
# Razvoj mobilne aplikacije
# Elektronski dnevnik u Kotlin programskom jeziku

ZAVRŠNI RAD 1.CIKLUSA STUDIJA

Student:

Ismail Ičanović

Mentor:

V.prof.dr. Samir Omanović

Sarajevo, Septembar 2022

**Nastavnik:** V.prof.dr Samir Omanović, dipl.ing.el.

**Tema br. 07 :** Završni rad 1. ciklusa za 2021/22 studijsku godinu

**Razvoj mobilne aplikacije Elektronski dnevnik u Kotlin programskom jeziku**

**Student: Ismail Ičanović**

**Cilj:**

Dizajn i razvoj mobilne aplikacije Elektronski dnevnik u Kotlin programskom jeziku.

**Opis:**

Potrebno je razviti potpuno funkcionalnu mobilnu aplikaciju Elektronski dnevnik u Kotlin programskom jeziku što uključuje brojne zadatke – od analize i specifikacije zahtjeva, dizajna interfejsa i baze podataka, programiranja do testiranja. Kratak opis očekivanih funkcionalnosti aplikacije:

- Nastavnik i učenik posjeduje svoj račun preko nekog e-mail servisa. Potrebno je omogućiti kreiranje i održavanje korisničkih računa.
- Student ima pravo da vidi samo svoj profil a nastavnik ima pravo da vidi profile svih svojih učenika.
- Svaki učenik unutar svog profila ima tematske jedinice (može biti jedan nivo ili hijerarhija tema i podtema) za koje dobija ocjene. Cilj je omogućiti kontinuirano praćenje znanja pa je potrebno da se za svaku tematsku jedinicu ima kontinuirano praćenje ocjenjivanja (datum, ocjena i komentar prilikom svake provjere poznavanja neke od tematskih jedinica). Omogućiti unos i ažuriranje tematskih jedinica za administratora sistema.
- Prikaz profila sa zadnjim ocjenama i datumima provjere znanja za svaku tematsku jedinicu radi mogućnosti ispisa.

Pored prethodno nabrojanih osnovnih funkcionalnosti se mogu dodati i druge korisne funkcionalnosti ukoliko se pokažu važnim. Podaci trebaju biti pohranjivani u bazi kao i cijel aistorija promjena podataka (log tabele za promjene ocjena).

Za testiranje je poželjno koristiti što više unit testove. Poželjno bi bilo razvijenu aplikaciju učiniti javno dostupnom na GitHub platformi kako bi što više ljudi imalo koristi od nje.

**Plan rada:**

1. Analiza i specifikacija zahtjeva, dizajn interfejsa i baze podataka koristeći odabrane alate, odabir tehnologija.
2. Razvoj aplikacije i izrada unit testova, testiranje, po potrebi vraćenje u prethodne korake radi određenih izmjena.
3. Pisanje dokumenta: rezultati analize i specifikacije zahtjeva, prezentacija dizajna interfejsa i baze podataka, kratak pregled odabranih tehnologija, prezentacija procesa razvoja i testiranja sa opisom eventualnih vraćanja u prethodne korake radi izmjena, prezentacija finalne verzije mobilne aplikacije i njenih mogućnosti, analiza uspješnosti realizacije specificiranih zahtjeva, zaključak, literatura.

**Očekivani rezultati:**

1. Mobilna aplikacija Elektronski dnevnik.
2. Dokument završnog rada.
3. Dobro poznavanje razvoja malih do srednje velikih mobilnih aplikacija u Kotlin programskom jeziku.

**Polazna literatura**:


1. Kotlin Foundation: „Kotlin - A modern programming language that makes developers happier.“, dostupno on-line: https://kotlinlang.org/ [23.02.2022]
2. Imf: „Build Your First Android App in Kotlin“, dostupno on-line: https://developer.android.com/codelabs/build-your-first-android-app-kotlin#0 [23.02.2022]
3. Christina Kopecky: „How to develop your first Android app with Kotlin“, dostupno on-line: https://www.educative.io/blog/android-development-app-kotlin [23.02.2022]
4. Google Developers: „Develop Android apps with Kotlin“, dostupno on-line: https://developer.android.com/kotlin [23.02.2022]

# Izjava o autentičnosti radova

Seminarski rad, završni (diplomski odnosno magistarski) rad za I i II ciklus studija i integrirani studijski program I i II ciklusa studija, magistarski znanstveni rad i doktorska disertacija*.

Ime i prezime ____***Ismail Ičanović***___

Naslov rada __***Elektronski dnevnik u Kotlin programskom jeziku***___

Vrsta rada ___***Diplomski rad za I ciklus studija***__

Broj stranica _____

Potvrđujem:

- da sam pročitao/la dokumente koji se odnose na plagijarizam, kako je to definirano Statutom Univerziteta u Sarajevu, Etičkim kodeksom Univerziteta u Sarajevu i pravilima studiranja koja se odnose na I i II ciklus studija, integrirani studijski program I i II ciklusa i III ciklus studija na Univerzitetu u Sarajevu, kao i uputama o plagijarizmu navedenim na web stranici Univerziteta u Sarajevu;
- da sam svjestan/na univerzitetskih disciplinskih pravila koja se tiču plagijarizma; •
- da je rad koji predajem potpuno moj, samostalni rad, osim u dijelovima gdje je to naznačeno;
- da rad nije predat, u cjelini ili djelimično, za stjecanje zvanja na Univerzitetu u Sarajevu ili nekoj drugoj visokoškolskoj ustanovi;
- da sam jasno naznačio/la prisustvo citiranog ili parafraziranog materijala i da sam se referirao/la na sve izvore;
- da sam dosljedno naveo/la korištene i citirane izvore ili bibliografiju po nekom od preporučenih stilova citiranja, sa navođenjem potpune reference koja obuhvata potpuni bibliografski opis korištenog i citiranog izvora;
- da sam odgovarajuće naznačio/la svaku pomoć koju sam dobio/la pored pomoći mentora/ice i akademskih tutora/ica.

Mjesto, datum __***Sarajevo, 01.09.2022***__

Potpis _____

---

* U radu su korišteni slijedeći dokumenti: Izjava autora koju koristi Elektrotehnički fakultet u Sarajevu; Izjava o autentičnosti završnog rada Centra za interdisciplinarne studije – master studij „Evropske studije", Izjava o plagijarizmu koju koristi Fakultet političkih nauka u Sarajevu.

# Sažetak

Svrha ovoga završnog rada jeste razvoj mobilne i serverske aplikacije koje zajedno komuniciraju i pružaju korisnicima specifičnu uslugu. Želja je bila pronaći oblast primjene gdje se ovakav sistem pojavljuje prvi put i olakšava ljudima njihov svakodnevni život i rad. Jedan od ciljeva jeste automatizacija ljudskih radnji.

U uvodu su spomenuti neki tipovi dnevnika koji postoje i odabrana je specifična oblast primjene sistema.

Svoju primjenu ovaj sistem pronalazi u velikom broju škola Kur'ana unutar i van BiH, te predstavlja nešto novo u toj oblasti obrazovanja.

Potrebno je bilo istražiti i poznavati oblast primjene da bi se unutar aplikacije korisnicima pružile nužne funkcionalnosti i zadovoljile njihove potrebe.

ANDROID je odabrani operativni sistem za koji je razvijena mobilna aplikacija. Aplikacija je razvijana sa svim tehnologijama i bibliotekama koje su preporučene od strane kompanije Google koja je vlasnik ANDROID operativnog sistema.

Aplikacija je napravljena od mnogih funkcionalnosti koje predstavljaju generalnu osnovu za sve tipove dnevnika uz specifične funkcionalnosti koji se prilagođavaju odabranoj oblasti primjene. Osnova aplikacije je lahko nadogradiva i podložna izmjenama zahvaljujući odabranim tehnologijama i arhitekturama.

Odabrane su tehnologije koje poštuju principe objektno orijentiranog programiranja, čistog koda, koda koji je podložan izmjenama, dogradnjama i testiranju. Bitan faktor u odabiru korištenih tehnologija su bili aktuelni trendovi.

Unutar rada u početnim teoretskim poglavljima su objašnjene korištene tehnologije i alati. Neke od objašnjenih tehnologija su KOTLIN, KTOR, MongoDB, MVVM, DAGGER HILT, RETROFIT itd.

Puno pažnje unutar rada je posvećeno KOTLIN programskom jeziku koji je odabrani jezik za serversku i klijentsku aplikaciju.

Prikazana je i odabrana arhitektura aplikacije i način na koji se osigurava sigurnost podataka korisnika koji koriste aplikaciju.

Kroz poglavlje funkcionalnosti je kreirana tehnička dokumentacija klijentske aplikacije sa dijagramima procesa i snimcima ekrana uz pojedine isječke koda. Pojašnjen je način perzistencije podataka kroz lokalnu i serversku bazu podataka. Napravljeni su dijagrami korištene relacione i nerelacione baze podataka.

# Sadržaj

# 1. Uvod

Elektronski dnevnici su neizostavni i podrazumijevani u današnjim obrazovnim sistemima zbog velikih beneficija koje nude. Oni imaju puno veće i efikasnije funkcionalnosti od dnevnika na papiru. Nedostaci su nestanak električne energije ili prazna baterija na uređaju, dok za razliku od onih na papiru pružaju bolju sigurnost podataka sa spašavanjem podataka na serveru. To omogućuje korištenje istih podataka na više uređaja i lokacija. Puno je lakše vršiti izmjene nad elektronskim podacima nego nad onim na papiru. Statistika je jako bitna i prikupljanje podataka za statističke analize kao i izvještaje pomoću elektronskih dnevnika se izvodi u par koraka, dok sa podacima na papiru to predstavlja mukotrpan poduhvat.

Termin dnevnik posjeduje opće značenje te postoji jako puno vrsta dnevnika. Neke od njih su akademski i školski dnevnici, dnevnici ishrane, zdravstveni dnevnici, tajni dnevnici, radni dnevnici i tako dalje. Fokus ovoga rada jeste na akademskim i školskim dnevnicima koji opet mogu da imaju više različitih vrsta za različita polja primjene.

Jedna od verzija školskih dnevnika u upotrebi su dnevnici za kontinuirano ocjenjivanje naučenog gradiva. Ovo predstavlja jedan od najgeneralnijih i najzastupljenijih tipova u upotrebi u obrazovnim institucijama. Međutim, s obzirom da obrazovne institucije same po sebi posjeduju razlike i bave se različitim životnim oblastima time se razlikuje i način na koji se ocjenjuje naučeno gradivo. Razlike koje se pojavljuju su samo „estetske“. To znači da postoje razlike u tipovima podataka koji se skladište i načinu formiranja klasa podataka, ali funkciju koju dnevnik obavlja ostaje ista.

Iz navedenog zaključujemo da postojanje elektronskog dnevnika za obrazovne institucije, koji služi za kontinuirano ocjenjivanje naučenog gradiva, u bilo kojoj konkretnoj oblasti omogućuje vrlo lahko modificiranje i doradu tog dnevnika da ispunjava više konkretnih ciljeva ili da pruža neku generalnu uslugu.

Uopšteni naziv ovoga rada je “ Elektronski dnevnik“ dok konkretno projektovan i prilagođen dnevnik je elektronski dnevnik za obrazovne institucije. On služi za kontinuirano praćenje i verificiranje naučenog gradiva. Konkretne institucije za koje je ovaj dnevnik projektovan su škole Kur’ana, udruženja, fondacije i druge islamske škole koje posjeduju u svom programu rada učenje Kur'ana napamet.

Nakon analiziranja nekoliko škola Kur'ana u Sarajevu, uočen je zastario način ocjenjivanja učenika i vođenja evidencije o njihovom znanju. Postoji veliki broj polaznika ovih škola, gdje su polaznici osobe svih uzrasta bez ikakvih ograničenja. Također postoji veliki broj ovih i sličnih škola unutar i van BiH.

Da bi se unaprijedio i olakšao rad uposlenika ovih škola, kao i pružio polaznicima bolji uvid u svoj uspjeh i rad, kreiran je ovaj elektronski dnevnik.

Zbog lahke dostupnosti i ubjedljive popularnosti mobilnih uređaja u današnjem vremenu, kao klijentska aplikacija odabrana je mobilna aplikacija koju mogu koristiti svi korisnici ANDROID mobilnih uređaja.

Očekivanja od ovoga sistema u samom početku su bila da pronađe upotrebu u nekim od škola i da korisnici budu zadovoljni na koji način ova aplikacija unaprjeđuje njihov rad.

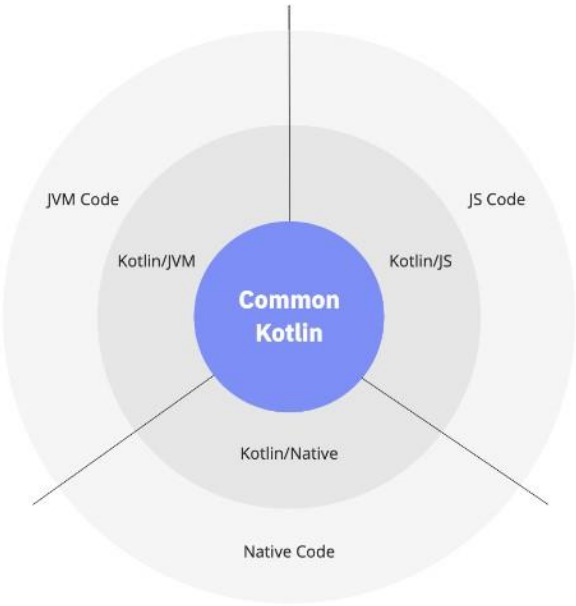
# 1. Tehnološki okvir

U tehnološkom okviru biti će obrađene korištene tehnologije, biblioteke, tipovi baza podataka i softverski okviri koji su korišteni za izradu ANDROID i serverske aplikacije. Kao posebno poglavlje biti će opisan KOTLIN programski jezik, odabran za programiranje cjelokupnog sistema.

## 1.1. Programski jezik – KOTLIN

KOTLIN je programski jezik za razvoj multiplatformskih aplikacija (engl. Multi platform apps), razvijen od strane JetBranis kompanije [3]. On spada u grupu statički tipiziranih jezika. To podrazumijeva da se provjera tipova varijabli obavlja za vrijeme kompajliranja programa, a ne za vrijeme njegovog izvršenja. U statički tipiziranim jezicima program se neće moći izvršavati ako posjeduje neku grešku, dok kod dinamički tipiziranih jezika (primjer *JavaScript*) program će se izvršavati dok ne dođe do mjesta gdje postoji greška kada će doći do narušavanja rada programa ili skripte.[4]

KOTLIN posjeduje i proširenje koje se naziva KOTLIN MULTIPLATFORM. On predsavlja SDK (engl. Software Development Kit) softverski razvojni alat koji omogućuje razvoj ANDROID, iOS i Web aplikacija u KOTLINU. Taj projekat je relativno mlad, još nije potpuno zaživio i nije korišten u ovom radu.
KOTLIN MULTIPLATFORM kod koji je za ANDROID se kompajlira u kod za Java virtuelnu mašinu (engl. Java Virtual Machine), dok kod za iOS operativne sisteme se pomoću LLVM (engl. Low level virtual machine) kompajlera kompajlira u izvorni mašinski kod. Također je moguć razvoj Web aplikacija gdje se Kotlin kod kompajlira u JavaScript kod. [2]




*Slika 1. Kotlin za više platformi. [2]*

KOTLIN je dizajniran da bude zamjena ili nasljednik JAVE. Sintaksa ova dva jezika nije jednaka i ne vrijede ista pravila. KOTLIN kod je dizajniran da može kooperirati sa Java kodom što je omogućilo postepenu adaptaciju i migraciju starih programa pisanih u Javi, bez da prethodno pisani kod bude potpuno neupotrebljiv.
KOTLIN se smatra novim i mladim jezikom, prvi put se pojavio 2011. godine, a prva verzija dostupna javnosti bila je 2012. godine. Prva stabilna verzija Kotlin 1.0 je zvanično objavljena 2016. godine.
Od 2019-te godine preporučeni jezik za razvoj ANDROID mobilnih aplikacija od strane *Google-a* je KOTLIN. [5]

Kotlin nudi velike prednosti od kojih su neke [1]:

- Kombinacija manje koda sa boljom čitljivošću
- „Zreo“ jezik i okruženje (KOTLIN je razvijan od 2011-te godine i potpuno je integrisan u ANDROID STUDIO)
- Podrška KOTLINA u mnogim ANDROID bibliotekama (korutine, proširene funkcije, lambda funkcije itd)
- Nudi divne funkcionalnosti u vidu null-sigurnosti i imutabilnosti
- Velika zajednica programera ( preko 60% aplikacija na *Google*-ovoj trgovini aplikacija (*Play Store*) su razvijene u *Kotlinu*)
- Velika upotreba u industriji

Većina trenutnih korisnika KOTLIN jezika baziraju se na razvoju ANDROID aplikacija, dok su ostale oblasti kao razvoj *Web* i *iOS* aplikacija trenutno jako malo zastupljene.

## 1.2. ANDROID aplikacija

Pokazna ANDROID aplikacija je rađena u ANDROID STUDIO integrisanom razvojnom okruženju (engl. Integrated Development Environment (IDE)). Izrađen je na InteliJ IDEA softveru firme JetBrains i predstavlja njegovu prilagođenu verziju za razvoj ANDROID aplikacija.

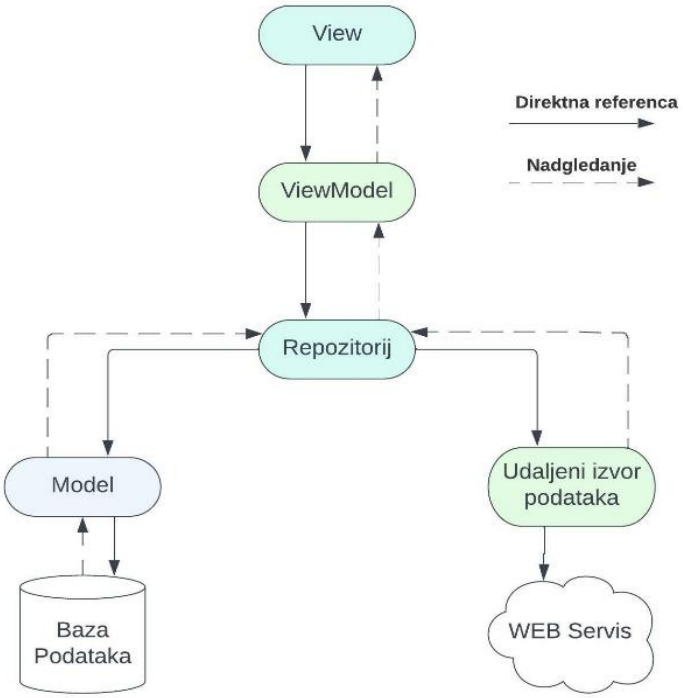
KOTLIN je potpuno podržan u ANDROID STUDIJU, oni zajedno predstavljaju jezik i okruženje preporučeno za razvoj ANDROID aplikacija od strane Google-a.

### 1.2.1. MVVM

Za razvoj kvalitetne aplikacije od krucijalne važnosti je korištenje adekvatne arhitekture koja će pomoći da kod koji se piše bude lagahan za testiranje, održiv i pogodan za nadogradnje.

Za pisanje kvalitetnog koda koji se smatra „čistim“ (engl. Clean Code) potrebo je poštovati i slijediti određena pravila kao što su pravilno imenovanje (klasa, atributa, varijabli, funkcija itd), a funkcije treba da rade samo jednu stvar. Oko pisanja komentara postoji razilaženje. Neki navode da treba pisati razumljive komentare, a drugi navode da komentara uopšte ne treba biti jer kod treba „sam o sebi da priča“. Vrlo je bitno slijediti šablone dobrog dizajna, ali i nakon svega toga kod može da ne zadovoljava određene karakteristike koje su jako bitne za kvalitetan kod. Ono što će upotpuniti pravila pisanja kvalitetnog koda jeste slijeđenje nekog arhitekturnog šablona za organizaciju koda.[6]

Najpopularnija arhitektura u razvoju Android aplikacija je MVVM → Model-Pogled-Pogled Model (engl. Model-View-ViewModel).




*Slika 2. MVVM dijagram[7]*

Glavni cilj svake arhitekture jeste razdvajanje odgovornosti koje se kod MVVM-a odvija pomoću Modela, View-a i ViewModel-a.

Dodatak MVVM strukturi predstavlja Repozitorij koji dohvaća podatke ViewModel-u. Njegova uloga je da odluči sa kojeg izvora će da preuzme podatke, sa lokalnog ili sa nekog udaljenog servera. Nakon što ih preuzme može ih učiniti dostupnim ViewModel-u koji uzima „gotove“ podatke i „ne brine“ za njihov izvor.

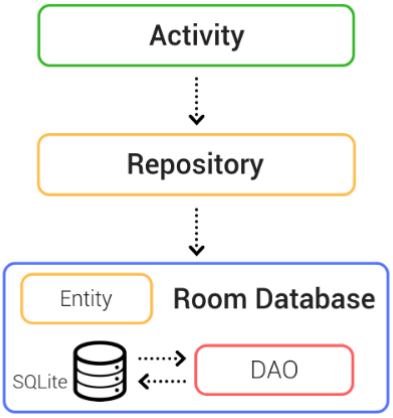
View ili pogled upravlja sa onim što korisnik vidi i sa čim komunicira na ekranu. Drugim riječima View radi sve ono što rade Fragmenti ili Aktivnosti. View nikako ne bi smio sadržavati neku poslovnu logiku, dohvaćanje podataka ili slično.

ViewModel je veza između pogleda i poslovne logike. Dobavlja podatke View-u iz Repozitorija. S obzirom da postoji direktna veza samo u smjeru od View-a ka ViewModelu, a to znači da ViewModel uopšte ne zna koji pogledi ga koriste. Cilj ViewModela jeste da podatke učini vidljivim View-u i View jednostavno samo nadgleda podatke iz ViewModela. ViewModel podatke učini dostupnim i obavještava o promjenama sve one koji ga nadgledaju. Ovakva vrsta nadgledanja moguća je sa nekoliko biblioteka, ali najpoznatija je LiveData.

Na sličan način na koji View posmatra podatke sa ViewModela, tako i ViewModel posmatra podatke sa Repozitorija. ViewModel ima direktnu referencu na Repozitorij, a Repozitorij ne šalje podatke direktno prema ViewModelu nego ih ViewModel nadgleda na osnovu te reference. Na sličan način se odvija komunikacija između Repozitorija i izvora podataka.

Model predstavlja apstrakciju izvora podataka. [7]

### 1.2.2. SQLITE i ROOM




*Slika 3.Pristup SQLite bazi pomoću ROOM biblioteke [10]*

Sve aplikacije koje zahtijevaju prikupljanje i skladištenje podataka za vrijeme njihovog rada posjeduju mehanizam za čuvanje i spremanje podataka u lokalnu bazu podataka. Kao najčešće korištena baza podataka u upotrebi na Android aplikacijama pojavljuje se SQLite zbog svoje jednostavnosti, ekonomičnosti i malih zahtjeva za resursima uređaja. [8]

Na ovom projektu je korištena SQLite baza podataka koja omogućuje da se na struktuiran način sačuvaju podaci. Ona predstavlja relacionu bazu podataka čiji su podaci privatni i dostupni samo za onoga ko ih je kreirao, a to predstavlja ono što nam je potrebno na Android uređajima.

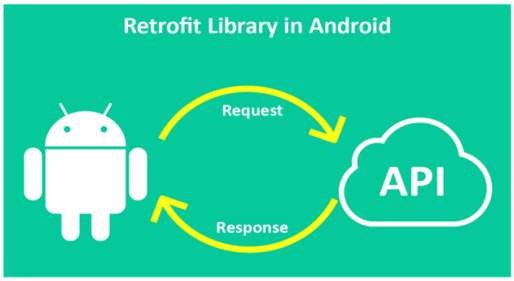
Prilikom komunikacije sa SQLite bazom preporučeno je koristiti neku biblioteku koja vrši apstrakciju komunikacije i olakšava rad. U ovom projektu odabrana je ROOM biblioteka koja omogućuje jednostavan način pristupa SQLite bazi uz potpunu iskoristivost njene snage.

Room pruža sljedeće pogodnosti [9]:

- Provjera SQL-upita tokom kompajliranja.
- Praktične anotacije koje minimiziraju ponavljajući i pogreškama podložan kod.
- Pojednostavljena migracija baze podataka.

Komponente ROOM biblioteke su Database (apstraktna klasa sa anotacijom @Database), Entity (klasa koja predstavlja kolone tabele u bazi) i DAO (engl. Data Access Objects) koji omogućuje korištenje SQL upita uz dodatne mogućnosti. [9]

### 1.2.3. RETROFIT




*Slika 4. RETROFIT klijent za Android [11]*

Pristup izgradnje mobilnih aplikacija koje posjeduju samo neki interfejs i možda spašavaju lokalno podatke u bazu, nije dovoljan za potpuno zadovoljstvo korisnika. Mnogo bolji pristup je da podatke spašavamo također na server i onda omogućimo korisnicima da podatke preuzimaju. Ukoliko uz ovo dodamo i želje korisnika da te podatke kreiraju i mijenjaju, dolazimo do zaključka da nam treba neki servis koji će omogućiti sve te operacije sa klijentskog uređaja.

Kao najbolje rješenje postoje REST (engl. *Representational state transfer*) pravila koja zajedno sa API (engl. *Application Programming Interface*) formiraju takozvani REST API koji koriste serveri da komuniciraju sa klijentima.

Serveri koji koriste REST API posjeduju slijedeće osobine [14]:

- Jednostavan i standardizovan pristup komunikaciji
- Skalabilnost bez stanja
- Velike performanse i keširanje

Za korištenje REST API-ja potrebna je komunikacija preko HTTP-a [12] (engl. Hyper Text Transfer Protokol), podrška za serijalizaciju, deserijalizaciju i JSON [13] (engl. JavaScript Object Notation). Također potrebno je da se ova komunikacija uspostavlja i obavlja asinhrono van glavne niti, da ne dođe do blokade korisničkog interfejsa. [14]

Da bi se sve ovo omogućilo na ANDROIDU potrebno je puno linija koda koji se ponavlja više puta. Iz tih i još mnogo drugih razloga kompanija Square kreirala je biblioteku koja je javno dostupna pod imenom RETROFIT. Ova biblioteka je izuzetno popularna među ANDROID programerima zbog lakog korištenja i odličnih performansi.

RETROFIT predstavlja REST (engl. *Representational state transfer*) klijenta za ANDROID i olakšava web servise.

Međutim RETROFIT nije jedini klijent koji omogućuje navedenu uslugu. Kao alternative pojavljuju se KTOR, VOLLEY i druge manje poznate biblioteke. Za ovaj projekt odabran je RETROFIT.

### 1.2.4. DAGGER HILT

U objektno orijentisanom programiranju postoje velike ovisnosti između klasa. Sve klase bi trebale da posjeduju samo jednu vrstu odgovornosti, dok za ostale odgovornosti uključuju se druge klase koje ih preuzimaju. Injektiranje ovisnosti predstavlja ubacivanje instanci drugih klasa u konstruktor klase koja ga koristi. Klase ne bi trebale same da kreiraju svoje ovisnosti, nego bi trebale biti instancirane negdje drugo i proslijeđene klasi koja ih koristi.

Injektiranje ovisnosti je moguće bez bilo kakve biblioteke, ali često učini kod obimnim sa puno konstruktora i ponavljajućeg koda. Također ako promijenimo neki od konstruktora moramo na svakom mjestu u kodu gdje ga pozivamo da vršimo izmjene.

Iz navedenih razloga su kreirane biblioteke koje same vrše injektiranje ovisnosti (engl. Dependency Injection). Dvije najpoznatije biblioteke su DAGGER i COIN. Postoji još jedna alternativa a to je DAGGER HILT, biblioteka kreirana kao apstrakcija DAGGERA prilagođena za ANDROID aplikacije. DAGGER HILT je odabran za ovaj projekt, te će biti ukratko opisan.

DAGGER je potpuno statički, kompajlirajući, softverski okvir za ubrizgavanje ovisnosti za JAVU, KOTLIN i ANDROID. Kreiran je od kompanije Square, a trenutno održavan od strane Google-a. DAGGER generiše sličan kod koji bi se pisao i manuelno. On interno kreira grafik objekata da bi mogao pronaći način da na potrebnom mjestu dostavi instancu neke klase.[15]

Ciljevi HILTA su [15]:

- Pojednostaviti infrastrukturu DAGGER-a za ANDROID aplikacije.
- Kreiranje standardnog skupa komponenti i opsega za lakše postavljanje, čitljivost/razumijevanje i dijeljenje koda između aplikacija.
- Pružanje jednostavnog načina za pružanje različitih veza za različite tipove izrade softvera (npr. testiranje, otklanjanje grešaka ili izdavanje (engl. *Deployment*)).

DAGGER HILT automatski generiše kod, koji je zamjena za kod koji bi inače pisali ručno. Budući da se kod generiše u vrijeme kompajliranja, on je učinkovitiji od drugih tipova koji se ne kompajliraju nego izvršavaju svaki put kada se aplikacija pokrene. Primjer takve biblioteke je Google GUICE čiji su nedostaci usporavanje rada aplikacije, neočekivane greške u radu aplikacije i slično. Iz navedenih razloga GUICE se više skoro nikako ne koristi a Google preporučuje upotrebu upravo DAGGER-a.

## 1.3. Serverska aplikacija

Serverska aplikacija za ovaj rad je rađena u InteliJ IDEA integrisanom razvojnom okruženju (engl. Integrated development environment (IDE)) kompanije JetBrains.

KOTLIN je potpuno podržan u InteliJ IDEA kao i KTOR softverski okvir (engl. framework) koji je odabran za razvoj serverske aplikacije. Ktor će biti opisan u narednom poglavlju.

### 1.3.1. KTOR


*Slika 5. KTor logo [17]*

KTOR je softverski okvir (engl. framework) koji omogućuje programerima da pišu asinhrone klijentske i serverske aplikacije u KOTLINU. Iako nije u potpunosti kompatibilan sa svim, KTOR cilja na više platformi kao što su JVM, JavaScript, Android i iOS. [16]

U ovom tekstu će fokus biti na serverskoj strani KTORA, jer je on odabran za serversku aplikaciju. KTOR se može koristiti na klijentskoj strani kao zamjena za RETROFIT kako samo naveli u tom poglavlju. KTOR je trenutno potpuno kompatabilan da radi sa JVM (engl. Java Virtual Machine). Može se razviti (engl. Deploy) s nekim kontejnerima kao što su Tomcat, Jetty ili putem Dockera.

Kada se KTOR server pokrene, on prihvata zahtjeve na određenom portu. Za upravljanje vezom i svim ostalim radnjama, KTOR radi sa cjevovodima (engl. pipelines) i oni su to što čini KTOR asinhronim.

Pipeline ili cjevovod je asinhroni lanac kodnih blokova ili funkcija koje pokreću asinhrone radnje, sa implementacijom korutina. Cjevovod konfigurišu moduli i funkcije učitane pri pokretanju aplikacije. Aplikacija je sama po sebi cjevovod (engl. Pipeline), koja će obraditi dolazni zahtjev kroz specifičnu putanju, uspostavljenu pri pokretanju servera.

Na primjer, u slučaju HTTP poziva, cjevovod aplikacije (engl. Application Call Pipeline) će izvršiti više faza, kao što je obrada zahtjeva ili čak slanje HTTP odgovora.

KTOR moduli su naslijeđene funkcije klase Application. Oni predstavljaju konfiguraciju aplikacije. Možemo konfigurisati server sa njegovim portom, ili čak instalirati specifične opcije za svaki modul (rutiranje, sesija, izvještavanje...). [17]

Opcije (engl. Features) proširuju mogućnosti aplikacije, neke od njih su [17] :

- Rutiranje: definisanje HTTP ruta koje će rukovati nadolazećim zahtjevima.
- Sesija: održavanje veze, ili informacija koje identificiraju klijenta, pružajući mogućnost usluge sa kontekstom. Naprimjer, ograničavanje prava pristupa pojedinim korisnicima.
- WebSockets: podrška za KTOR, koja je više od jednostavnih HTTP zahtjeva, i cilja održavanje veze servera i klijenta, omogućujući serveru da šalje notifikacije klijentu.

### 1.3.2. MongoDB

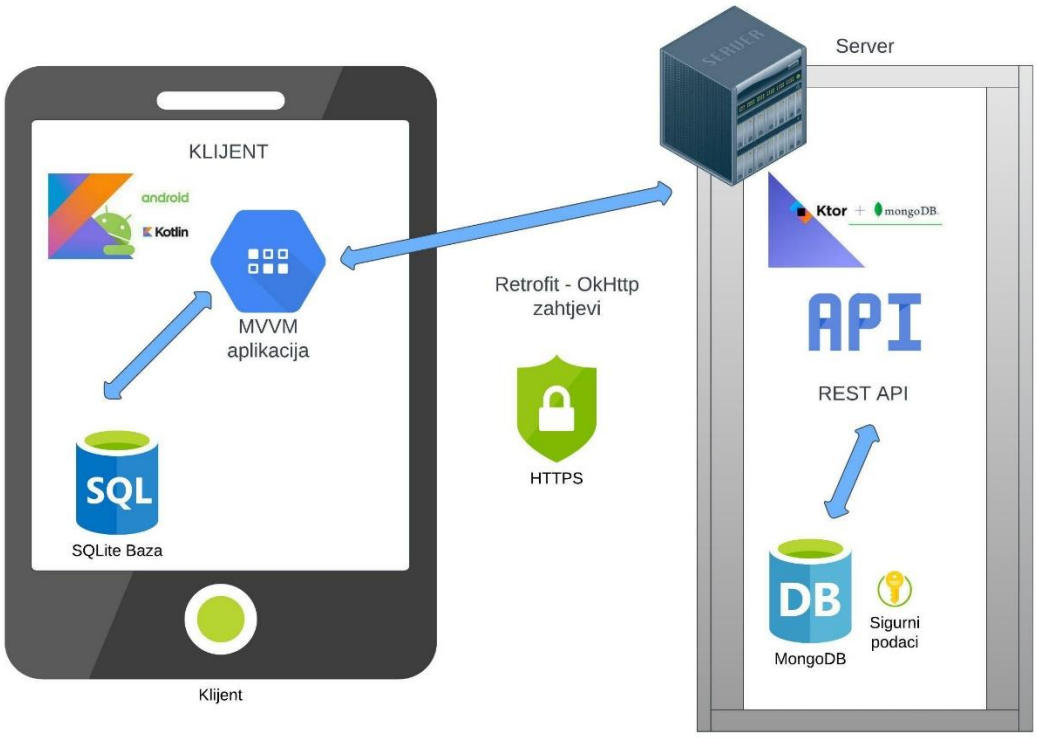

Slika 6. *MongoDB logo [18]*

Neralcione ili još poznate kao „Ne-SQL“ (engl. No-SQL) baze podataka vrše spašavanje i dohvaćanje podataka iz baze na drugačiji način u odnosu na relacione. Kako samo ime ukazuje, ne postoji relacija između podataka, također podaci se ne skladište u tabele i nemaju osobine kao što su primarni, strani ključevi i ograničenja. Umjesto toga podaci se spašavaju kao neka vrsta dokumenta a obično je to JSON.

Za ovaj projekt je korištena nerelaciona baza, podaci se spašavaju kao kolekcija dokumenata a ne u tabele. Razlog odabira nerelacione baze jeste obično veća brzina u odnosu na relacione. One pružaju laganu razmjenu s velikim količinama podataka kod velikih opterećenja od strane korisnika. Nerelacione posjeduju određena ograničenja jer ne dozvoljavaju komplikovane upite kao relacione, što u ovom projektu neće biti problem zbog jednostavnijeg dizajna podataka.

MongoDB spada u NoSQL baze podataka. Otvorenog je koda koji je dostupan na raznim platformama poput GitHub-a. Dokumenti se pohranjuju i preuzimaju kod kao JAVASCRIPT objekti. Tačnije MongoDB koristi BSON (engl. Binary JavaScript Object Notation) za kodirano skladištenje podataka što mu omogućuje jedno od najboljih indeksiranja i upita u industriji. [19]

# 2. Arhitektura




Slika 7. *Arhitektura sistema*

Odabrana je Klijent-Server arhitektura cjelokupnog sistema.

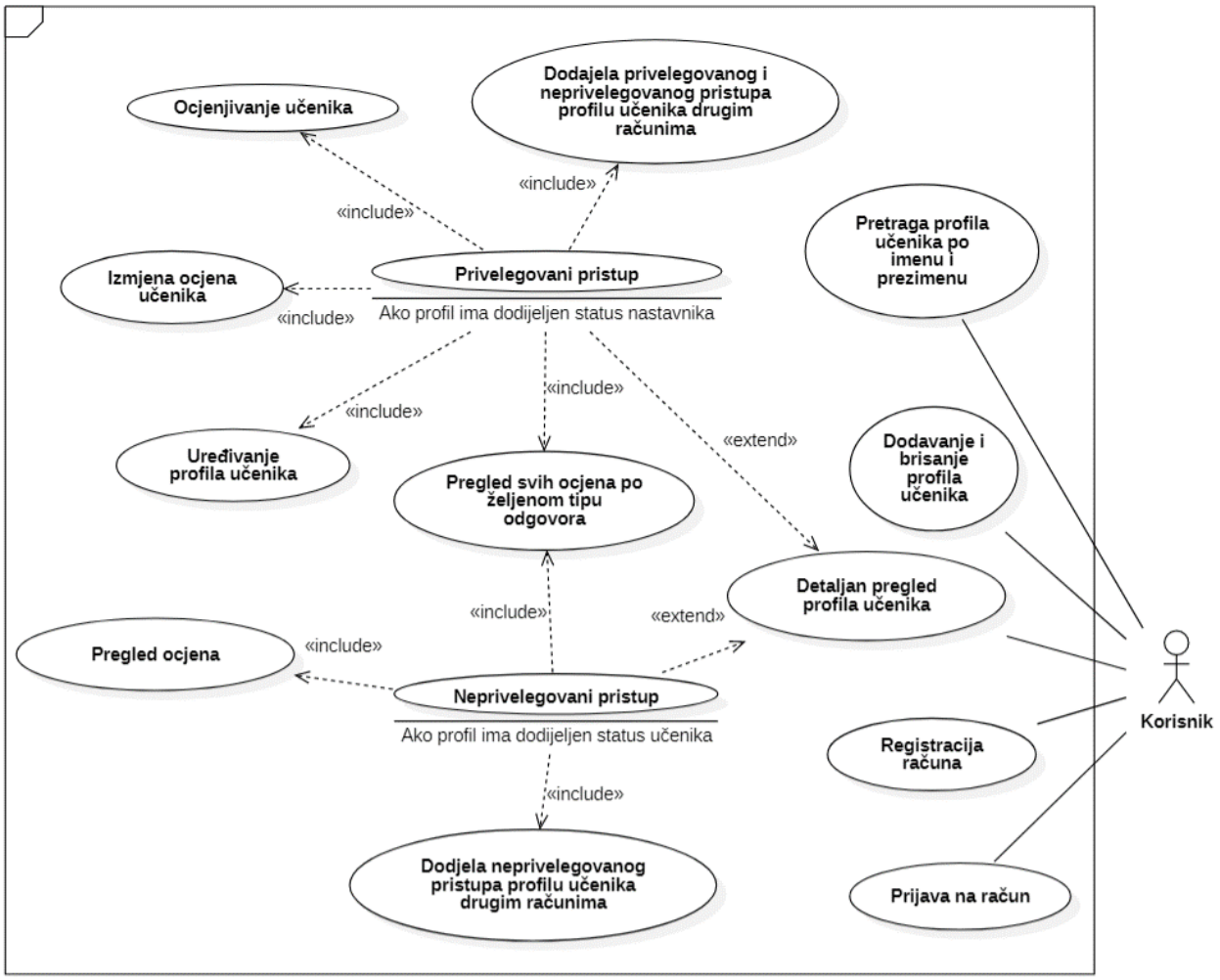
Klijentska aplikacija izrađena u KOTLINU uz poštovanje MVVM principa nalazi se na uređajima korisnika. Aplikacija posjeduje lokalnu bazu koja omogućuje korištenje aplikacije bez internet konekcije. Kada se uspostavi konekcija vrši se ažuriranje podataka na serveru onim koji su lokalno spašeni.

Posjedovanje servera osigurava sigurnost i perzistenciju podataka neovisno o uređaju koji korisnik koristi. Korisnik može na taj način pristupiti svojim podacima sa različitih uređaja bez gubljenja podataka. Na serveru to omogućuje serverska aplikacija koja komunicira sa *MongoDB* nerelacionom bazom podataka. Ova baza posjeduje sve podatke o korisnicima kao i podatke koje poslužuje korisnicima.

Važno je napomenuti da je sav saobraćaj između klijenta i servera odvija putem HTTPS protokola a kriptuje pomoću SSL algoritma. Također korisničke lozinke i drugi osjetljivi podaci su heširani i ne spremaju se u izvornom obliku.

# 3. Dijagram slučaja upotrebe (engl. *USE CASE DIAGRAM*)

Na slijedećem dijagramu je prikazana interakcija između sistema i korisnika. On predstavlja funkcionalnosti na visokom nivou i granice sistema.




Slika 8. *USE CASE dijagram*

## 3.1. Učesnici u sistemu (Akteri)

Jedini učesnik sistema je korisnik. Postojanje aktera nastavnika i učenika nije obavezno. Aplikacija omogućuje da učenici budu i nastavnici, što može biti korisno prilikom međusobne provjere znanja između učenika. To je česta pojava u oblasti sticanja znanja za koju je namijenjen elektronski dnevnik razvijen u okviru ovog završnog rada.

Prijava i korištenje aplikacije je jednaka za nastavnike i učenike kao korisnike. Razlika se pojavljuje dodavanjem profila učenika u sistem od strane bilo kojeg korisnika. Korisnik koji doda novog učenika automatski postaje privilegovani korisnik nad tim profilom učenika i posjeduje status nastavnika.

## 3.2. Slučajevi upotrebe

Svi korisnici aplikacije koriste jedinstven način registracije i prijave. Svi mogu dodati novi profil učenika i vršiti pretragu svojih učenika.

Kada korisnik doda novog učenika ostvaruje privilegovan pristup i sve mogućnosti koje taj pristup nudi. Privilegovan pristup predstavlja status nastavnika i na taj način je korisnik dodao novog učenika kojeg ima pravo ocjenjivati.

Korisnik ostvaruje neprivilegovani pristup ukoliko mu drugi korisnik isti omogući nad nekim profilom učenika. Neprivilegovani pristup predstavlja pristup sa statusom učenika i namijenjen je za učenike kao korisnike da im omogući samo pregled onoga što nastavnici uređuju.

Dodjeljivanje privilegovanog i neprivilegovanog pristupa između korisnika će biti detaljnije objašnjeno u posebnom poglavlju.

# 4. Funkcionalnosti

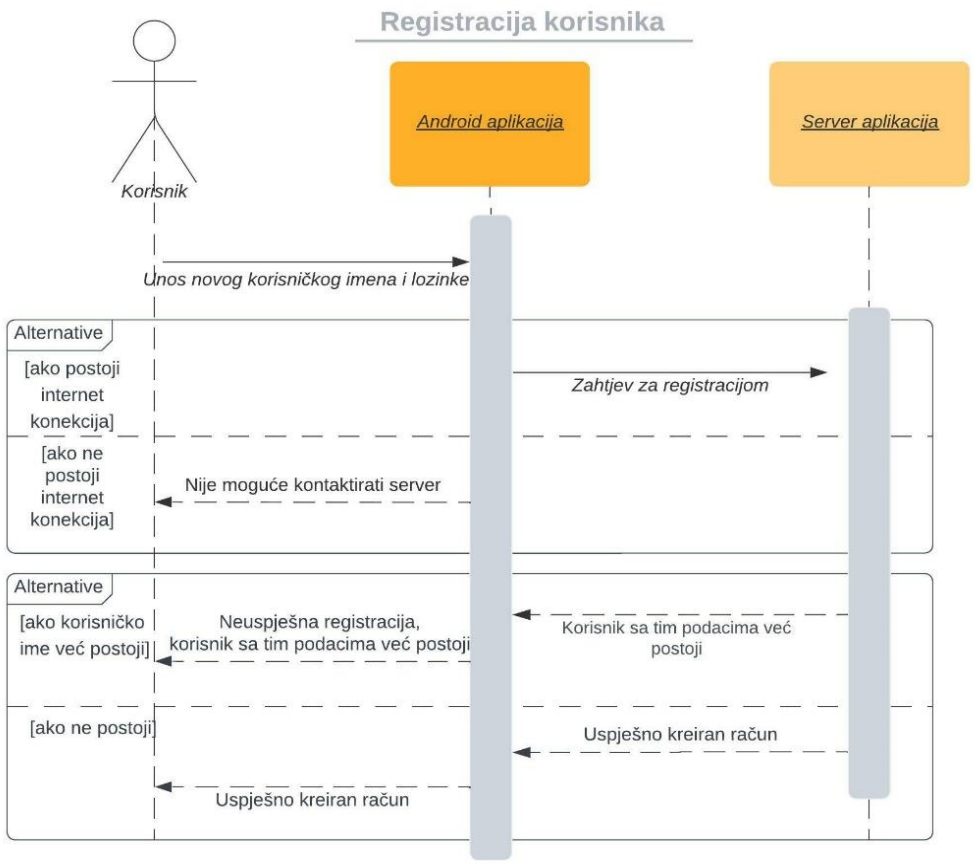
U ovom poglavlju će biti detaljno objašnjene sve funkcionalnosti koje nudi aplikacija kroz prezentaciju procesa pomoću dijagrama sekvence, nekih isječaka koda koji vrše dio navedene funkcionalnosti na klijentskoj aplikaciji i slika ekrana aplikacije. Dijagrami sekvence će prikazati komunikaciju između korisnika i sistema kao i interakciju mobilne aplikacije sa serverskom aplikacijom. Dok će slike ekrana prikazati implementirane funkcionalnosti na mobilnom uređaju.

## 4.1. Registracija računa

Prvi uslov za korištenje aplikacije je registracija sa korisničkim imenom i lozinkom. Da bi se uspješno izvršila registracija potrebna je komunikacija sa serverom a to zahtjeva internet konekciju. Ukoliko korisnik ne posjeduje internet konekciju niti registrovan račun na koji se prijavio, ne može koristiti opcije koje ova aplikacija nudi.

Još jedan od uslova uspješne registracije je jedinstveno korisničko ime. U slučaju da korisnik proba registraciju sa već postojećim korisničkim imenom biti će obaviješten o tipu greške.

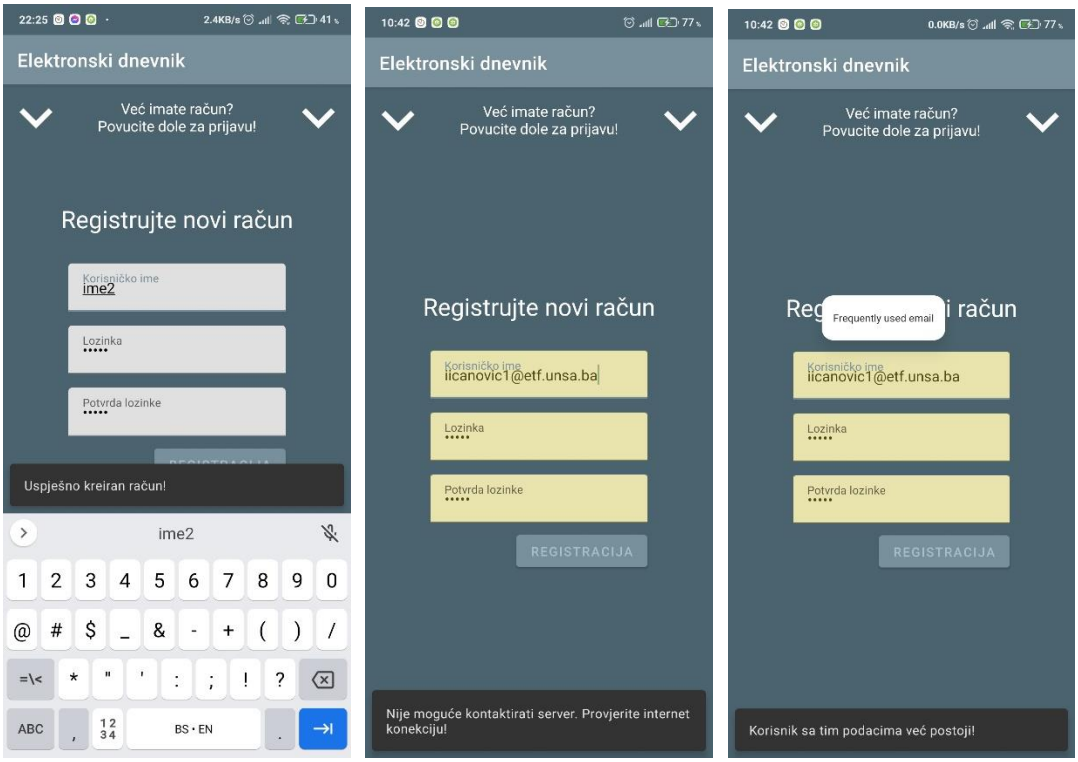
Lozinka koju korisnik unese će prilikom transporta podataka biti zaštićena SSL algoritmom prilikom prenosa preko HTTPS protokola, također heširana na serveru i spašena u heširanom obliku unutar baze. Na taj način je sistem osiguran protiv malverzacija i zloupotrebe korisničkih podataka.




*Slika 9. Dijagram sekvence – Registracija korisnika*

```
suspend fun register(email : String, password :String)  = withContext(Dispatchers.IO) { // osi
    try {
        val response = studentApi.register(AccountRequest(email, password))
        if(response.isSuccessful && response.body()!!.successful){
            Resource.success(response.body()?.message) ^withContext
        }else{
            Resource.error( msg: response.body()?.message ?: response.message(), data: null) ^with
        }
    }catch (e: Exception){
        Resource.error( msg: "Nije moguće kontaktirati server. Provjerite internet konekciju!",
            data: null) ^withContext
    }
}
```

*Isječak koda 1. Registracija korisnika, funkcija unutar repozitorija zadužena za asinhrone pozive prema serveru*




Slika 10. *Snimci ekrana – Registracija korisnika*

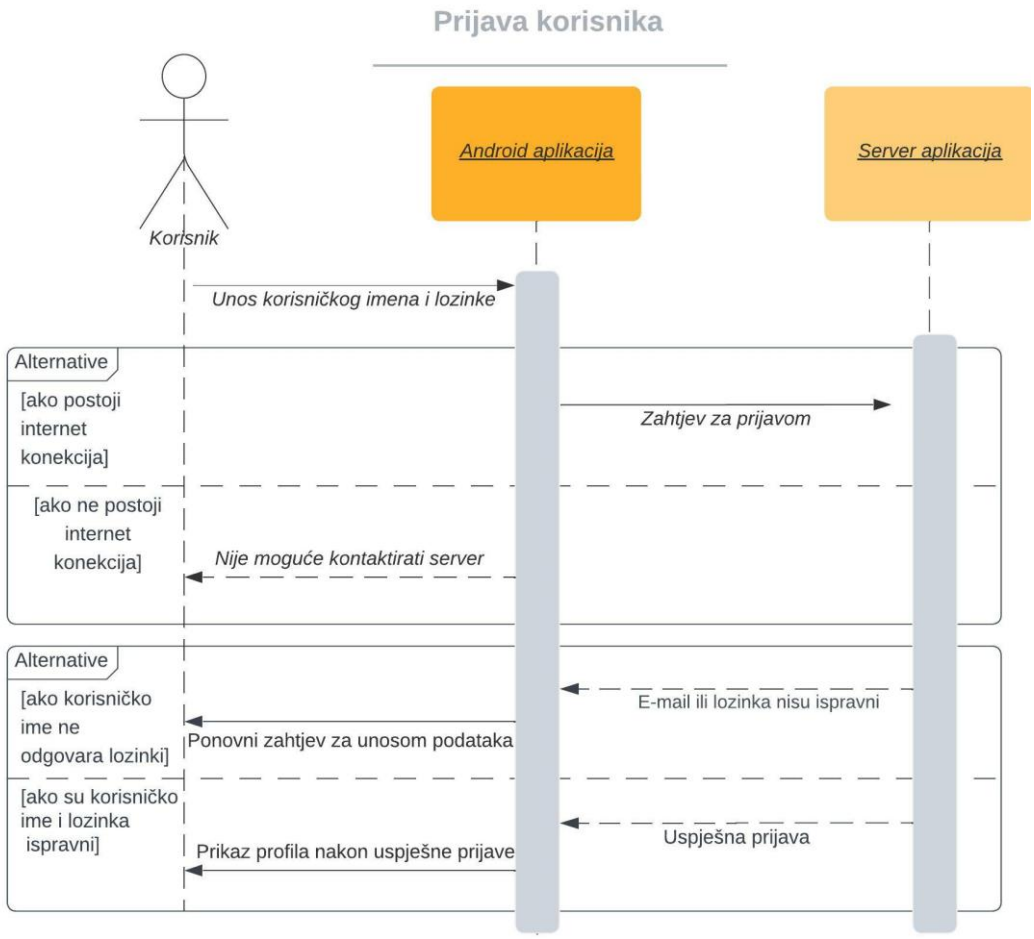
## 4.2. Prijava na aplikaciju

Prvi uslov za prijavu je registracija, to jeste posjedovanje korisničkog računa. Da bi se uspješno izvršila prijava potrebna je komunikacija sa serverom, a to zahtjeva internet konekciju. Ukoliko korisnik ne posjeduje internet konekciju i nije se prijavio, ne može koristiti opcije koje ova aplikacija nudi.

Nakon što korisnik uspostavi internet konekciju može ostvariti uspješnu prijavu. Za to je potrebno poznavanje vlastitog korisničkog imena i lozinke.

Ukoliko korisnik unese pogrešne podatke biti će obaviješten o tipu greške.

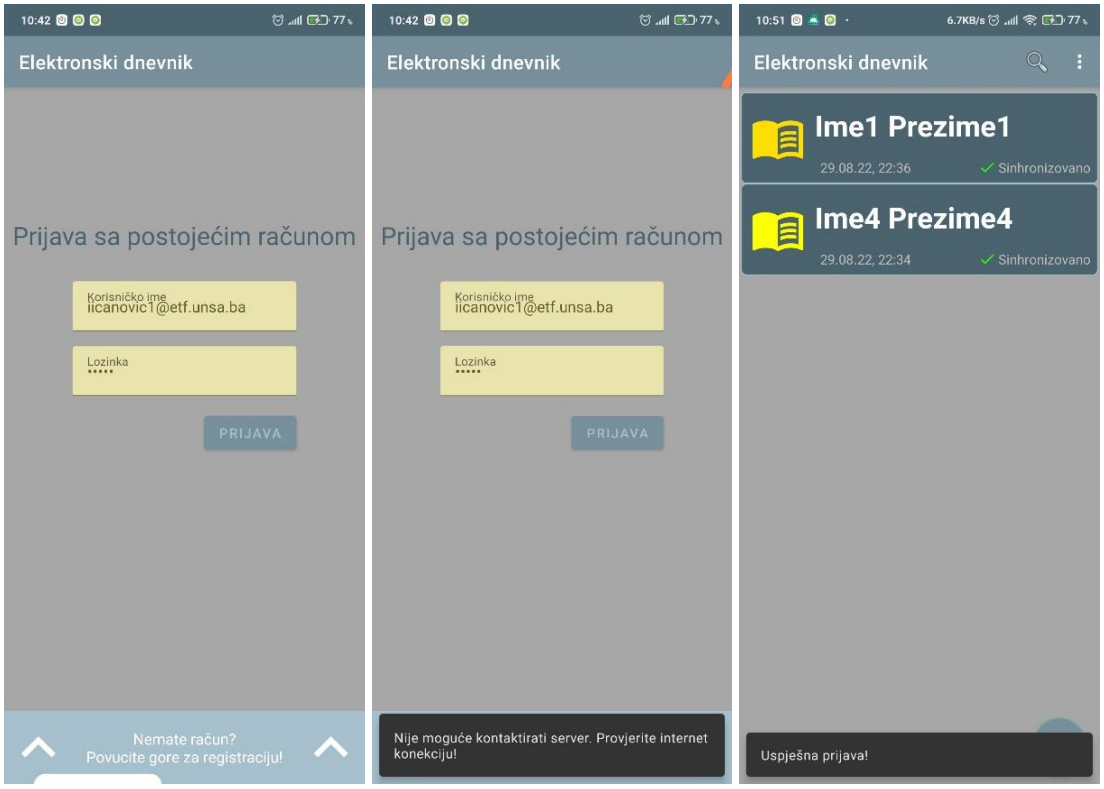
Nakon što je obavljena uspješna prijava moguće je koristiti sve funkcionalnosti aplikacije bez internet konekcije.




Slika 11. *Dijagram sekvence – Prijava korisnika*

```
suspend fun login(email : String, password :String)  = withContext(Dispatchers.IO) {
    try {
        val response = studentApi.login(AccountRequest(email, password))
        if(response.isSuccessful && response.body()!!.successful){
            Resource.success(response.body()?.message)
        }else{
            Resource.error( msg: response.body()?.message ?: response.message(), data: null)
        }
    }catch (e: Exception){
        Resource.error( msg: "Nije moguće kontaktirati server. Provjerite internet konekciju!"
            , data: null)
    }
}
```

*Isječak koda 2. Prijava korisnika, funkcija unutar repozitorija zadužena za asinhrone pozive prema serveru*




*Slika 12. Snimci ekrana, prijava korisnika*

## 4.3. Dodavanje i uređivanje učenika

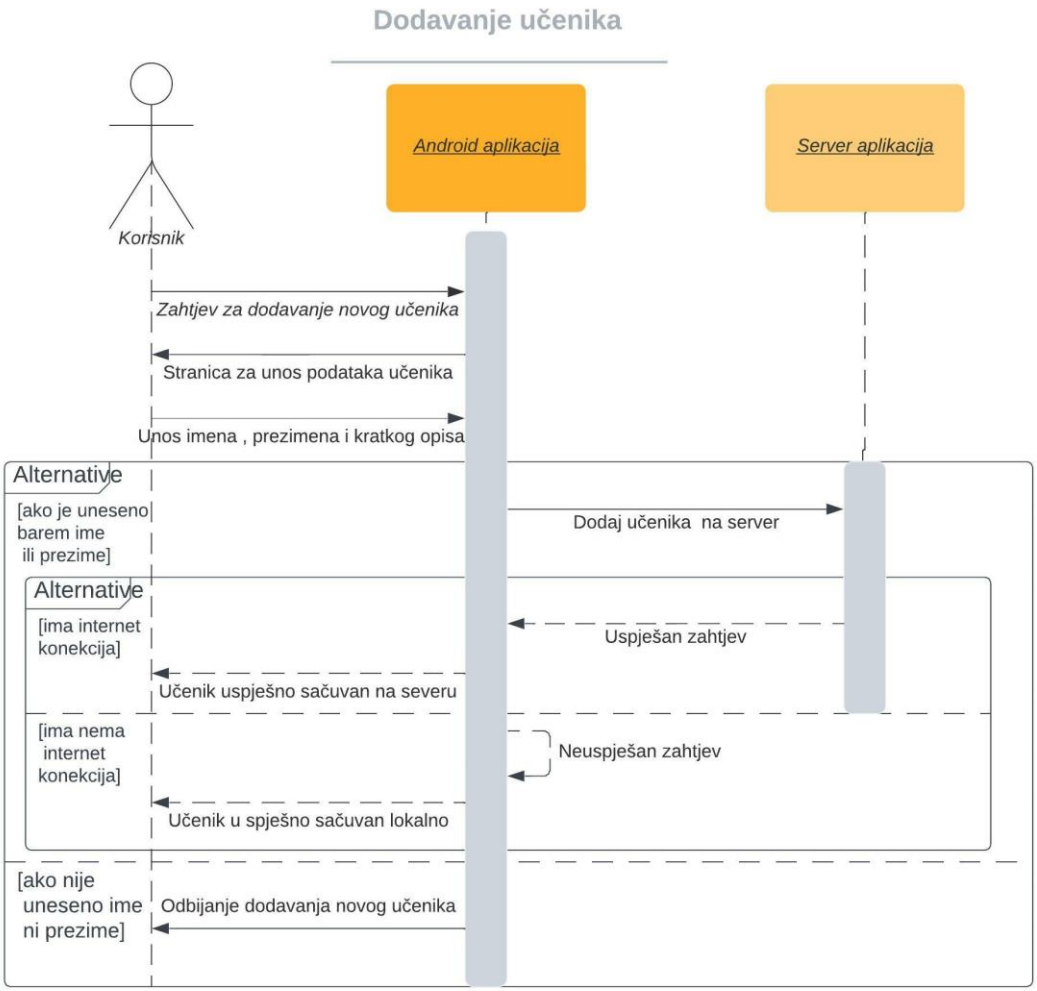
Dodavanje novih učenika predstavlja funkcionalnost dostupnu svim korisnicima i ona predstavlja kreiranje profila učenika sa imenom, prezimenom i kratkim opisom koji definišu novog učenika. Za uspješan završetak ove radnje, uslov je da se podacima popuni polje za ime ili za prezime dok je detaljan opis opcionalan.

Nakon uspješnog dodavanja učenika, korisnik koji je tu radnju obavio postaje i privilegovani korisnik nad tim profilom učenika i može ga ocjenjivati. Korisnik tada može i urediti osnovne podatke koje smo naveli da definišu učenika kada god to želi.

Korisnik tada ostvaruje i pristup opcijama dodavanja pristupa profilu učenika drugim korisnicima. Drugačije rečeno, korisnik koji je napravio profil učenika postaje njegov nastavnik, nakon toga može svom učeniku koji mora biti registrovani korisnik aplikacije dodijeliti pristup njegovom profilu, to jeste pružiti mu pregled ocjena.

Ova funkcionalnost ne zahtjeva internet konekciju. Ukoliko korisnik ne posjeduje konekciju, podaci se spašavaju lokalno i korisnik se obavještava o tome da nije bilo moguće kontaktirati server.

Omogućeno je i vizuelno prikazivanje koji dijelovi podataka ( profili učenika ) su sinhronizovani sa serverom, a koji nisu. Ukoliko je korisnik uređivao ili dodao novi profil učenika dok je imao internet konekciju, uz učenika će biti prikazana zelena kvačica sa tekstom „sinhronizovano“ a u suprotnom crveni „X“ sa tekstom „nije sinhronizovano“.

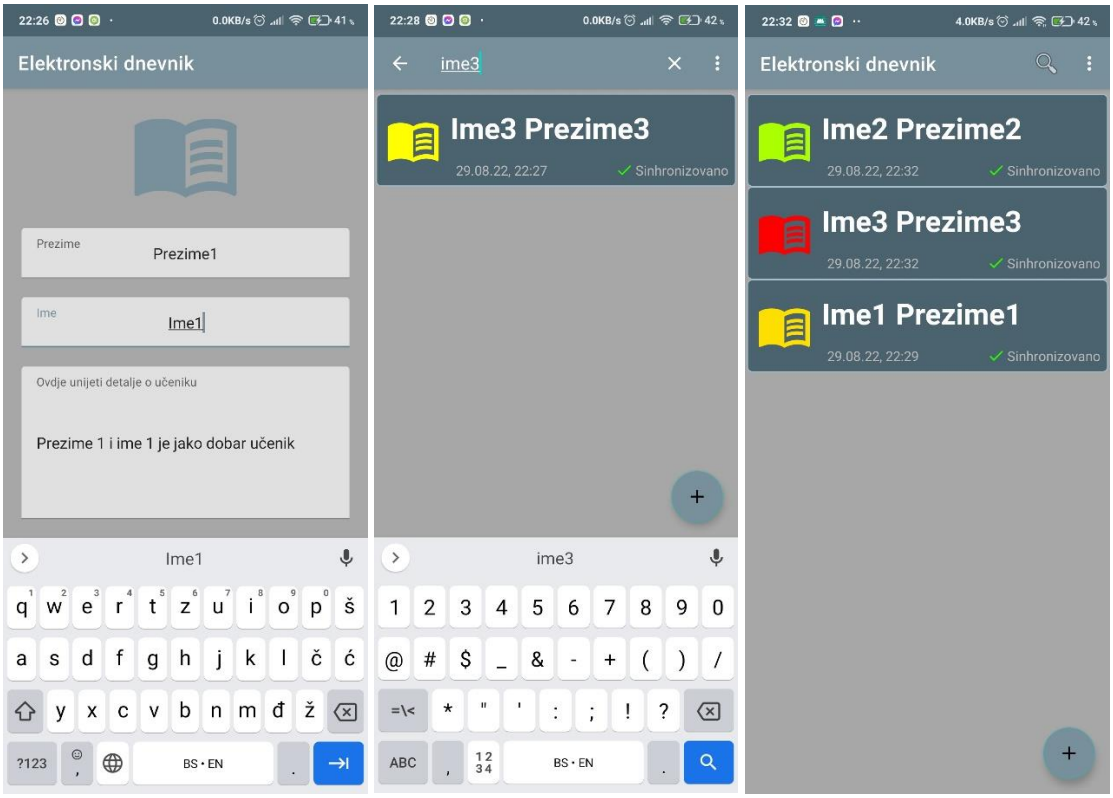



*Slika 13. Dijagram sekvence, dodavanje učenika*

```
suspend fun insertStudent(student: Student) {
    val response = try {
        studentApi.addStudent(student)
    }catch (e: Exception){
        null
    }
    if (response != null && response.isSuccessful){
        studentDao.insertStudent(student.apply { isSynced = true })
    }else{
        studentDao.insertStudent(student)
    }
}

suspend fun insertStudents (studets: List<Student>){
    studets.forEach{student -> insertStudent(student)}
}
```

*Isječak koda 3. Dodavanje učenika, funkcije koja dodaju učenike lokalno i na server*




*Slika 14. Snimci ekrana, dodavanje učenika*

## 4.4. Brisanje učenika

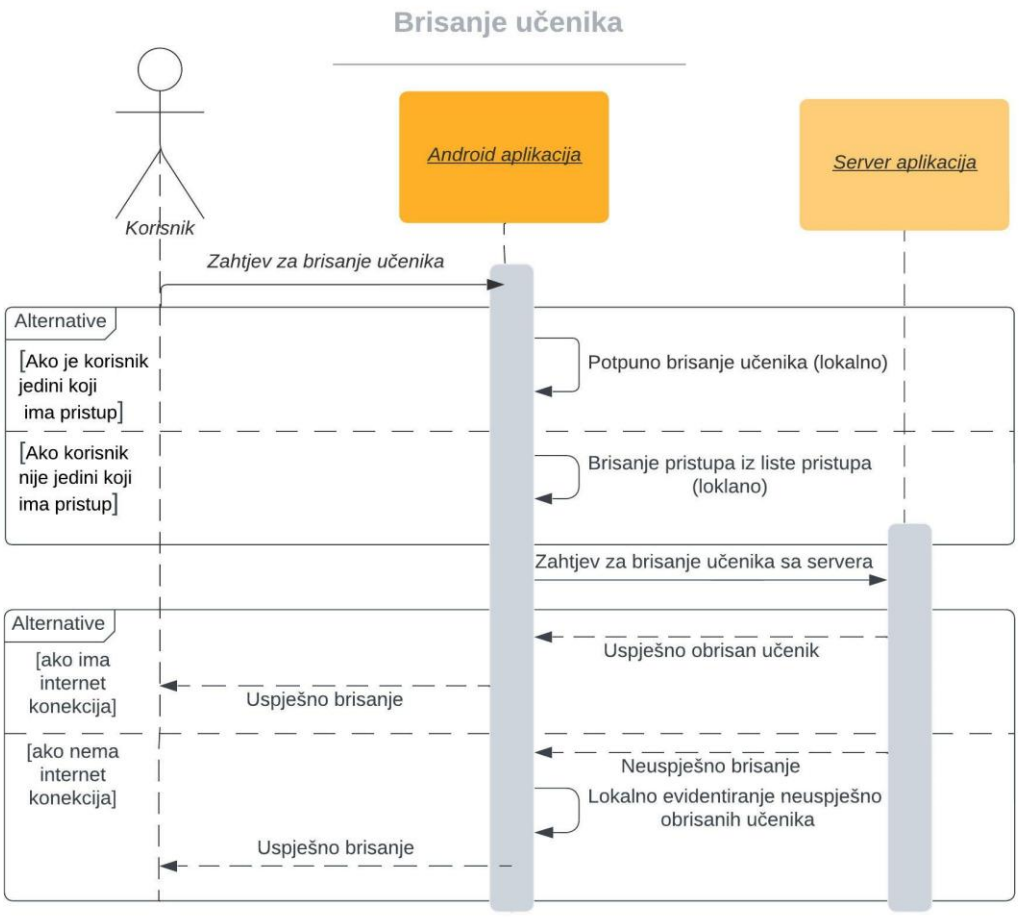
Profil učenika može obrisati bilo koji korisnik koji ima bilo koju vrstu pristupa nad tim profilom. Ta radnja brisanja se neće u svakom slučaju na isti način obaviti. Postoje različiti scenariji koji uzrokuju slijedeća ponašanja :

| **Scenario** | **Ponašanje** |
|---|---|
| Korisnik koji jedini posjeduje privilegovan ili neprivilegovan pristup profilu tog učenika briše taj profil učenika | Profil učenika se potpuno briše |
| Korisnik sa privilegovanim pristupom koji nije jedini koji ima pristup podacima (profilu) učenika briše profil učenika | Profil učenika ostaje dostupan svim ostalim korisnicima koji imaju pristup i privilegija pristupa se ne mijenja |

Drugim riječima, osigurano je da učenik ne može obrisati svoj profil jer nastavnik ima pravo pristupa tom profilu. Može ga obrisati za sebe i neće mu više moći pristupiti dok ponovo ne dobije pristup od nekog drugog ko ima pristup, a to je obično nastavnik.

Nakon završene radnje brisanja, korisnik će dobiti potvrdu o brisanju i mogućnost da tu radnju opovrgne ako želi. Brisanje se odvija sa akcijom povlačenja u lijevo preko profila učenika.

Brisanje je radnja koja također ne zahtjeva internet konekciju, ali posjedovanje konekcije bitno utiče na način na koji se ova radnja obavlja. Ukoliko je dostupna konekcija podaci se brišu lokalno i sa servera u isto vrijeme. Ako nije dostupna konekcija a podaci se obrišu samo lokalno, prilikom uspostavljanja konekcije doći će do učitavanja podataka sa servera i prethodno brisanje neće imati efekta. Stoga brisanje bez konekcije zahtjeva da se vodi evidencija o lokalno obrisanim podacima. Nakon uspostavljanja konekcije vrši se sinhronizacija lokalno obrisanih podataka sa onim na serveru i podaci će biti uspješno obrisani.

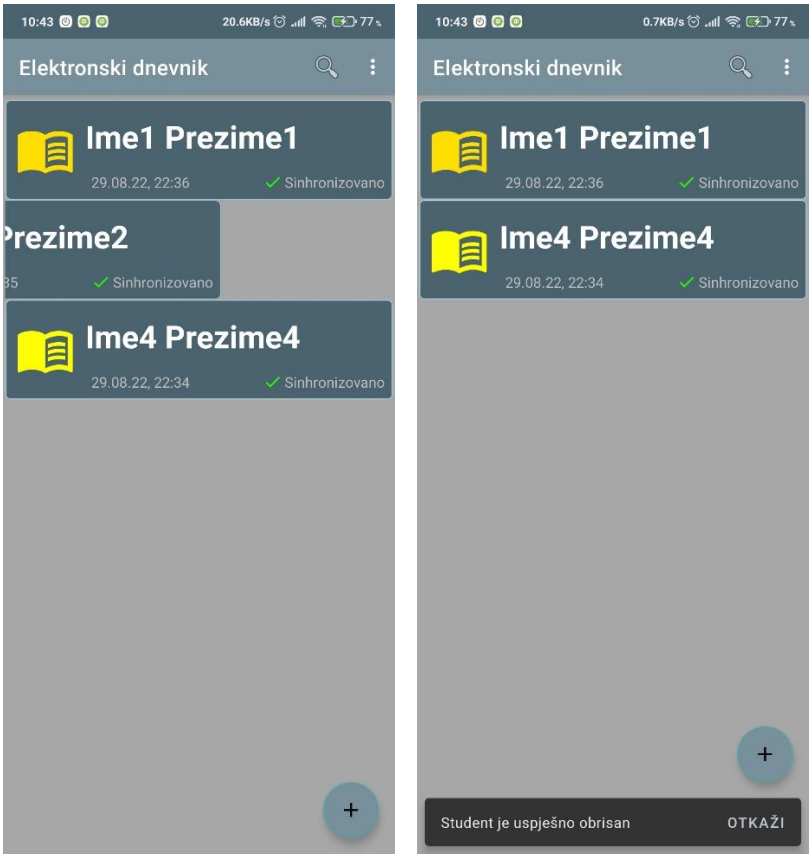



*Slika 15. Dijagram sekvence, brisanje učenika*

```
suspend fun deleteLocallyDeletedStudentID(deletedStudentID: String) {
    studentDao.deleteLocallyDeletedStudentIDs(deletedStudentID)
}

suspend fun deleteStudent(studentID: String) {
    val response = try {
        studentApi.deleteStudent(DeleteStudentRequest(studentID))
    } catch (e: Exception) {
        null
    }
    studentDao.deleteStudentById(studentID)
    if(response == null || !response.isSuccessful) {
        studentDao.insertLocallyDeletedStudentIDs(LocallyDeletedStudentID(studentID))
    }else{
        deleteLocallyDeletedStudentID(studentID)
    }
}
```

*Isječak koda 4 - Brisanje učenika, funkcije koje osiguravaju brisanje učenika lokalno i sa servera*



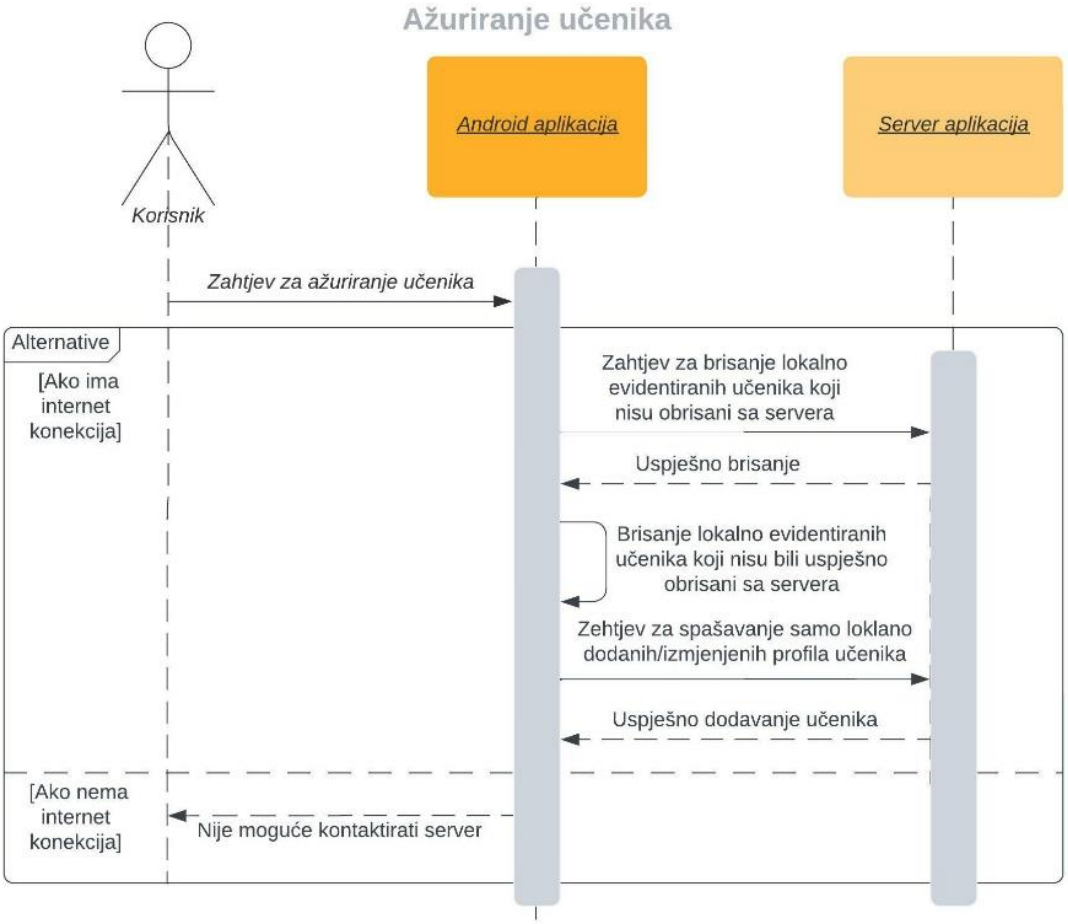

*Slika 16. Snimci ekrana, brisanje učenika*

## 4.5. Ažuriranje (sinhronizacija) učenika

Ažuriranje učenika ili proces sinhronizacije lokalnih podataka sa serverom je radnja koja zahtjeva internet konekciju za uspješan završetak. Sinhronizacija se odvija automatski prilikom otvaranja aplikacije, dok korisnik može i samostalno da je izvrši ako ima za tim potrebu. Sinhronizacija se inicira povlačenjem po ekranu prema dole. Ukoliko nije moguće izvršiti sinhronizaciju, korisnik će biti o tome obaviješten sa odgovarajućom greškom.

Kako je navedeno, brisanje bez internet konekcije bilježi lokalno obrisane podatke i sinhronizuje ih sa serverom prilikom uspostavljanja internet konekcije. Ta radnja se izvršava u procesu ažuriranja kao i radnja sinhronizacije novih podataka koji su spašeni ili izmijenjeni lokalno.

Ažuriranje uključuje uvid u tabelu lokalno obrisanih učenika i sinhronizuje ih sa serverom. Također se vrši sinhronizacija lokalno dodanih ili izmijenjenih učenika za vrijeme rada bez internet konekcije.



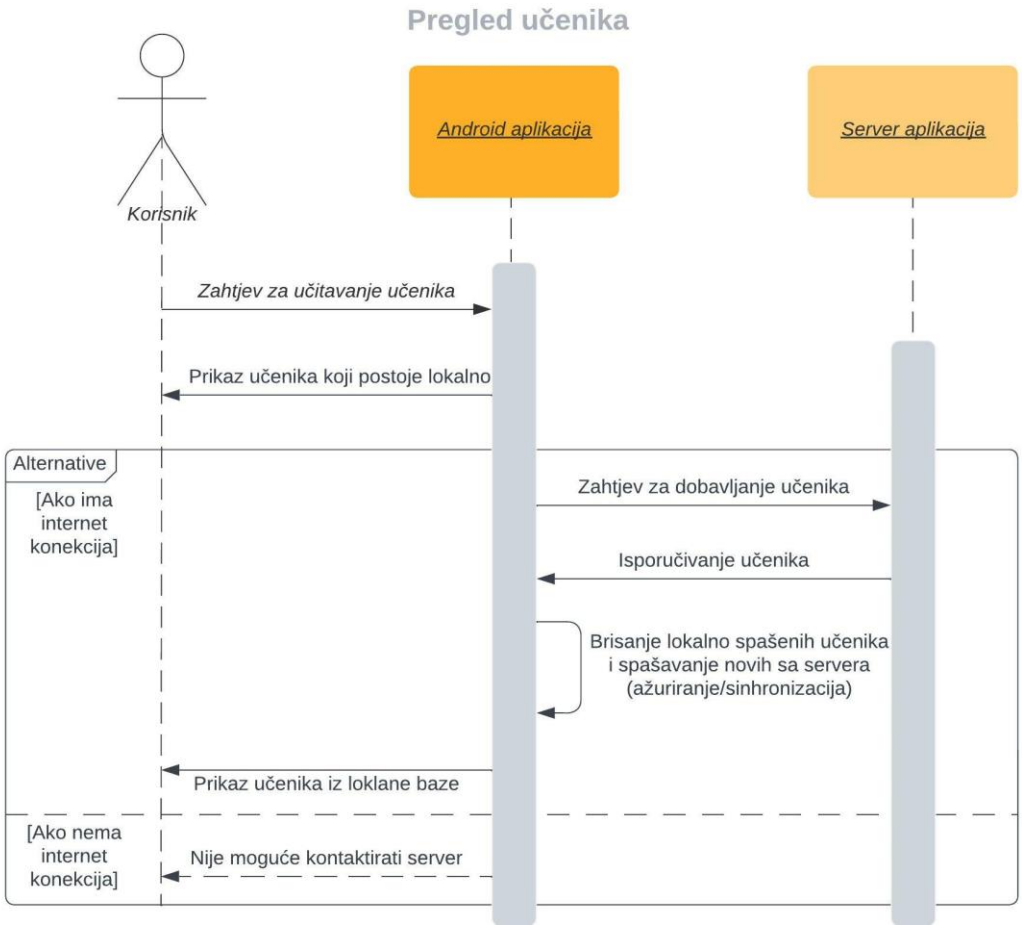

*Slika 17. Dijagram sekvence, ažuriranje učenika*

```
suspend fun syncStudents(){
    val locallyDeletedStudentIDs = studentDao.getAllLocallyDeletedStudentIDs()
    locallyDeletedStudentIDs.forEach{
        id -> deleteStudent(id.deletedStudentID)
    }
    val unsyncedStudents = studentDao.getAllUnsyncedStudents()
    unsyncedStudents.forEach {
        student -> insertStudent(student)
    }
}
```

*Isječak koda 5. Ažuriranje učenika*

## 4.6. Učitavanje i pregled učenika

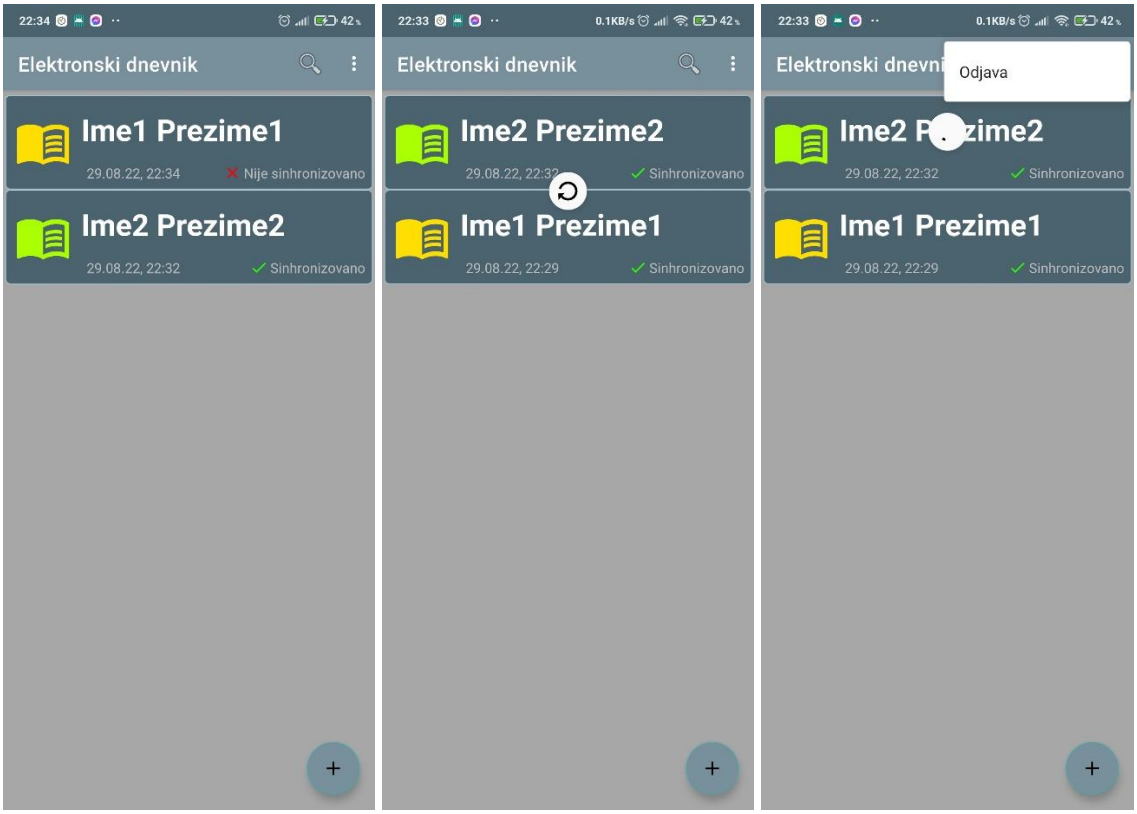
Pregled učenika podrazumijeva učitavanje učenika u pogled na aplikaciji, gdje korisnik može da pregleda sve učenike i pristupa željenim radnjama. Dobavljanje podataka se uvijek vrši sa jednog izvora a to je lokalna baza, a nikad direktno server. Nakon što se učitaju lokalni podaci, uslov dobavljanja podataka sa servera je internet konekcija. Ako postoji, dobavljaju se učenici sa servera u lokalnu bazu i iz baze ponovo učitavaju u pogled. Prije samog učitavanja vrši se već analizirani proces ažuriranja. Ako nema interneta ostaje isti prikaz u pogledu koji je učitan lokalno, ali se korisnik obavještava da nije moguće kontaktirati server. Pregled učenika je moguć uz filtriranje ili pretragu učenika po imenu i prezimenu. Učenici se u pogledu redaju po datumu izmjene podataka i prvi se prikazuju oni koji su zadnji uređivani. Mobilna aplikacija prilikom pregleda učenika pruža vizuelnu reprezentaciju uspjeha učenika preko boja. Boja koja se prikazuje uz učenika predstavlja direktnu vezu sa prosjekom svih ocjena . Boje su raspoređene u RGB (engl. Red Green Blue) rasponu između crvene i zelene, a sredinu predstavlja žuta boja. Konačan ton boje se dobije kada se izračuna tačan prosjek. Potpuno zelena je 5.0 a crvena 1.0.




*Slika 18. Dijagram sekvence, pregled učenika*

```
fun getAllStudents() : Flow<Resource<List<Student>>> {
    return networkBoundResource(
            query = {
                studentDao.getAllStudents()
            },
            fetch = {
                syncStudents()
                studentApi.getStudnets() ^lambda
            },
            saveFetchResult = { response ->   // što smo dobili sa api-ja
                response.body()?.let{ it: List<Student>
                    studentDao.deleteAllStudents()
                    insertStudents(it.onEach { student -> student.isSynced = true })
                }
            },
            shouldFetch = { it: List<Student>
                checkForInternetConnection(context)
            }
    )
}
```

*Isječak koda 6. Dobavljanje učenika za prikaz korisniku*



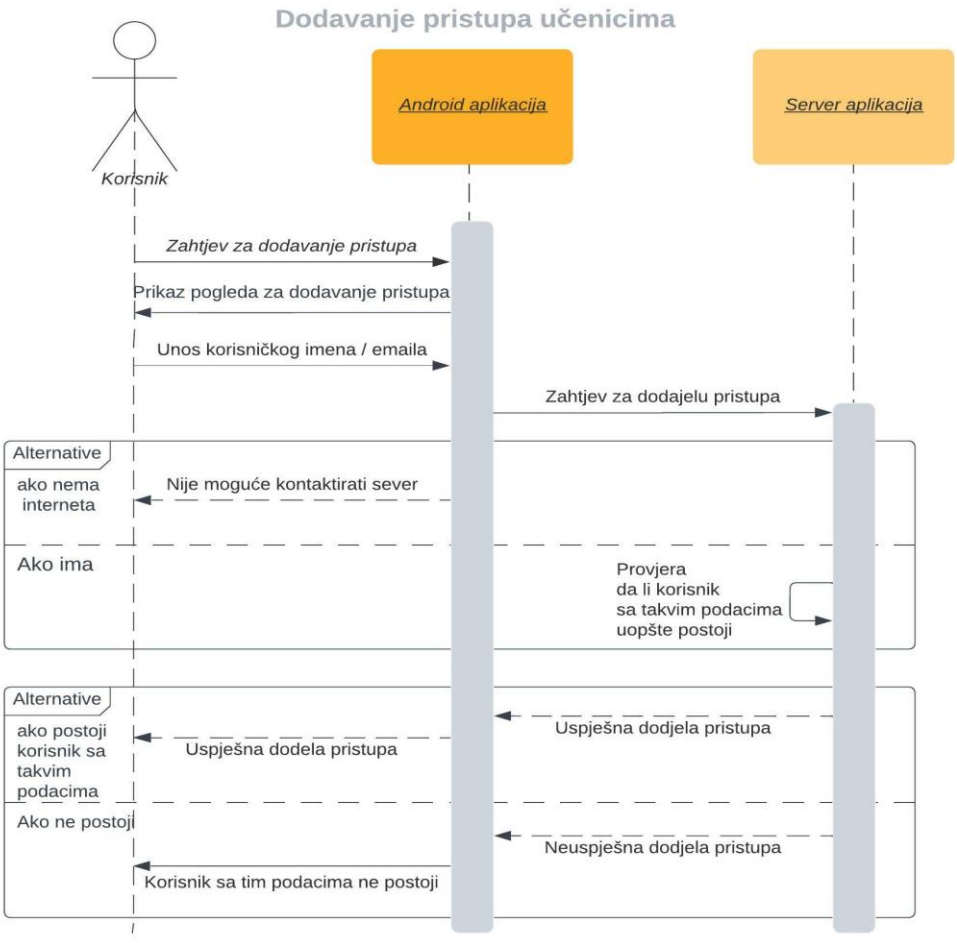

*Slika 19. Snimci ekrana, pregled učenika*

## 4.7. Dodavanje pristupa profilima učenika

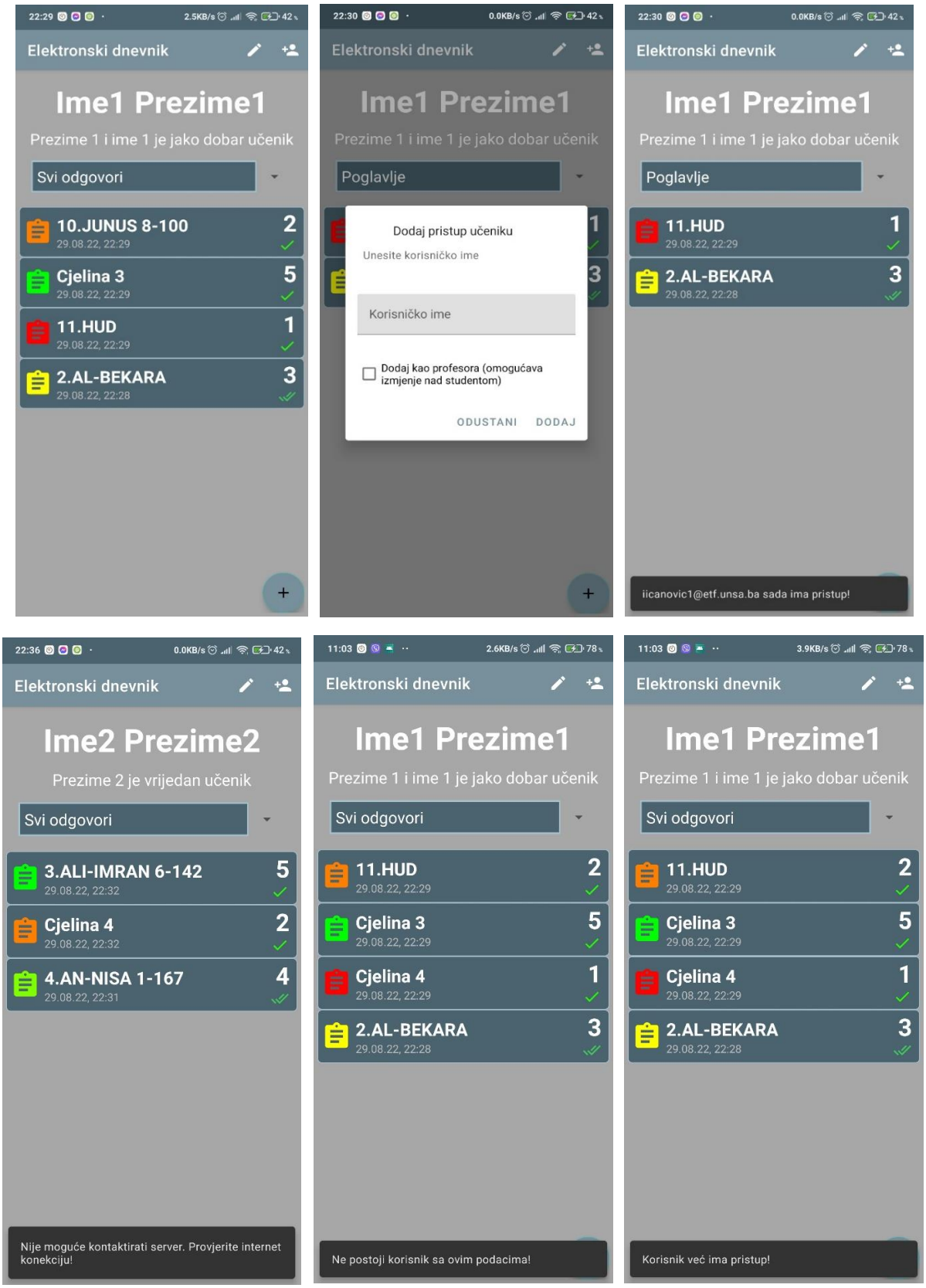
Ova funkcionalnost aplikacije omogućuje korisnički odnos između učenika i nastavnika. Ukoliko nastavnik dodijeli učeniku neprivilegovani pristup nad njegovim profilom, omogućio mu je pregled svih ocjena kao i mogućnost da on opet dodijeli pristup svome profilu nekom drugom korisniku , ali bez rasta u hijerarhiji pristupa. Može naprimjer svojim roditeljima ili prijateljima omogućiti isti pristup koji on ima.

Nastavnik može dodijeliti sve vrste pristupa pa i privilegovani, to je korisno ukoliko dođe do privremene ili stalne zamjene nastavnika. Na taj način, profil studenta može nastaviti voditi drugi nastavnik bez gubljenja podataka.

Dodjela pristupa zahtjeva internet konekciju, te ukoliko korisnik nema konekcije ili postoji greška na serveru biti će obaviješten o tipu greške. Još jedan od uslova uspješne dodjele jeste unošenje validnog korisničkog imena za dodjelu. Ako se pokuša dodijeliti pristup nepostojećem korisniku, biti će prikazana poruka o grešci.




*Slika 20. Dijagram sekvence, dodavanje pristupa učenicima*




*Slika 21. Snimci ekrana, dodavanje pristupa učenicima*

## 4.8. Dodavanje / Uređivanje odgovora (ocjena)

Ocjenjivanje u školama Kur'ana, za koje se primarno Elektronski dnevnik razvija u okviru ovog završnog rada, specifično je i potrebno je razumjeti nekoliko termina koji se pojavljuju.

Prvenstvena nauka koja se izučava u školama Kur'ana jeste pamćenje Kur'ana. Ovo predstavlja bitnu oblast u Islamskoj nauci jer učenje i pamćenje Kur'ana predstavlja jedno od najboljih dobrih dijela.

Božija knjiga Kur'an se sastoji od 114 poglavlja (ar. *Sura*), sa ukupno 6236 rečenica (ar. *Ajet*) podjeljenih u 30 jednakih cijelina (ar. *Džuz*) od po 20 stranica.

Zavisno od sposobnosti učenika, nastavnik mu zadaje da uči određene rečenice, poglavlja ili cjeline. Postoji jako puno načina ocjenjivanja i učenja ali neki od najzastupljenijih su gore navedeni.

Najniži nivo je učenje samih rečenica koje predstavljaju najmanju količinu gradiva i učenik može da dobije ocjenu na osnovu učenja određenog broja rečenica koje pripadaju određenom poglavlju i cjelini.

Slijedeći nivo jeste učenje poglavlja. Postoje poglavlja različitih veličina i svako ima određeni broj rečenica, stoga ako učenik dobije da uči poglavlje podrazumijeva se da će učiti sve rečenice toga poglavlja.

Najveći nivo učenja jeste učenje po cjelinama, jer su obično cjeline količinski najobimnije. Izuzetak predstavlja nekoliko poglavlja, koja se nalaze u više cjelina.

Aplikacija omogućuje nastavnicima ocjenjivanje na sve navedene načine kao i jednostavnu filtraciju svih tipova ocjenjivanja prilikom pregleda ocjena. S obzirom da su cjeline sačinjene od poglavlja, a poglavlja od rečenica odabir odgovora mora da poštuje skupove pripadnosti između navedenih veličina.

Unos novog odgovora podrazumijeva unos barem cjeline i ako je potreban unos cjeline kao tip odgovora na tome se stane, odgovor će biti označen kao odgovor cjeline. Odgovor koji se pokuša unijeti bez odabira barem cjeline, biti će odbijen i korisnik će dobiti povratnu poruku.

Ukoliko je potreban unos odgovora tipa poglavlja, nakon odabira cjeline u kojoj se to poglavlje nalazi pristupa se odabiru poglavlja. Svako poglavlje posjeduje određeni broj rečenica i prilikom odabira poglavlja moguće je odabrati donji i gornji redni broj rečenice koja se odgovara. Ako se ostavi inicijalna donja i gornja vrijednost podrazumijeva se da je naučeno čitavo poglavlje te se odgovor spašava kao tip odgovora po poglavlju.

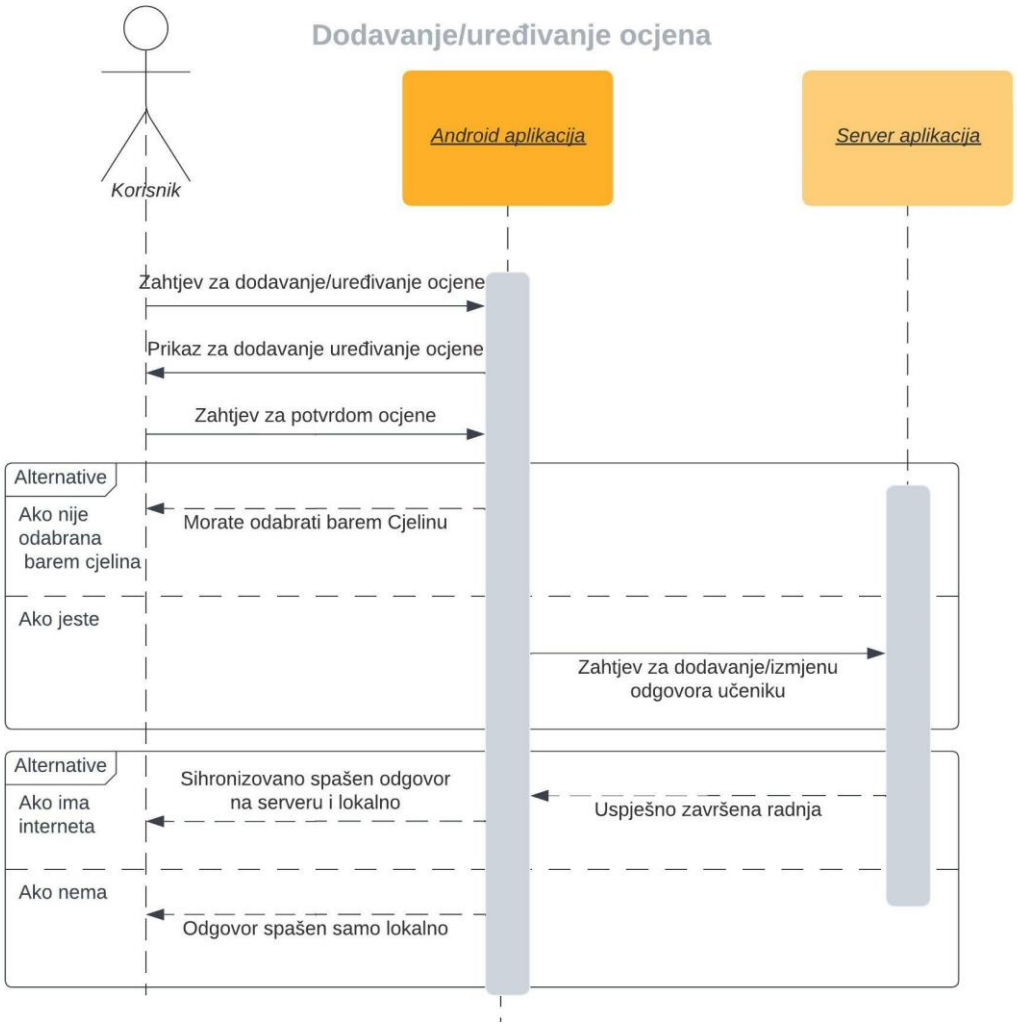
Unos odgovora po rečenicama se ostvaruje odabirom cjeline i poglavlja unutar kojeg se rečenice nalaze. Odabire se redni broj prve i zadnje rečenice u rednom broju rečenica,

unutar poglavlja za koji se unosi ocjena. Odgovor može biti za samo jednu rečenicu i taj redni broj rečenice unutar poglavlja unosi se kao donja i gornja vrijednost.

Učenici koji nauče određeni dio knjige da bi čuvali svoje znanje potrebno je da ga ponavljaju. Nastavnici također nakon određenog vremena vrše ponovno ispitivanje učenika i to predstavlja drugu vrstu ispitivanja a to je ispitivanje starog gradiva.

Moguć je višestruki unos istog odgovora uz razlikovanje u tipu odgovora. Odgovor može da bude odgovaranje novog ili starog gradiva, što prilikom prikaza odgovora posjeduje jasnu vizuelnu diferencijaciju. Ocjenjivanje se obično vrši ocjenama od 1 do 5. Nakon kreiranja odgovora i dodjele ocjene odgovor je uvijek moguće izmijeniti od strane nastavnika ako je došlo do neke greške u unosu.

Za prikaz odgovora je omogućena ista vizuelna prezentacija pomoću boja kao za ukupni prosjek ocjena.

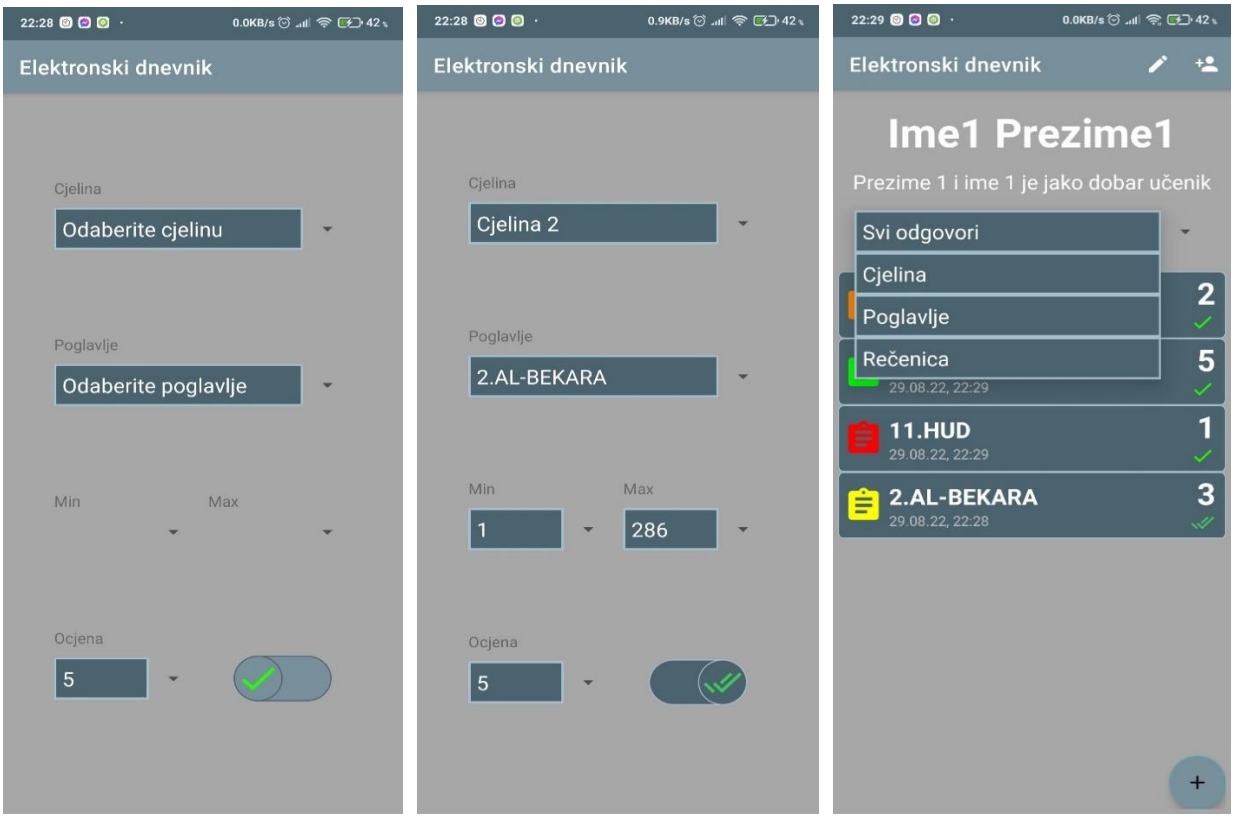



*Slika 22. Dijagram sekvence, dodavanje/uređivanje odgovora*

```
private fun saveAnswer(){
    var answerType : AnswerType = SECTION
    if (chapter != Chapter_NULL){
        if (sentenceMinSelectedNumber == 1 &&
                sentenceMaxSelectedNumber == chapter.numberOfSentences)
            answerType = CHAPTER
        else
            answerType = SENTENCE
    }
    val revision = swRevision.isChecked
    val id = oldAnswer?.id ?: UUID.randomUUID().toString()
    val date = oldAnswer?.date ?:  System.currentTimeMillis()
    val answer = Answer(answerType, section, chapter, sentenceMinSelectedNumber,
        sentenceMaxSelectedNumber, date = date, mark, id = id, revision)

    if(answer.section == Section_NULL){
        showSnackbar( text: "Odgovor nije spašen, morate odabrati barem cjelinu!")
        return
    }
    addAnswerToCurStudent(answer)
}
```

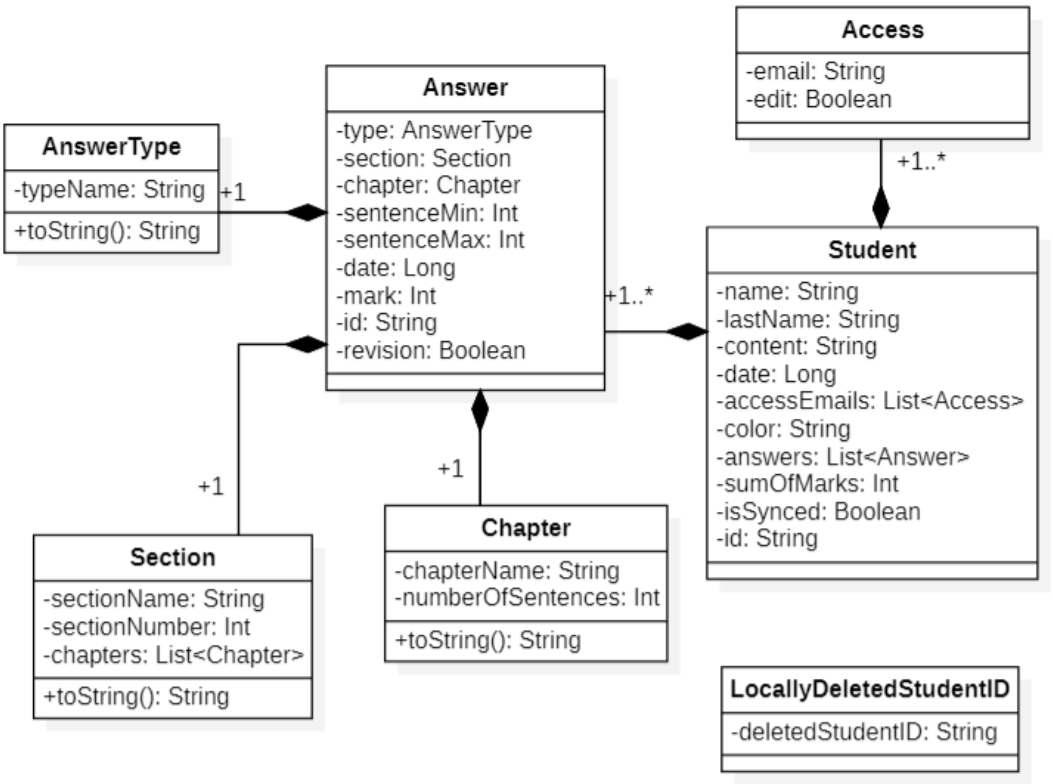
*Isječak koda 7. Dodavanje/uređivanje odgovora*




*Slika 23. Snimci ekrana, dodavanje/uređivanje odgovora*

# 5. Dijagram klasa

Dijagram klasa predstavlja strukturu sistema prikazom klasa, njihovih atributa i funkcija. Engleski jezik je korišten za imenovanje svih varijabli, klasa, atributa, funkcija i tako dalje dok su komentari pisani u bosankom jeziku.




*Slika 24. Dijagram klasa*

Glavna klasa je klasa Student koja u sebi sadrži sve atribute koji opisuju nekog učenika.

Svi pristupi tom studentu se evidentiraju listom *accessEmails*. Ona sadrži listu klase *Access* čiji atributi su *email* (korisničko ime) i atribut *edit* koji označava koji pristupi imaju mogućnost uređivanja.

Unutar *Student* objekta se također kao lista spašavaju svi njegovi odgovori. Sam odgovor mora imati definisan tip koji se predstavlja klasom *AnswerType*. *AnswerType* ili tipovi odgovora su definisani kao konstante i mogu biti Odjeljak, Poglavlje ili Rečenice. Svakom odgovoru se definiše iz kojeg je Odjeljka.

Odjeljak je predstavljen klasom *Section* koja definiše njegovo ime, broj i listu poglavlja koje ga čine. Odjeljci su unaprijed kreirani kao konstante 30 postojećih odjeljaka Kur'ana.

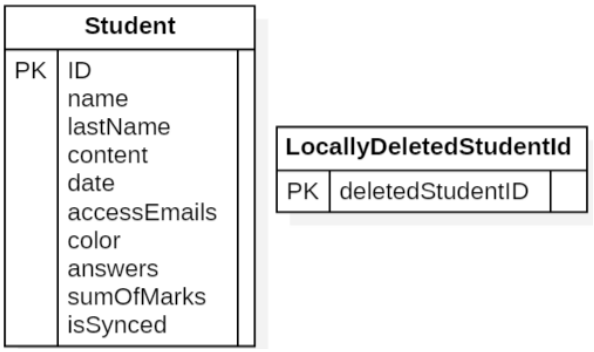
Definisanje odgovora uključuje i definisanje poglavlja iz tog odabranog odjeljka, koji god tip odgovora bio, osim ako je odgovor samo odgovor tipa *Section*. Atributi klase *Chapter* (Poglavje) su ime i broj rečenica koji se nalaze u poglavlju. Poglavlja su kreirana kao konstante postojećih 114 poglavlja Kur'ana.

Klasa *LocallyDeletedStudentId* predstavlja klasu koja omogućuje lokalno skladištenje svih *ID* učenika koje korisnik obriše samo lokalno kada nema internet konekciju. Uspostavljanjem internet konekcije se gleda u ove unose i ako postoje, ti studenti se brišu i sa servera. U suprotnom brisanje studenata samo lokalno ne bi imalo nikakvog efekta jer bi se opet učitali obrisani učenici sa servera.

Dodavanje i izmjena učenika bez interneta je implementirano dodavanjem atributa *isSynced* kod klase *Student*, gdje se uspostavljanjem konekcije vrši uvid u sve učenike kojima ovaj atribut ima vrijednost *false* te ih ažurira sa serverom.

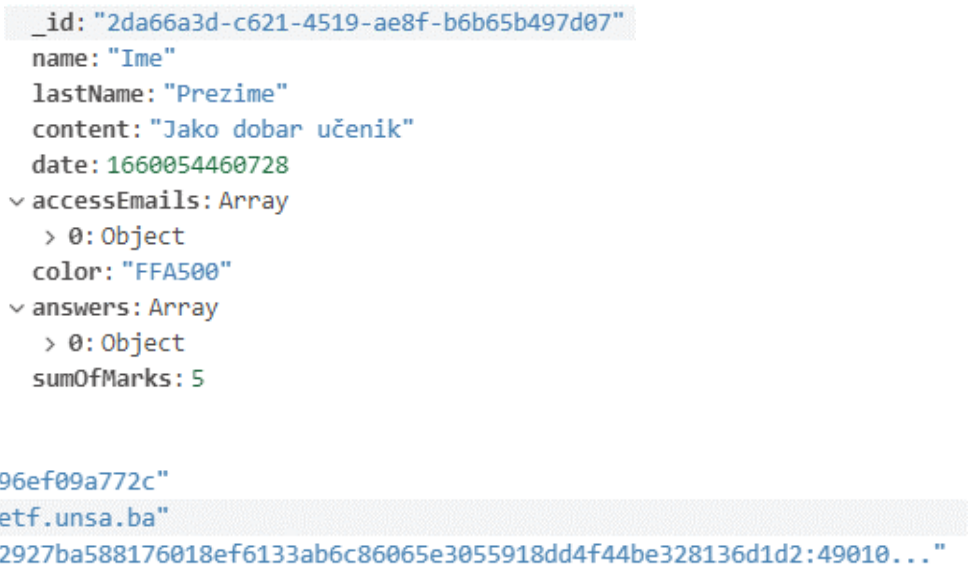
# 6. Diagrami baza podataka

Lokalna perzistencija podataka se vrši u relacionu SQLite bazu podataka i biti će prikazan njen diagram. Lokalno se samo pohranjuju podaci o studentu, pa je baza sačinjena od tabele *Student*. Postoji još jedna pomoćne tabela *LocallyDeletedStudentId*. Ona omogućuje lokalno skladištenje svih ID-vrijednosti učenika koje korisnik obriše lokalno, kada nema internet konekciju.




*Slika 25. Šema lokalne baze podataka*

Skladištenje podataka u nerelacionu bazu na serveru podrazumjeva spašavanje podataka o profilima učenika kao i podataka o svim korisnicima aplikacije. MongoDB serverska baza se sastoji od dva dokumenta. Nakon što se korisnik registruje njegovi podaci se spašavaju u poseban dokument sa njegovim korisničkim imenom, lozinkom i id-ijem. Podaci i profili učenika se spašavaju u drugi dokument na isti način kao i lokalno uz nedostatak atributa „*isSynced*“. Prikaz unosa u navedene dokumente je prikazan ispod.



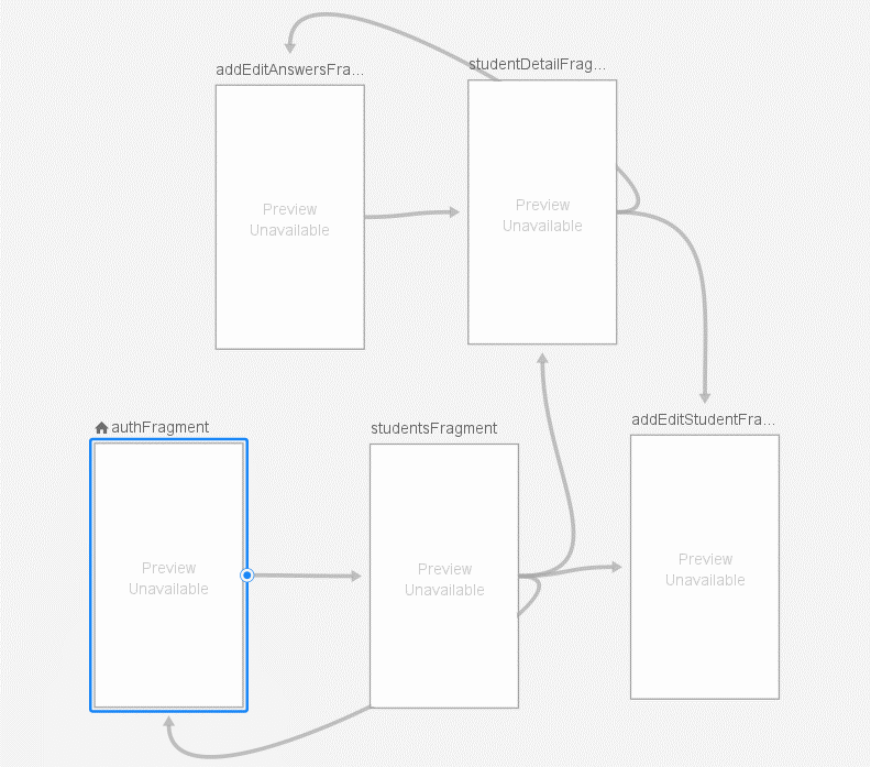

*Slika 26. Primjer unosa unutar serverske baze podataka*

# 7. Implementacija pogleda (engl. Views)

Poštujući MVVM principe kreirane su posebne klase za poglede. Svi pogledi su predstavljeni fragmentima i smjenjuju se unutar jedne glavne aktivnosti. Aplikacija je sačinjena od 7 Fragmenata koji su:

- *authFragment* – prikazuje pogled za akcije prijave i registracije
- *studentsFragment* – prikazuje početni pogled nakon prijave koji uključuje prikaz svih učenika
- *addEditStudentFragment* – prikazuje pogled za dodavanje novih studenata ili izmjenu postojećih
- *studentDetailFragment* – prikazuje sve detalje o student uključujući ocjene
- *addEditAnswerFragment* – prikazuje pogled za dodavanje i uređivanje odgovora
- *AddAccedDialog* – druga vrsta fragmenta tipa *Dialog*, predstavlja skočni prozor koji korisniku omogućuje dodavanje pristupu učenicima

Tranzicije i kretanje između fragmenata je implementirano uz pomoć biblioteke *Navigation Component*. Uz pomoć nje možemo lahko pomoću vizuelnog editora napraviti rute koje su moguće između fragmenata kao i sve parametre koji se prosljeđuju iz jednog fragmenta u drugi. U pozadini će biblioteka za nas kreirati sve potrebne klase i parametre koji se koriste kao gotov proizvod za kretanje između pogleda. Na slijedećoj slici je prikaz svih fragmenata sa putanjama.




*Slika 27. Prikaz fragmenata aplikacije unutar Android Studio okruženja pomoću Navigation Component biblioteke*

Preostali šesti fragment jeste *BaseFragment* koji predstavlja apstraktnu klasu koja nasljeđuje klasu Fragment. Razlog postojanja ovog fragmenta jeste smanjenje dupliciranog koda. Svi prije navedeni fragmenti će da naslijede *BaseFragment*, a samim time i *Fragment* klasu što je obavezno prilikom kreiranja fragmenta. Nasljeđujući *BaseFragment* klasu preostali fragmenti će moći pristupati njenoj implementiranoj funkciji *showSnackbar*. Funkcija *showSnackbar* predstavlja funkciju koja pravi takozvanu povratnu poruku koja se prikaže korisniku u nekom dijelu ekrana. S obzirom da je u ovoj aplikaciji to korišteno unutar svih fragmenata na ovaj način su sve ostale klase mogle da koriste ovu funkciju bez da je same pišu. Njihovo je da je samo pozovu uz odgovarajuću poruku kao parametar koja će se prikazati korisniku.

```
abstract class BaseFragment(layoutId : Int) : Fragment(layoutId) {

    fun showSnackbar(text : String){
        val snackbar = Snackbar.make(
            requireActivity().rootLayout,
            text,
            Snackbar.LENGTH_LONG
        ).show()
    }
}
```

*Isječak koda 8. Klasa baznog fragmenta*

# 8. Zaključak

Tokom izrade ovoga rada i implementiranja sistema, cilj je bila potpuna funkcionalnost uz poštovanje određenih principa koji se smatraju obaveznim u ozbiljnim sistemima na tržištu. Pored funkcionalnosti koda taj kod je lahko čitljiv, izmjenjiv i pogodan za nadogradnje. Lahkoj čitljivosti koda najviše doprinosi KOTLIN programski jezik koji je sa svojom pojavom nadomjestio sve nedostatke JAVE. Uveo je jako puno noviteta koji doprinose performansama kao što su korutine. *Null* sigurnost je osigurala da aplikacije koje se rade u KOTLIN-u su puno stabilnije i manje podložne iznenadnim rušenjima aplikacija u radu. Iako JAVA nije prošlost i nešto što se izbacuje iz upotrebe, KOTLIN je njegov adekvatan nasljednik. S obzirom na sve navedene činjenice i samu činjenicu da Google preporučuje KOTLIN, odabir KOTLIN-a kao programskog jezika predstavlja jako dobru odluku. Zadovoljenje MVVM principa kao i objektno orijentiranog šablona injekcije ovisnosti pomoću biblioteke DAGGER HILT kod je vrlo lahko mijenjati i popravljati. Odabrana MongoDB nerelaciona baza na serveru omogućuje usluživanje jako velikog broja korisnika bez narušavanja performansi ukoliko hardver na kojem se aplikacija izvršava ima zadovoljavajuće osobine. Biblioteke RETROFIT i ROOM se u svojoj oblasti primjene smatraju najpopularnijim izborom među ANDROID programerima. Priliku bi trebalo pružiti  i KTOR-u na klijentskoj strani jer je kao softverski okvir odabran u ovom radu za serversku aplikaciju. KTOR na serveru se pokazao kao dobro rješenje, iako ima veliku konkurenciju od strane SPRING-a predstavlja bolji izbor prilikom izrade manjih do srednje velikih aplikacija.

Osnovi razlog prilikom odabira izrade aplikacije za škole Kur'ana predstavlja manjak ovakvih aplikacija u toj oblasti primjene. Stoga može se očekivati veliko zadovoljstvo korisnika. Napravljene su mnoge generalne funkcionalnosti ( registracija, prijava, lokalno keširanje, spašavanje podataka na serveru...) koje je lahko iskoristiti prilikom nadopunjavanja i prilagođavanja novim željama korisnika.

# 9. Literatura


1. Kotlin Foundation: „Kotlin - A modern programming language that makes developers happier.“, dostupno on-line: https://kotlinlang.org/ [24.09.2022]
2. Kotlin Multiplatform, dostupno on-line: https://kotlinlang.org/docs/multiplatform.html [24.09.2022]
3. Kotlin (programming language), dostupno online: https://en.wikipedia.org/wiki/Kotlin_(programming_language) [24.09.2022]
4. Dynamic typing vs. Static Typing. Dostupno online: https://docs.oracle.com/cd/E57471_01/bigData.100/extensions_bdd/src/cext_transform_typing.html[24.09.2022]
5. Uvod u Kotlin — Koje su prednosti novog zvaničnog jezika za razvoj Android aplikacija. Dostupno online: https://startit.rs/uvod-u-kotlin-koje-su-prednosti-zvanicnog-jezika-za-razvoj-android-aplikacija/[24.09.2022]
6. Usporedba MVVM i MVP arhitektura pri izradi Android aplikacija na primjeru vođenja skladišta. Dostupno online: https://repozitorij.etfos.hr/islandora/object/etfos%3A2612/datastream/PDF/view[24.09.2022]
7. Introduction to MVVM on Android, Dostupno online: https://resocoder.com/2018/08/31/introduction-to-mvvm-on-android/[24.09.2022]
8. SQLite. Dostupno na : https://www.sqlite.org/index.html[24.09.2022]
9. Save data in a local database using Room. Dostupno online: https://developer.android.com/training/data-storage/room[24.09.2022]
10. SQLite and the Room Persistence Library. Dostupno na: https://codingwithmitch.com/blog/sqlite-and-the-room-persistence-library/[24.09.2022]
11. Working with Retrofit. Dostupno na : https://medium.com/eoraa-co/working-with-retrofit-f4bdf4fe7aa9[24.09.2022]
12. Hypertext Transfer Protocol . Dostupno na: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol[24.09.2022]
13. JSON, dostupno na : https://en.wikipedia.org/wiki/JSON[24.09.2022]
14. REST APIs . Dostupno na : https://www.ibm.com/cloud/learn/rest-apis?mhsrc=ibmsearch_a&mhq=what%20is%20rest%20API[24.09.2022]
15. Hilt, dostupno na : https://dagger.dev/hilt/[24.09.2022]
16. Ktor, dostupno na : https://ktor.io/[24.09.2022]
17. Ktor as a backend Application, dostupno na : https://www.codingame.com/playgrounds/36835/ktor-as-a-backend-application[24.09.2022]
18. MongoDB , dostupno na : https://www.mongodb.com/[24.09.2022]
19. Priručnik za NoSQL - MongoDB kroz primjere, dostupno na : https://zir.nsk.hr/islandora/object/infri:702/datastream/PDF/view[24.09.2022]