

UNIVERZITET SINGIDUNUM

Softversko i informaciono inženjerstvo

**Razvoj LessonLynx, aplikacije za administraciju nastave stranog
jezika**

- diplomski rad -

Mentor:

prof. dr Miodrag Živković

Kandidat:

Nikola Đermanović

Beograd, 2026.

Sadržaj

Uvod	3
Problem istraživanja	3
Predmet istraživanja	3
Ciljevi istraživanja	3
Značaj rada	4
Metode istraživanja	4
Struktura rada	5
1. Pregled tehnologija	6
1.1. React Native i Expo	6
1.2. TypeScript	6
1.3. Firebase, Authentication i Firestore	6
1.4. React Navigation	7
1.5. Eksterni API, ExchangeRate-API	7
1.6. Ostale korišćene biblioteke	8
2. Analiza i dizajn sistema	9
2.1. Analiza zahteva	9
Modul Students (studenti)	9
Modul Lessons (časovi)	9
Modul Payments (uplate)	9
Modul Dashboard (pregled/statistika)	9
2.2. Arhitektura aplikacije	11
2.3. Model podataka	13
2.4. Navigacioni tok i UI/UX dizajn	16
3. Implementacija	18
3.1. Autentifikacija	18
3.2. Modul Studenti	22
3.3. Modul Lekcije	26
3.4. Modul Plaćanja i FIFO logika	30
3.5. Dashboard i eksterni API	33
4. Testiranje	36
4.1. Funkcionalno testiranje	36
4.2. Testiranje autentifikacije i poznatih problema	40
Zaključna razmatranja	42
Literatura	46

Uvod

Problem istraživanja

Sa razvojem digitalnih tehnologija i porastom popularnosti učenja stranih jezika putem interneta, sve veći broj nastavnika engleskog kao stranog jezika (*ESL, English as a Second Language*) odlučuje se da napusti rad preko online škola i platformi i posluje samostalno. Razlog za ovakav korak najčešće su bolji uslovi rada: veća zarada, fleksibilnije radno vreme i direktan odnos sa studentima, bez posrednika koji zadržava deo prihoda. Međutim, prelazak na samostalan rad nastavnika stavlja u poziciju sličnu preduzetničkoj, gde pored same nastave mora sam da vodi i administrativne poslove: evidenciju održanih časova, praćenje dugovanja studenata i obračun mesečne zarade, često u više različitih valuta. Ručno vođenje ovakve evidencije, najčešće putem tabelarnih proračuna (npr. Excel ili Google Sheets), postaje nepregledno i podložno greškama kako broj studenata raste, a proces obračuna plaćanja i praćenja neplaćenih časova dodatno je otežan kada su u pitanju studenti iz različitih zemalja koji plaćaju u različitim valutama.

Aktuelnost ove teme dodatno potvrđuje širenje BaaS (Backend as a Service) pristupa u razvoju softvera, koji nezavisnim programerima i malim timovima omogućava da funkcionalnosti poput autentifikacije i skladištenja podataka koriste kao gotov servis, bez potrebe za razvojem i održavanjem sopstvene serverske infrastrukture (Moroney, 2017). Paralelno s tim, sve veći broj korisnika mobilne uređaje koristi kao primarni alat za obavljanje svakodnevnih poslovnih zadataka, što nameće potrebu da aplikacije namenjene ovakvoj upotrebi budu jednostavne za korišćenje i prilagođene ograničenjima malog ekrana i povremene mrežne konekcije (Nielsen & Budiu, 2012).

Predmet istraživanja

Predmet ovog diplomskog rada je razvoj mobilne aplikacije ESL Lesson Tracker, namenjene ESL nastavnicima za objedinjeno praćenje studenata, održanih časova, plaćanja i mesečne zarade. Aplikacija je razvijena korišćenjem tehnologije React Native (Expo) za mobilni interfejs, uz Firebase (Firestore i Authentication) kao cloud backend rešenje, i integraciju eksternog API-ja za konverziju valuta u realnom vremenu.

Ciljevi istraživanja

Osnovni cilj rada je dizajn i implementacija funkcionalnog mobilnog rešenja koje ESL nastavniku omogućava: evidenciju studenata i njihovih podataka za kontakt; unos i pregled održanih časova sa automatskim obračunom cene; praćenje dugovanja po studentu uz automatsko označavanje plaćenih časova (*FIFO metod*); i pregled statistike zarade kroz mesečni i godišnji izveštaj, uključujući konverziju u različite valute. Poseban cilj jeste demonstriranje integracije eksternog API-ja u realnom vremenu i primene principa reaktivnog ažuriranja podataka (real-time sinhronizacija) u mobilnoj aplikaciji.

Značaj rada

Značaj rada ogleda se u dvostrukoj prirodi ostvarenih rezultata. Sa praktične strane, razvijeno rešenje predstavlja funkcionalnu aplikaciju koja se realno koristi u svakodnevnom radu ESL nastavnika, čime rad prevazilazi okvire akademske vežbe i predstavlja konkretan doprinos rešavanju stvarnog problema. Praktična vrednost rešenja potvrđena je i kroz sopstveno korišćenje aplikacije od strane autora rada, koji je istovremeno i ESL nastavnik za čije je svakodnevne potrebe aplikacija prvobitno i razvijena, prema proceni autora, redovno korišćenje aplikacije umesto ranijeg, ručnog vođenja evidencije uštedi približno četiri do pet sati nedeljno. Neformalne povratne informacije nekolicine drugih ESL nastavnika, kojima je aplikacija demonstrirana u toku izrade rada, dodatno ukazuju na to da potreba za ovakvim rešenjem nije ograničena na pojedinačan slučaj, već je prepoznata i među širim krugom nastavnika u istoj situaciji. Sa stručne strane, rad dokumentuje ceo proces razvoja mobilne aplikacije zasnovane na savremenom tehnološkom steoku (React Native, Expo, Firebase): od analize zahteva, preko dizajna arhitekture i modela podataka, do implementacije i testiranja, čime može poslužiti kao koristan referentni materijal studentima koji se prvi put susreću sa razvojem mobilnih aplikacija oslonjenih na BaaS platforme.

Metode istraživanja

U izradi rada korišćene su sledeće metode: metoda studije slučaja, kroz razvoj konkretnog softverskog rešenja od analize zahteva do implementacije i testiranja; analitičko-sintetička metoda, prilikom razlaganja problema na funkcionalne module (studenti, časovi, plaćanja, statistika) i njihovog objedinjavanja u celovit sistem; i metoda analize sadržaja sekundarnih izvora, kroz proučavanje tehničke dokumentacije korišćenih tehnologija (React Native, Expo, Firebase). Dodatno, u izradi poglavlja koje opisuje implementaciju rešenja (poglavlje 3) korišćena je metoda analize izvornog koda aplikacije, čime se obezbeđuje da

opisana funkcionalnost i mehanizmi u radu tačno odgovaraju stvarnom ponašanju razvijenog sistema, a ne uopštenom ili idealizovanom opisu planiranih funkcionalnosti.

Struktura rada

Rad je organizovan u pet poglavlja. U prvom poglavlju daje se pregled korišćenih tehnologija. Drugo poglavlje obuhvata analizu zahteva i dizajn sistema. U trećem poglavlju opisana je implementacija aplikacije po modulima. Četvrto poglavlje posvećeno je testiranju razvijenog rešenja. U zaključnim razmatranjima sumirani su rezultati rada i date su preporuke za dalja unapređenja aplikacije. Radi lakšeg praćenja i preciznijeg razumevanja opisanih rešenja, tekst je kroz treće i četvrto poglavlje praćen odgovarajućim izvorima koda, tabelama sa strukturom podataka i test slučajevima, kao i snimcima ekrana same aplikacije, čime se opisana funkcionalnost direktno povezuje sa njenim stvarnim izgledom i ponašanjem u radu. Na ovaj način, čitalac dobija uvid ne samo u teorijski opis rešenja, već i u njegovu konkretnu, proverljivu realizaciju.

1. Pregled tehnologija

1.1. React Native i Expo

React Native je open-source framework koji je razvila kompanija Meta (ranije Facebook), a omogućava izradu mobilnih aplikacija za iOS i Android platforme korišćenjem jedinstvene JavaScript/TypeScript kodne baze (Eisenman, 2017; React Native, 2026). Za razliku od klasičnog pristupa nativnom razvoju, gde se aplikacija posebno piše za svaku platformu (Swift/Objective-C za iOS, Kotlin/Java za Android) (Živković, 2024), React Native prevodi komponente napisane u JavaScript-u u nativne UI elemente, čime se postiže brzina razvoja uz zadržavanje performansi bliskih nativnim aplikacijama (Boduch & Derks, 2024).

Expo predstavlja skup alata i servisa izgrađenih iznad React Native-a, koji dodatno pojednostavljuje proces razvoja (Expo, 2026). Expo obezbeđuje tzv. managed workflow, u okviru kojeg programer ne mora ručno da konfiguriše native Xcode/Android Studio projekte, već se aplikacija razvija, testira i pokreće direktno preko Expo Go aplikacije ili razvojnog build-a. U okviru rada, Expo je korišćen za inicijalizaciju projekta, upravljanje zavisnostima i pokretanje aplikacije tokom razvoja.

1.2. TypeScript

TypeScript predstavlja nadskup JavaScript programskog jezika koji uvodi statičku tipizaciju (Boduch & Derks, 2024). Za razliku od JavaScript-a, gde se tip promenljive određuje tek u toku izvršavanja programa, TypeScript zahteva da tipovi budu definisani još u fazi pisanja koda, čime se veliki broj grešaka otkriva pre pokretanja aplikacije, a ne tek prilikom testiranja. U okviru aplikacije ESL Lesson Tracker, TypeScript je korišćen za definisanje tipova podataka koji odgovaraju Firestore kolekcijama (Student, Lesson, Payment), kao i za tipizaciju parametara koji se prenose između ekrana putem React Navigation biblioteke, čime se smanjuje mogućnost greške prilikom pristupa nepostojećim poljima objekta.

1.3. Firebase, Authentication i Firestore

Firebase je platforma kompanije Google koja pripada kategoriji BaaS (Backend as a Service) rešenja (Moroney, 2017). BaaS pristup podrazumeva da programer ne mora sam da razvija i održava serversku infrastrukturu, bazu podataka, autentifikacioni sistem, hosting, već te funkcionalnosti koristi kao gotov servis, dostupan putem SDK-a i API poziva.

Firestore Authentication korišćen je za autentifikaciju korisnika aplikacije putem email adrese i lozinke (Google LLC, 2026). Servis se brine o bezbednom čuvanju kredencijala i generisanju pristupnih tokena, dok aplikacija, korišćenjem AsyncStorage mehanizma, čuva sesiju korisnika lokalno na uređaju, čime se postiže da korisnik ne mora ponovo da se prijavljuje prilikom svakog pokretanja aplikacije.

Cloud Firestore je NoSQL dokument baza podataka koja podatke organizuje u kolekcije (npr. students, lessons, payments) i dokumente unutar njih, za razliku od relacionih baza koje koriste tabele i redove (Moroney, 2017). U radu su korišćena dva pristupa čitanju podataka iz Firestore-a: real-time onSnapshot listeneri, koji automatski ažuriraju korisnički interfejs čim se podaci promene na serveru (npr. na ekranu Student Detail), i jednokratni getDocs pozivi, korišćeni na mestima gde real-time ažuriranje nije neophodno (npr. prilikom agregacije podataka za Dashboard).

1.4. React Navigation

React Navigation je biblioteka koja omogućava implementaciju navigacije unutar React Native aplikacije (React Navigation, 2026). U radu su korišćena dva tipa navigatora: Stack Navigator, koji ekrane organizuje u vidu steka, sa mogućnošću povratka na prethodni ekran, i Bottom Tab Navigator, koji omogućava korisniku da se kreće između glavnih celina aplikacije (Students, Lessons, Payments, Dashboard) putem trake sa ikonicama pri dnu ekrana. Svaki od tabova sadrži sopstveni Stack Navigator, čime se omogućava dublja navigacija, na primer, sa liste studenata ka detaljima pojedinačnog studenta, bez gubljenja konteksta trenutno izabrane tabove. Prenos podataka između ekrana ostvaren je putem navigacionih parametara: prilikom prelaska sa liste studenata na ekran detalja prenosi se identifikator studenta (studentId), dok se prilikom prelaska na ekran za izmenu podataka prenosi ceo objekat studenta.

1.5. Eksterni API, ExchangeRate-API

Za potrebe prikaza zarade u različitim valutama, aplikacija koristi eksterni servis ExchangeRate-API (ExchangeRate-API, 2026). Servis se poziva putem HTTP GET zahteva na endpoint https://v6.exchangerate-api.com/v6/{API_KEY}/latest/USD, koji kao odgovor vraća trenutne kurseve za veliki broj svetskih valuta u odnosu na američki dolar. Poziv se izvršava asinhrono, prilikom svakog otvaranja Dashboard ekrana, uz prikaz loading indikatora dok se odgovor ne primi. Dobijeni kursevi (RSD, HKD, EUR) koriste se za

konverziju ukupne zarade nastavnika iz osnovne valute u izabranu valutu prikaza. Besplatni plan servisa omogućava 1500 poziva mesečno, što je dovoljno za potrebe ove aplikacije s obzirom na frekvenciju poziva.

1.6. Ostale korišćene biblioteke

Pored navedenih tehnologija, u razvoju aplikacije korišćen je i niz dodatnih biblioteka iz šireg React/React Native ekosistema (Banks & Porcello, 2020). Za prikaz statističkih podataka na Dashboard ekranu (stubičasti grafikon zarade po mesecima) korišćena je biblioteka `react-native-gifted-charts`, uz prateće zavisnosti `react-native-svg` i `react-native-linear-gradient`, neophodne za renderovanje grafičkih elemenata. Za unos datuma (datum lekcije, datum plaćanja) korišćena je biblioteka `@react-native-community/datetimepicker`, koja na iOS uređajima prikazuje modalni spinner, a na Android uređajima nativni dijalog za izbor datuma. Ikonice u navigacionoj traci pri dnu ekrana obezbeđene su putem biblioteke `@expo/vector-icons` (skup ikonica Ionicons).

2. Analiza i dizajn sistema

2.1. Analiza zahteva

U okviru analize zahteva razmotreni su funkcionalni i nefunkcionalni zahtevi aplikacije LessonLynx, alata za praćenje individualne nastave stranog jezika namenjenog nastavniku kao jedinom korisniku sistema. Zahtevi su izvedeni iz stvarne implementacije aplikacije i organizovani su prema funkcionalnim modulima koji odgovaraju glavnim navigacionim celinama aplikacije: Students, Lessons, Payments i Dashboard.

Funkcionalni zahtevi

Modul Students (studenti)

- Kreiranje, izmena i brisanje zapisa o studentu, uz čuvanje osnovnih podataka: imena, opcionih kontakt podataka (e-mail, WeChat ID), valute u kojoj se student zadužuje (USD, HKD, EUR ili RSD) i satnice (hourlyRate).
- Prikaz liste svih studenata, sortirane po imenu.
- Prikaz detalja pojedinačnog studenta, uključujući tekući iznos duga (totalOwed).
- Sistem automatski, prilikom kreiranja novog studenta, inicijalizuje totalOwed na nulu; ovo polje se dalje ne unosi ručno, već se ažurira isključivo kroz operacije nad časovima i uplatama (videti 2.3).

Modul Lessons (časovi)

- Evidentiranje održanog časa: student kome je čas održan, datum, trajanje u minutima, cena, valuta, opcionalna napomena i status naplaćenosti (isPaid).
- Izmjena i brisanje postojećeg časa, uz automatsko usklađivanje duga studenta (videti 2.3).
- Prikaz svih časova (hronološki, od najnovijeg ka najstarijem) ili časova filtriranih po pojedinačnom studentu.

Modul Payments (uplate)

- Evidentiranje uplate: student, iznos, valuta i datum.
- Prilikom unosa uplate, sistem automatski umanjuje dug studenta (totalOwed) za iznos uplate i, po FIFO principu (first in, first out), označava kao naplaćene (isPaid = true) najstarije neplaćene časove tog studenta, sve dok se iznos uplate ne iscrpi.
- Prikaz hronološke liste svih uplata.

Modul Dashboard (pregled/statistika)

- Prikaz agregiranih finansijskih pokazatelja po mesecima, u vidu grafikona (BarChart), za izabranu kalendarsku godinu.
- Prikaz raspodele zarade po studentima za tekući mesec (ako je izabrana tekuća godina) ili za celu izabranu godinu (u suprotnom), sortirane opadajuće.

- Prikaz broja aktivnih studenata (studenata sa bar jednim časom) u tekućem u odnosu na prethodni mesec.
- Konverzija svih iznosa u jedinstvenu referentnu valutu (USD) radi uporedivosti i agregacije, korišćenjem tekućeg kursa preuzetog sa spoljnog servisa ExchangeRate-API.

Nefunkcionalni zahtevi

- **Sinhronizacija podataka u realnom vremenu:** podaci o studentima, časovima i uplatama prikazuju se korišćenjem Firestore-ovih real-time listenera (onSnapshot), čime se svaka promena podataka (lokalna ili sa drugog uređaja) odražava na korisnički interfejs bez potrebe za ručnim osvežavanjem.
- **Izolacija podataka po korisniku:** svi podaci (studenti, časovi, uplate) čuvaju se kao potkolekcije dokumenta konkretnog korisnika (users/{uid}/...), čime je na nivou strukture baze obezbeđeno da svaki korisnik ima pristup isključivo sopstvenim podacima.

Izolacija podataka po korisniku dodatno je potvrđena postojanjem Firestore Security Rules, konfigurisanih na nivou Firebase Console-a (van git repozitorijuma klijentske aplikacije, pa stoga nisu vidljive pretragom koda): pravilo `match /users/{uid}/{document=**} { allow read, write: if request.auth != null && request.auth.uid == uid; }` eksplicitno ograničava čitanje i pisanje nad svim podkolekcijama (students, lessons, payments) isključivo na autentikovanog korisnika čiji se uid poklapa sa putanjom dokumenta.

- **Perzistentna korisnička sesija:** autentikacija je implementirana korišćenjem Firebase Auth-a, uz eksplicitnu konfiguraciju perzistencije sesije preko AsyncStorage (initializeAuth sa getReactNativePersistence), tako da korisnik ostaje prijavljen između pokretanja aplikacije, bez potrebe za ponovnim unosom kredencijala.
- **Otpornost na nedostupnost spoljne usluge:** u slučaju da servis za kurseve valuta (ExchangeRate-API) nije dostupan, aplikacija to detektuje i ne prekida rad (ratesAvailable stanje), već obaveštava korisnika o nemogućnosti prikaza agregiranih finansijskih podataka.

2.2. Arhitektura aplikacije

Arhitektura aplikacije zasnovana je na klijent-servis modelu, u kojem mobilna aplikacija (React Native/Expo klijent) komunicira direktno sa dva spoljna sistema: Firebase platformom (Authentication i Firestore) i spoljnim REST servisom ExchangeRate-API. Aplikacija nema sopstveni backend server, sva poslovna logika koja bi se tradicionalno nalazila na serverskoj strani (validacija, agregacija, ažuriranje zavisnih polja) implementirana je na klijentu, unutar servisnog sloja.

Tok podataka na visokom nivou

1. Korisnik (nastavnik) se autentikuje preko Firebase Authentication-a (e-mail/lozinka); token sesije se perzistira lokalno preko AsyncStorage.
2. Nakon uspešne autentikacije, klijent uspostavlja real-time konekcije (onSnapshot pretplate) ka Firestore potkolekcijama users/{uid}/students, users/{uid}/lessons i users/{uid}/payments.
3. Svaka promena podataka (bilo da je inicirana lokalno kroz UI, bilo od strane Firestore-a kao potvrda upisa) automatski se propagira kroz pretplate nazad ka komponentama korisničkog interfejsa, bez eksplicitnog ručnog osvežavanja.
4. Za potrebe modula Dashboard, klijent dodatno šalje HTTP zahtev ka ExchangeRate-API (GET .../latest/USD) radi dobijanja tekućih kurseva konverzije, koji se koriste isključivo lokalno, u trenutku izračunavanja, i ne perzistiraju se u Firestore-u.

Slojevi u kodu

Kod aplikacije organizovan je u direktorijumu src/ kroz sledeće slojeve, sa jasnom podelom odgovornosti:

- **types/ (src/types/index.ts):** centralizovana definicija domenskih tipova (Student, Lesson, Payment) i pomoćnih tipova za prikaz na Dashboard-u (MonthData, StudentEarning, ActiveStudentCounts), koji predstavljaju jedini izvor istine za oblik podataka u čitavoj aplikaciji.
- **services/:** sloj pristupa podacima i poslovne logike, podeljen po domenu (firebase.ts, students.ts, lessons.ts, payments.ts, dashboard.ts). Ovaj sloj direktno komunicira sa Firestore SDK-om (CRUD operacije, real-time pretplate) i sa spoljnim ExchangeRate-API servisom, i sadrži logiku koja prevazilazi prostu perzistenciju (npr. denormalizaciju totalOwed, FIFO raspodelu uplata, agregaciju podataka za grafikone).
- **hooks/ (useAuthUser.ts):** enkapsulira pristup Firebase Auth stanju korisnika kroz React hook, koji screens komponente koriste da bi dobile identifikator trenutno prijavljenog korisnika (uid), neophodan za sve pozive ka services sloju.
- **navigation/ (index.tsx):** definiše strukturu navigacije i tipizirane parametre ruta (videti 2.4), i sadrži logiku prebacivanja između neautentikovanog i autentikovanog dela aplikacije.

- **screens/:** prezentacioni sloj, organizovan po istim domenskim celinama kao services (auth, students, lessons, payments, dashboard). Screens komponente pozivaju funkcije iz services sloja, drže lokalno UI stanje (React useState/useEffect) i renderuju korisnički interfejs.
- **components/:** deljene prezentacione komponente koje ne pripadaju jednom ekranu (npr. StudentEarningsChart).
- **constants/:** statične vrednosti koje se koriste na više mesta (npr. nazivi meseci u constants/months.ts).

Ova organizacija odgovara slojevitoj (layered) arhitekturi u kojoj je pristup podacima strogo odvojen od prikaza: screens komponente ne komuniciraju direktno sa Firestore SDK-om, već isključivo preko funkcija izloženih iz services sloja, što olakšava održavanje i izolovano testiranje poslovne logike od korisničkog interfejsa.

2.3. Model podataka

Model podataka aplikacije definisan je u fajlu `src/types/index.ts` i realizovan je u Firestore-u kao tri potkolekcije unutar dokumenta korisnika: `users/{uid}/students`, `users/{uid}/lessons` i `users/{uid}/payments`. Ovakva struktura (potkolekcije ispod dokumenta korisnika) istovremeno definiše i model podataka i mehanizam izolacije podataka po korisniku opisan u 2.1.

Tabela 1: Struktura kolekcije students

Polje	Tip	Opis
id	string	identifikator dokumenta (dodeljuje Firestore)
name	string	ime studenta
email	string (opciono)	kontakt e-mail
wechatId	string (opciono)	kontakt preko WeChat platforme
currency	'USD' 'HKD' 'EUR' 'RSD'	valuta u kojoj se student zadužuje
hourlyRate	number	satnica
totalOwed	number	tekući (denormalizovan) dug studenta
createdAt	Date/Timestamp	vreme kreiranja zapisa

Izvor: autor

Tabela 2: Struktura kolekcije lessons

Polje	Tip	Opis
id	string	identifikator dokumenta
studentId	string	referenca na dokument studenta
studentName	string	ime studenta (denormalizovano)
date	Date/Timestamp	datum održanog časa
durationMinutes	number	trajanje časa u minutima
price	number	cena časa
currency	'USD' 'HKD' 'EUR' 'RSD'	valuta cene
notes	string (opciono)	napomena
isPaid	boolean	da li je čas naplaćen
createdAt	Date/Timestamp	vreme kreiranja zapisa

Izvor: autor

Tabela 3: Struktura kolekcije payments

Polje	Tip	Opis
id	string	identifikator dokumenta
studentId	string	referenca na dokument studenta
studentName	string	ime studenta (denormalizovano)
amount	number	iznos uplate
currency	'USD'/'HKD'/'EUR'/'RSD'	valuta uplate
date	Date/Timestamp	datum uplate
createdAt	Date/Timestamp	vreme kreiranja zapisa

Izvor: autor

Denormalizacija polja totalOwed

Polje totalOwed na dokumentu studenta predstavlja denormalizovan, unapred izračunat zbir, umesto da se dug studenta računa dinamički (agregacijom svih njegovih neplaćenih časova) u trenutku prikaza. Razlog za ovakav pristup leži u ograničenjima Firestore-a kao dokumentno orijentisane baze: Firestore ne podržava agregacione upite tipa SUM preko kolekcije u realnom vremenu na strani klijenta bez dodatnog čitanja svih relevantnih dokumenata, što bi za svaki prikaz liste studenata zahtevalo N dodatnih upita (po jedan za svakog studenta) nad kolekcijom časova. Umesto toga, totalOwed se održava inkrementalno, korišćenjem Firestore atomarne operacije increment():

- pri kreiranju časa (addLesson), totalOwed se uvećava za cenu časa (increment(price));
- pri izmeni časa (updateLesson), ako je student isti, totalOwed se koriguje za razliku nove i stare cene (increment(newPrice - oldPrice)); ako je čas premešten na drugog studenta, stara cena se oduzima od prethodnog studenta, a nova se dodaje novom;
- pri brisanju časa (deleteLesson), totalOwed se umanjuje za cenu obrisano časa;
- pri unosu uplate (addPayment), totalOwed se umanjuje za iznos uplate (increment(-amount)).

Operacija increment() je atomarna na nivou Firestore servera, čime se izbegava tzv. race condition problem koji bi nastao pri klasičnom obrascu „pročitaj – izmeni u klijentu – upiši”, posebno relevantnom u kontekstu real-time sinhronizacije opisane u 2.1.

Duplikacija polja studentName

Polje studentName duplirano je na dokumentima kolekcija lessons i payments (pored postojeće reference studentId), iz istog osnovnog razloga kao denormalizacija totalOwed:

Firestore ne podržava relacione JOIN upite. Bez ove denormalizacije, prikaz liste svih časova ili svih uplata (koji uključuje ime studenta uz svaku stavku) zahtevao bi, za svaku stavku u listi, dodatno čitanje dokumenta odgovarajućeg studenta radi dobijanja njegovog imena, što bi značajno povećalo broj Firestore čitanja (i, posledično, troškove i latenciju), naročito imajući u vidu da se liste časova i uplata osvežavaju u realnom vremenu putem onSnapshot pretplata. Cena ove optimizacije jeste potencijalna nekonzistentnost: ukoliko korisnik naknadno promeni ime studenta, već upisani časovi i uplate zadržavaju staro ime, jer se studentName ne ažurira retroaktivno na povezanim dokumentima.

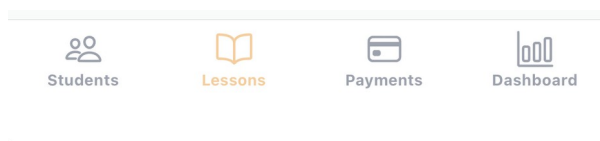
2.4. Navigacioni tok i UI/UX dizajn

Struktura navigacije

Navigacija je implementirana korišćenjem biblioteke React Navigation (@react-navigation/native, verzija 7), i centralizovana je u fajlu src/navigation/index.tsx. Korenska komponenta Navigation prati stanje autentikacije korisnika preko Firebase Auth callback-a onAuthStateChanged i, na osnovu njega, uslovno renderuje jednu od dve potpuno odvojene navigacione celine:

- **Auth Stack (AuthStack, createNativeStackNavigator):** prikazuje se kada korisnik nije prijavljen. Sadrži dva ekrana: Login i Register, bez prikazanog zaglavlja (headerShown: false).
- **Main Tabs (MainTabs, createBottomTabNavigator):** prikazuje se kada je korisnik prijavljen. Sadrži četiri taba u dnu ekrana koji odgovaraju modulima opisanim u 2.1: StudentsTab, LessonsTab, PaymentsTab i DashboardTab.

Izgled trake sa četiri taba pri dnu ekrana, sa vizuelno istaknutim aktivnim tabom (primarna boja #FFCC99), prikazan je na Slici 1.



Slika 1 - Traka sa tabovima (Bottom Tab Navigator)

Svaki od četiri taba ne sadrži pojedinačan ekran, već sopstveni ugnježdeni stek navigator (npr. StudentsNavigator, LessonsNavigator), čime je omogućeno da svaki modul ima sopstvenu dubinu navigacije (npr. lista → detalji → forma za izmenu) nezavisno od ostalih tabova, uz očuvanje sopstvene istorije navigacije po tabu. Na primer, StudentsNavigator sadrži ekrane StudentsList → StudentDetail → AddEditStudent. Prijavljivanje/odjavljivanje trenutno prijavljenog korisnika (signOut) obavlja se iz zaglavlja Dashboard ekrana (dugme u headerRight opciji DashboardNavigator-a), a ne kroz posebnu stavku menija.

Prenos parametara kroz navigaciju

Parametri ruta tipizirani su TypeScript tipovima definisanim uz svaki stek navigator u navigation/index.tsx (npr. StudentsStackParamList, LessonsStackParamList, PaymentsStackParamList), čime se na nivou kompajlera proverava da svaki poziv navigation.navigate() prosleđuje ispravan oblik parametara za ciljani ekran. Konkretni primeri prenosa podataka kroz parametre rute:

- **StudentDetail:** prima samo { studentId: string }, na osnovu kog ekran samostalno učitava (pretplaćuje se na) pun objekat studenta preko services sloja;
- **AddEditStudent i AddEditLesson:** primaju opcioni pun objekat ({ student?: Student }, odnosno { lesson?: Lesson }), prisustvo objekta signalizira režim izmene, a njegovo odsustvo (undefined) režim kreiranja novog zapisa, čime se izbegava dupliranje forme za dodavanje i izmenu u dva odvojena ekrana;
- **AddPayment:** prima kombinaciju identifikatora i već poznatih vrednosti sa prethodnog ekrana ({ studentId, studentName, totalOwed, currency }), čime se ekranu za unos uplate odmah prosleđuju podaci potrebni za prikaz konteksta (npr. trenutni dug i valuta), bez dodatnog čitanja iz Firestore-a pri otvaranju ekrana.

Dizajn sistem

Na osnovu pregleda stilova u komponentama ekrana, utvrđeno je da aplikacija ne koristi centralizovan fajl dizajn tokena/teme (nije pronađen zaseban theme.ts, colors.ts i sl.); boje i tipografija definisane su lokalno, unutar StyleSheet.create poziva svakog ekrana pojedinačno. I pored odsustva centralizacije, analiza upotrebe heksadecimalnih vrednosti boja kroz kod pokazuje dosledno ponavljanje iste, ograničene palete, iz čega se može rekonstruisati implicitan dizajn sistem:

- **primarna akcenat boja:** #FFCC99 (bledonarandžasta/pastelna), korišćena za aktivne tabove u donjoj navigaciji i primarne akcije;
- **tekst i naslovi:** #111827 (skoro crna, najčešće korišćena boja u kodu);
- **sekundarni/pomoćni tekst i neaktivne ikone:** #6B7280 i #9CA3AF (sive nijanse);
- **pozadine i separatori:** #F9FAFB, #F3F4F6, #E5E7EB (svetle sive nijanse);
- **semantičke boje stanja:** #EF4444 (crvena, za greške/brisanje/neplaćeno) i #22C55E (zelena, za uspeh/naplaćeno).

Tipografija se oslanja na podrazumevani sistemski font platforme (u kodu nije pronađen uvoz prilagođenog fonta niti fontFamily svojstvo), dok se hijerarhija postiže variranjem veličine i debljine slova: telesni tekst i oznake polja pretežno koriste fontSize u rasponu 12–16, naslovi i istaknute numeričke vrednosti (npr. iznosi na Dashboard-u) fontSize u rasponu 24–32, uz fontWeight '600' ili '700' za isticanje.

3. Implementacija

3.1. Autentifikacija

Inicijalizacija Firebase-a nalazi se u fajlu `src/services/firebase.ts`. Aplikacija koristi `initializeApp` za povezivanje sa Firebase projektom (konfiguracija se učitava iz promenljivih okruženja sa prefiksom `EXPO_PUBLIC_`), dok se Firebase Auth eksplicitno inicijalizuje preko `initializeAuth` uz parametar `persistence: getReactNativePersistence(ReactNativeAsyncStorage)`. Ovim se sesija korisnika trajno čuva u `AsyncStorage`-u na uređaju, čime se obezbeđuje da korisnik ostane prijavljen između uzastopnih pokretanja aplikacije, bez potrebe za ponovnim unosom kredencijala pri svakom pokretanju.

Kod 1: Inicijalizacija Firebase Auth sa AsyncStorage perzistencijom (src/services/firebase.ts)

```
export const auth = initializeAuth(app, {
  persistence: getReactNativePersistence(ReactNativeAsyncStorage),
});
export const db = getFirestore(app);
```

Tok registracije implementiran je u `src/screens/auth/RegisterScreen.tsx`. Pre poziva ka Firebase-u, forma sprovodi tri nezavisne provere na klijentu: da su sva polja popunjena, da se lozinka i potvrda lozinke poklapaju, i da lozinka ima najmanje 6 karaktera. Tek nakon prolaska ovih provera poziva se `createUserWithEmailAndPassword(auth, email, password)`. U slučaju greške sa Firebase strane, korisniku se prikazuje čitljiva poruka mapirana preko istog rečnika grešaka koji koristi i ekran za prijavu (videti dalje u tekstu).

Tok prijave implementiran je u `src/screens/auth/LoginScreen.tsx`, korišćenjem `signInWithEmailAndPassword(auth, email, password)`. Ekran za prijavu koristi rečnik grešaka izdvojen u zajednički modul `src/constants/authErrors.ts` (`AUTH_ERROR_MESSAGES`, `AUTH_ERROR_FALLBACK`), po uzoru na već postojeći obrazac deljenih konstanti (`constants/months.ts`). Isti rečnik koristi i ekran za registraciju, čime je uklonjena ranija asimetrija u obradi grešaka između dva ekrana:

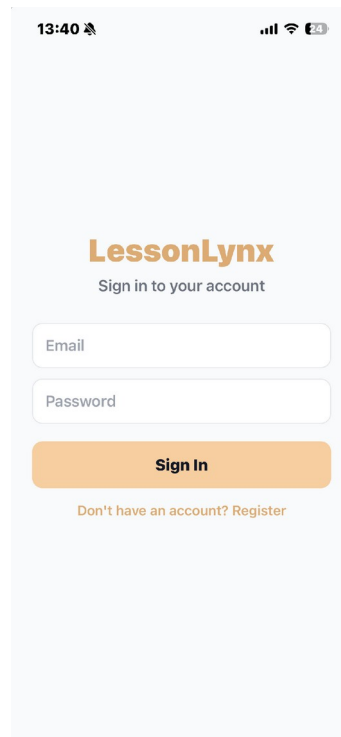
Kod 2: Zajednički rečnik za mapiranje Firebase grešaka (src/constants/authErrors.ts)

```
export const AUTH_ERROR_MESSAGES: Record<string, string> = {
  'auth/invalid-credential': 'Incorrect email or password.',
  'auth/user-not-found': 'No account found with this email.',
  'auth/wrong-password': 'Incorrect password.',
  'auth/invalid-email': 'Please enter a valid email address.',
  'auth/too-many-requests': 'Too many failed attempts. Try again later.',
```

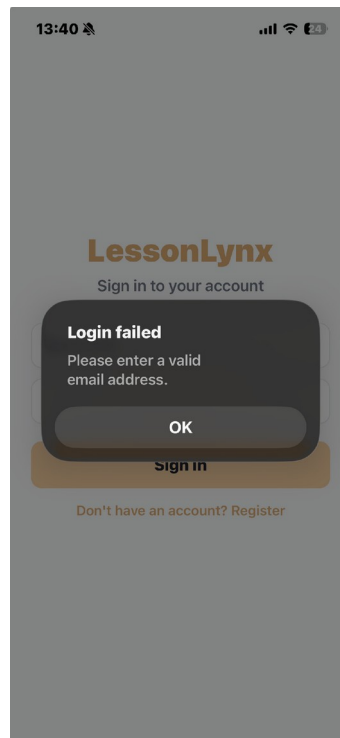
```
'auth/email-already-in-use': 'An account with this email already exists.',
'auth/weak-password': 'Password must be at least 6 characters.',
'auth/operation-not-allowed': 'Registration is currently disabled. Please try again later.',
};
export const AUTH_ERROR_FALLBACK = 'Something went wrong. Please try again.';
```

Oba ekrana pozivaju `AUTH_ERROR_MESSAGES[e.code]` ?? `AUTH_ERROR_FALLBACK` prilikom prikazivanja Alert dijaloga. Ukoliko kod greške koji vrati Firebase nije eksplicitno naveden u rečniku, korisniku se prikazuje generička rezervna poruka, umesto tehničkog opisa greške.

Izgled ekrana za prijavu prikazan je na Slici 2, dok je primer prikaza mapirane poruke greške (u slučaju unosa nevalidnog formata e-mail adrese) prikazan na Slici 3.



Slika 2 - Ekran za prijavu (Login)



Slika 3 - Mapirana poruka greške pri prijavi

Prebacivanje između Auth i Main navigacije upravlja se u korenskoj komponenti Navigation (src/navigation/index.tsx), koja drži sopstveno lokalno stanje user i loading, i pretplaćuje se na Firebase-ov `onAuthStateChanged` callback nezavisno od hook-a `useAuthUser`:

Kod 3: `onAuthStateChanged` pretplata u korenskoj navigacionoj komponenti

```
useEffect(() => {
  const unsub = onAuthStateChanged(auth, (u) => {
    setUser(u);
    setLoading(false);
  });
  return unsub;
}, []);
```

Dok traje inicijalna provera stanja autentikacije (`loading === true`), prikazuje se `ActivityIndicator`. Nakon što callback prvi put vrati rezultat, komponenta uslovno renderuje `<MainTabs />` ako je user postavljen, odnosno `<AuthStack />` (sa ekranima Login i Register) ako korisnik nije prijavljen. Odjava korisnika (`signOut(auth)`) poziva se iz zaglavlja Dashboard ekrana, nakon potvrde u Alert dijalogu.

Zaseban hook `useAuthUser` (src/hooks/useAuthUser.ts) enkapsulira istu pretplatu na `onAuthStateChanged`, ali se u kodu koristi samo unutar `DashboardScreen` komponente, radi

dobijanja identifikatora prijavljenog korisnika (user.uid) potrebnog za učitavanje podataka za grafikone.

3.2. Modul Studenti

CRUD operacije nad studentima implementirane su u `src/services/students.ts` i direktno se oslanjaju na Firestore Web SDK, bez dodatnog sloja apstrakcije. Sve operacije adresiraju potkolekciju `users/{uid}/students`. Kreiranje studenta (`addStudent`) prima podatke bez polja `id`, `createdAt` i `totalOwed`, i eksplicitno postavlja `totalOwed` na 0 i `createdAt` preko `serverTimestamp()` pre upisa. Izmena (`updateStudent`) prosleđuje samo izmenjena polja `updateDoc`-u. Forma za dodavanje/izmenu studenta, sa izborom jedne od četiri podržane valute, prikazana je na Slici 8.

Slika 8 - Forma za dodavanje studenta (Add/Edit Student)

Brisanje studenta (`deleteStudent`) implementirano je kao atomarna grupna operacija preko Firestore-ovog `writeBatch` mehanizma: funkcija prvo upitom (`where('studentId', '==', studentId)`) pronalazi sve povezane dokumente u kolekcijama `lessons` i `payments`, dodaje njihovo brisanje u batch zajedno sa brisanjem dokumenta samog studenta, i izvršava sve izmene jednim pozivom `batch.commit()`. Time se izbegava ostajanje „osirotelih” zapisa u bazi nakon brisanja studenta, a operacija je, za razliku od ostalih CRUD operacija opisanih u ovom poglavlju, izvedena kao sve-ili-ništa (atomarna), bez rizika od delimičnog izvršenja u slučaju greške između pojedinačnih poziva.

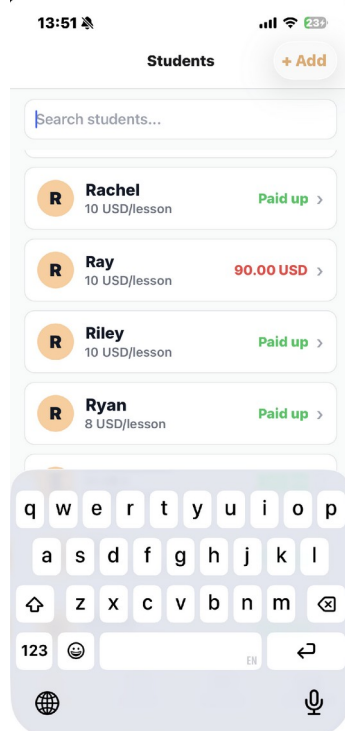
`StudentsListScreen` (`src/screens/students/StudentsListScreen.tsx`) prikazuje listu svih studenata korišćenjem real-time pretplate `subscribeToStudents`. Pretraga je implementirana

potpuno na klijentskoj strani, kao filtriranje već učitano niza u memoriji, bez dodatnih Firestore upita:

Kod 4: Klijentsko filtriranje pretrage studenata

```
const filtered = students.filter((s) =>
  s.name.toLowerCase().includes(search.toLowerCase())
);
```

Sortiranje liste ne vrši se u samoj komponenti, već je već primenjeno na nivou Firestore upita u servisnom sloju: `subscribeToStudents` koristi `query(..., orderBy('name'))`, tako da klijent uvek prima podatke već sortirane po imenu. Izgled liste studenata, sa vizuelnim razlikovanjem studenata koji duguju (crveno, sa iznosom) od onih koji su izmirili dug (zeleno, „Paid up“), prikazan je na Slici 4.



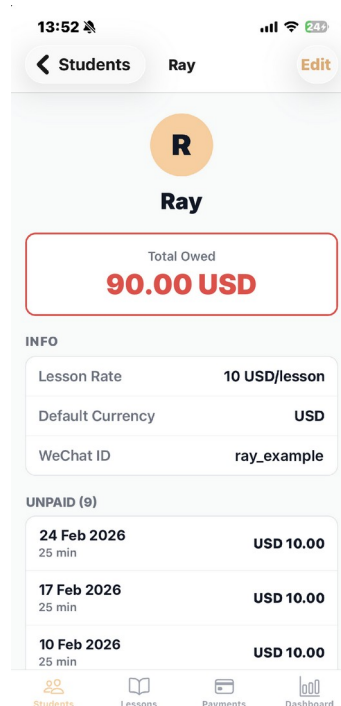
Slika 4 - Lista studenata (StudentsList) sa statusima dugovanja

`StudentDetailScreen` (`src/screens/students/StudentDetailScreen.tsx`) koristi dve nezavisne real-time pretplate istovremeno: `subscribeToStudent` (za osnovne podatke i `totalOwed` konkretnog studenta) i `subscribeToStudentLessons` (za sve časove tog studenta). Obe pretplate se raskidaju (`unsubStudent()`, `unsubLessons()`) pri `unmount`-ovanju komponente, u `cleanup` funkciji `useEffect` hook-a. Prikazani časovi dele se u dve sekcije na osnovu polja `isPaid`:

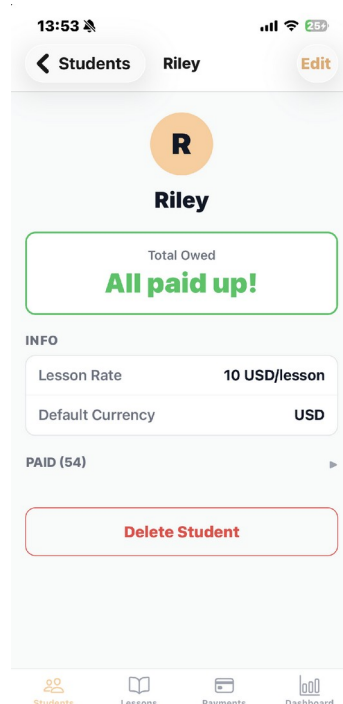
Kod 5: Podela lekcija na neplaćene i plaćene

```
const unpaidLessons = lessons.filter((l) => !l.isPaid);  
const paidLessons = lessons.filter((l) => l.isPaid);
```

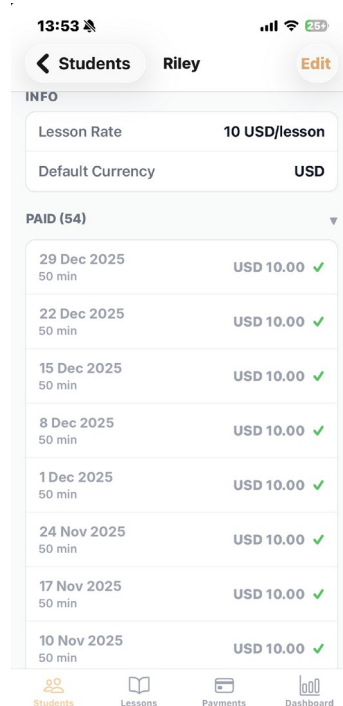
Sekcija neplaćenih časova prikazuje se uvek razvijena (ako postoji bar jedan neplaćen čas), dok je sekcija plaćenih časova podrazumevano skupljena i može se proširiti/skupiti preko lokalnog UI stanja `paidExpanded`, čime se korisniku prioriteto prikazuju podaci relevantni za tekuće dugovanje. Primer studenta sa dugovanjem, sa razvijenom sekcijom neplaćenih časova, prikazan je na Slici 5. Primer studenta bez dugovanja, sa skupljenom (Slika 6) i razvijenom (Slika 7) sekcijom plaćenih časova, pri čemu svaki plaćen čas nosi zelenu oznaku potvrde, prikazan je na narednim slikama.



Slika 5 - Detalj studenta sa neplaćenim časovima (StudentDetail)



Slika 6 - Detalj studenta, sekcija plaćenih časova skupljena



Slika 7 - Detalj studenta, sekcija plaćenih časova razvijena

3.3. Modul Lekcije

Forma za dodavanje i izmenu časa nalazi se u `src/screens/lessons/AddEditLessonScreen.tsx`. Trajanje časa ograničeno je na tri unapred definisane vrednosti (25, 50 i 75 minuta), svakoj od kojih odgovara fiksna, unapred definisana cena preko konstantnog mapiranja:

Kod 6: Fiksne cene po trajanju časa

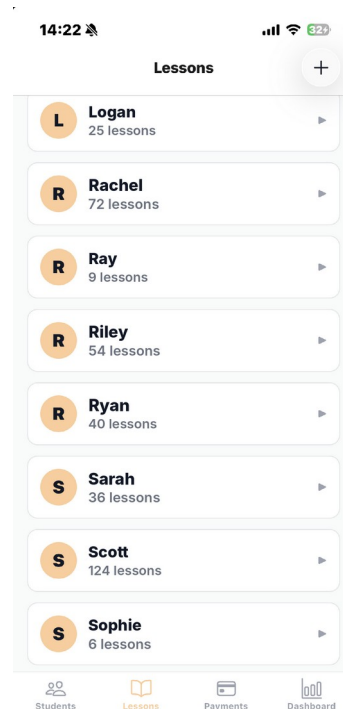
```
const DURATIONS = [25, 50, 75];  
const DURATION_PRICES: Record<number, number> = { 25: 10, 50: 20, 75:  
30 };
```

U režimu dodavanja novog časa (`isEdit === false`), cena polja se automatski postavlja na odgovarajuću vrednost iz `DURATION_PRICES` svaki put kada korisnik promeni izabrano trajanje, preko posebnog `useEffect` efekta. Polje za cenu (`TextInput`) ostaje editabilno u svakom trenutku, tako da korisnik može ručno prepisati automatski predloženu vrednost (`override`); ova automatska sinhronizacija se, međutim, isključuje u režimu izmene postojećeg časa (`isEdit === true`), gde se zadržava originalno unesena cena i ne prepisuje se automatski pri promeni trajanja. Primer ekrana za izmenu časa prikazan je na Slici 9, čas u trajanju od 50 minuta zadržava cenu od 10 (izvorno unetu vrednost), umesto da bude automatski postavljena na 20, upravo zbog ovog ponašanja.

Slika 9 - Forma za izmenu časa (Edit Lesson), sa izborom trajanja i cene

Prilikom čuvanja, valuta časa preuzima se od izabranog studenta u trenutku njegovog biranja sa liste (`handleSelectStudent` postavlja `setCurrency(student.currency)`), ali ostaje naknadno izmenljiva preko posebnih dugmadi za valutu.

`LessonsListScreen` (`src/screens/lessons/LessonsListScreen.tsx`) ne prikazuje ravnu (flat) listu časova, već ih grupiše po studentu preko `useMemo` hook-a, formirajući presek (mapu) po `studentId`, a zatim sortira dobijene sekcije po imenu studenta (`localeCompare`). Svaka sekcija je proširiva/skupljiva pojedinačno, preko lokalnog `Set`-a `expanded` koji čuva identifikatore trenutno otvorenih sekcija. Prikaz liste sa više skupljenih sekcija po studentu dat je na Slici 10, dok je primer jedne razvijene sekcije, sa pojedinačnim časovima, prikazan na Slici 11.



Slika 10 - Lista časova grupisana po studentu (skupljene sekcije)

Lessons		
Hailey 47 lessons		
5 Jan 2026 50 min	USD 10.00	
2 Jan 2026 50 min	USD 10.00	
29 Dec 2025 50 min	USD 10.00	
22 Dec 2025 50 min	USD 10.00	
19 Dec 2025 50 min	USD 10.00	
15 Dec 2025 50 min	USD 10.00	
12 Dec 2025 50 min	USD 10.00	
8 Dec 2025 50 min	USD 10.00	
5 Dec 2025 50 min	USD 10.00	
27 Nov 2025		

Slika 11 - Lista časova, razvijena sekcija jednog studenta

Mehanizam ažuriranja `totalOwed` implementiran je u `src/services/lessons.ts` i predstavlja konkretnu realizaciju denormalizacije opisane u poglavlju 2.3. Pri kreiranju časa, nakon upisa dokumenta u kolekciju `lessons`, izvodi se poseban, sekvencijalni poziv koji uvećava `totalOwed` studenta preko atomarne operacije `increment()`:

Kod 7: Ažuriranje `totalOwed` pri kreiranju časa (`addLesson`)

```
export async function addLesson(uid, data) {
  await addDoc(lessonsCol(uid), { ...data, createdAt:
serverTimestamp() });
  await updateDoc(studentDoc(uid, data.studentId), {
    totalOwed: increment(data.price),
  });
}
```

Pri izmeni časa (`updateLesson`), logika zavisi od toga da li je čas premešten na drugog studenta. Ako student ostaje isti, `totalOwed` se koriguje samo za razliku nove i stare cene; ako je student promenjen, cena se prvo oduzima od starog studenta, a zatim dodaje novom:

Kod 8: Ažuriranje `totalOwed` pri izmeni časa (`updateLesson`)

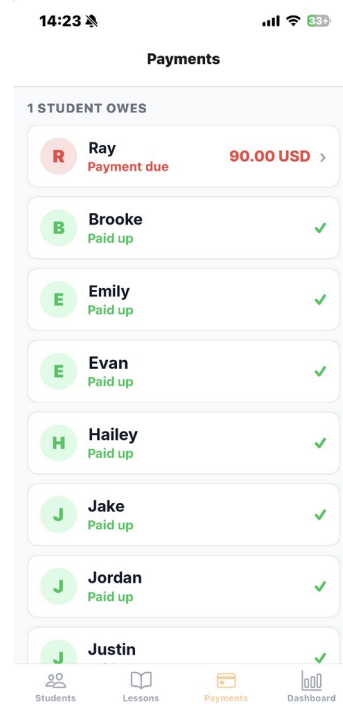
```
if (oldLesson.studentId === data.studentId) {
  const diff = data.price - oldLesson.price;
  if (diff !== 0) {
    await updateDoc(studentDoc(uid, data.studentId), { totalOwed:
increment(diff) });
  }
}
```

```
} else {  
  await updateDoc(studentDoc(uid, oldLesson.studentId), { totalOwed:  
increment(-oldLesson.price) });  
  await updateDoc(studentDoc(uid, data.studentId), { totalOwed:  
increment(data.price) });  
}
```

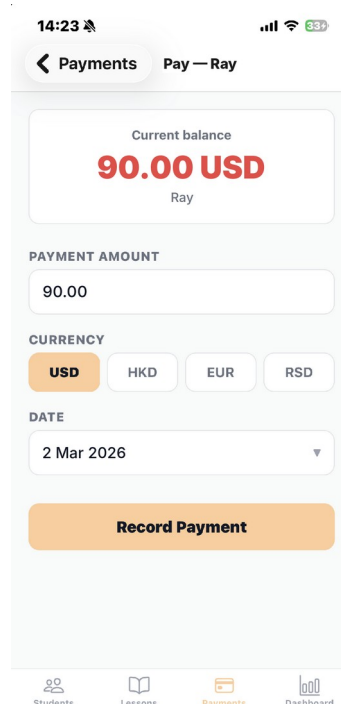
Pri brisanju časa (`deleteLesson`), dokument se prvo briše iz kolekcije `lessons`, a zatim se `totalOwed` studenta umanjuje za cenu obrisano časa (`increment(-lesson.price)`). U sva tri slučaja operacije nad dokumentom časa i dokumentom studenta izvode se kao dva odvojena, sekvencijalna `async` poziva, a ne unutar Firestore transakcije (`runTransaction`) ili grupnog pisanja (`writeBatch`).

3.4. Modul Plaćanja i FIFO logika

Payments Screen prikazuje listu svih studenata, pri čemu su studenti sa dugovanjem izdvojeni u odeljak „Payment due“ na vrhu ekrana (crveno), dok su studenti bez dugovanja prikazani ispod (zeleno, „Paid up“), kao na Slici 12. Dodirom na studenta sa dugovanjem otvara se ekran za unos uplate (Slika 13), na kom je iznos unapred popunjen tekućim dugovanjem studenta.



Slika 12 - Ekran Payments sa studentima koji duguju i onima koji su izmirili dug



Slika 13 - Ekran za unos uplate (Add Payment)

Funkcija `addPayment` u `src/services/payments.ts` izvodi tri operacije uzastopno pri svakom unosu uplate. Prva je jednostavan upis dokumenta u kolekciju `payments`. Druga je umanjeње `totalOwed` studenta preko `increment(-data.amount)`, analogno mehanizmu opisanom u 3.3. Treća, i najsloženija, jeste raspodela uplate na pojedinačne časove po FIFO principu (*first in, first out*), najstariji neplaćeni časovi se naplaćuju prvi.

Algoritam se izvodi u četiri koraka:

1. Preuzimanje svih časova studenta. Umesto da se već filtrira po `isPaid` na nivou `Firestore` upita, preuzimaju se svi časovi datog studenta (`where('studentId', '==', data.studentId)`), a filtriranje po statusu naplaćenosti vrši se naknadno, na klijentu.
2. Filtriranje i sortiranje neplaćenih časova. Od preuzetih dokumenata izdvajaju se samo oni sa `isPaid === false`, a zatim se sortiraju rastuće po datumu (najstariji prvi).

Kod 9: Filtriranje i sortiranje neplaćenih časova (FIFO priprema)

```
const unpaid = lessonsSnap.docs
  .filter((d) => !d.data().isPaid)
  .map((d) => ({ ref: d.ref, price: d.data().price, date:
    (d.data().date).toDate() })))
  .sort((a, b) => a.date.getTime() - b.date.getTime());
```

3. Iterativno trošenje iznosa uplate. Kroz sortiranu listu se prolazi redom, a promenljiva `remaining` prati preostali, još neraspoređeni iznos uplate. Čas se označava kao naplaćen (`isPaid: true`) samo ako preostali iznos u potpunosti pokriva njegovu cenu.

Kod 10: Iterativno označavanje časova kao naplaćenih (FIFO petlja)

```
let remaining = data.amount;
for (const lesson of unpaid) {
  if (remaining <= 0) break;
  if (remaining >= lesson.price) {
    await updateDoc(lesson.ref, { isPaid: true });
    remaining -= lesson.price;
  }
}
```

4. Prekid petlje. Petlja se prekida čim remaining padne na 0 ili ispod te vrednosti, ili čim se iscrpi lista neplaćenih časova.

Granični slučajevi

Ukoliko je remaining manji od cene tekućeg časa (lesson.price), grana if (remaining >= lesson.price) se ne izvršava, čas se ne označava kao naplaćen, a petlja nastavlja na sledeći čas u nizu, umesto da stane, što znači da uplata koja delimično pokriva jedan čas ne „preskače” ka manjem sledećem času koji bi mogla u potpunosti pokriti, već se iterira strogo po hronološkom redosledu bez pokušaja optimalnog uparivanja. Ukoliko je iznos uplate veći od ukupnog zbira svih neplaćenih časova (npr. korisnik plati unapred ili preplati), svi neplaćeni časovi bivaju označeni kao isPaid: true, a totalOwed studenta (već umanjen za pun iznos uplate) postaje negativan broj. Za razliku od ranije verzije aplikacije, ovaj višak se sada eksplicitno prikazuje korisniku: StudentsListScreen, StudentDetailScreen i PaymentsScreen proveravaju, pored postojećeg uslova totalOwed > 0 („owes”), i uslov totalOwed < 0 („hasCredit”), pri čemu se u tom slučaju umesto oznake „Paid up” prikazuje „Credit: X” sa apsolutnom vrednošću preplaćenog iznosa. Grupisanje studenata (owing/paidUp) ostaje nepromenjeno, jer student sa kreditom formalno ništa ne duguje, menja se samo vizuelni prikaz pojedinačne stavke.

Takođe, pošto operacije unutar for petlje (updateDoc(lesson.ref, ...)) nisu grupisane u writeBatch, u slučaju greške na mreži usred iteracije deo časova može ostati označen kao naplaćen, a deo ne, bez mogućnosti automatskog vraćanja na prethodno stanje (rollback).

3.5. Dashboard i eksterni API

Servisni sloj Dashboard modula (src/services/dashboard.ts) razdvaja preuzimanje sirovih podataka od njihove agregacije. Funkcija `fetchAllLessons` preuzima sve časove korisnika iz Firestore-a (jednokratno, preko `getDocs`, a ne kao real-time pretplata), dok `fetchExchangeRates` poziva spoljni REST servis `ExchangeRate-API`:

Kod 11: Poziv eksternog API-ja za kurseve valuta (`fetchExchangeRates`)

```
export async function fetchExchangeRates(): Promise<Record<string,
number> | null> {
  try {
    const apiKey = process.env.EXPO_PUBLIC_EXCHANGERATE_API_KEY;
    const res = await fetch(`https://v6.exchangerate-api.com/v6/${
      apiKey}/latest/USD`);
    const json = await res.json();
    if (json.result === 'success') return json.conversion_rates as
Record<string, number>;
    return null;
  } catch {
    return null;
  }
}
```

Oba poziva (`fetchAllLessons` i `fetchExchangeRates`) izvode se paralelno preko `Promise.all` unutar `DashboardScreen` komponente, prilikom inicijalnog učitavanja i pri promeni prijavljenog korisnika. Ukoliko `fetchExchangeRates` vrati `null` (mrežna greška, neuspešan HTTP odgovor ili `json.result !== 'success'`), komponenta postavlja lokalno stanje `ratesAvailable` na `false` i prekida dalje izračunavanje agregata.

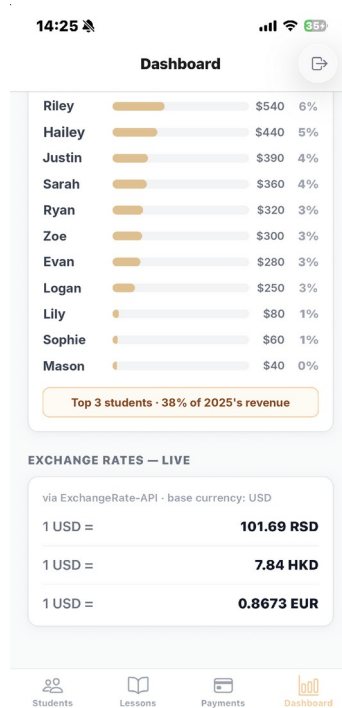
Agregacija po mesecima vrši funkcija `buildAnnualData`, koja za izabranu godinu generiše niz od tačno 12 elemenata, i za svaki mesec izračunava zbir cena časova (konvertovanih u USD preko `convertToUSD`) i broj časova održanih u tom mesecu. Za tekuću godinu, meseci koji tek dolaze označavaju se poljem `isFuture: true` i njihov zbir se ne računa, dok se tekući mesec posebno obeležava poljem `isCurrent: true` radi isticanja na grafikonu.

Agregacija po studentima vrši se preko `buildStudentMonthlyEarnings` (za tekuću godinu, grupisanje po tekućem mesecu) ili `buildStudentAnnualEarnings` (za prethodne godine, grupisanje po celoj godini); `DashboardScreen` bira jednu od ove dve funkcije na osnovu toga da li je `selectedYear` jednak tekućoj godini. Obe funkcije grupišu časove po `studentId`, sabiraju konvertovane iznose, i računaju procentualno učešće svakog studenta u ukupnoj sumi (`percentage`).

Renderovanje grafikona obavlja se preko komponente BarChart iz biblioteke react-native-gifted-charts. Podaci za grafikon formiraju se mapiranjem izlaza buildAnnualData funkcije, pri čemu se boja svakog stubića određuje uslovno: izabrani (tapnuti) stubić dobija boju #F59E0B, tekući mesec #FFCC99, budući meseci (bez podataka) #F3F4F6, a prošli meseci #E5C08A. Tapanje na stubić (onPress) postavlja lokalno stanje selectedBarIdx, koje zatim upravlja i bojom i tekstualnim prikazom detalja iznad grafikona. Širina grafikona izračunava se dinamički na osnovu broja stubića, kako bi grafikon sa više meseci podataka postao horizontalno skrolabilan umesto da se stubovi sabiju ispod širine ekrana. Kada ratesAvailable iznosi false, i BarChart i StudentEarningsChart komponenta zamenjuju se tekstualnom porukom o nedostupnosti kursa, umesto da pokušaju prikaz sa nepotpunim ili netačnim podacima. Izgled statističkih kartica i interaktivnog grafikona zarade po mesecima prikazan je na Slici 14, dok je rang lista prihoda po studentima, zajedno sa karticom prikaza live kurseva valuta (ExchangeRate-API), prikazana na Slici 15.



Slika 14 - Dashboard: statistika i grafikon zarade po mesecima



Slika 15 - Dashboard: rang lista prihoda po studentima i live kursevi valuta

4. Testiranje

U projektu nije pronađen nijedan automatizovani test: ne postoji direktorijum tests/, ne postoje fajlovi sa nastavkom .test.ts(x) ili .spec.ts(x), nema konfiguracionog fajla za Jest (jest.config.*) niti za Detox (.detoxrc), a u package.json ne postoji definisana test skripta niti su među zavisnostima navedene biblioteke jest, jest-expo, detox ili @testing-library/*. Iz ovoga proizlazi da testiranje aplikacije nije bilo automatizovano, već isključivo manuelno/funkcionalno, izvođeno direktnim korišćenjem aplikacije (Expo Go / simulator) tokom razvoja. Scenariji opisani u nastavku predstavljaju rekonstrukciju takvog manualnog testiranja, zasnovanu na stvarnoj poslovnoj logici implementiranoj u servisnom sloju (services/), a ne na formalno definisanim i izvršenim test slučajevima u okviru testing frameworka.

4.1. Funkcionalno testiranje

Modul Studenti (services/students.ts)

Tabela 4: Test slučajevi: modul Studenti

ID	Scenario	Koraci	Očekivan rezultat
TC-ST-01	Kreiranje studenta i inicijalna vrednost totalOwed	1. Otvoriti StudentsList → „+ Add”. 2. Uneti ime, satnicu i valutu. 3. Sačuvati (addStudent).	Novi dokument se pojavljuje preko real-time pretplate; totalOwed je 0, iako nije eksplicitno uneto (addStudent ga eksplicitno postavlja pre upisa).
TC-ST-02	Izmena podataka studenta	1. Otvoriti StudentDetail. 2. „Edit” → izmeniti satnicu/valutu. 3. Sačuvati (updateStudent).	Izмене se odražavaju odmah bez ručnog osvežavanja (onSnapshot). totalOwed ostaje nepromenjen.
TC-ST-03	Brisanje studenta	1. StudentDetail → „Delete Student” → potvrda. 2. deleteStudent.	Dokument se uklanja iz students; korisnik se vraća na StudentsList.
TC-ST-04	Brisanje studenta briše i povezane zapise	1. Kreirati studenta sa bar jednim časom i jednom uplatom. 2. Obrisati studenta (deleteStudent). 3. Proveriti kolekcije lessons i payments za tog studenta.	Dokumenti u lessons i payments koji referenciraju obrisaniog studenta bivaju obrisani zajedno sa studentom, u okviru iste atomarne batch operacije (writeBatch), potvrđeno da ne ostaju osiroteli zapisi u bazi.
TC-ST-05	Pretraga na StudentsList	1. Uneti deo imena u polje za pretragu.	Lista se filtrira klijentski, bez dodatnog Firestore upita.

Izvor: autor

Modul Lekcije (services/lessons.ts)

Tabela 5: Test slučajeви: modul Lekcije

ID	Scenario	Koraci	Očekivan rezultat
TC-LS-01	Dodavanje časa i uvećanje totalOwed	1. Zabeležiti totalOwed pre unosa. 2. Dodati čas cene 20 (addLesson).	totalOwed se uvećava za 20, posledica increment(price).
TC-LS-02	Izmena cene kod istog studenta	1. Izmeniti cenu sa 20 na 30, isti student (updateLesson).	totalOwed se koriguje za razliku (+10), ne postavlja se direktno na 30.
TC-LS-03	Premeštanje časa na drugog studenta	1. Promeniti studentId postojećeg časa sa A na B.	totalOwed(A) se umanjuje, totalOwed(B) se uvećava za istu cenu; zbir dugovanja ostaje isti.
TC-LS-04	Brisanje časa	1. Obrisati neplaćen čas (deleteLesson).	totalOwed se umanjuje tačno za cenu obrisanog časa.
TC-LS-05	Auto-cena pri dodavanju	1. Izabrati trajanje 25, zatim 50, zatim 75 min.	Cena se automatski postavlja na 10, 20, 30 (DURATION_PRICES).
TC-LS-06	Ručni override cene	1. Trajanje 50 min (auto-cena 20). 2. Ručno uneti 25. 3. Sačuvati.	Čas se čuva sa cenom 25, polje ostaje editabilno i nadjačava predlog.
TC-LS-07	Grupisanje po studentu na LessonsList	1. Dodati časove za dva studenta. 2. Otvoriti LessonsList.	Časovi prikazani grupisano, sortirano po imenu (localeCompare).

Izvor: autor

Modul Plaćanja (services/payments.ts)

Tabela 6: Test slučajevi: modul Plaćanja

ID	Scenario	Koraci	Očekivan rezultat
TC-PM-01	Uplata koja tačno pokriva dug (FIFO)	1. Student ima neplaćen čas od 20. 2. Uplata od tačno 20.	totalOwed pada na 0; čas isPaid: true (remaining >= price je tačno na granici).
TC-PM-02	Uplata koja delimično pokriva čas (granični slučaj)	1. Neplaćen čas od 30. 2. Uplata od 20 (remaining=20 < 30).	totalOwed pada na 10, ali čas ostaje isPaid: false, nijedan čas nije obeležen kao naplaćen.
TC-PM-03	Uplata koja premašuje ukupan dug (preplata)	1. Student ima neplaćene časove u ukupnom iznosu od 20 (jedan čas). 2. Uneti uplatu od 50.	Svi neplaćeni časovi studenta se označavaju isPaid: true. totalOwed pada na -30 (increment(-50) primenjen na prethodnih 20).
TC-PM-04	Prikaz preplate u korisničkom interfejsu	1. Nastaviti od TC-PM-03 (totalOwed = -30). 2. Otvoriti StudentsList, StudentDetail i PaymentsScreen.	Student se prikazuje sa oznakom „Credit: 30.00 [valuta]" umesto „Paid up", StudentsListScreen prikazuje „Credit: X" u boji kartice, StudentDetailScreen menja naslov kartice iz „Total Owed" u „Credit" i prikazuje apsolutnu vrednost, a PaymentsScreen prikazuje status „Credit" i iznos umesto kvačice (✓). Student i dalje spada u „paid up" grupu (ne duguje ništa), ali se vizuelno razlikuje od studenta sa totalOwed === 0.
TC-PM-05	Redosled naplate kod više neplaćenih časova	1. Stariji čas 10 (1.3.), noviji čas 20 (15.3.). 2. Uplata od 10.	Naplaćuje se isključivo stariji čas, lista je sortirana rastuće po datumu.

Izvor: autor

Dashboard (services/dashboard.ts)

Tabela 7: Test slučajevi: Dashboard

ID	Scenario	Koraci	Očekivan rezultat
TC-DB-01	Agregacija kad je ExchangeRate-API dostupan	1. Otvoriti Dashboard sa validnim API ključem i mrežom.	fetchExchangeRates vraća conversion_rates; ratesAvailable=true; prikazuju se grafikoni i kursevi.
TC-DB-02	Ponašanje kad API nije dostupan	1. Simulirati nedostupnost servisa pre otvaranja Dashboard-a.	ratesAvailable=false; agregati se ne računaju (rani return); prikazuju se poruke „unavailable” umesto grafikona.
TC-DB-03	„This month” naspram „Annual total”	1. Tekuća godina → proveriti oznaku. 2. Preći na prethodnu godinu strelicom.	Za tekuću godinu prikazuje se „This month”; za prethodnu „Annual total” (zbir cele godine).
TC-DB-04	Aktivni studenti — poređenje meseci	1. Zabeležiti broj aktivnih studenata tekućeg i prethodnog meseca.	Prikazuje se delta („+N/-N/Same as last month”) sa odgovarajućom bojom.

Izvor: autor

4.2. Testiranje autentifikacije i poznatih problema

Scenariji prijave i sesije

Tabela 8: Test slučajevi: autentifikacija

ID	Scenario	Koraci	Očekivan rezultat
TC-AU-01	Uspešna prijava	1. Ispravan e-mail i lozinka. 2. „Sign In“.	onAuthStateChanged postavlja user; prikazuje se MainTabs.
TC-AU-02	Pogrešna lozinka	1. Ispravan e-mail, pogrešna lozinka.	Alert „Incorrect password.“ ili „Incorrect email or password.“
TC-AU-03	Nepostojeći nalog	1. Uneti neregistrovan e-mail.	Poruka „No account found with this email.“
TC-AU-04	Nevalidan format e-maila	1. Uneti tekst koji nije validna e-mail adresa.	Poruka „Please enter a valid email address.“
TC-AU-05	Nemapiran kod greške	1. Izazvati grešku van rečnika (npr. auth/network-request-failed).	Rezervna poruka „Something went wrong. Please try again.“
TC-AU-06	Lozinke se ne poklapaju (registracija)	1. Različite vrednosti u Password/Confirm Password.	Lokalna poruka „Passwords do not match.“ pre poziva ka Firebase-u.
TC-AU-07	Perzistencija sesije nakon restarta	1. Prijaviti se. 2. Force-close app. 3. Ponovo pokrenuti.	Kratak ActivityIndicator, zatim direktno MainTabs, bez Login ekrana.
TC-AU-08	Odjava	1. Dashboard → ikonica odjave → potvrda.	signOut(auth); user postaje null; prikazuje se AuthStack.
TC-AU-10	Neuspešna registracija, mejl već zauzet	1. Na Register ekranu uneti e-mail postojećeg naloga.	Prikazuje se poruka „An account with this email already exists.“ (auth/email-already-in-use), preko istog rečnika AUTH_ERROR_MESSAGES koji koristi i Login ekran.

Izvor: autor

Studija slučaja: pronađen i ispravljen bug, prazan (beo) DatePicker spinner na iOS-u

U dokumentaciji projekta (PRD.md, sekcija 17.0 „Bug: Date Picker Blank on iOS“) formalno je zaveden i opisan bug otkriven tokom manuelnog testiranja unosa datuma na iOS-u. Ekran: AddEditLessonScreen (naknadno potvrđeno da se isti problem javlja i u AddPaymentScreen). Simptom: dodirom na polje za datum, modal sa spinner biračem (display="spinner") ispravno se otvara, ali je sam sadržaj spinnera prazan/nevidljiv, beo tekst na beloj pozadini. Uzrok: DatePicker komponenta sa display="spinner" na novijim verzijama iOS-a podrazumevano renderuje tekst u beloj boji; bez eksplicitno postavljenog svojstva themeVariant="light", tekst spinnera ostaje nevidljiv na svetloj pozadini modala.

Rešenje: dodavanje props-a `themeVariant="light"` na `DateTimePicker` komponentu unutar iOS modala u oba pogođena ekrana.

Ispravka je potvrđena i u git istoriji projekta, komit e59c056 („final bugs and changes completed”) sadrži identičnu izmenu u oba fajla:

Kod 12: Ispravka DateTimePicker bug-a (git diff, komit e59c056)

```
<DateTimePicker
  value={tempDate}
  mode="date"
  display="spinner"
+  themeVariant="light"
  onChange={(_, selected) => {
    if (selected) setTempDate(selected);
  }}
/>
```

Regresioni test nakon ispravke (TC-AU-09, po analogiji sa gore navedenim scenarijima) sastojao bi se od ponovnog otvaranja date-picker modala na oba ekrana (`AddEditLessonScreen` i `AddPaymentScreen`) na iOS uređaju/simulatoru i vizuelne provere da su cifre datuma, meseca i godine u spinneru čitljive (tamni tekst na svetloj pozadini).

Napomena o razvoju poruka grešaka pri prijavi

U `PRD.md` (sekcija 17.3, „Custom Auth Error Messages”) planirano je uvođenje čitljivih poruka grešaka pri neuspešnoj prijavi, sa predloženom implementacijom u vidu switch iskaza unutar pomoćne funkcije `getAuthErrorMessage`. Kontrolne stavke ove sekcije u `PRD.md` ostale su neobeležene (nezavršeno), međutim, stvarna implementacija (opisana u poglavlju 3.1) ostvaruje funkcionalno istovetan, pa i širi ishod, mapiranje proširenog skupa Firebase kodova grešaka, zajedničkog za `Login` i `Register` ekran, ali kroz statički `Record<string, string>` rečnik izdvojen u poseban modul (`constants/authErrors.ts`), a ne kroz switch konstrukciju unutar samog ekrana.

Zaključna razmatranja

U okviru ovog diplomskog rada razvijena je mobilna aplikacija LessonLynx (ESL Lesson Tracker), namenjena nastavniku engleskog kao stranog jezika koji posluje samostalno, za objedinjeno praćenje studenata, održanih časova, uplata i mesečne/godišnje zarade. Aplikacija je realizovana korišćenjem React Native/Expo tehnologije za mobilni interfejs, uz Firebase (Authentication i Firestore) kao cloud backend rešenje, i integraciju eksternog ExchangeRate-API servisa za konverziju zarade u više valuta u realnom vremenu.

Kroz analizu zahteva, dizajn sistema, implementaciju i testiranje pokazano je da razvijeno rešenje ispunjava postavljene ciljeve: omogućena je evidencija studenata sa osnovnim kontakt podacima, unos i pregled časova sa automatskim predlogom cene na osnovu trajanja, praćenje dugovanja po studentu uz automatsko raspoređivanje uplata na najstarije neplaćene časove (FIFO princip), kao i pregled agregirane statistike zarade kroz interaktivni grafikon po mesecima i rang listu prihoda po studentima. Real-time sinhronizacija podataka, ostvarena preko Firestore-ovih onSnapshot pretplata, obezbeđuje da se promene odmah odražavaju na korisnički interfejs bez potrebe za ručnim osvežavanjem, dok perzistentna Firebase Auth sesija korisniku omogućava neprekidan rad bez ponovljenog prijavljivanja.

Ostvarenost ciljeva istraživanja

Poređenjem sa ciljevima istraživanja definisanim u uvodnom delu rada, može se konstatovati da su svi postavljeni ciljevi u potpunosti ostvareni:

- **Evidencija studenata i njihovih podataka za kontakt:** ostvarena je kroz modul Students, uključujući ime, kontakt podatke, valutu zaduživanja i satnicu, uz automatsko inicijalizovanje dugovanja na nulu za svakog novog studenta (poglavljje 2.1 i 3.2).
- **Unos i pregled časova sa automatskim obračunom cene:** ostvareni su kroz modul Lessons, sa fiksnim trajanjima časa (25/50/75 minuta) kojima odgovara automatski predlog cene, uz mogućnost ručnog prepisivanja predložene vrednosti (poglavljje 3.3).
- **Praćenje dugovanja uz automatsko označavanje plaćenih časova:** ostvareno je kroz modul Payments i FIFO algoritam implementiran u addPayment funkciji, koji najstarije neplaćene časove naplaćuje prve, uz atomarno ažuriranje dugovanja preko Firestore operacije increment() (poglavljje 3.4).
- **Pregled statistike zarade sa konverzijom valuta:** ostvaren je kroz Dashboard modul, koji prikazuje mesečnu i godišnju zaradu, rang listu prihoda po studentima i konverziju u različite valute korišćenjem tekućih kurseva sa ExchangeRate-API servisa (poglavljje 3.5).

Poseban cilj, demonstriranje integracije eksternog API-ja u realnom vremenu i primene principa reaktivnog ažuriranja podataka, takođe je ostvaren, kako kroz opisanu ExchangeRate-API integraciju, tako i kroz dosledno korišćenje Firestore onSnapshot pretplata na više ekrana aplikacije (Students, Lessons, StudentDetail).

Ograničenja rešenja

Tokom analize i implementacije uočen je niz ograničenja trenutnog rešenja, koja predstavljaju prostor za unapređenje pre eventualnog šireg korišćenja aplikacije:

- **Nekonzistentnost pri paralelnim izmenama:** operacije koje ažuriraju totalOwed studenta (pri dodavanju/izmeni/brisanju časa, kao i pri unosu uplate) izvode se kao dva odvojena, sekvencijalna Firestore poziva, a ne unutar transakcije (runTransaction) ili grupnog pisanja (writeBatch), što u slučaju greške na mreži usred izvršavanja teorijski može dovesti do neusklađenosti između totalOwed vrednosti i stvarnog stanja časova. Delimičan izuzetak predstavlja operacija brisanja studenta, koja je, za razliku od ostalih, već izvedena kao atomarna writeBatch operacija (videti 3.2).
- **Odsustvo automatizovanog testiranja:** aplikacija ne poseduje automatizovane testove (jedinичne ili integracione), zbog čega se ispravnost poslovne logike oslanja isključivo na manuelno testiranje opisano u četvrtom poglavlju. Izolacija podataka po korisniku je, s druge strane, potvrđena i na nivou Firestore Security Rules konfigurisanih u Firebase Console-u (videti 2.1).
- **Implicitan dizajn sistem:** paleta boja i tipografija nisu definisane na jednom centralizovanom mestu (npr. theme.ts), već su ponovljene lokalno kroz stilove svakog ekrana, što otežava buduće izmene vizuelnog identiteta aplikacije.
- **Zastarelo ime studenta na starim zapisima:** polje studentName na dokumentima kolekcija lessons i payments ne ažurira se retroaktivno ukoliko korisnik naknadno promeni ime studenta (videti 2.3), svestan kompromis denormalizacije, a ne propust.

Refleksija o procesu razvoja

Proces razvoja aplikacije potvrdio je nekoliko principa koji su u fazi analize tehnologija (poglavlje 1) predstavljeni teorijski. Denormalizacija podataka (totalOwed, studentName), uvedena radi prilagođavanja Firestore-ovom modelu upita, pokazala se kao neophodna već pri malom broju studenata i časova korišćenih tokom testiranja, bez nje bi svaki prikaz liste zahtevao dodatna čitanja iz baze za svaku stavku. Sa druge strane, upravo je ova ista denormalizacija otvorila najveći broj ograničenja opisanih u prethodnom odeljku (rizik neusklađenosti pri paralelnim izmenama, zastarelo ime studenta na starim zapisima), što ukazuje na tipičan kompromis u projektovanju aplikacija zasnovanih na dokumentno orijentisanim bazama podataka, dobitak u broju čitanja i brzini prikaza plaća se dodatnom složenošću održavanja konzistentnosti. Implementacija FIFO logike raspodele uplata

pokazala se kao najsloženiji pojedinačni deo poslovne logike u radu, upravo zbog potrebe da se ispravno obrade granični slučajevi (delimično pokriven čas, preplata) koji na prvi pogled nisu očigledni iz osnovnog opisa zahteva, a otkriveni su tek detaljnom analizom koda i osmišljavanjem test scenarija u četvrtom poglavlju.

Moguća proširenja

Na osnovu uočenih ograničenja i potreba koje se prirodno nameću kroz svakodnevnu upotrebu aplikacije, izdvajaju se sledeći pravci za dalji razvoj:

- **Nedeljni prikaz rasporeda:** uvođenje nedeljnog kalendarskog prikaza časova (dani u nedelji kao kolone, vremenski termini kao redovi) predstavljalo bi značajno unapređenje u odnosu na trenutni prikaz liste grupisane po studentu, olakšavajući planiranje, uočavanje slobodnih termina i eventualnih preklapanja u rasporedu.
- **Generisanje i slanje faktura:** za studente sa većim brojem neplaćenih časova, aplikacija bi mogla automatski da generiše fakturu (izvod dugovanja) u deljivom formatu, koju bi nastavnik jednim tapom mogao poslati studentu putem aplikacija za razmenu poruka, uz mogućnost masovnog označavanja svih časova studenta kao plaćenih nakon potvrde uplate.
- **Praćenje stope otkazivanja po studentu:** dodatna statistika koja bi za svakog studenta pratila odnos zakazanih i otkazanih časova (procenat otkazivanja) pomogla bi nastavniku da prepozna manje pouzdane studente i shodno tome prilagodi prioritet pri zakazivanju novih termina.
- **Pregled opterećenja nastavnika:** vizuelni pregled (heatmap) ukupnog broja nastavnih sati raspoređenih po danima u nedelji i delovima dana (jutro/popodne/veče) omogućio bi nastavniku bolji uvid u sopstveno radno opterećenje i lakše planiranje slobodnog vremena.
- **Push notifikacije za dugovanja:** slanje automatskih podsetnika studentima sa dugovanjem, umesto ručnog praćenja preko Dashboard i Payments ekrana.
- **Izvoz podataka:** mogućnost izvoza podataka (studenti, časovi, uplate) u PDF ili CSV format, radi arhiviranja ili knjigovodstvenih potreba.
- **Podrška za više nastavnika (multi-teacher):** proširenje modela podataka tako da više nastavnika može nezavisno koristiti istu instalaciju aplikacije, uz odgovarajuću izolaciju podataka po nastavniku.
- **Rad bez internet konekcije (offline sync):** keširanje podataka lokalno na uređaju, kako bi osnovne funkcionalnosti (pregled već učitanih studenata i časova) ostale dostupne i bez aktivne internet konekcije.

Uzimajući sve navedeno u obzir, može se zaključiti da razvijeno rešenje u potpunosti ispunjava postavljene funkcionalne i tehničke zahteve za potrebe izrade ovog diplomskog rada, dok istovremeno ostavlja jasno definisan i realno ostvariv prostor za dalje unapređenje u pravcu produkcione, šire primenjive aplikacije. Vrednost rešenja opisanog u ovom radu ne ogleda se samo u ispunjenim akademskim zahtevima, već i u njegovoj potvrđenoj

primenjivosti u realnom kontekstu, princip da BaaS pristup omogućava nezavisnim programerima razvoj funkcionalnih rešenja bez sopstvene serverske infrastrukture (Moroney, 2017), pomenut u uvodnom delu rada kao teorijska osnova, ovde je potvrđen kroz aplikaciju koja je u trenutku pisanja rada u svakodnevnoj upotrebi. Slično tome, pažnja posvećena jednostavnosti korišćenja na malom ekranu (Nielsen & Budiu, 2012) odrazila se u pozitivnim reakcijama ostalih nastavnika kojima je aplikacija demonstrirana, što rad zaokružuje kao primer u kome akademski i praktični rezultati direktno potvrđuju jedni druge.

Literatura

- [1] Banks, A., & Porcello, E. (2020) Learning React: Modern Patterns for Developing React Apps, 2. izd., O'Reilly Media, Sebastopol
- [2] Boduch, A., & Derks, R. (2024) React and React Native: Build Cross-Platform Web and Mobile Applications Using React 18 and TypeScript, 5. izd., Packt Publishing, Birmingham
- [3] Eisenman, B. (2017) Learning React Native: Building Native Mobile Apps with JavaScript, 2. izd., O'Reilly Media, Sebastopol
- [4] Expo (2026) Documentation [on-line]. Expo (650 Industries, Inc.). Preuzeto sa <https://docs.expo.dev> [Dostupno dana 07.08.2026.]
- [5] ExchangeRate-API (2026) API Documentation — Standard Requests [on-line]. Preuzeto sa <https://www.exchangerate-api.com/docs/standard-requests> [Dostupno dana 07.08.2026.]
- [6] Google LLC (2026) Firebase Documentation [on-line]. Preuzeto sa <https://firebase.google.com/docs> [Dostupno dana 07.08.2026.]
- [7] Moroney, L. (2017) The Definitive Guide to Firebase: Build Android Apps on Google's Mobile Platform, Apress, New York
- [8] Nielsen, J., & Budiu, R. (2012) Mobile Usability, New Riders, Berkeley
- [9] React Native (2026) Documentation [on-line]. Meta Platforms, Inc. Preuzeto sa <https://reactnative.dev/docs/getting-started> [Dostupno dana 07.08.2026.]
- [10] React Navigation (2026) Getting Started [on-line]. Preuzeto sa <https://reactnavigation.org/docs/getting-started/> [Dostupno dana 07.08.2026.]
- [11] Živković, M. (2024) Razvoj mobilnih aplikacija: Android Java programiranje, 2. izd., Univerzitet Singidunum, Beograd