Razvoj mobilne aplikacije za praćenje stanja automobila pomoću React Native softverskog okvira

Vuk, Daniel

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DANIEL VUK

RAZVOJ MOBILNE APLIKACIJE ZA PRAĆENJE STANJA AUTOMOBILA POMOĆU REACT NATIVE SOFTVERSKOG OKVIRA

Završni rad

Pula, ožujak 2025. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

DANIEL VUK

RAZVOJ MOBILNE APLIKACIJE ZA PRAĆENJE STANJA AUTOMOBILA POMOĆU REACT NATIVE SOFTVERSKOG OKVIRA

Završni rad

JMBAG: 0303094620, redoviti student
Studijski smjer: Sveučilišni prijediplomski studij informatika
Predmet: Programiranje
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: prof. dr. sc. Tihomir Orehovački

Pula, ožujak 2025. godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Daniel Vuk, kandidat za prvostupnika informatike ovime izjavljujem da je ovaj Završni rad rezultat isključivo mojega vlastitog rada, da se temelji na mojim istraživanjima te da se oslanja na objavljenu literaturu kao što to pokazuju korištene bilješke i bibliografija. Izjavljujem da niti jedan dio Završnog rada nije napisan na nedozvoljeni način, odnosno da je prepisan iz kojega necitiranog rada, te da ikoji dio rada krši bilo čija autorska prava. Izjavljujem, također, da nijedan dio rada nije iskorišten za koji drugi rad pri bilo kojoj drugoj visokoškolskoj, znanstvenoj ili radnoj ustanovi.

Student

_____

U Puli, 12.03.2025.

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Daniel Vuk dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj Završni rad pod nazivom „Razvoj mobilne aplikacije za praćenje stanja automobila pomoću React Native softverskog okvira" koristi na način da gore navedeno autorsko djelo, kao cjeloviti tekst trajno objavi u javnoj internetskoj bazi Sveučilišne knjižnice Sveučilišta Jurja Dobrile u Puli te kopira u javnu internetsku bazu završnih radova Nacionalne i sveučilišne knjižnice (stavljanje na raspolaganje javnosti), sve u skladu s Zakonom o autorskom pravu i drugim srodnim pravima i dobrom akademskom praksom, a radi promicanja otvorenoga, slobodnoga pristupa znanstvenim informacijama.

Za korištenje autorskog djela na gore navedeni način ne potražujem naknadu.

U Puli, 12.02.2025.

Potpis

_____

SAŽETAK

U ovom završnom radu izrađena je mobilna aplikacija za praćenje stanja automobila koristeći React Native softverski okvir. Aplikacija omogućava korisnicima unos vozila i kategorija te praćenje aktivnosti za svako vozilo i kategorije, za koje će korisnik primati podsjetnike za nadolazeće aktivnosti. Osim toga, aplikacija nudi personalizirane kategorije, za koje se može postavljati kilometarske ili vremenske intervale. Za autentifikaciju korisnika i pohranu podataka koristi se Firebase. Sve funkcionalnosti aplikacije detaljno su opisane u radu.

KLJUČNE RIJEČI: React Native, praćenje stanja automobila, mobilna aplikacija, upravljanje vozilima

ABSTRACT

This thesis presents the development of a mobile application for monitoring the condition of a car using the React Native software framework. The application allows users to input vehicle details and categories, as well as track activities for each vehicle and category, with reminders for upcoming tasks. Additionally, the app offers personalized categories, where users can set mileage or time intervals. Firebase is used for user authentication and data storage. All functionalities of the application are described in detail in the thesis.

KEYWORDS: React Native, vehicle condition monitoring, mobile application, fleet management

SADRŽAJ

1. UVOD

U današnjem brzom i tehnološki naprednom društvu, praćenje stanja vozila može biti korisno, ne samo za tvrtke s velikim flotama vozila, već i za pojedince. Svatko tko posjeduje automobil, imao je potrebu nešto zapamtiti, bio to, datum za registraciju ili što je sve popravljano na automobilu. U velikim tvrtkama koje upravljaju flotama vozila, praćenje stanja automobila, troškova ili prihoda, postaje ključno za održivost poslovanja. Tim načinom rada postiže se pravovremeno održavanje automobila, smanjenje broja nesreća, troškova te povećanje produktivnosti što poboljšava cjelokupnu učinkovitost organizacije (Machaba & Ndou, 2024).

Slični izazovi s održavanjem i praćenjem vozila javljaju se i kod privatnih vlasnika, koji često nemaju sustav za praćenje svih važnih podataka o svom vozilu. Shodno tome, mogu nastati neočekivani troškovi, propuštene prilike za uštedu ili čak do ozbiljnih problema za vozilo.

Razvojem mobilnih aplikacija, korisnicima se omogućuje da na jednostavan način prate stanje svojih automobila. Korištenjem tehnologija poput React Native softverskog okvira omogućava se jednostavni razvoj aplikacije za više platformi.

Cilj ovog rada je razviti mobilnu aplikaciju koja omogućuje korisnicima dodavanje automobila, praćenje servisa, troškove i ostalih informacija vezane za automobile te primanje obavijesti za nadolazeće aktivnosti.

Korisnik može kreirati kategorije za praćenje aktivnosti i odrediti vremenske ili kilometarske intervale. Aplikacija na temelju tih postavki šalje obavijesti isključivo za odabrane aktivnosti. Tako korisnik ima potpunu kontrolu nad održavanjem vozila, što doprinosi smanjenju troškova i duljem vijeku trajanja automobila.

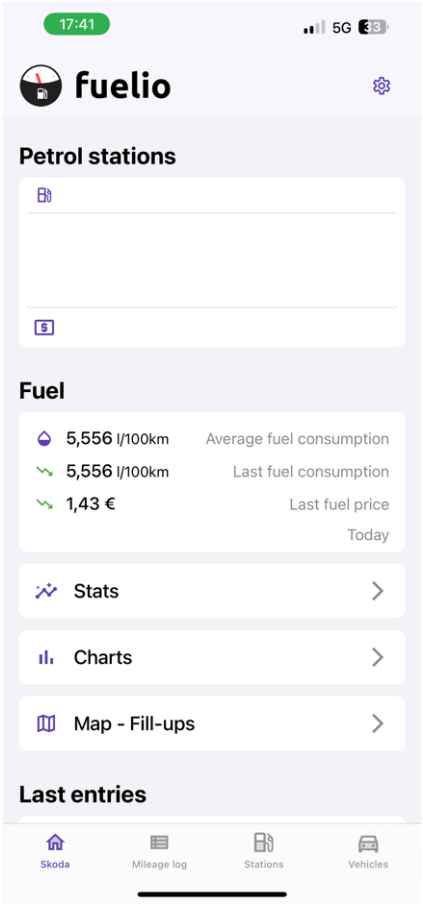
Ovaj završni rad se sastoji od šest poglavlja, u drugom poglavlju je napravljena analiza srodnih aplikacija, gdje su opisane tri najpopularnije aplikacije za upravljanje stanjima automobila. U trećem poglavlju opisan je proces izrade mobilnih aplikacija gdje se prolazi kroz svaku fazu životnog ciklusa razvoja softvera. Nadalje, u četvrtom poglavlju su opisane tehnologije koje se koriste za izradu aplikacije u ovome radu te peto poglavlje, koje je središnji dio rada, u kojem je predstavljena izrađena aplikacija i detaljno opisane funkcionalnosti aplikacije. Na kraju, se navode prednosti i ograničenja aplikacije te moguća proširenja i poboljšanja.

2. ANALIZA SRODNIH APLIKACIJA

Na tržištu postoji niz mobilnih aplikacija koje omogućuju praćenje troškova i održavanje vozila. Svaka od njih nudi različite funkcionalnosti, ali često imaju određena ograničenja. U ovom poglavlju analizirane su tri srodne aplikacije, kako bi se usporedile njihove mogućnosti i identificirale prednosti implementiranog rješenja u odnosu na postojeće alternative.

2.1. Fuelio

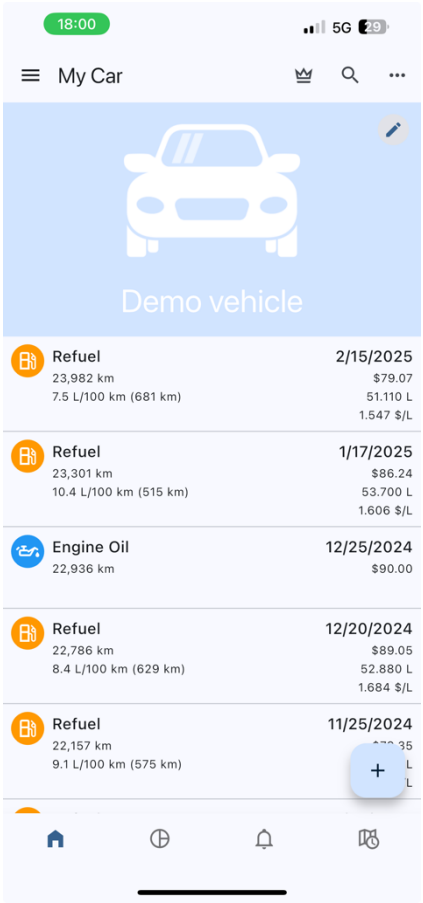
Fuelio je jednostavna i lako upotrebljiva aplikacija za praćenje potrošnje goriva, prijeđene kilometraže i troškova goriva (Fuelio, 2025). Kroz jednostavno sučelje, korisnici mogu unositi podatke o točenju goriva, bilježiti servisne troškove te analizirati potrošnju putem grafova i izvještaja. Slika 1 prikazuje glavni ekran aplikacije Fuelio.



Slika 1. Prikaz glavnog ekrana iz aplikacije Fuelio (Izvor: Autorski rad)

2.2. My Car

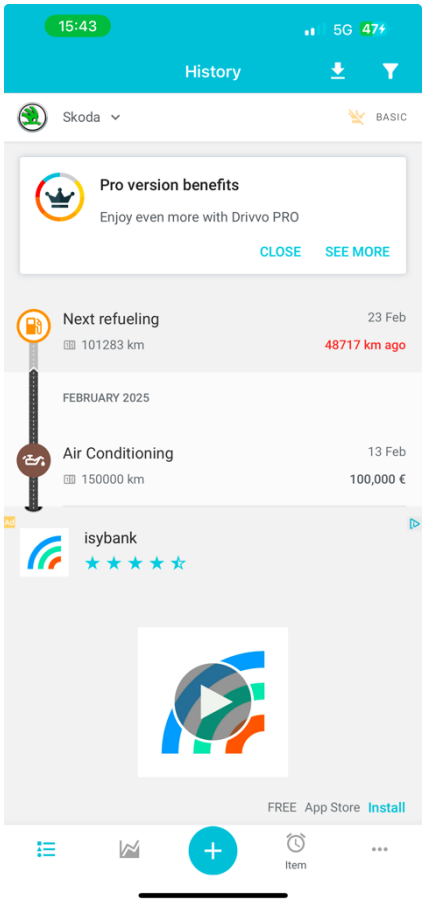
Aplikacija My Car predstavlja novu generaciju modernih aplikacija za održavanje vozila i upravljanje voznim parkom. Omogućuje praćenje potrošnje goriva, servisnih aktivnosti, troškova i prijeđene kilometraže. Korisnici mogu analizirati grafikone, detaljne statistike i izvještaje. My Car pomaže u učinkovitom održavanju vozila i uštedi novca (My Car, 2025). Na slici 2 je prikazan glavni ekran aplikacije My Car.



Slika 2. Prikaz glavnog ekrana iz aplikacije My Car (Izvor: Autorski rad)

2.3. Drivvo

Drivvo je aplikacija za upravljanje vozilima za osobnu upotrebu. Omogućuje korisniku kontrolu nad punjenjem goriva, održavanjem, zamjenom ulja i ostalim servisima koji se obavljaju na vozilu (Drivvo, 2025). Nudi detaljne statistike i izvještaje koji pomažu u analizi potrošnje i optimizaciji troškova. Međutim, aplikacija podržava unos različitih podataka, uključujući gorivo, troškove, prihode, rute, kontrolne popise i podsjetnike, ali nisu svi dostupni u besplatnoj verziji te je korisnik ograničen na samo dva vozila te mu se prikazuju reklame. Slika 3 prikazuje glavni ekran Drivvo aplikacije.



Slika 3. Prikaz glavnog ekrana iz aplikacije Drivvo (Izvor: Autorski rad)

2.4. Analiza nedostataka postojećih rješenja

Analizirane aplikacije pružaju korisnicima mogućnost upravljanja potrošnjom goriva, servisnim aktivnostima i troškovima povezanima s vozilima. Međutim, svaka od njih ima određena ograničenja:

- **Fuelio** se fokusira isključivo na praćenje goriva i osnovnih troškova, dok ne nudi podršku za praćenje svih aspekata održavanja vozila i drugih aktivnosti povezanih s voznim parkom.
- **My Car** omogućuje praćenje različitih podataka vezanih uz vozila, ali korisnici mogu dodavati samo unaprijed definirane kategorije te nema mogućnost dodavanja prihoda, podsjetnika i još ponekih opcija u besplatnoj verziji.
- **Drivvo** pruža širok raspon funkcionalnosti, uključujući unos servisa, troškova i ruta te postavljanje podsjetnika, ali besplatna verzija dolazi s ograničenjima poput maksimalnog broja vozila i prisutnosti reklama i nemogućnosti dodavanja prihoda i kontrolnih popisa, što može smanjiti korisničko iskustvo.

Kako bi se adresirali identificirani nedostaci, razvijena aplikacija nudi potpunu fleksibilnost u upravljanju stanjima automobila. Za razliku od Fuelio-a, omogućuje se praćenje ne samo potrošnje goriva, već svih aktivnosti povezanih s održavanjem, troškovima i prihodima. U odnosu na My Car, korisnicima se omogućuje stvaranje vlastitih kategorija za koje žele pratiti aktivnosti, čime korisnik nije ograničen na unaprijed definirane opcije. U usporedbi s Drivvom, aplikacija ne ograničava broj vozila niti osnovne funkcionalnosti što je važno za potpuno korisničko iskustvo.

3. PROCES IZRADE MOBILNIH APLIKACIJA

Razvoj mobilnih aplikacija odvija se kroz nekoliko faza koje osiguravaju kvalitetu, funkcionalnost i stabilnost gotovog proizvoda. Standardni životni ciklus razvoja softvera (eng. Software Development Life Cycle) primjenjuje se i na mobilne aplikacije (Miller, 2020). U nastavku su detaljno opisane faze razvoja te njihova primjena u kontekstu izrade aplikacije:

3.1. Faza zahtjeva

Faza zahtjeva je prvi korak u životnom ciklusu razvoja softvera, koji uključuje definiranje problema, zahtjeva i resursa potrebnih za razvoj aplikacije. U ovoj fazi se analiziraju poslovne potrebe, identificiraju problemi koji se žele riješiti te se planira vremenski okvir i budžet. Za aplikaciju praćenja stanja automobila, planiranje uključuje razumijevanje korisničkih zahtjeva i definiranje ključnih funkcionalnosti, poput praćenja vozila, kategorija i aktivnosti. Ovisno o metodologiji razvoja, ovaj proces može biti temeljen na klasičnom (vodopadnom) pristupu, gdje su zahtjevi unaprijed definirani i realiziraju se u kasnijim fazama, ili agilnom pristupu, koji omogućuje fleksibilnost i stalne prilagodbe tijekom razvoja (Miller, 2020).

3.2. Dizajn

Dizajn faza uključuje stvaranje arhitekture i izgleda aplikacije. To obuhvaća odabir tehnologija, razvoj strukture podataka, definiranje sučelja i dizajniranje korisničkog iskustva (UX/UI). U ovom koraku, odabran je React Native za razvoj mobilne aplikacije, dok je Firebase odabran za autentifikaciju i pohranu podataka. Također, razmatrani su dizajnerski principi za intuitivno i jednostavno korisničko sučelje. Agilni pristup u ovoj fazi omogućava kontinuirane promjene i prilagodbe na temelju povratnih informacija, dok klasični model zahtijeva detaljno planiranje svih aspekata dizajna unaprijed (Miller, 2020).

3.3. Implementacija

Ovo je središnja faza SDLC, gdje se razvija aplikacija prema zahtjevima i nacrtima stvorenim u prethodnim fazama. U ovoj fazi se razvijaju funkcionalnosti i povezuju različite komponente u sustav, također se faze počnu preklapati pa postoji vjerojatnost da će se morati ponovno razmotriti prethodni zahtjevi koji se utvrde netočnim ili nepraktičnim. Funkcionalnosti koje su bile previše male da bi se unaprijed dizajnirale, morat će se doraditi. Ako se koristi agilni pristup, razvoj se odvija kroz kratke cikluse (sprintove), s mogućnošću ponovnog pregleda i prilagodbe u svakoj iteraciji (Miller, 2020). Tijekom razvoja aplikacije implementirane su CRUD operacije za upravljanje vozilima, kategorijama i aktivnostima, čime je osigurano potpuno upravljanje voznim parkom. Korišten je iterativni razvoj kroz kratke cikluse, što je omogućilo kontinuirane prilagodbe i poboljšanja.

3.4. Testiranje

Testiranje je faza razvoja mobilnih aplikacija koja osigurava stabilnost, sigurnost i funkcionalnost aplikacije. Cilj je identificirati i ispraviti greške prije distribucije aplikacije. Testiranje se može provoditi ručno ili automatiziranim alatima, a posebna pažnja posvećuje se kompatibilnosti s različitim uređajima i operativnim sustavima kako bi se osigurala optimalna izvedba aplikacije. Kod agilnog modela, testiranje se vrši kontinuirano tijekom razvoja, dok kod vodopadnog modela testiranje obično dolazi nakon što su sve funkcionalnosti razvijene (Miller, 2020). U kontekstu razvoja FleetManager aplikacije svaka nova funkcionalnost, poput registracije korisnika ili dodavanja novog vozila, testirana je odmah nakon implementacije.

3.5. Distribucija i održavanje

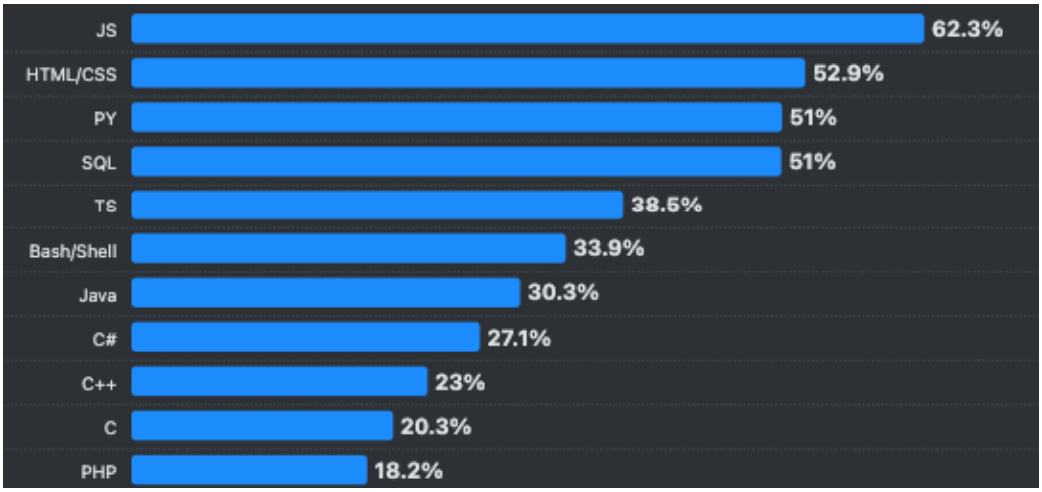
Nakon testiranja, kada aplikacija zadovolji sve kriterije za distribuciju, aplikacija se objavljuje na platformama kao što su Google Play Store i Apple Store. No, razvoj aplikacije ne završava njezinom distribucijom. Potrebno je pratiti korisničke povratne informacije, redovito ažurirati aplikaciju te osigurati sigurnost i performanse. Održavanje aplikacije ključno je za njezinu dugoročnu upotrebu i zadovoljstvo korisnika. Agilni pristup omogućava kontinuirane nadogradnje na temelju povratnih informacija korisnika, dok klasični model zahtijeva planirane nadogradnje nakon određenih vremenskih razdoblja (Miller, 2020). Umjesto objave na Google Play Store-u ili Apple Store-u, aplikacija FleetManager je distribuirana putem Expo platforme. Expo omogućuje jednostavnu distribuciju aplikacije putem Expo Go aplikacije. Korištenjem EAS (Expo Application Services) za izgradnju i distribuciju aplikacije, kreirana je produkcijska verzija aplikacije koja je dostupna korisnicima putem posebnog produkcijskog kanala. Ovaj kanal omogućuje automatske nadogradnje aplikacije svim korisnicima koji koriste aplikaciju putem Expo Go. Za razliku od objavljivanja aplikacija na Google Play Store-u ili Apple Store-u, koji uključuju naknade za objavu aplikacije (25 USD jednokratno za Google Play Store i 99 USD godišnje za Apple Developer Program; Apple Inc., n.d.; Google, n.d.), distribucija putem Expo platforme nije povezana s dodatnim troškovima te nema potrebe za procesom odobravanjem od strane Google-a ili Apple-a.

4. KORIŠTENE TEHNOLOGIJE

Za izradu aplikacije korištene su tehnologije koje omogućuju njezinu funkcionalnost, performanse i jednostavnost u razvoju. JavaScript je korišten kao programski jezik, jer je osnova za rad s React Native-om, omogućujući dinamičko upravljanje podacima i logikom aplikacije. React Native je korišten kao softverski okvir za razvoj aplikacije, omogućujući izradu izvornih aplikacija za iOS i Android platforme s istom bazom koda. Expo je korišten kao alat za pojednostavljenje procesa razvoja i distribucije aplikacije, jer omogućuje brzo testiranje i objavu aplikacije putem svojih alata kao što su Expo Go i EAS (Expo Application Services). Firebase je odabran kao backend platforma, jer pruža usluge kao što su autentifikacija korisnika, baza podataka u stvarnom vremenu i pohrana podataka, što omogućava jednostavno upravljanje korisnicima i podacima aplikacije. Cloudinary je korišten za pohranu i upravljanje medijskim sadržajem, poput slika vozila, osiguravajući brzu i učinkovitu distribuciju tih medija unutar aplikacije.

4.1. JavaScript

JavaScript je programski jezik weba. Jedan od najpopularnijih i najčešće korištenih programskih jezika u svijetu za razvoj web i mobilnih aplikacija. Ogromna većina modernih korisnika web stranica koristi JavaScript i svi moderni web preglednici, igraće konzole, tableti i pametni telefoni uključuju JavaScript tumače, čineći JavaScript najprisutnijim programskim jezikom u povijesti. Podržava objektno-orijentirano i funkcionalno programiranje te podržava modularnost koda, čime poboljšava skalabilnost i održavanje aplikacija (Flanagan, 2020). Na slici 4. prikazan je popis najpopularnijih tehnologija u 2024. godini, gdje vidimo da je JavaScript na samom vrhu.



Slika 4. Prikaz najpopularnijih tehnologija 2024. godine (Izvor: Stackoverflow, 2025)

4.2. React Native

React Native je JavaScript biblioteka nastala 2015. od tvrtke Meta Platforms (izvorno Facebook). Ona omogućuje razvoj aplikacija na više platformi, Android i iOS, koristeći istu bazu koda. Ovaj softverski okvir pruža razvoj aplikacija koristeći JavaScript programski jezik i React komponente koje se prikazuju kao izvorno sučelje platforme (React Native, 2025). Veliki razlog popularnosti React Native je to što može napraviti most između JavaScript koda i izvornog koda da međusobno komuniciraju. Također, omogućuje integraciju s izvornim funkcijama i bibliotekama ako je potreban pristup specifičnim funkcionalnostima uređaja. Cilj React Native je pružiti najbolje iskustvo programeru, što postiže „Hot Reloading-om“, odnosno da programer odmah vidi promjene u aplikaciji bez potrebe za ponovnim pokretanjem aplikacije. React Native preporučuje Expo koji pojednostavljuje razvoj aplikacija te pruža pristup raznim komponentama.

4.3. Expo

Expo je okvir koji pojednostavljuje stvaranje i distribuciju aplikacija na više platformi. Daje alate za testiranje i otklanjanje grešaka te nudi niz alata i usluga koje se mogu koristiti za razvoj i objavljivanje React Native aplikacija (Expo, 2025). Pruža brzi početak rada bez potrebe za postavljanjem složenih razvojnih okruženja kao što su Xcode ili Android Studio, tako što nudi neutralno platformno aplikacijsko programsko sučelje koji oslobađa programere za pisanjem koda specifičnog za platformu kako bi pristupili izvornim funkcija uređaja te se mogu fokusirati na funkcionalnosti svoje aplikacije umjesto da brinu o aspektima specifičnim za platformu.

4.4. Firebase

Firebase je razvojna platforma, koja nudi niz usluga koje olakšavaju razvoj, implementaciju i skaliranje mobilnih i web aplikacija. Pruža integracije s najnovijim Googleovom AI tehnologijom, pohranjivanje i sinkronizaciju podataka aplikacije na globalnoj razini bez upravljanja poslužiteljima, pruža sigurnost i zaštitu korisničkih podataka te izgradnju i objavljivanje statičke ili dinamičke web aplikacije (Firebase, 2025). U ovome radu se koristi Firebase RealTime Database baza podataka smještena u oblaku. Podaci se pohranjuju kao JSON (eng. JavaScript Object Notation) i sinkroniziraju u stvarnom vremenu sa svakim povezanim klijentom.

4.5.Cloudinary

Cloudinary je platforma za upravljanje medijskim datotekama koja omogućava pohranu, obradu i isporuku slika i videa putem API-ja. U ovom projektu, Cloudinary je korišten za pohranu slika vozila. Platforma nudi brojne prednosti, uključujući automatsku optimizaciju slika, transformaciju slika u različite formate i veličine te brz pristup putem globalno distribuiranog CDN-a (eng. Content Delivery Network), što poboljšava brzinu učitavanja slika (Cloudinary, 2025). Cloudinary nudi besplatan plan za osnovne potrebe i omogućava jednostavno integriranje s aplikacijama putem RESTful API-ja. Ova platforma je omogućila optimizaciju slika vozila, čime je poboljšano korisničko iskustvo i performanse aplikacije. Na slici 5 je prikazan kod za

spremanje slika na Cloudinary.

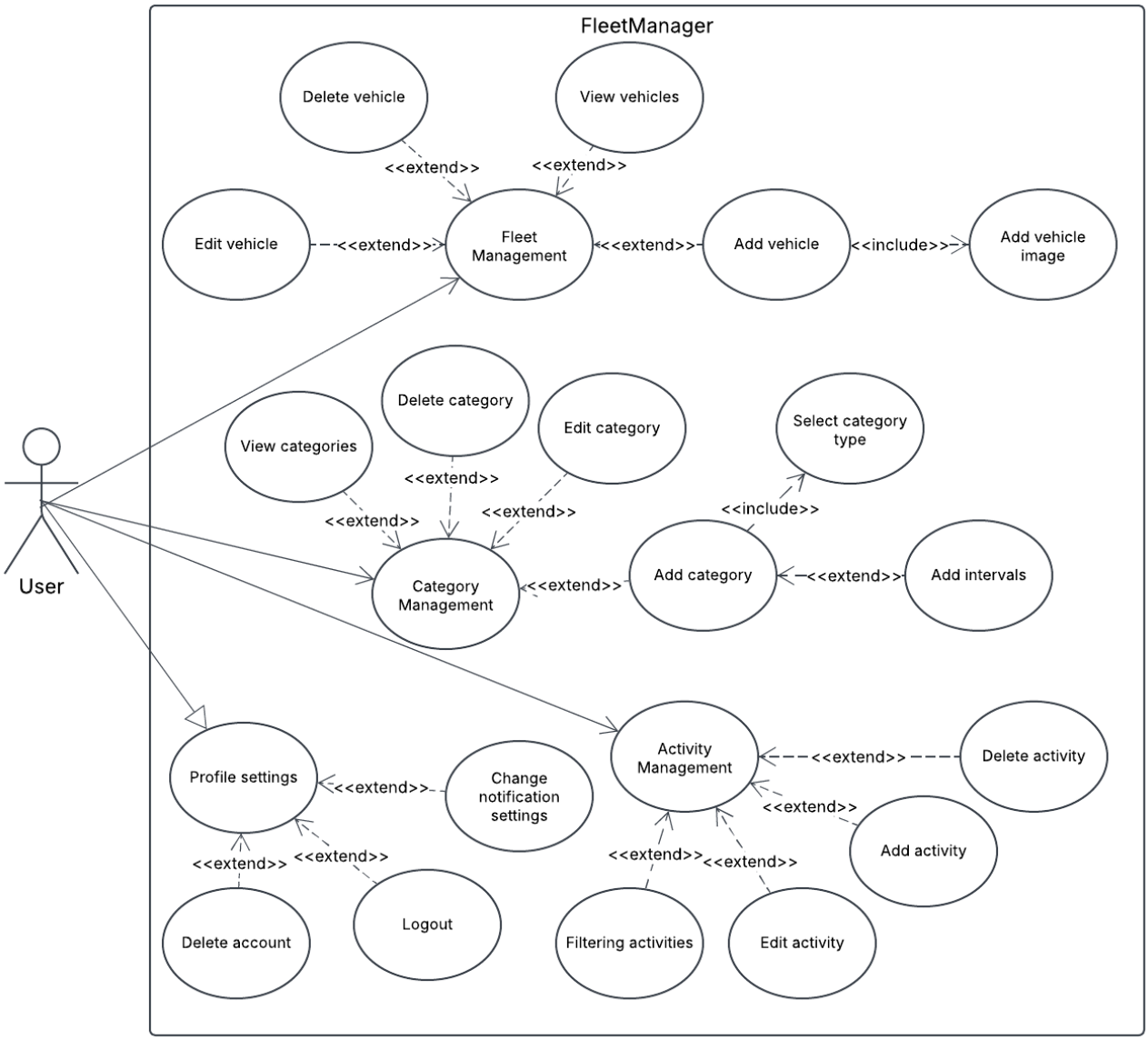
```
import axios from "axios";  62.3k (gzipped: 23.1k)
import { UPLOAD_PRESET, CLOUD_NAME } from "@env";

const uploadImage = async (photo) => {
  if (!photo) return;

  const filename = photo.split("/").pop();
  const fileType = filename.split(".").pop();

  const data = new FormData();
  data.append("file", {
    uri: photo,
    type: `image/${fileType}`,
    name: filename,
  });
  data.append("upload_preset", UPLOAD_PRESET);
  data.append("cloud_name", CLOUD_NAME);

  try {
    const res = await axios.post(
      `https://api.cloudinary.com/v1_1/${CLOUD_NAME}/image/upload`,
      data,
      {
        headers: {
          "Content-Type": "multipart/form-data",
        },
      }
    );
    return res.data.secure_url;
  } catch (error) {
    throw error;
  }
};

export { uploadImage };

```

Slika 5. Programski kod za spremanje slika na Cloudinary (Izvor: Autorski rad)

5. OPIS APLIKACIJE

FleetManager je aplikacija koja omogućuje praćenje stanja automobila izrađena pomoću React Native softverskog okvira. Namijenjena je vlasnicima automobila, ali mogu je koristiti i firme koje imaju istu potrebu te žele na jednostavni način pratiti stanja svojih automobila. Kartična navigacija, omogućuje korisnicima intuitivno korisničko sučelje te jednostavno kretanje kroz aplikaciju. Aplikacija se sastoji od 4 kartice. Prva kartica je Home kartica, ona prikazuje korisniku pregled informacija o floti, o njihovim troškovima i prihodima te omogućuje grafički prikaz o floti. Također, prikazuje i nadolazeće aktivnosti i akcije za vozila. Druga kartica Activity, omogućuje korisnicima dodavanje, filtriranje i pregled aktivnosti za vozila, tu isto korisnik ima mogućnost kreiranja vlastitih kategorija za koje želi pratiti aktivnosti. Treća kartica je Fleet, ona omogućuje korisnicima pregled, brisanje i dodavanje automobila u aplikaciji te zadnja kartica je Profil, gdje korisnik može vidjeti svoje postavke za podsjetnike, mogućnost brisanja računa i odjaviti se iz aplikacije.

Dijagram slučajeva upotrebe koji se nalazi na slici 6 prikazuje ključne funkcionalnosti aplikacije, koji ilustrira kako korisnici (vlasnici flote) komuniciraju s aplikacijom.



Slika 6. Use case dijagram aplikacije FleetManager (Izvor: Autorski rad)

5.1. Upravljanje podacima u aplikaciji

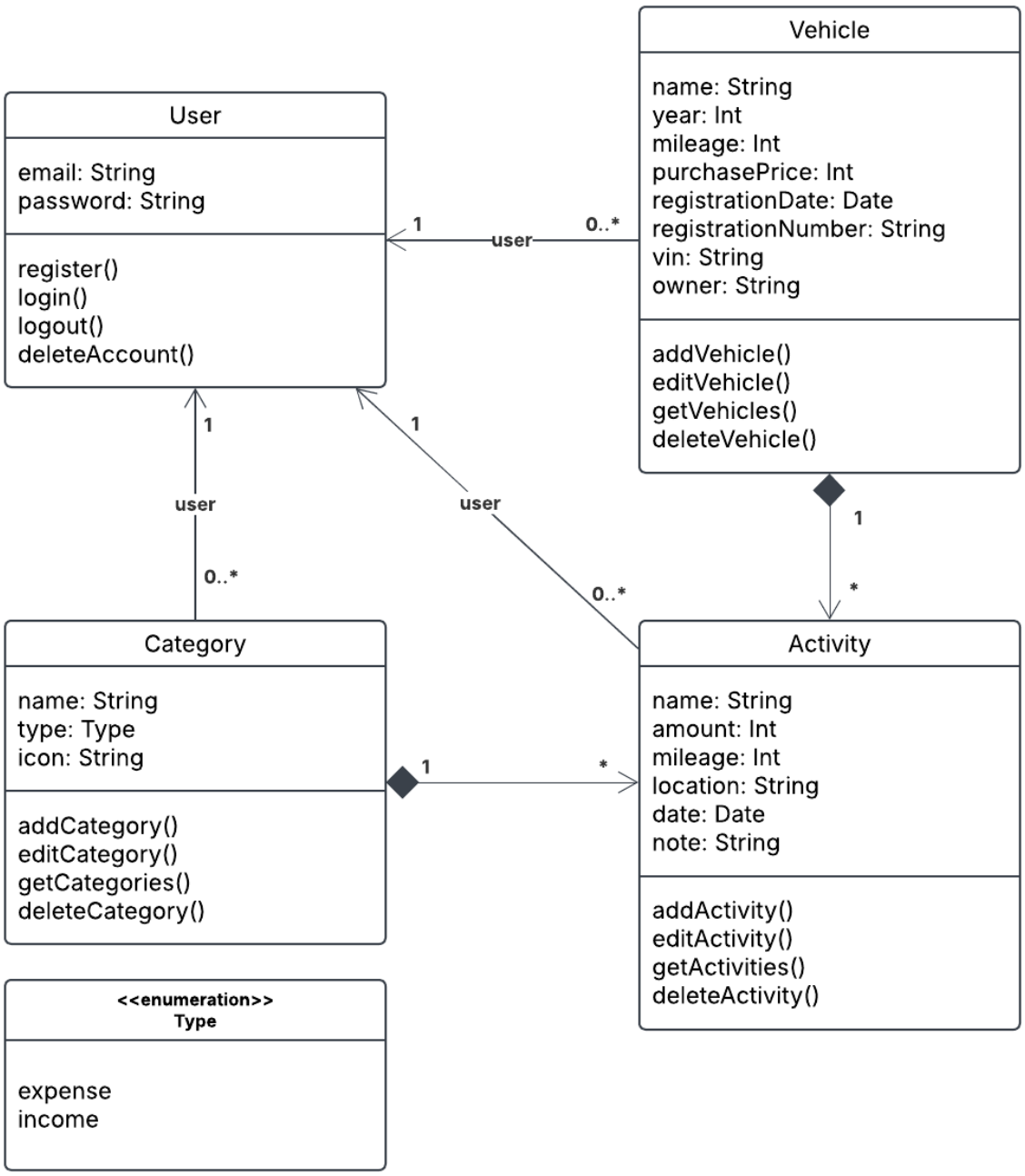
U aplikaciji FleetManager, podaci se upravljaju pomoću konteksta u React Native-u. Korištenje konteksta omogućuje centralizirano upravljanje stanjem aplikacije, što pojednostavljuje komunikaciju između različitih dijelova aplikacije i održavanje konzistentnosti podatka. Kontekst dopušta roditelj komponenti da neke informacije učini dostupnima bilo kojoj komponenti u stablu ispod sebe, bez obzira koliko duboko, bez da ih se eksplicitno prosljeđuje kroz props (React, 2025). Za upravljanje korisničkim podacima, uključujući informacije o vozilima, kategorijama, aktivnostima i postavkama, kreiran je AppContekst. Na slici 7 je prikazana implementacija

AppContext u kodu.

```jsx
import React, { createContext, useState } from "react";

const initialState = {
  loading: false,
  user: null,
  fleet: [],
  categories: [],
  activities: [],
  settings: null,
};

const AppContext = createContext();

const AppStateProvider = ({ children }) => {
  const [state, setState] = useState(initialState);

  return (
    <AppContext.Provider value={[state, setState]}>
      {children}
    </AppContext.Provider>
  );
};

export { AppContext, AppStateProvider, initialState };
```

Slika 7. Implementacija AppContext u aplikaciji FleetManager (Izvor: Autorski rad)

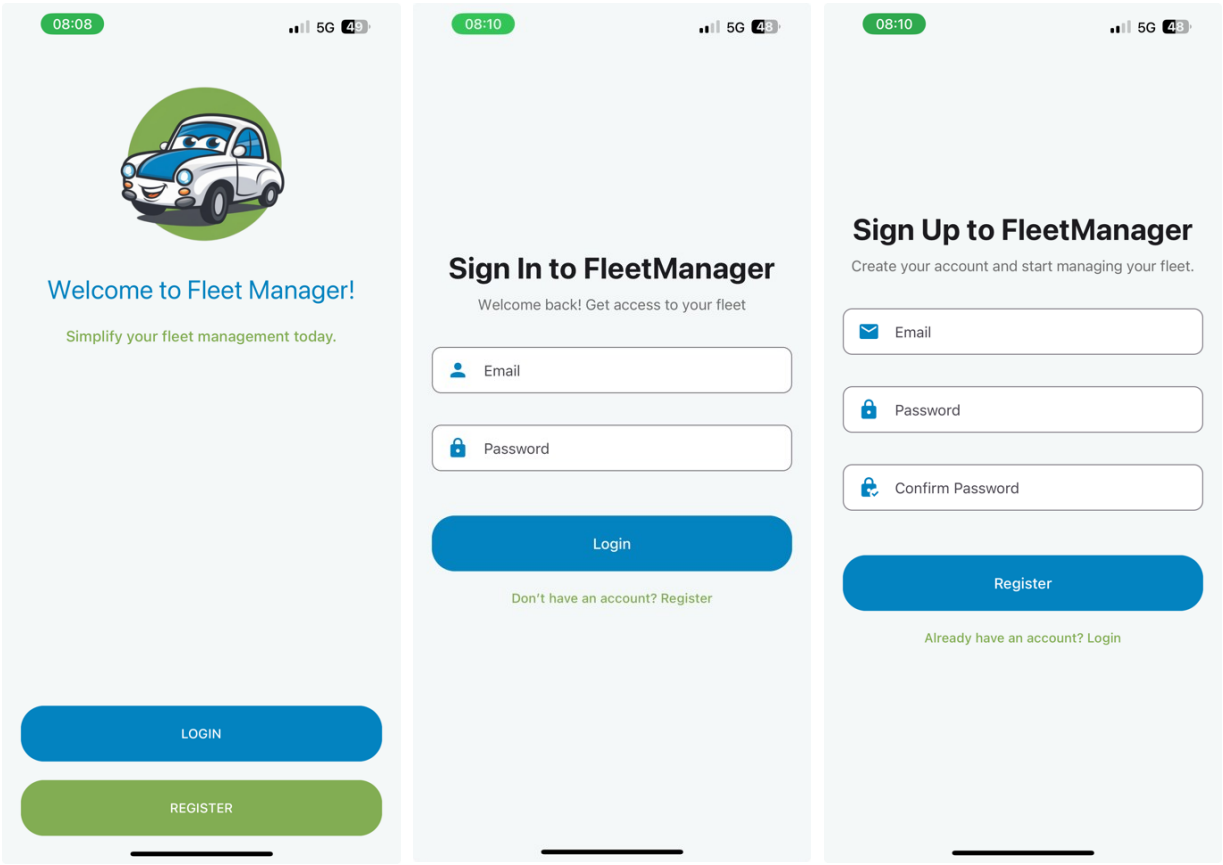
Kako bi se dodatno prikazala organizacija podataka u aplikaciji, na slici 8 prikazan je klasni dijagram. Dijagram ilustrira glavne entitete u aplikaciji, njihove atribute i metode te odnose među njima. Korisnik dodaje vozila, kategorije i aktivnosti. Svaka aktivnost mora biti povezana s određenom kategorijom i vozilom te brisanje bilo kojeg entiteta kategorije ili vozila dovodi do brisanja svih povezanih aktivnosti.



Slika 8. Klasni dijagram aplikacije FleetManager (Izvor: Autorski rad)

5.2. Registracija i prijava korisnika

Aplikacija kreće s početnom stranicom dobrodošlice s logom koja dalje usmjerava na stranice za prijavu ili registraciju korisnika u aplikaciju. Ekrani za dobrodošlicu te prijavu i registraciju korisnika prikazani su na slici 9.



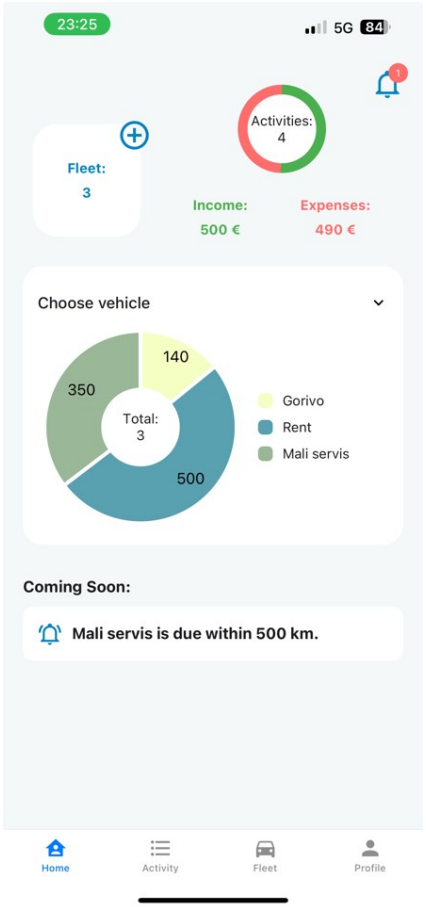
Slika 9. Ekrani za prijavu i registraciju korisnika (Izvor: Autorski rad)

Za autentifikaciju korisnika koristi se Firebase Authentication, koji omogućuje prijavu i registraciju putem e-mail adrese i lozinke. Implementacija autentifikacije obavlja se slanjem HTTP zahtjeva prema Firebase REST API-ju koristeći biblioteku Axios. Slika 10 prikazuje kod za registraciju i prijavu korisnika implementirana pomoću asinkronih funkcija koje šalju zahtjeve Firebase servisu.

```
const registerUser = async (email, password) => {
  const endpoint = `https://identitytoolkit.googleapis.com/v1/accounts:signUp?key=${API_KEY}`;
  return await axios.post(endpoint, {
    email,
    password,
    returnSecureToken: true,
  });
};

const loginUser = async (email, password) => {
  const endpoint = `https://identitytoolkit.googleapis.com/v1/accounts:signInWithPassword?key=${API_KEY}`;
  return await axios.post(endpoint, {
    email,
    password,
    returnSecureToken: true,
  });
};
```

Slika 10. Kod za registraciju i prijavu korisnika (Izvor: Autorski rad)

Nakon prijave korisnika, dolazi se do glavnog ekrana aplikacije koji je prikazan na slici 11.

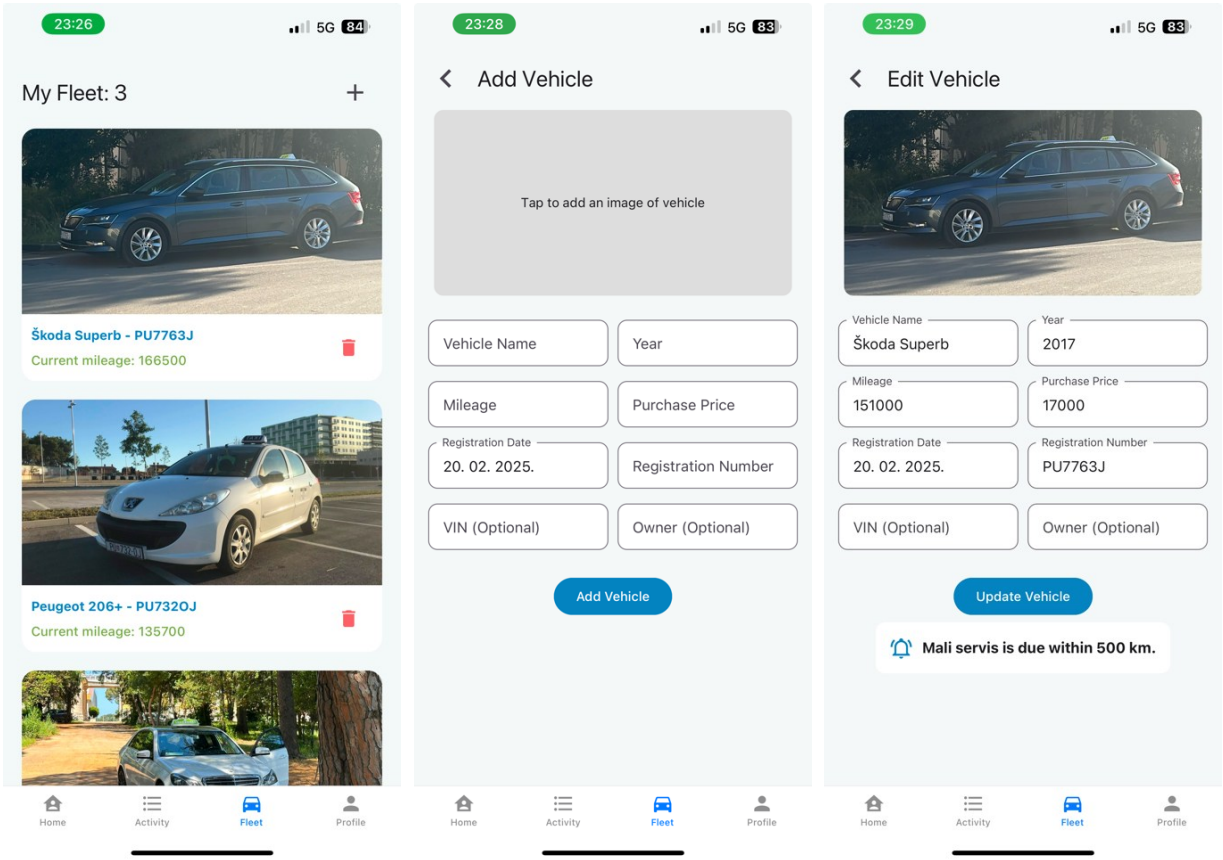


Slika 11. Glavni ekran aplikacije (Izvor: Autorski rad)

Kada je korisnik prijavljen, dolazi do glavnog ekrana koji prikazuje informacije o njegovoj floti, odnos troškova i prihoda te raspodjela njegovih kategorija koja se može filtrirati po vozilu te je grafički prikazana. Na ekranu su vidljive nadolazeće aktivnosti te mogućnost pregleda svih podsjetnika i dodavanje novog vozila.

5.3. Dodavanje, uređivanje i brisanje vozila

Za kreiranje vozila potrebno je unijeti sliku vozila, naziv, godinu, kilometražu, cijenu, datum registracije i registracijsku oznaku te opcionalno broj šasije i vlasnika vozila. Na ekranu za pregled vozila je moguće brisanje i dodavanje neograničenog broja vozila. Klikom na plus se ide na ekran za dodavanje vozila, a klikom na vozilo se ide na ekran za uređivanje vozila. Brisanjem vozila, brišu se sve aktivnosti koje su povezane s tim vozilom, o čemu korisnik bude obaviješten. Ekrani za pregled, uređivanje i dodavanje vozila su prikazana na slici 12.



Slika 12. Ekran za dodavanje, uređivanje i brisanje vozila (Izvor: Autorski rad)

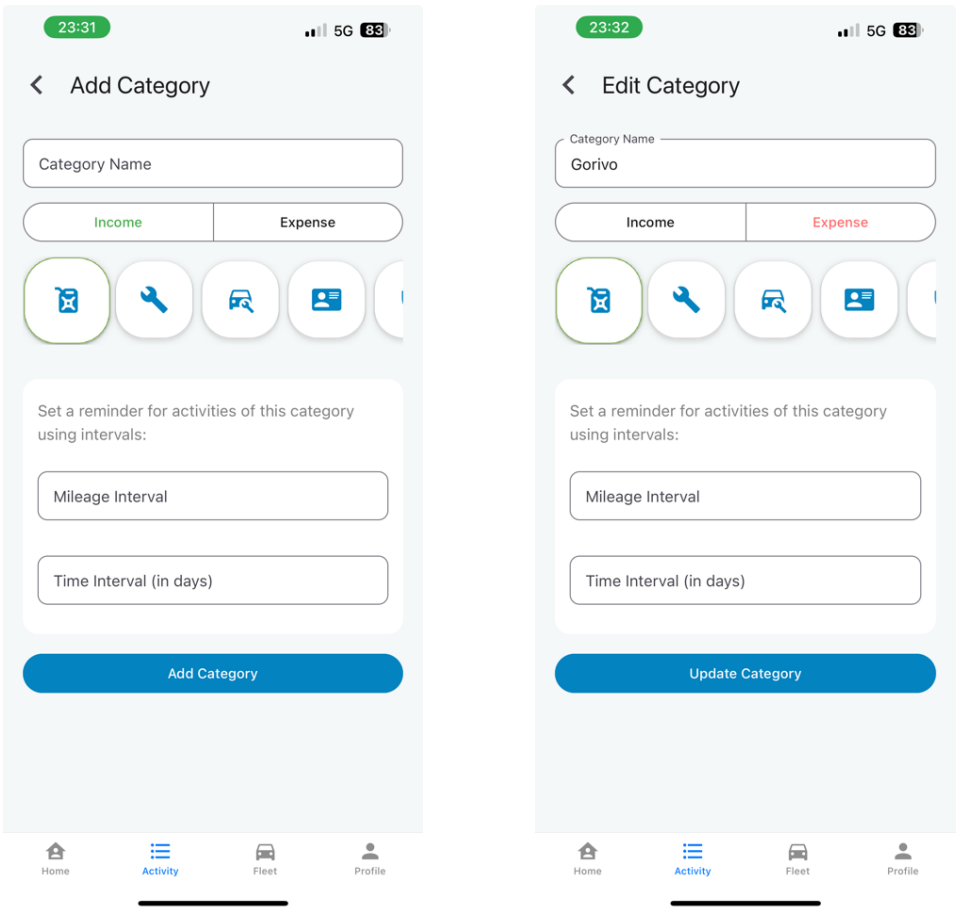
Za rad s podacima o vozilima koriste se HTTP zahtjevi prema Firebase Realtime Database, pri čemu su implementirane funkcije za dodavanje, uređivanje i brisanje vozila. Na slici 13 prikazan je kod koji implementira ove funkcionalnosti.

```
const addVehicle = async (vehicle) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/fleet.json?auth=${token}`;

  try {
    const response = await axios.post(endpoint, {
      ...vehicle,
    });

    return { id: response.data.name, ...vehicle };
  } catch (error) {
    throw error;
  }
};

const deleteVehicle = async (vehicleId) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/fleet/${vehicleId}.json?auth=${token}`;
  try {
    return await axios.delete(endpoint);
  } catch (error) {
    throw error;
  }
};

const editVehicle = async (vehicle) => {
  const { id, ...vehicleData } = vehicle;
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/fleet/${id}.json?auth=${token}`;

  try {
    return await axios.put(endpoint, vehicleData);
  } catch (error) {
    throw error;
  }
};
```

Slika 13. Implementacija funkcija za upravljanje flotom (Izvor: Autorski rad)

5.4. Dodavanje, uređivanje i brisanje kategorija

Da bi se mogle kreirati aktivnosti, potrebne su kategorije. Svaka kategorija sadrži naziv, tip kategorije koji može biti prihod ili rashod, ikonu te opcionalno može odrediti kilometarske ili vremenske intervale. Intervali služe da bi korisnik dobivao obavijesti za aktivnosti te kategorije. Brisanjem kategorije, brišu se sve aktivnosti koje su povezane s tom kategorijom, o čemu korisnik bude obaviješten. Na slici 14 su prikazani ekrani za unos i izmjenu kategorija.



Slika 14. Ekrani za dodavanje i uređivanje kategorija (Izvor: Autorski rad)

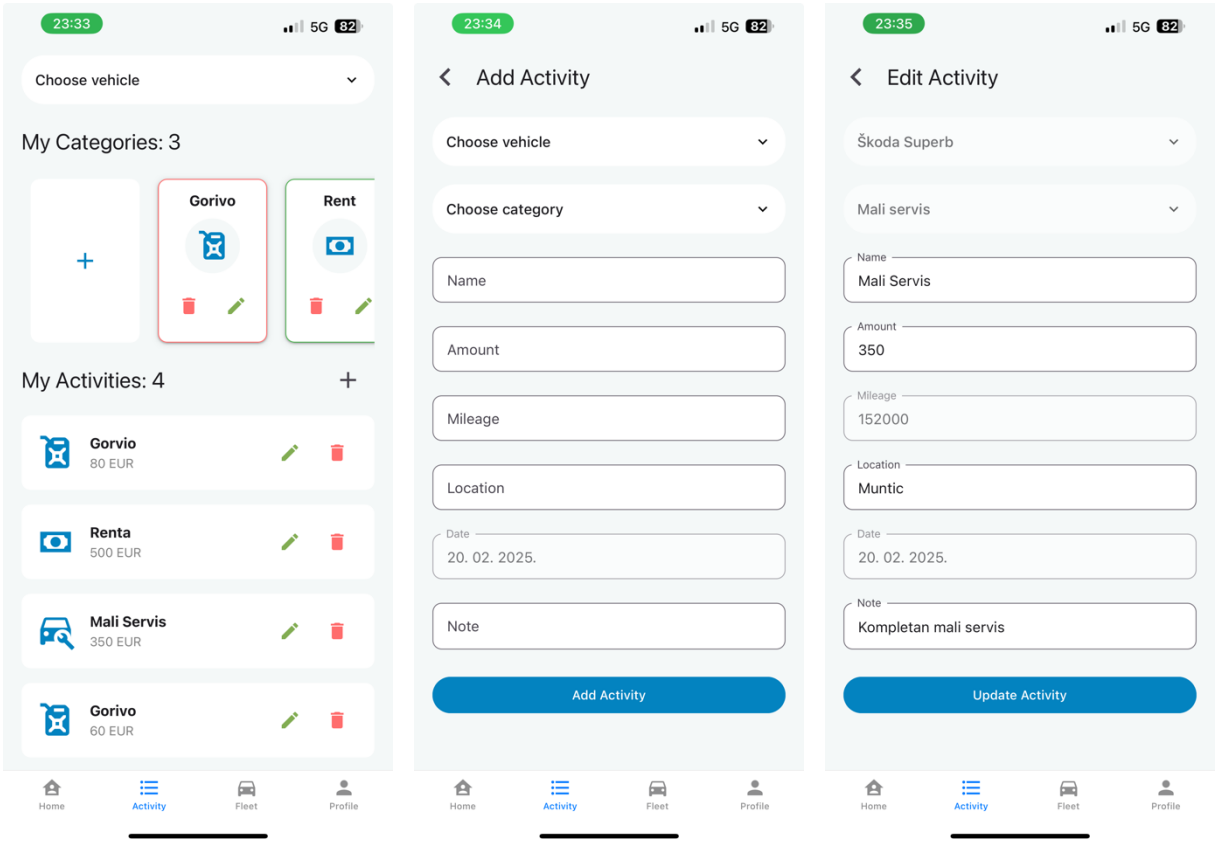
Za upravljanje kategorijama u aplikaciji koriste se funkcije za dodavanje, uređivanje i brisanje podataka u Firebase bazi. Sljedeća slika 15 prikazuje isječak koda implementacije ovih funkcionalnosti.

```
const addCategory = async (category) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/categories.json?auth=${token}`;

  try {
    const categories = await getUserCategories(category.userId);
    const exists = categories.some(
      (cat) => cat.name.toLowerCase() === category.name.toLowerCase()
    );
    if (exists) {
      throw new Error("Category with this name already exists.");
    }

    const res = await axios.post(endpoint, {
      ...category,
    });

    return { id: res.data.name, ...category };
  } catch (error) {
    throw error;
  }
};

const deleteCategory = async (categoryId) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/categories/${categoryId}.json?auth=${token}`;

  try {
    return await axios.delete(endpoint);
  } catch (error) {
    throw error;
  }
};

const editCategory = async (category) => {
  const { id, ...categoryData } = category;
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/categories/${id}.json?auth=${token}`;

  try {
    return await axios.put(endpoint, categoryData);
  } catch (error) {
    throw error;
  }
};
```

Slika 15. Implementacija funkcionalnosti za upravljanje kategorijama (Izvor: Autorski rad)

5.5. Dodavanje, uređivanje i brisanje aktivnosti

Nakon što su kreirane kategorije, moguće je dodavanje aktivnosti. Prilikom dodavanja aktivnosti potrebno je odabrati vozilo i kategoriju te unijeti naziv, iznos i kilometražu te opcionalno lokaciju i opis aktivnosti. Na ekranu Activity moguće je i filtriranje aktivnosti po vozilu i po kategoriji. Na slici 16 su prikazani ekrani pregled, dodavanje i uređivanje aktivnosti.



Slika 16. Ekrani za pregled, dodavanje i uređivanje aktivnosti (Izvor: Autorski rad)

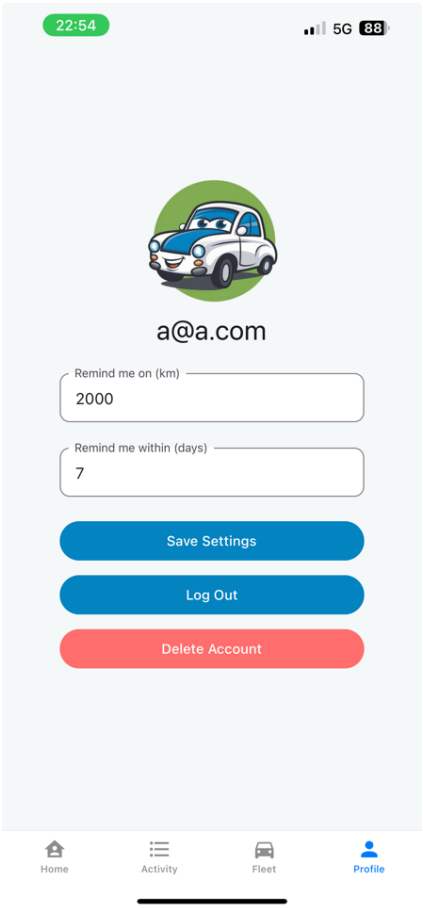
Za rad s aktivnostima u aplikaciji koriste se funkcije za njihovo dodavanje, uređivanje i brisanje u Firebase bazi podataka. Prilikom dodavanja aktivnosti potrebno ju je povezati s određenim vozilom i kategorijom. Slika 17 prikazuje implementaciju ovih operacija.

```
const addActivity = async (activity) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/activities.json?auth=${token}`;

  try {
    const res = await axios.post(endpoint, {
      ...activity,
    });

    return { id: res.data.name, ...activity };
  } catch (error) {
    throw error;
  }
};

const deleteActivity = async (activityId) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/activities/${activityId}.json?auth=${token}`;

  try {
    return await axios.delete(endpoint);
  } catch (error) {
    throw error;
  }
};

const editActivity = async (activity) => {
  const { id, ...activityData } = activity;
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/activities/${id}.json?auth=${token}`;

  try {
    return await axios.put(endpoint, activityData);
  } catch (error) {
    throw error;
  }
};
```

Slika 17. Implementacija funkcionalnosti za upravljanje aktivnostima (Izvor: Autorski rad)

5.6. Ekran profila

Ovdje se korisniku omogućava upravljanje svojim računom te prilagodbu postavki aplikacije. Korisnik se može odjaviti iz aplikacije ili trajno obrisati svoj račun, čime se trajno brišu svi podaci. Također, profil uključuje postavke za podsjetnike, gdje korisnik može odrediti prag za prikaz nadolazećih aktivnosti. Zadana vrijednost praga je postavljena na 2000 km ili 7 dana, što znači da će se podsjetnik za određenu aktivnost prikazivati tek kada preostane manje ili jednako od 2000 km do nadolazeće aktivnosti ili 7 dana do njenog dolaska. Na slici 18 je prikazan ekran profila.



Slika 18. Prikaz ekrana Profila (Izvor: Autorski rad)

Za upravljanje korisničkim postavkama o podsjetnicima koje korisnik može prilagoditi prema vlastitim potrebama, sljedeća slika 19 prikazuje kod za upravljanje postavkama.

```
const addSettings = async (settings) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/settings.json?auth=${token}`;

  try {
    const res = await axios.post(endpoint, settings);
    return { id: res.data.name, ...settings };
  } catch (error) {
    console.error("Error saving settings:", error);
    throw error;
  }
};
const updateSettings = async (settings) => {
  const { id, ...settingsData } = settings;
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/settings/${id}.json?auth=${token}`;

  try {
    await axios.put(endpoint, settingsData);
  } catch (error) {
    console.error("Error updating settings:", error);
    throw error;
  }
};
const deleteSetting = async (settingId) => {
  const token = await storage.getToken();
  const endpoint = `${BASE_URL}/settings/${settingId}.json?auth=${token}`;

  try {
    await axios.delete(endpoint);
  } catch (error) {
    throw new Error("Failed to delete setting: " + error.message);
  }
};
```

Slika 19. Kod za upravljanje postavkama (Izvor: Autorski rad)

5.7. Brisanje korisničkih podataka

Kako bi se osigurala privatnost i omogućilo korisnicima potpuno brisanje svojih podataka, implementirana je funkcija koja briše sve podatke povezane s korisnikom iz baze podataka putem REST API-ja.

Prilikom brisanja računa, potrebno je ukloniti sve podatke povezane s korisnikom. To uključuje uklanjanje svih vozila (Fleet) koja su bila dodana u aplikaciju, sve kategorije (Categories) koje je korisnik kreirao za praćenje aktivnosti, kao i sve aktivnosti (Activities) koje su bile povezane s tim vozilima i kategorijama te je potrebno obrisati postavke korisnika. Time se osigurava da svi podaci povezani s korisnikom budu potpuno uklonjeni.

Funkcija deleteUser poziva pojedinačne funkcije za brisanje podataka iz svake kategorije. Kako bi se osigurala ispravna izvedba, koristi se Promise.all, što omogućuje paralelno izvršavanje svih zahtjeva za brisanje. Slika 20 prikazuje implementaciju za brisanje korisnika i njegovih podataka.

```
const deleteUser = async (userId, settingId) => {
  try {
    const token = await storage.getToken();

    await Promise.all([
      deleteUserFleet(userId),
      deleteUserCategories(userId),
      deleteUserActivities(userId),
      deleteSetting(settingId),
    ]);

    const endpoint = `https://identitytoolkit.googleapis.com/v1/accounts:delete?key=${API_KEY}`;

    return await axios.post(endpoint, {
      idToken: token,
    });
  } catch (error) {
    console.error("Error deleting user data:", error.message);
    throw error;
  }
};
```

Slika 20. Kod za brisanje korisnika i njegovih podataka (Izvor: Autorski rad)

6. ZAKLJUČAK

U ovom završnom radu izrađena je mobilna aplikacija za praćenje stanja automobila pomoću React Native softverskog okvira. Razvoj ove aplikacije omogućio je detaljno razumijevanje procesa izrade mobilnih aplikacija, od planiranja i dizajna korisničkog sučelja do implementacije složenih funkcionalnosti poput praćenja troškova, bilježenja aktivnosti i slanja obavijesti. Korištenje React Native okvira omogućilo je razvoj aplikacije koja podržava više platformi, dok je Firebase pružio sigurno i učinkovito upravljanje korisničkim podacima.

Tijekom razvoja aplikacije fokus je bio na jednostavnosti korištenja i intuitivnom korisničkom sučelju, kako bi korisnici aplikacije mogli lako unositi podatke i pratiti stanje svojih automobila. Implementacija obavijesti dodatno poboljšava korisničko iskustvo, omogućujući pravovremeno reagiranje na nadolazeće troškove i obveze povezane s održavanjem vozila.

Razvoj ove aplikacije pokazao je kako digitalna rješenja mogu doprinijeti boljoj organizaciji i upravljanju voznim parkom, bilo da je riječ o privatnim vlasnicima vozila ili tvrtkama koje upravljaju flotama. Ovaj rad istaknuo je važnost korištenja modernih tehnologija u optimizaciji svakodnevnih zadataka i poboljšanju učinkovitosti.

Glavne prednosti ove aplikacije jesu njezina jednostavnost i intuitivno korisničko sučelje, osim toga, aplikacija nudi fleksibilnost, jer korisnici mogu kreirati vlastite kategorije, prilagođavajući aplikaciju svojim potrebama. Korisnici nisu ograničeni u broju vozila ili kategorija koje mogu dodati, što im omogućuje potpunu slobodu u upravljanju podacima.

Međutim, aplikacija ima i svoja ograničenja. Trenutačno podržava samo jednu razinu korisničkog pristupa, samo za vlasnika vozila. Također, aplikacija ne nudi naprednu analitiku za praćenje stanja automobila.

Mogućnosti daljnjeg razvoja uključuje dodavanje podrške za više korisnika s različitim razinama pristupa, naprednije analitičke funkcionalnosti, kao i mogućnost integracije s vanjskim sustavima poput GPS praćenja ili automatskog prikupljanja podataka s vozila.

Aplikacija razvijena u ovom radu predstavlja korak prema modernizaciji upravljanja voznim parkom i može poslužiti kao temelj za daljnja poboljšanja i nadogradnje u budućnosti.

7. LITERATURA

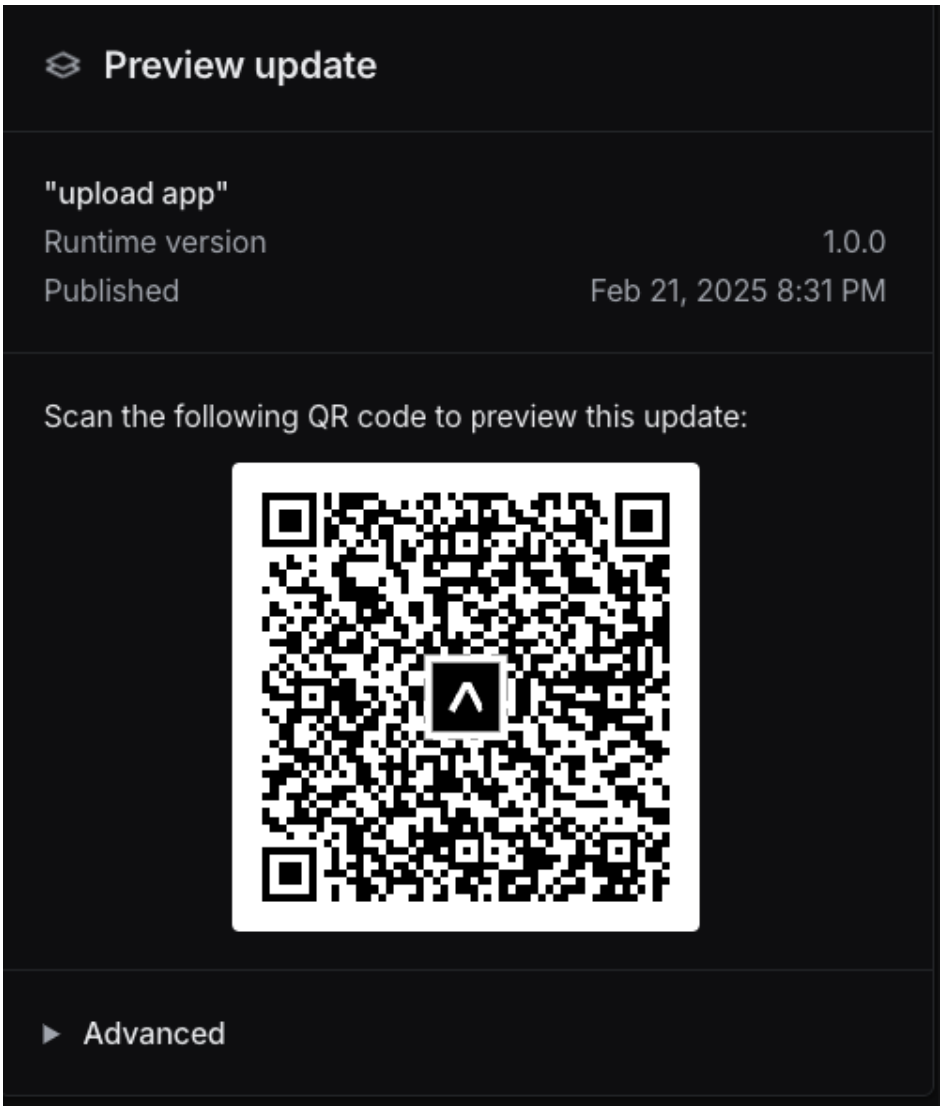
- Apple Inc. (n.d.). Compare memberships. Dostupno na: https://developer.apple.com/support/compare-memberships/. Pristupljeno 5. ožujka 2025.
- Cloudinary. (n.d.). Cloudinary – Cloud-based media management platform. Dostupno na: https://cloudinary.com/. Pristupljeno 16. veljače 2025.
- Expo. (n.d.). *Expo – Open-source platform for making universal native apps.* Dostupno na: https://expo.dev. Pristupljeno 16. veljače 2025.
- Flanagan, D. (2020). JavaScript: The Definitive Guide, Canada. Pristupljeno 16. veljače 2025.
- Google. (n.d.). *Firebase – App development platform.* Dostupno na: https://firebase.google.com. Pristupljeno 16. veljače 2025.
- Google. (n.d.). Pricing for Google Play Console. Dostupno na: https://support.google.com/googleplay/android-developer/answer/6112435?hl=en. Pristupljeno 5. ožujka 2025.
- Machaba, M., & Ndou, J. M. (2024). Investigating Fleet Management Challenges and their Impact on Service Delivery: A Case Study of a Selected Municipality in Limpopo Province, South Africa. Pristupljeno 13. veljače 2025.
- Meta. (n.d.). *React Native: Learn once, write anywhere. Dostupno na:* https://reactnative.dev. Pristupljeno 16. veljače 2025.
- Miller, M. G. (2020). Professional Software Development. Dostupno na: https://mixmastamyk.bitbucket.io/pro_soft_dev/. Pristupljeno 19. veljače 2025.
- React team. (n.d.). Passing dana deeply with context. React. Dostupno na: https://react.dev/learn/passing-data-deeply-with-context. Pristupljeno 17. veljače 2025.
- Stack Overflow. (2024). *Stack Overflow Developer Survey 2024 – Most popular technologies*. Dostupno na: https://survey.stackoverflow.co/2024/technology#most-popular-technologies, Pristupljeno 16. veljače 2025.

8. POPIS SLIKA

9. PRILOZI

- FleetManager
  https://github.com/DanielVuk/FleetManger

- Aplikacija dostupna na: FleetManager



Slika 21. QR kod na aplikaciju FleetManager (Izvor: Autorski rad)