, osim samog jezika, ključan element u razvoju Android aplikacija je i korisničko sučelje. Tradicionalno, izgradnja korisničkog sučelja za Android aplikacije odvijala se putem XML layout datoteka, ali s pojavom Jetpack Compose-a, situacija se dramatično mijenja. Jetpack Compose predstavlja revolucionarni pristup izradi korisničkog sučelja, omogućavajući deklarativno programiranje umjesto imperativnog, što rezultira čistijim, fleksibilnijim i reaktivnijim kodom.

Kroz ovo poglavlje, istražit ćemo kako kombinacija Kotlina i Jetpack Compose-a transformira način na koji se razvijaju Android aplikacije. Fokusirat ćemo se na najbolje prakse, tehnike i obrasce dizajna koji omogućuju razvoj modernih, responzivnih i intuitivnih korisničkih sučelja. Osim toga, istražit ćemo i napredne koncepte kao što su arhitekturni uzorci, upravljanje stanjem i integracija s ostalim dijelovima Android ekosustava kako bismo stvorili cjelovito iskustvo za krajnje korisnike. Kroz primjere, savjete i analize, cilj nam je osposobiti čitatelje da uspješno kreiraju visokokvalitetne Android aplikacije koje odražavaju najnovije trendove i standarde u mobilnom razvoju.

## 8.1. Jetpack Compose

Jetpack Compose predstavlja revolucionarni alat za izradu korisničkog sučelja (UI) u Android aplikacijama, koji omogućuje deklarativno programiranje umjesto tradicionalnog

imperativnog pristupa. Umjesto pisanja XML layout datoteka kao što je bio slučaj s prethodnim alatima poput XML-a, programeri koriste Kotlin DSL (Domain Specific Language) za definiranje UI komponenata i njihovih svojstava. Ovaj pristup donosi niz prednosti, uključujući veću čitljivost koda, lakšu održivost, reaktivnost i mogućnost dinamičke promjene UI-a na temelju stanja aplikacije ili korisničkih interakcija.

Kroz Jetpack Compose, programeri imaju pristup bogatom setu predefiniranih komponenata koje mogu koristiti za izgradnju različitih dijelova korisničkog sučelja, kao što su gumbi, tekstualna polja, liste, kartice i mnoge druge. Osim toga, Compose omogućuje lako grupiranje komponenata i definiranje njihovih međusobnih odnosa, čime se olakšava izrada složenih i dinamičnih UI-ja.

Još jedna značajna prednost Jetpack Compose-a je njegova integracija s ostatkom Android ekosustava i Jetpack biblioteka. Programeri mogu kombinirati Compose sa ostalim Jetpack komponentama poput Navigation-a, ViewModel-a, i sličnoga, kako bi stvorili cjelovita i skalabilna rješenja za razvoj Android aplikacija. Ova integracija olakšava upravljanje stanjem, navigacijom, lokalnom pohranom podataka i drugim ključnim aspektima razvoja aplikacija.

U konačnici, Jetpack Compose predstavlja budućnost razvoja korisničkog sučelja u Android aplikacijama, nudeći programerima moderni, efikasan i ugodan način za izgradnju responzivnih i atraktivnih korisničkih iskustava. Ova tehnologija donosi mnoge prednosti za razvojni proces, omogućujući brži razvoj, manje grešaka i veću fleksibilnost u prilagođavanju aplikacija različitim zahtjevima i trendovima.

### 8.1.1. Jetpack Compose vs XML

Jetpack Compose i XML su dva različita pristupa izradi korisničkog sučelja (UI) u Android aplikacijama. Dok XML predstavlja tradicionalni pristup, Jetpack Compose donosi modernu deklarativnu paradigmu.

XML se koristi za definiranje izgleda sučelja i njegovih komponenata putem oznaka i atributa. Na primjer, definicija jednostavnog gumba u XML-u izgledala bi ovako:

```
<Button
    android:id="@+id/button"
```

```
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="Click me" />
```

S druge strane, Jetpack Compose koristi Kotlin DSL (Domain Specific Language) za definiranje UI komponenata. Primjer istog gumba u Jetpack Compose-u bio bi:

```
Button(onClick = {}) {
    Text("Click me")
}
```

Jedna od glavnih prednosti u odnosu na XML i to što direktno u samoj definiciji i povezujemo komponentu sa funkcionalnosti. Tako u slučaju gumba definiramo koja će se funkcija baviti na njegov klik, dok bi u XML-u trebali povezati komponentu s samom logikom aplikacije preko „id“ definiranog u komponenti te bi to zahtijevalo još dodatnog koda.

Glavna razlika između ova dva pristupa je u načinu definiranja komponenata. Dok je XML imperativan i zahtijeva detaljnu specifikaciju svakog atributa komponente, Jetpack Compose je deklarativan i omogućava programerima da opišu kako žele da UI izgleda, a ne kako to postići korak po korak.

Usporedba ova dva pristupa može se izraziti kroz čitljivost koda, fleksibilnost i dinamičnost. Jetpack Compose često rezultira čišćim i manje verbose kodom, olakšavajući razumijevanje i održavanje UI-a. Također, Compose omogućava dinamičke promjene UI-a na temelju stanja aplikacije ili korisničkih interakcija, što može biti izazovnije u XML-u.

### 8.1.2. Sličnost s tehnologijama raznih platformi

Jetpack Compose i SwiftUI su slične tehnologije koje donose revolucionarne pristupe izradi korisničkog sučelja (UI) u svojim platformama. Obe tehnologije koriste deklarativni pristup programiranju za definiranje UI komponenata, umjesto tradicionalnog imperativnog pristupa.

Sličnost između Jetpack Compose-a i SwiftUI-a se ogleda u njihovoj sposobnosti da omoguće programerima opisivanje izgleda i ponašanja UI-a kroz deklarativni kod. Oba alata omogućavaju programerima da opišu što žele postići, a ne kako to postići korak po korak, što rezultira čišćim, manje verbose i lakše održivim kodom.

Još jedna slična tehnologija je React Native, koja omogućava izradu mobilnih aplikacija koristeći JavaScript i React. Iako nije ista kao Jetpack Compose i SwiftUI, React Native također koristi deklarativni pristup programiranju i omogućava razvoj responzivnih korisničkih sučelja.

Za programere je bitno razumjeti slične tehnologije jer im omogućava lakše prelazak s jedne platforme na drugu ili sa jednog jezika na drugi. Poznavanje sličnih tehnologija olakšava učenje novih alata i povećava fleksibilnost programera, što ih čini konkurentnijima na tržištu rada. Također, razumijevanje sličnih tehnologija može olakšati dijeljenje znanja i resursa unutar razvojnog tima.

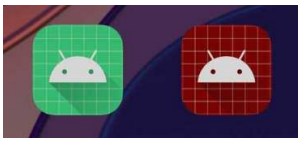
## 8.2. Struktura Android aplikacije

Struktura praznog Android projekta generiranog od strane Android Studija obično uključuje nekoliko osnovnih direktorija i datoteka koje služe kao temelj za razvoj aplikacije.

1. **app/** direktorij:
   1.1. **src/main/** direktorij: Ovaj direktorij sadrži izvorni kod aplikacije.
      1.1.1. **java/** direktorij: Ovdje se nalazi Java (ili Kotlin) izvorni kod aplikacije.
      1.1.2. **res/** direktorij: Sadrži resurse aplikacije kao što su slike, ikone, tekstovi, itd.
         1.1.2.1. **values**: sadrži datoteku za tekstove na engleskom jeziku
         1.1.2.2. **values**-hr: sadrži datoteku za tekstove na hrvatskom jeziku
      1.1.3. **AndroidManifest.xml**: Osnovna konfiguracija aplikacije, uključujući dozvole, aktivnosti i ostale komponente.
   1.2. **config/** direktorij: Ovaj direktorij služi za konfiguracijske dataotke.
   1.3. **google-service.json**: Datoteka s konfiguracijom za Firebase servis
2. **gradle/** direktorij:
   2.1. **wrapper/** direktorij: Sadrži skripte potrebne za izradu i upravljanje Gradleom, alatom za izgradnju Android projekata.
3. **gradle.properties:** Konfiguracijska datoteka za Gradle projekte, uključujući postavke verzija i putanja.
4. **build.gradle:** Konfiguracijska datoteka koja definira postavke izgradnje projekta, uključujući ovisnosti, verzije i dodatne alate.

Uz main/ direktorij koji sadrži naš kod aplikacije u našem src/ direktoriju nalaze se još i:

- dev/ direktorij – služi za specifičnosti „dev“ okruženja
- prod/ direktorij – služi za specifičnosti „prod“ okruženja



Slika 19. Istoimeni resurs u različitim okruženjima

- /test direktorij – služi za jedinične testove pojedinih dijelova koda

U samom main/ direktoriju aplikaciju smo podijelili na više paketa/modula. Svaki od modula zadužen je za određeni skup funkcionalnosti. Skup funkcionalnosti može biti arhitekturalan (svrha mu je postaviti i olakšati korištenje odabrane arhitekture) ili može biti orijentiran na same mogućnosti aplikacije.

## 8.3. Moduli

Moduli naše aplikacije su :

- app – služi za generalne stvari vezane uz aplikaciju, poput Application klase, MainActivity klase, klasa za navigaciju, generalni DI modul, i slično
- common – služi za dijelove koji se koriste na više mjesta u aplikaciji
- data – zadužen za svu komunikaciju s vanjskim svijetom, jedini modul koji ima pristup Android kontekstu (uz naravno app), sadrži implementacije svih repository-a i store-ova
- domain – zadužen je za definiciju modela, repository-a i svih ostalih komponenata koje će se koristit kroz „feature“ module
- download – zadužen za set funkcionalnosti preuzimanja formula sa servera
- formulas zadužen za set funkcionalnosti vezane uz same formule, poput pregleda, uređivanja, brisanja…
- home – zadužen je za početni ekran prilikom prolaska „login“ faze aplikaciju
- onboarding – zadužen je za inicijalni ekran aplikacije u kojem korisnik ima opciju kreiranja računa, preskakanje računa, i sl.

### 8.3.1. app – aplikacijski modul

Aplikacijski modul u našoj aplikaciji sadrži:

- **Aplikacijsku klasu** koja služi za inicijaliziranje Koin modula, to mora biti obavljeno prije svega ostalog kako bismo izbjegli „crasheve“.
- **Koin modul** u kojem definiramo sve povezane module u kojima se nalaze definicije svih potrebnih klasa u aplikaciji.
- **MainActivity klasu** u kojoj postavljamo navigacijski kontroler, navigacijsko stablo, temu aplikacije, animacije prilikom tranzicije ekrana, i sl.
- **Navigation direktorij** u kojem se nalaze *Navigator* koji koristimo za samu navigaciju kroz aplikaciju, te *NavHostControllerProvider* koji služi za posluživanje NavHsotController mogućnosti samom Navigatoru.

### 8.3.2. common – generalni modul

Common modul nam služi kako bismo definirali komponente koje ćemo koristit na više mjesta u samoj aplikaciji. To bi bili modeli poput *AppError* i *Epoch* (razlika u odnosu na domenski model je to što se mogu koristit kroz sve slojeve aplikacije a ne samo na feature sloju), uz modele to bi bile i UI komponente kao što su *NormalButton*, *Spacers*, *Spinners*, i sl.

Common modul sadrži:

- **DI direktorij** – sadrži *CommonModule* u kojem su implementirane klase Common modula
- **Infrastructure direktorij** – sadrži direktorije logger i configuration
    - **Logger** – sadrži definiciju funkcija dostupnih za logiranje (implementacija u data modulu)
    - **Configuration** – sadrži definiciju i dvije implementacije aplikacijske konfiguracije (*ProdConfiguration*, *DevConfiguration*)
    - **Analytics** – sadrži definiciju servisa za analitiku (implementacija u data modulu)

- **Model direktorij** – Sadrži nekoliko modela, *AppError* (služi za definiranje različitih grešaka kroz aplikaciju), *Epoch* (služi za definiranje vremenskih oznaka kroz aplikaciju) i AnalyticsModel (sadrži definiciju analitičkih evenata i ekrana koji se šalju iz aplikacije)
- **UI direktorij** – sadrži implementacije svih dijeljenih UI komponenti u ostalim modula poput *ForMatInputField*, *LatexView*, *Spacers*, *Spinner*…
- **Util direktorij** – sadrži samo jednu metodu za izračunavanje sažetka korisnikove lozinke.

### 8.3.3. data – podatkovni modul

U data modulu nalaze se sve komponente koje komuniciraju s vanjskim svijetom. Pod vanjski svijet ne misli samo dio koji komunicira s API-em već se pod to podrazumijeva svaka komponenta koja „izlazi“ iz same aplikacije. To je dakle svako čitanje i pisanje u trajnu memoriju, pristup web servisu i sl.

Dana modul se sastoji od:

- **API direktorija** – sadrži dva API-a, *PublicApi* i *RestrictedApi*
    - **PublicApi** – služi za sav pristup serveru bez potrebe da korisnik bude ulogiran, tako se tamo nalaze metode za ulazak korisnika u korisnički račun, registracija korisnikovog računa, dohvat svih objavljenih formula…

      ```
      @GET("/api/groups")
      suspend fun groups(): List<GroupDto>
      ```

    - **RestrictedApi** – služi za sve metode za koje korisnik mora biti ulogiran na svoj korisnički račun, kako bismo spriječili nepoznatog korisnika od mijenjanja seta formula koje mu ne pripadaju ili da objavljuje nove setove formula bez računa.

      ```
      @POST("/api/formula/react")
      suspend fun formulaReact(
          @Body request: FormulaReactDto
      ): FormulaReactResponseDto
      ```

- **DI direktorij** - sadrži *DataModule* u kojem su implementirane klase Data modula

- **Formulas** direktorij – sadrži implementaciju *FormulasRepository*-a i *FormulaStore*-a
- **Infrastructure direktorij** – sadrži implementacije generalnih bitnih za samu aplikaciju
    - **DateTimeProvider** – služi za dohvaćanje vremena i sustavskih postavki
    - **ForMatLogger** – služi za zapisivanja logova kroz kod
    - **ForMatPreferences** – služi za pristup aplikacijskoj kriptiranoj memoriji
    - **FirebaseAnalytics** – služi za slanje evenata i ekrana na Firebase servis
- **User direktorij** – sadrži implementaciju *UserRepository*-a
- **Networking direktorij** – direktorij koji se bavi u uspostavom konekcije, upravljanje tokenima, serijalizacijom…

#### 8.3.3.1 networking – modul komunikacije sa internetom

Pošto je networking modul poprilično bitan za samu aplikaciju, jer većina funkcionalnosti vezana je uz sami API objašnjen ranije, ovaj je modul dobio svoju sekciju.

Ovaj modul sadrži:

- **DI direktorij** - sadrži *NetworkingModule* u kojem su implementirane klase Networking modula
- **Interceptor** – sadrži *AuthorizationInterceptor* koji služi kako bi se na svakom pozivu prema API-u (iz *RestrictedApi*-a) korisniku automatski provjerila validnost njegovog pristupnog tokena i ukoliko je on istekao odrađuje se poziv prema *PublicApi*-u koji nam pomoću našeg osvježnog tokena vraća novi pristupni token.
- **Serialization** – sadrži *JsonProvider* implementaciju koju koristimo za kreiranje *Retrofit*-a
- **Token** – sadrži sami model *Token* koji se koristi za komunikaciju s API-em i *TokenStore* implementaciju koja služi za lokalno pohranjivanje dohvaćenih tokena
- **Util** – sastoji se od nekoliko pomoćnih klasa koje sadrže metode za provjeru validnosti tokena, za dohvaćanje novog tokena i pretvaranje network greške u nama poznat model.

### 8.3.4. domain – domenski modul

Domenski modul je zadužen za definiciju svih potrebnih repozitorija i modela koje će ostatak modula koristit kao svoj izvor istine. Ovaj sloj možemo smatrati „Model“ u MVVM arhitekturi.

Domenski modul sastoji se od:

- **Formulas direktorija** – sadrži definiciju *FormulasRepository*-a i *FormulaStore*-a
- **Model direkotrija** – sadrži sve modele definirane za ostatak aplikacije poput *User*, *Reaction*, *FormulaGroup*, i sl.
- **User direktorija** – sadrži definiciju UserRepository-a

Možemo vidjeti kako u domenskom modulu ne postoji **DI** modul, a to je upravo iz razloga što se domenski modul bavi samo definicijom, dok data modul bavi samom implementacijom. Time smo zadovoljili ranije spomenut *DDD* arhitekturalni obrazac

### 8.3.5. download – feature modul

Download je naš prvi modul koji se bavi mogućnostima u aplikaciji. Odgovornost download modula je omogućiti korisniku da preuzme objavljene formule sa API-a.

Download modul sadrži:

- **DI direktorij** – sadrži DI modul u kojem su navedene implementirane klase feature modula
- **UI direktorij** – sadrži *DownloadFormulaScreen* koji nam služi za preuzimanje odabrane objavljene formule.. UI direktorij spada pod „View“ dio MVVM-a
- **viewModel direktorij** – sadrži *DownloadFormulaViewModel*, koji kako i samo ime govori spada pod „ViewModel“ dio MVVM-a
- **viewState direktorij** – sadrži podatkovnu klasu u kojoj se nalazi lista formula dostupnih za preuzimanje sa API-a. Podatkovnu klasu sa listama formula popunio je ViewModel.

### 8.3.6. formulas – feature modul

Formulas modul je modul koji se bavi prikazivanjem groupe formula, uređivanje, kreiranje i brisanjem groupa formula kao i sam pregled pojedinačnih formula. Isto tako njegova odgovornost je i objavljivanje reakcija na pojedine formule te dohvaćanje statistike za grupu i formulu.

- **DI direktorij** – sadrži DI modul u kojem su navedene implementirane klase feature modula
- **UI direktorij** – sadrži sve ekrane, komponente i UI elemente koji su jedinstveni za ovaj skup funkcionalnosti. UI direktorij spada pod „View“ dio MVVM-a.
    - **EditGroupScreen** – sadrži mogućnost dodavanja novih formula, brisanja starih formula i izmjena postojećih formula i imena grupe.
    - **FormulaDetailsScreen** – sadrži mogućnost prikazivanja formule, objavljivanja rekacije na nju te pregled statistike (ukoliko smo vlasnik)
    - **GroupDetailsScreen** – sadrži mogućnost prikazivanja groupe, objavljivanja, uređivanje i brsanje grupe te pregled statistike (ukoliko smo vlasnik)
- **viewModel direktorij** – sadrži 3 viewModela od gore navedenih ekrana. Svaki od ViewModela služi kako bih ekranu omogućio funkcionalnost, a kako i samo ime govori spada pod „ViewModel“ dio MVVM-a
- **viewState direktorij** – sadrži podatkovne klase potrebne za prikazivanje ekrana poput „isGroupPublished“ varijable koja će se koristit za prikazivanje animacije učitavanja

### 8.3.7. home – feature modul

Home modul služi za inicijalni pregled stanja aplikacije, tu se nalaze prikazi svih grupa formula koje imamo lokalo kreiranih, preuzetih sa API-a te posebno označene grupe formula koje nam se sviđaju.

- **DI direktorij** – sadrži DI modul u kojem su navedene implementirane klase feature modula
- **UI direktorij** – sadrži HomeScreen koji korisniku nudi mogućnost dodavanja nove grupe formula, otvaranje ekrana za preuzimanje nove formule, izlogiravanja

korisnika iz njegovog korisničkog računa i ulaz u pregled grupe formula. UI direktorij spada pod „View“ dio MVVM-a

- **viewModel direktorij** – sadrži *HomeViewModel* koji dohvaća i pohranjuje podatke u odgovarajući viewState, koji kako i samo ime govori spada pod „ViewModel“ dio MVVM-a
- **viewState direktorij** – sadrži podatkovnu klasu u kojoj se nalaze liste pojedinih grupa formula koje se prikazuju na samom ekranu.

### 8.3.8. onboarding – feature modul

- **DI direktorij** – sadrži DI modul u kojem su navedene implementirane klase feature modula
- **UI direktorij** – sadrži *LoginScreen*, *WelcomeScreen* i *RegisterScreen*. UI direktorij spada pod „View“ dio MVVM-a
- **viewModel direktorij** – sadrži 3 viewModela od gore navedenih ekrana. Svaki od ViewModela služi kako bih ekranu omogućio funkcionalnost, a kako i samo ime govori spada pod „ViewModel“ dio MVVM-a
- **viewState direktorij** – sadrži dvije podatkovne klase za Login i Register ekrane u kojim se nalaze podaci o trenutnom stanju poziva, te poruka o grešci ukoliko je ima.

# Zaključak

Zaključak ovog diplomskog rada pruža sintezu naučenih lekcija i refleksiju na cijeli proces razvoja aplikacije za podršku učenju matematike na tehničkim fakultetima, koristeći bogat set tehnologija i alata. Rad je pokazao da iako je krajnji proizvod, funkcionalna aplikacija, važan, jednako su važni procesi koji vode do njenog stvaranja.

Kroz diplomski rad, razvijena aplikacija služila je kao praktični primjer kako se tehnološke kompetencije mogu primijeniti u stvarnom projektu, te kako teorija i praksa mogu uspješno koegzistirati i nadopunjavati jedna drugu. Pokriven je širok spektar tehnologija koje se uključuju u moderni softverski razvoj, uključujući Kotlin, Android Studio, Git, Firebase, i mnoge druge, svaka sa svojom specifičnom ulogom i doprinosom konačnom proizvodu.

Razvoj aplikacije nije bio bez izazova, što je neizbježan dio bilo kojeg inženjerskog pothvata. Međutim, kroz rješavanje ovih izazova, stekla su se dragocjena znanja o važnosti planiranja, analize, upravljanja projektima i adaptacije na promjene, koje su sve ključne komponente uspješnog softverskog projekta.

Ovaj rad također naglašava važnost neprekidnog učenja i prilagodbe u brzo mijenjajućem tehnološkom svijetu. Softverski inženjeri moraju biti spremni na stalno usavršavanje svojih vještina i znanja kako bi ostali relevantni i učinkoviti u svojim profesionalnim ulogama. Diplomski rad je praktični primjer kako teoretsko znanje stečeno na fakultetu može biti primijenjeno na stvarne probleme, te kako akademski programi mogu biti dizajnirani da pripreme studente za stvarne izazove u industriji.

U konačnici, ovaj rad demonstrira kako procesi, iako ponekad zanemareni u korist krajnjih produkata, formiraju temelj na kojem se gradi svaki uspješan softverski projekt. S razumijevanjem i poštovanjem ovih procesa, budući programeri će biti bolje opremljeni za suočavanje s izazovima koje donosi profesionalni razvoj softvera.

# Literatura


[1] Martin R. C., *Clean Code*. 1. izdanje. London: Pearson, 2008.

[2] Erich G., Richard H., Ralph J., John V., *Design Patterns*. 1. izdanje, United States: Addison-Wesley, 1994.

[3] Alexey S., *Kotlin Design Patterns and Best Practices*. 2. izdanje. Brimingham: Packt Publishing, 2021.

[4] Martin F., *Refactoring*. 2. izdanje. United States: Addison-Wesley, 2019.

[5] Brigit, P., Rolf B., Ghislaine G., *Mathematics in Engineering Education: a Review of the Recent Literature with a View towards Innovative Practices*, International Journal of Research in Undergraduate Mathematics Education, 7, (2021), str. 163-188

[6] JetBrains (bez dat.) *Kotlin Documentation*; Poveznica https://kotlinlang.org/docs/home.html; pristupljeno 5. svibnja 2024.

[7] TJ Holowaychuk (bez dat.) *Express 4.x*; Poveznica: https://expressjs.com/en/4x/api.html; pristupljeno 5. svibnja 2024.

[8] Google (bez dat.) *Get stated with Jetpack Compose*; Poveznica: https://developer.android.com/develop/ui/compose/documentation; pristupljeno 5. svibnja 2024.

[9] Prisma Data (bez dat.) *Prisma Documentation*; Poveznica: https://www.prisma.io/docs; pristupljeno 5. svibnja 2024.

[10] Google (bez dat.) *Firebase Documentation*; Poveznica: https://firebase.google.com/docs; pristupljeno 5. svibnja 2024.

[11] Linus Torvalds (bez dat.) *Git - Documentation*; Poveznica: https://git-scm.com/doc; pristupljeno 5. svibnja 2024.

[12] Square (bez dat.) *Retrofit*; Poveznica: https://square.github.io/retrofit/; pristupljeno 5. svibnja 2024.

# Sažetak

**Naslov**: Razvoj aplikacije za podršku učenju matematike na tehničkim fakultetima

**Sažetak**: Diplomski rad usmjeren je na razvoj mobilne aplikacije namijenjene podršci učenju matematike na tehničkim fakultetima, koristeći moderne tehnologije kao što su Kotlin, Android Studio, Git, i Firebase. Fokus rada nije samo na razvoju funkcionalne aplikacije, već i na detaljnom objašnjavanju procesa i tehnologija koje su ključne u softverskom inženjerstvu. Rad pruža uvid u izazove i rješenja u svakoj fazi projekta, od planiranja i analize, preko implementacije, do testiranja i distribucije aplikacije, naglašavajući važnost razumijevanja softverskih procesa u praksi.

**Ključne riječi**: aplikacija, matematika, Kotlin, Android Studio, Git, Firebase, softversko inženjerstvo, proces razvoja softvera

# Summary

**Title**: Development of an Application for Supporting Mathematics Learning at Technical Universities

**Abstract**: This thesis focuses on the development of a mobile application designed to support mathematics learning at technical universities, utilizing modern technologies such as Kotlin, Android Studio, Git, and Firebase. The emphasis of the work is not only on developing a functional application but also on a detailed explanation of the processes and technologies that are pivotal in software engineering. The thesis provides insights into the challenges and solutions at every stage of the project, from planning and analysis, through implementation, to testing and distribution of the application, highlighting the importance of understanding software development processes in practice.

**Keywords**: application, mathematics, Kotlin, Android Studio, Git, Firebase, software engineering, software development process.

# Skraćenice

| | | |
|---|---|---|
| KMM | *Kotlin Multiplatform Mobile* | Kotlin za više mobilnih platformi |
| IDE | *Integrated Development Environment* | integrirano razvojno okruženje |
| API | *Application Programming Interface* | sučelje za programiranje aplikacija |
| CI | *Continuous Integration* | kontinuirana integracija |
| CD | *Continuous Delivery* | kontinuirana isporuka |
| HTTP | *Hypertext Transfer Protocol* | protokol za prijenos hiperteksta |
| UI | *User Interface* | korisničko sučelje |
| JSON | *JavaScript Object Notation* | oznaka objekta JavaScript |
| FCM | *Firebase Cloud Messaging* | oblačno slanje poruka Firebase |
| DI | *Dependency Injection* | ubrizgavanje ovisnosti |
| DDD | *Domain-Driven Design* | dizajn vođen domenom |

# Privitak

### Programski kod mobilne aplikacije ForMat

### Programski kod REST API-a aplikacije ForMat