Razvoj mobilne aplikacije za javni prijevoz u gradu Puli

Ostović, Robert

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

Robert Ostović

Razvoj mobilne aplikacije za javni prijevoz u gradu Puli

Diplomski rad

Pula, rujan 2023. godine

Sveučilište Jurja Dobrile u Puli
Fakultet informatike

Robert Ostović

Razvoj mobilne aplikacije za javni prijevoz u gradu Puli

Diplomski rad

JMBAG: 0303068667, redoviti student
Studijski smjer: Informatika
Kolegij: Napredni algoritmi i strukture podataka
Znanstveno područje: Društvene znanosti
Znanstveno polje: Informacijske i komunikacijske znanosti
Znanstvena grana: Informacijski sustavi i informatologija
Mentor: izv. prof. dr. sc. Tihomir Orehovački

Pula, rujan 2023. Godine

IZJAVA O AKADEMSKOJ ČESTITOSTI

Ja, dolje potpisani Robert Ostović, kandidat za magistra informatike ovime izjavljujem da je ovaj Diplomski 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 Diplomskog 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

Robert Ostović

U Puli, 15.09.2023 godine

IZJAVA O KORIŠTENJU AUTORSKOG DJELA

Ja, Robert Ostović dajem odobrenje Sveučilištu Jurja Dobrile u Puli, kao nositelju prava iskorištavanja, da moj diplomski rad pod nazivom

Razvoj mobilne aplikacije za javni prijevoz u gradu Puli

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, 15.09.2023. godine

Potpis

Robert Ostović

SADRŽAJ

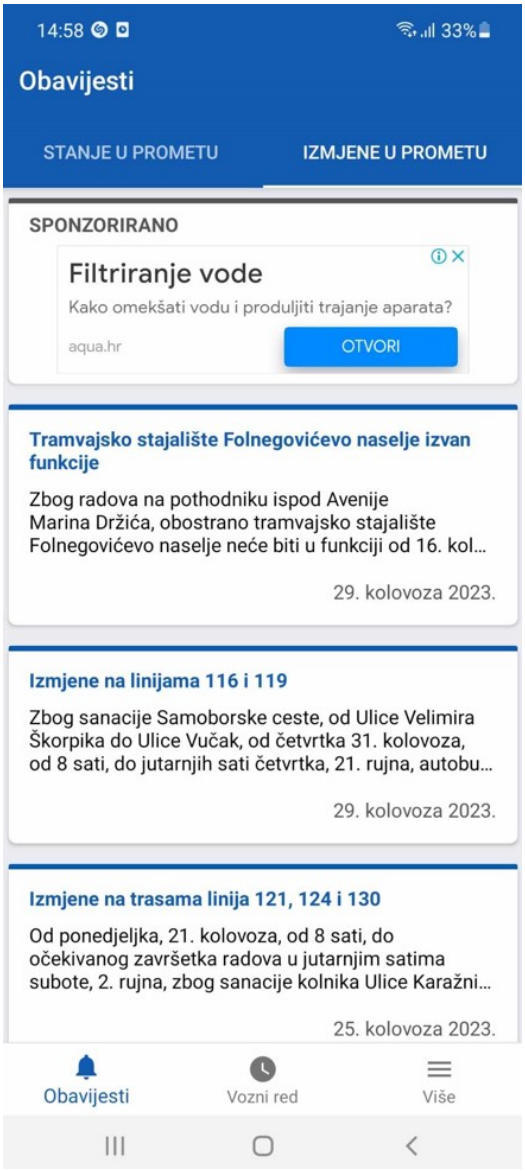
1. Uvod

U vremenu brzog tehnološkog napretka koji je pridonio ubrzanom razvoju pametnih telefona, mobilne aplikacije su postale nezamjenjivi alati koji transformiraju različite aspekte svakodnevnog života. Sektor prijevoza, posebno, svjedoči pomaku u načinu na koji se informacije distribuiraju i pristupaju zahvaljujući integraciji mobilne tehnologije. Ovaj diplomski rad se usmjerava na kreiranje mobilne aplikacije osmišljene kako bi unaprijedila informacijski softver autobusnog sustava u gradu Puli. Efikasnost urbanih prijevoznih sustava igra ključnu ulogu u osiguranju besprijekorne mobilnosti stanovnika i posjetitelja. Pravovremene i točne informacije o autobusnim rutama, rasporedima, praćenju u stvarnom vremenu i drugim važnim ažuriranjima su presudne za glatko iskustvo putovanja. Tradicionalne metode dobivanja takvih informacija često su bile nedovoljne, dovodeći do neugodnosti i neefikasnosti. Grad Pula, poznat po svojoj povijesnoj važnosti i načinu života, ima značajan javni prijevozni mreži sustav koji služi stanovnicima i turistima. Međutim, grad Pula nema moderan informacijski sustav vezan za autobuse i autobusne stanice koji bi olakšao prijenos informacija, te postoji neiskorišteni potencijal za poboljšanje pristupačnosti i upotrebljivosti informacija o autobusnom sustavu grada Pule. Cilj diplomskog rada je razviti novu i modernu mobilnu aplikaciju koja će korisnicima biti pristupačna te će imati pristup stanicama, rasporedu, autobusnim linijama i u budućnosti čak dodatnim funkcionalnostima poput kupnje autobusnih karata koje bi mogle poboljšati ukupno iskustvo korištenja prijevoza i planiranje svojeg putovanja u gradu Puli. Kroz integraciju interaktivne mape unutar mobilne aplikacije, korisnici će imati pristup vizualnoj prezentaciji stvarnog svijeta, omogućujući im da istraže različite autobusne linije, autobusne stanice, rasporede i slično. Osim interaktivne mape, mobilna aplikacija će omogućiti korisnicima da klikom na ikonice dobiju detaljne informacije o stanicama, predviđenim vremenima dolaska autobusa i drugim relevantnim aspektima poput slike autobusne stanice, udaljenost od stanice i slično. Ova personalizirana interakcija s podacima pomoći će korisnicima da donesu informirane odluke o svojim putovanjima te tako smanje neizvjesnost i vrijeme čekanja. Kroz sve navedene inovativne karakteristike, ovaj diplomski rad ima za cilj osigurati da mobilna aplikacija postane ključni alat za poboljšanje iskustva putovanja javnim prijevozom, pridonoseći tako modernizaciji i unaprjeđenju gradskog prijevoza grada Pule. Cijeli programski kôd se može pogledati na poveznici: https://github.com/rostovic/PBS

2. Analiza postojećih rješenja

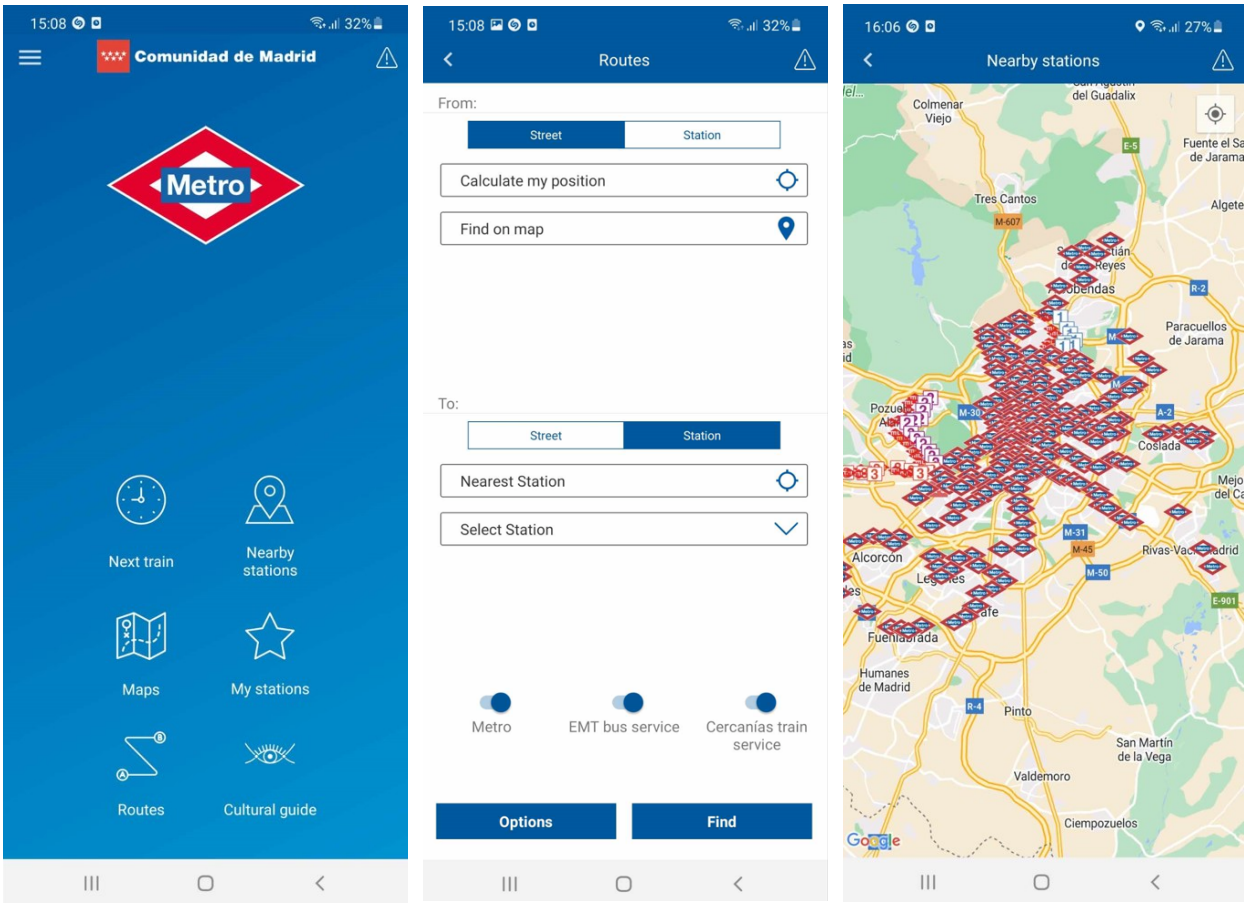
Analiza postojećih rješenja obuhvaća primjere aplikacija koje služe za informativni sustav javnog gradskog prijevoza. Za primjer usporedbe analizirane su aplikacije „ZET info'' i „Metro de Madrid''. Naime ZET info aplikacija služi za obavijesti i najjednostavnije informiranje o prijevozu tramvaja. U nastavku su prikazane slike aplikacije „ZET info''.

Slika 1. Prikaz sučelja aplikacije ZET info - 1



Slika 2. Prikaz sučelja aplikacije ZET info – 2

Iz prikazanih slika 1 i 2 se vidi da su aplikacije lijepo formatirane, međutim isto tako se vidi da ne postoji zanimljiva interakcija između korisnik i aplikacije, već se aplikacija bazira na običnim tekstualnim obavijestima. Također ne postoji nekakva interaktivna mapa, već sve informacije su u tekstualnom obliku, npr. vozni red, linije i slično. U nastavku su prikazane slike aplikacije „Metro de Madrid''. Slike 3, 4 i 5 prikazuju aplikaciju Metro de Madrid koji ima puno bolji i pristupačniji pristup od aplikacije ZET info, modernije izgleda i ima interaktivnu mapu, što jako pomaže u interakciji sa korisnikom.



Slika 3. (lijevo) Prikaz sučelja aplikacije Metro de Madrid – 1

Slika 4. (sredina) Prikaz sučelja aplikacije Metro de Madrid – 2

Slika 5. (desno) Prikaz sučelja aplikacije Metro de Madrid – 3

3. Korištene tehnologije

Korištene tehnologije u ovom diplomskom radu su:

- JavaScript
- React (React Native)
- NodeJS

JavaScript i React Native su korišteni za izradu sučelja i manipuliranje podacima, dok je NodeJS korišten za transformaciju podataka u format koji je potreban da aplikacije funkcionira.

3.1 JavaScript

JavaScript je visokostrukturirani, dinamički programski jezik koji je ključan za razvoj interaktivnih i dinamičnih web aplikacija. Ovaj jezik, često skraćeno nazivan JS, omogućava programerima da manipuliraju elementima na web stranici i ostvaruju različite funkcionalnosti s interakcijom korisnika. Razvijen je od strane Netscape Communications Corporationa s ciljem da omogući dinamično mijenjanje sadržaja web stranica, čime je označio početak nove ere u razvoju web aplikacija. Jedna od glavnih karakteristika JavaScripta je njegova sposobnost izvođenja u web pregledniku korisnika, čime se postiže brza interakcija s korisničkim sučeljem. Osim toga, JavaScript se također koristi izvan preglednika, u okruženjima poput servera (Node.js), što omogućava razvoj potpune stog aplikacija. Kroz vrijeme, JavaScript je doživio značajna poboljšanja i nadogradnje. Moderni JavaScript (često nazivan ECMAScript) standardi pružaju bogat skup značajki i sintakse za razvoj složenih aplikacija. Osim osnovnih funkcija i tipova podataka, uključuje koncepte kao što su funkcionalno programiranje, asinkrono izvođenje (promises i async/await), i modularnost (moduli u ES6+ standardima). JavaScript također ima bogatu zajednicu koja podržava razvoj aplikacija kroz brojne biblioteke i okvire. Biblioteke poput jQuery pružaju olakšane načine manipulacije DOM-om, dok okviri poput Reacta, Angulara i Vue-a omogućuju izgradnju većih i kompleksnijih web aplikacija [1][2].

3.2 React

React, također poznat kao React.js ili ReactJS, je otvorena JavaScript biblioteka za izradu korisničkih sučelja. Razvijen od strane Facebooka, React je osmišljen kako bi pojednostavio proces stvaranja interaktivnih i dinamičnih web aplikacija fokusiranjem na ponovnu upotrebu komponenata i efikasno ažuriranje korisničkog sučelja. Jedan od ključnih koncepta u Reactu je korištenje virtualnog DOM-a (Document Object Model). Umjesto izravnog manipuliranja stvarnim DOM elementima na stranici, React stvara virtualnu reprezentaciju DOM-a u memoriji. Kada dođe do promjena u podacima ili stanju, React učinkovito izračunava minimalne promjene potrebne virtualnom DOM-u, a zatim te promjene primjenjuje na stvarni DOM, što rezultira poboljšanom izvedbom i reaktivnošću. React potiče arhitekturu temeljenu na komponentama, gdje se korisničko sučelje razbija na manje, ponovno upotrebljive komponente. Ove komponente se mogu stvarati, kombinirati i koristiti diljem aplikacije. Ova modularnost ne samo da čini proces razvoja organiziranijim, već i poboljšava održivost i ponovnu upotrebu koda. React koristi deklarativan pristup izgradnji korisničkih sučelja. Razvojni programeri definiraju kako korisničko sučelje treba izgledati na temelju trenutnog stanja aplikacije, a React se brine za ažuriranje sučelja kako bi odgovaralo željenom stanju. Ovo se razlikuje od imperativnog programiranja, gdje razvojni programeri eksplicitno specificiraju kako promijeniti korisničko sučelje. React također podržava koncept unidirekcijskog tijeka podataka. Podaci teku u jednom smjeru, od nadređenih komponenata prema potomcima. Ovo olakšava razumijevanje načina na koji se podaci prenose i mijenjaju unutar aplikacije. Osim svojih osnovnih značajki, React ima velik ekosustav alata, biblioteka i proširenja. React Router je popularna biblioteka za upravljanje rutiranjem u React aplikacijama, dok se Redux i MobX često koriste za upravljanje stanjima. React također može biti integriran s drugim tehnologijama poput GraphQL-a za učinkovito dohvaćanje i manipulaciju podacima [3].

3.3 React Native

React Native zato ima takoreći ista obilježja kao i React, odnosno - izuzetno je popularna tehnologija koja pruža programerima sposobnost da izrađuju mobilne aplikacije visokih performansi koristeći svoje znanje u jeziku JavaScript i konceptima Reacta. Ona donosi

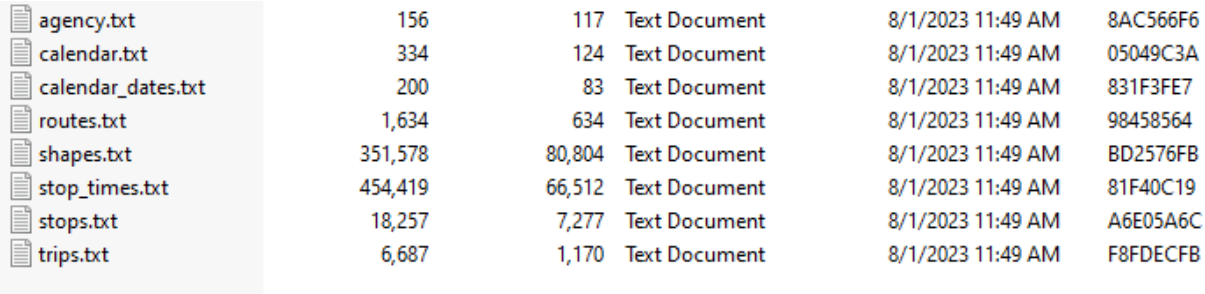
revolucionarne promjene u načinu na koji se razvijaju mobilne aplikacije omogućavajući istovremeno razvoj za više platformi kao što su iOS i Android. Ovaj pristup smanjuje znatnu količinu potrebne ponovne izrade koda, što ubrzava cijeli razvojni proces. Temelji se na jednostavnom konceptu komponenata, modularnih građevnih blokova aplikacija, čime se olakšava stvaranje bogatih i dinamičkih korisničkih sučelja. Ono što čini React Native posebno moćnim je njegova sposobnost reaktivnog ažuriranja. Promjene u podacima automatski se odražavaju na korisničkom sučelju, pružajući glatko iskustvo bez potrebe za ručnim osvježavanjem. Osim toga, React Native pruža "hot reloading" značajku, što znači da se promjene u kodu mogu trenutno vidjeti u stvarnom vremenu, čime se smanjuje vrijeme testiranja i ubrzava iterativni proces razvoja. Uz to, velika i aktivna zajednica razvija brojne dodatke i komponente koje olakšavaju razvoj različitih aspekata aplikacija, od složenih animacija do interaktivne navigacije. Sve više kompanija prepoznaje prednosti React Native-a te ga koristi za stvaranje brzih, responzivnih i intuitivnih mobilnih iskustava koja zadovoljavaju zahtjeve današnjih korisnika [4].

3.4 NodeJS

Node.js je open-source okruženje za izvođenje JavaScript koda izvan web preglednika. Razvijen na osnovi JavaScripta, Node.js omogućava programerima da kreiraju serverske aplikacije i alate koristeći isti jezik koji se tradicionalno koristi za izradu klijentske strane web aplikacija. Jedna od ključnih karakteristika Node.js-a je asinkronost. To znači da Node.js može izvoditi više operacija istovremeno, bez blokiranja izvođenja drugih dijelova koda. Ova asinkrona priroda čini Node.js iznimno pogodnim za rukovanje velikim brojem korisničkih zahtjeva u stvarnom vremenu. Node.js se temelji na V8 JavaScript engineu tvrtke Google, koji izvodi JavaScript kod brzo i učinkovito. Zbog toga, Node.js omogućava izvođenje brzih i responzivnih serverskih aplikacija. Jedna od velikih prednosti Node.js-a je njegova aktivna i proširena zajednica. Razvijatelji su stvorili mnoge dodatke i biblioteke koje olakšavaju razvoj web aplikacija. Express.js je popularni okvir koji pojednostavljuje izradu web aplikacija, a biblioteke poput Mongoose olakšavaju povezivanje s bazama podataka. Node.js se često koristi za razvoj serverskih aplikacija, RESTful API-ja (Application Programming Interface), real-time aplikacija, tehnologija temeljenih na događajima kao što su chat aplikacije i streaming usluge [5].

4. Transformacija podataka

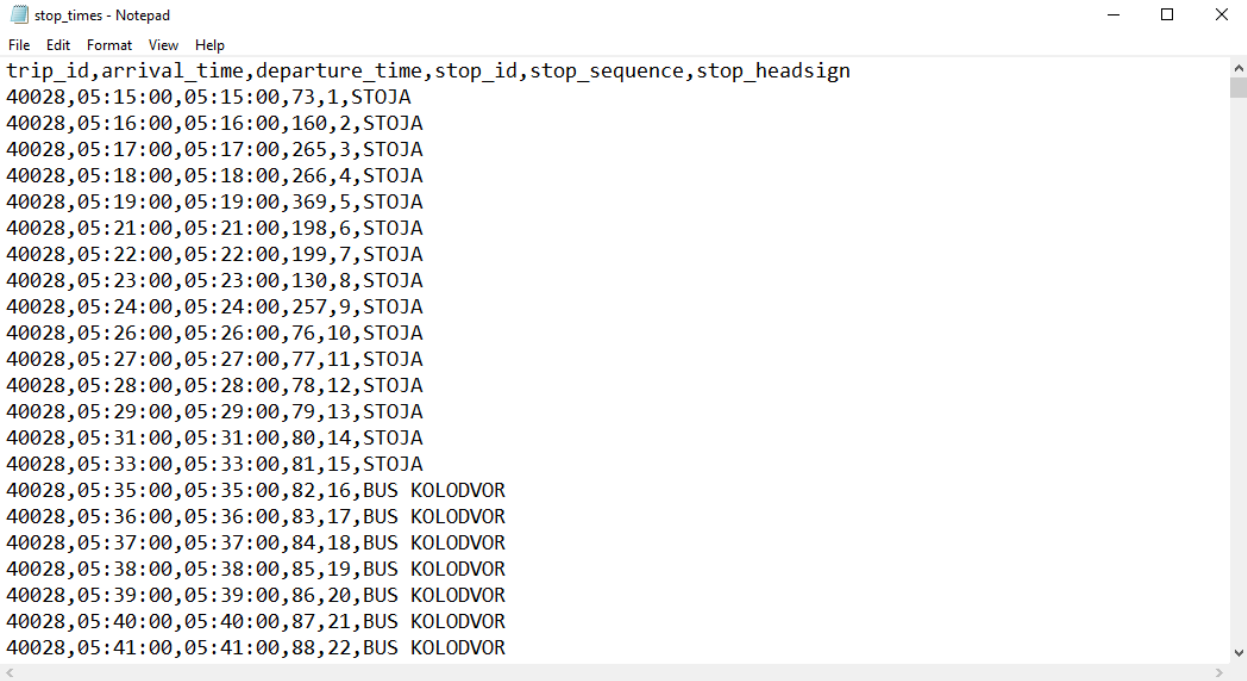
Uz pomoć gradonačelnika grada Pule gospodina Filipa Zoričića i djelatnika Pule Prometa dobiva se podaci vezani za stanice. Potrebna je transformacija podataka kako bi format podataka imao smisla i da se podaci mogu korektno prezentirati na aplikaciji. Zip datoteka sadrži:



agency.txt	156	117	Text Document	8/1/2023 11:49 AM	8AC566F6
calendar.txt	334	124	Text Document	8/1/2023 11:49 AM	05049C3A
calendar_dates.txt	200	83	Text Document	8/1/2023 11:49 AM	831F3FE7
routes.txt	1,634	634	Text Document	8/1/2023 11:49 AM	98458564
shapes.txt	351,578	80,804	Text Document	8/1/2023 11:49 AM	BD2576FB
stop_times.txt	454,419	66,512	Text Document	8/1/2023 11:49 AM	81F40C19
stops.txt	18,257	7,277	Text Document	8/1/2023 11:49 AM	A6E05A6C
trips.txt	6,687	1,170	Text Document	8/1/2023 11:49 AM	F8FDECFB

Slika 6. Prikaz datoteka unutar zip datoteke

U izradi projekta korištene su datoteke *stops.txt* i *stop_times.txt* (slika 7 i 8). Stop.txt označavaju stanice sa koordinatama, dok stop_times.txt označava raspored autobusa na svakoj stanici. Prikaz tih podataka možete vidjeti na sljedećim slikama.

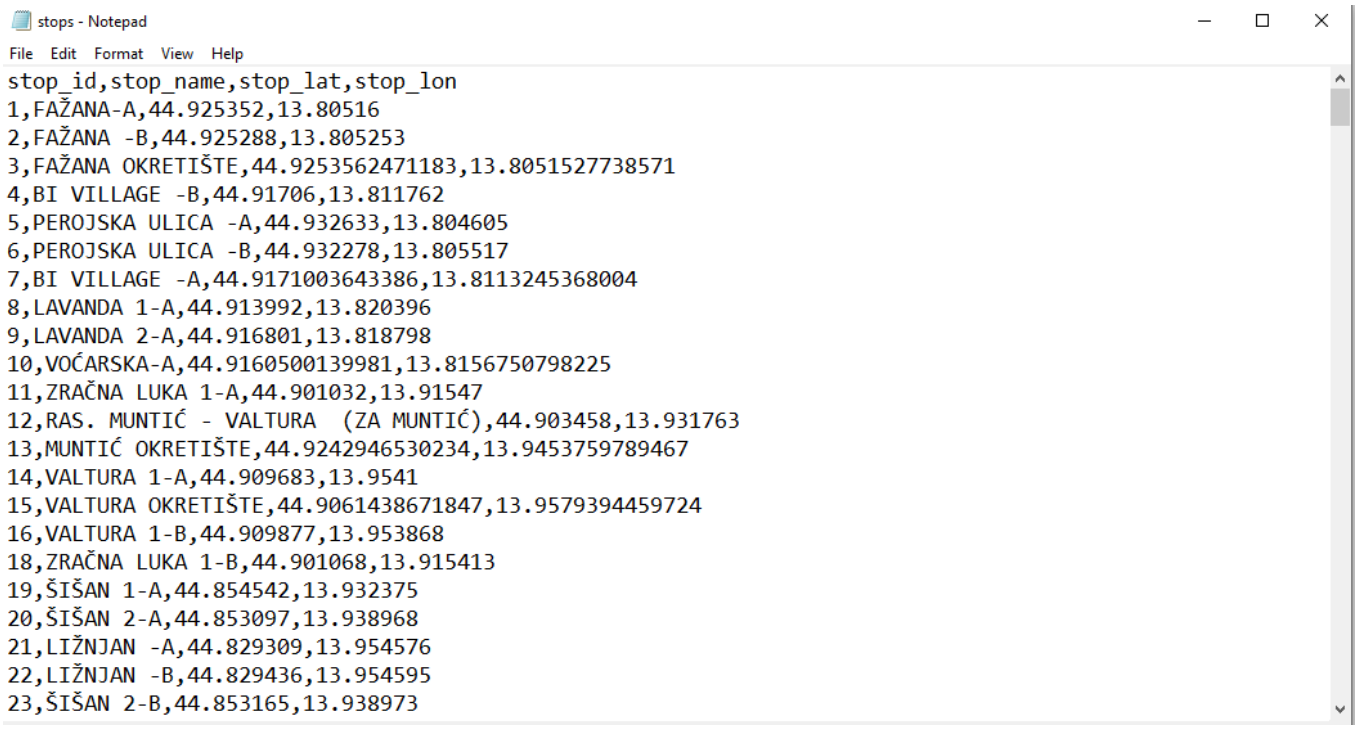


stop_times - Notepad

File Edit Format View Help

```
trip_id,arrival_time,departure_time,stop_id,stop_sequence,stop_headsign
40028,05:15:00,05:15:00,73,1,STOJA
40028,05:16:00,05:16:00,160,2,STOJA
40028,05:17:00,05:17:00,265,3,STOJA
40028,05:18:00,05:18:00,266,4,STOJA
40028,05:19:00,05:19:00,369,5,STOJA
40028,05:21:00,05:21:00,198,6,STOJA
40028,05:22:00,05:22:00,199,7,STOJA
40028,05:23:00,05:23:00,130,8,STOJA
40028,05:24:00,05:24:00,257,9,STOJA
40028,05:26:00,05:26:00,76,10,STOJA
40028,05:27:00,05:27:00,77,11,STOJA
40028,05:28:00,05:28:00,78,12,STOJA
40028,05:29:00,05:29:00,79,13,STOJA
40028,05:31:00,05:31:00,80,14,STOJA
40028,05:33:00,05:33:00,81,15,STOJA
40028,05:35:00,05:35:00,82,16,BUS KOLODVOR
40028,05:36:00,05:36:00,83,17,BUS KOLODVOR
40028,05:37:00,05:37:00,84,18,BUS KOLODVOR
40028,05:38:00,05:38:00,85,19,BUS KOLODVOR
40028,05:39:00,05:39:00,86,20,BUS KOLODVOR
40028,05:40:00,05:40:00,87,21,BUS KOLODVOR
40028,05:41:00,05:41:00,88,22,BUS KOLODVOR
```

Slika 7. Prikaz stop_times.txt datoteke



Slika 8. Prikaz sadržaja datoteke stops.txt

Transformaciju podataka izvršena je na sljedeći način: pomoću NodeJS-a kreiramo funkcije koje će pročitati stop_times.txt i stops.txt datoteke te transformati podatke u JSON format. Podaci koji su u stops.txt datoteci nisu 100% točni, odnosno neke autobusne stanice nisu u uporabi tako da treba filtrirati podatke pa se transformacija podataka vrši programskim kôdom koji je prikazan na slikama u nastavku:

Na slici 9 se nalazi programski kôd u NodeJS-u. Ovaj kratki program čita datoteku stop_times.txt i izvlači samo podatke koji su nama potrebni – odnosno vrijeme dolaska i ID autobusne stanice. Dobiveni rezultat je prikazan u nastavku na slici 10:

```
const fs = require("fs");

const filePath = "stop_times.txt";

fs.readFile(filePath, "utf8", (err, data) => {
  if (err) {
    console.error("Error reading the file:", err);
    return;
  }

  const lines = data.split("\n");
  const extractedData = [];

  lines.forEach((line) => {
    const params = line.split(",");
    if (params.length >= 4) {
      const time = params[1];
      const id = params[3];
      extractedData.push({ time, id });
    }
  });

  const outputFilePath = "extracted_data.txt";
  const outputData = extractedData
    .map((entry) => `${entry.time},${entry.id}`)
    .join("\n");

  fs.writeFile(outputFilePath, outputData, (err) => {
    if (err) {
      console.error("Error writing to the file:", err);
      return;
    }
    console.log("Data saved to extracted_data.txt");
  });
});
```

Slika 9. Prikaz kôda za transformaciju stop_times.txt – korak 1

Slika 10 prikazuje vrijeme i ID stanice. Rezultat, odnosno datoteka sadrži jako puno redova jer postoji više od 300 stanica. Sljedeći korak je grupiranje vremena dolaska po ID-u.

Slika 10. Rezultat prvog koraka transformacije

Slika 11 prikazuje programski kôd u NodeJS-u za grupiranje prijašnjeg rezultata, odnosno rezultata prikazanog na slici 10.

```
const fs = require("fs");
const filePath = "extracted_data.txt";

fs.readFile(filePath, "utf8", (err, data) => {
  if (err) {
    console.error("Error reading the file:", err);
    return;
  }

  const lines = data.split("\n");
  const groupedData = {};

  lines.forEach((line) => {
    const [time, id] = line.split(",");
    if (!groupedData[id]) {
      groupedData[id] = [];
    }
    groupedData[id].push(time);
  });

  const sortedIds = Object.keys(groupedData).sort((a, b) => b - a);
  const sortedGroupedData = {};

  sortedIds.forEach((id) => {
    sortedGroupedData[id] = groupedData[id];
  });

  const outputFilePath = "grouped_and_sorted_data.json";

  const outputData = JSON.stringify(sortedGroupedData, null, 2);

  fs.writeFile(outputFilePath, outputData, (err) => {
    if (err) {
      console.error("Error writing to the file:", err);
      return;
    }
    console.log("Data saved to grouped_and_sorted_data.json");
  });
});
```

Slika 11. Prikaz kôda - grupiranje prijašnjeg rezultata

Na slici 12 vidi se rezultat grupiranja. Za svaku stanicu je predstavljeno raspored vremena u kojem dolazi autobus te je sad potrebno sortirati podatke za svaku stanicu, odnosno svaki ID.

Slika 12. Rezultat grupiranja

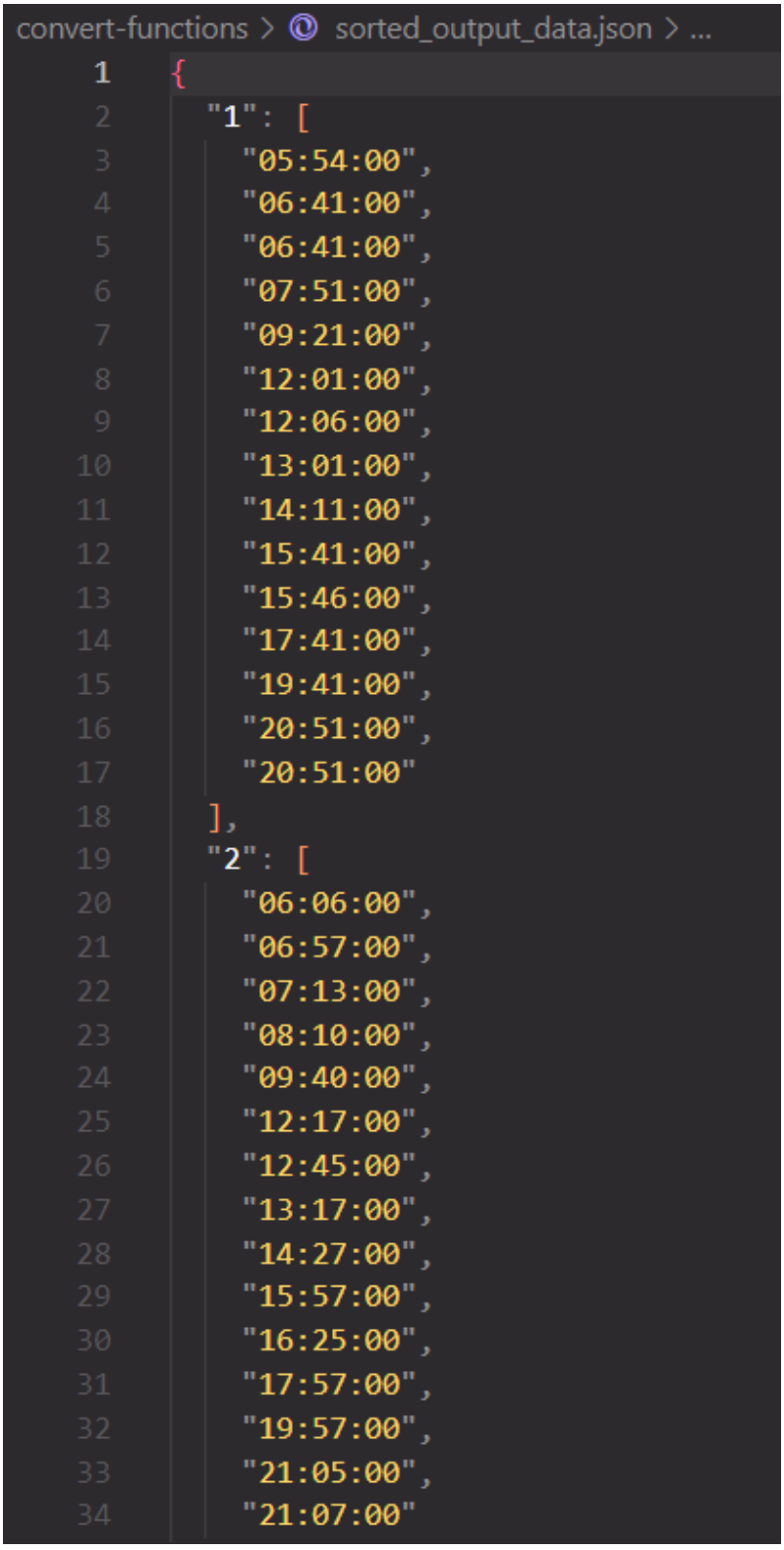
Slika 13 prikazuje programski kôd koji sortira podatke koji su prikazani na slici 12.

convert-functions > sort_time.js > ...

```
const fs = require("fs");

const inputFilePath = "grouped_and_sorted_data.json";

fs.readFile(inputFilePath, "utf8", (err, data) => {
  if (err) {
    console.error("Error reading the file:", err);
    return;
  }

  const jsonData = JSON.parse(data);
  const sortedData = {};

  for (const id in jsonData) {
    if (jsonData.hasOwnProperty(id)) {
      const times = jsonData[id];
      sortedData[id] = times.sort();
    }
  }

  const outputFilePath = "sorted_output_data.json";

  const outputData = JSON.stringify(sortedData, null, 2);

  fs.writeFile(outputFilePath, outputData, (err) => {
    if (err) {
      console.error("Error writing to the file:", err);
      return;
    }
    console.log("Data saved to sorted_output_data.json");
  });
});

```

Slika 13. Prikaz kôda – sortiranje prijašnjeg rezultata

Rezultat sortiranja prikazan je na slici 14 u nastavku:



Slika 14. Prikaz podataka nakon sortiranja vremena

Zadnji korak jest za svaki ID (stanicu) dodati polje „times“ kako bi format lijepo izgledao. Slika 15. prikazuje programski kôd koji upravo to radi

convert-functions > {} transform_data.js > ...

```
const fs = require("fs");

const jsonData = require("./sorted_output_data.json");

const transformedData = [];

for (const id in jsonData) {
  if (jsonData.hasOwnProperty(id)) {
    const times = jsonData[id];
    transformedData.push({ id: parseInt(id), times });
  }
}

const outputFilePath = "transformed_data.txt";
const outputData = JSON.stringify(transformedData, null, 2);

fs.writeFile(outputFilePath, outputData, (err) => {
  if (err) {
    console.error("Error writing to the file:", err);
    return;
  }
  console.log("Transformed data saved to transformed_data.txt");
});
```

Slika 15. Prikaz kôda za dodavanja polja „times"

Format koji se dobije nakon korištenja kôda sa slike 15, rezultat je prikazan u nastavku na slici 16:

convert-functions > transformed_data.txt

```
[
  {
    "id": 1,
    "times": [
      "05:54:00",
      "06:41:00",
      "06:41:00",
      "07:51:00",
      "09:21:00",
      "12:01:00",
      "12:06:00",
      "13:01:00",
      "14:11:00",
      "15:41:00",
      "15:46:00",
      "17:41:00",
      "19:41:00",
      "20:51:00",
      "20:51:00"
    ]
  },
  {
    "id": 2,
    "times": [
      "06:06:00",
      "06:57:00",
      "07:13:00",
      "08:10:00",
      "09:40:00",
      "12:17:00",
      "12:45:00",
      "13:17:00",
      "14:27:00",
```

Slika 16. Prikaz podataka nakon svih koraka

Sad je potrebno napraviti transformaciju datoteke stops.txt koja je navedena prije u diplomskom

radu. To radimo na način da kreiramo NodeJS program koji će izvući samo one podatke koji su nam potrebni, prikaz programskog kôda je u nastavku na slici 17.

```
const fs = require("fs");

const readFileAndConvertToJson = (filePath) => {
  try {
    const data = fs.readFileSync(filePath, "utf-8");
    const lines = data.trim().split("\n");
    const jsonData = [];

    lines.forEach((line) => {
      const [id, nameWithCommas, latitude, longitude] = line.trim().split(",");
      const name = nameWithCommas.replace(/-[^\w\s]/g, "").trim();
      jsonData.push({
        id: parseInt(id),
        name,
        latitude: parseFloat(latitude),
        longitude: parseFloat(longitude),
      });
    });

    return jsonData;
  } catch (error) {
    console.error("Error reading or processing the file:", error);
    return null;
  }
};

const filePath = "stops.txt";
const jsonData = readFileAndConvertToJson(filePath);
if (jsonData) {
  const jsonString = JSON.stringify(jsonData, null, 2);
  console.log(jsonString);
}
```

Slika 17. Prikaz kôda za transformaciju stops.txt

Rezultat programskog kôd je prikazan na slici 18.

```
{
  id: 1,
  name: "FAŽANA-A",
  latitude: 44.925352,
  longitude: 13.80516,
},
{
  id: 2,
  name: "FAŽANA -B",
  latitude: 44.925288,
  longitude: 13.805253,
},
{
  id: 3,
  name: "FAŽANA OKRETIŠTE",
  latitude: 44.9253562471183,
  longitude: 13.8051527738571,
},
```


Slika 18. Prikaz podataka nakon transformacije

Nakon transformacije je ručno dodana slika (koja je pohranjena uz pomoć Google Maps i print screena) za svaku stanicu te uz usporedbu sa stop_times podacima, odnosno autobusnim rasporedom za svaku stanicu, dodan je inUse property koji može biti 1 ili 0, odnosno ili je stanica u uporabi ili nije. Nakon svega navedenog, format stanica je prikazan na slici 19.

```
const busStopData = [
  {
    id: 1,
    name: "FAŽANA-A",
    image: require("../images/bus-stops/FAŽANA-A.jpg"),
    latitude: 44.925352,
    longitude: 13.80516,
    inUse: 1,
  },
  {
    id: 2,
    name: "FAŽANA -B",
    image: require("../images/bus-stops/FAŽANA-A.jpg"),
    latitude: 44.925288,
    longitude: 13.805253,
    inUse: 1,
  },
  {
    id: 3,
    name: "FAŽANA OKRETIŠTE",
    image: require("../images/bus-stops/FAŽANA-A.jpg"),
    latitude: 44.9253562471183,
    longitude: 13.8051527738571,
    inUse: 0,
  },
```

Slika 19. Prikaz formata stanica

Na kraju se dobiva konstanta busStopData koja sadrži podatke o svim autobusnim stanicama koje se koriste u aplikaciji. Svaki objekt sadrži:

ID – svaka stanica ima zasebni ID

name – ime stanice

image – slika stanice koja je pohranjena u aplikaciji

latitude i longitude – geografske koordinate stanice

inUse – property koji označava je li stanica u uporabi

Aplikacija također koristi podatke o autobusnim linija. Format je prikazan na slici 20.

```
const routesData = [
  {
    id: 0,
    name: "1",
    color: "magenta",
    stops: [
      73, 176, 265, 266, 369, 198, 199, 130, 257, 76, 77, 78, 79, 80, 81, 82,
      83, 84, 85, 86, 87, 88, 89, 90, 110, 189, 190, 370, 158, 94, 160, 98,
    ],
    pathCoords: [
      { latitude: 44.8762118075772, longitude: 13.8546985387802 },
      { latitude: 44.876282476100805, longitude: 13.854751233148491 },
      { latitude: 44.876317163847574, longitude: 13.85482767609816 },
      { latitude: 44.87634338670522, longitude: 13.854818707351694 },
      { latitude: 44.876846038922665, longitude: 13.854591114383584 },
      { latitude: 44.87691321770848, longitude: 13.854590875472827 },
      { latitude: 44.87699318460799, longitude: 13.854571909392797 },
      { latitude: 44.87707759096223, longitude: 13.854565874423086 },
      { latitude: 44.877144589638135, longitude: 13.854879692848055 },
      { latitude: 44.87736426413096, longitude: 13.856004478301529 },
      { latitude: 44.8773749, longitude: 13.8559997 },
      { latitude: 44.877370000000006, longitude: 13.856000000000002 },
      { latitude: 44.87742, longitude: 13.856200000000001 },
      { latitude: 44.87753, longitude: 13.856700000000002 },
      { latitude: 44.877660000000006, longitude: 13.857310000000002 },
      { latitude: 44.877810000000004, longitude: 13.858070000000001 },
      { latitude: 44.87802000000001, longitude: 13.85904 },
      { latitude: 44.87807, longitude: 13.85932 },
```

Slika 20. Prikaz formata autobusnih linija

Format sadrži:

ID – ručno dodan

name – ime autobusne linija

color – boja linije ručno dodana

stops – ručno dodane stanice od početka linije do kraja, stanice ide redoslijedom od prve do zadnje

pathCoords – koordinate točaka koje kad se spoje sastavljaju autobusnu liniju koja se ocrtava u aplikaciji

Koordinate za svaku liniju je napravljeno na sljedeći način: Korišten je Google Directions API. Međutim prilikom izrade linija događaju se par manjih problema: Google Directions API gleda najbliži put između dvije točke, što u slučaju aplikacije nije idealno pošto aplikacija mora sadržavati koordinate od stanice do stanice što dovodi do sljedećeg zaključka: za izradu jedne autobusne linije treba se podijeliti linija na puno manjih kako bi Google Directions API dao točke podatke. Za korištenje Google Directions API treba izraditi račun i na mjesečnoj razini ima besplatno otprilike 1000 poziva, stoga u aplikaciji je napravljeno više od sto API poziva te su koordinate spremljene u konstantu. Primjer korištenja Google Directions API poziva je prikazan na slici 21.

```
https://maps.googleapis.com/maps/api/directions/json
  ?destination=Montreal
  &origin=Toronto
  &key=YOUR_API_KEY
```

Slika 21. Primjer Google Directions API poziva
(https://developers.google.com/maps/documentation/directions/overview)
28.08.2023.

Poziv sadrži nekoliko parametra:

- destination – destinacija ili krajnja točka - može biti ime grada, u slučaju izrade aplikacija korištene su koordinate (longitude, latitude)
- origin – početka točka - može biti ime grada, u slučaju izrade aplikacija korištene su koordinate (longitude, latitude)
- key – ključ koji se dobiva izradom računa na Google Cloud Platform
- mode (optional) – način rada, odnosno način poziva, može se ubaciti opcionalni parametar npr. walking (hodanje), driving (vožnja), transit (javni prijevoz)

Nakon API poziva dobiva se sljedeći rezultat koji je prikazan na slici 22, 23, 24, 25.

```
{
  "geocoded_waypoints": [
    {
      "geocoder_status": "OK",
      "place_id": "ChIJQ-FChCDTfEcRpq8sdcRHwMI",
      "types": ["street_address"]
    },
    {
      "geocoder_status": "OK",
      "place_id": "ChIJQ6PlEeDSfEcRAZSChwPtU-4",
      "types": ["establishment", "point_of_interest", "transit_station"]
    }
  ],
  "routes": [
    {
      "bounds": {
        "northeast": {
          "lat": 44.8700678,
          "lng": 13.8523644
        },
        "southwest": {
          "lat": 44.86842009999999,
          "lng": 13.848521
        }
      },
      "copyrights": "Map data \u00a92023",
      "legs": [
        {
          "distance": {
            "text": "0.4 km",
            "value": 438
          },
          "duration": {
            "text": "2 mins",
            "value": 116
          },
          "end_address": "ISTARSKA -A (FINA), 52100, Pula, Croatia",
          "end_location": {
            "lat": 44.8700678,
            "lng": 13.8485338
          },
          "start_address": "Cankarova ul. 13, 52100, Pula, Croatia",
```


Slika 22. Prikaz prvog dijela rezultata API poziva

```
"start_address": "Cankarova ul. 13, 52100, Pula, Croatia",
"start_location": {
  "lat": 44.86842009999999,
  "lng": 13.852336
},
"steps": [
  {
    "distance": {
      "text": "8 m",
      "value": 8
    },
    "duration": {
      "text": "1 min",
      "value": 2
    },
    "end_location": {
      "lat": 44.8684849,
      "lng": 13.8523644
    },
    "html_instructions": "Head \u003cb\u003enorth\u003c/b\u003e on \u003cb\u003eCankarova ul.\u003c/b\u003e toward \u003cb\u003eZagreba\u010dka ul.\u003c/b\u003e",
    "polyline": {
      "points": "sjzpGcppsAKC"
    },
    "start_location": {
      "lat": 44.86842009999999,
      "lng": 13.852336
    },
    "travel_mode": "DRIVING"
  },
  {
    "distance": {
      "text": "0.3 km",
      "value": 322
    },
    "duration": {
      "text": "1 min",
      "value": 87
    },
```


Slika 23. Prikaz drugog dijela API poziva


```
    "end_location": {
      "lat": 44.8690951,
      "lng": 13.8485257
    },
    "html_instructions": "Turn \u003cb\u003eleft\u003c/b\u003e onto \u003cb\u003eZagreba\u010dka ul.\u003c/b\u003e",
    "maneuver": "turn-left",
    "polyline": {
      "points": "_kzpGgppsA[nBFfE?jAOlC?BATCVCJS`BIXo@p@"
    },
    "start_location": {
      "lat": 44.8684849,
      "lng": 13.8523644
    },
    "travel_mode": "DRIVING"
  },
  {
    "distance": {
      "text": "0.1 km",
      "value": 99
    },
    "duration": {
      "text": "1 min",
      "value": 25
    },
    "end_location": {
      "lat": 44.8699843,
      "lng": 13.8485275
    },
    "html_instructions": "Turn \u003cb\u003eright\u003c/b\u003e onto \u003cb\u003eGiardini\u003c/b\u003e",
    "maneuver": "turn-right",
    "polyline": {
      "points": "{nzpGixosAkACwADKA"
    },
    "start_location": {
      "lat": 44.8690951,
      "lng": 13.8485257
    },
```


Slika 24. Prikaz trećeg dijela API poziva

```
            "travel_mode": "DRIVING"
          },
          {
            "distance": {
              "text": "9 m",
              "value": 9
            },
            "duration": {
              "text": "1 min",
              "value": 2
            },
            "end_location": {
              "lat": 44.8700678,
              "lng": 13.8485338
            },
            "html_instructions": "Continue onto \u003cb\u003eIstarska ul.\u003c/b\u003e",
            "polyline": {
              "points": "ktzpGixosAQ?"
            },
            "start_location": {
              "lat": 44.8699843,
              "lng": 13.8485275
            },
            "travel_mode": "DRIVING"
          }
        ],
        "traffic_speed_entry": [],
        "via_waypoint": []
      }
    ],
    "overview_polyline": {
      "points": "sjzpGcppsAKC[nBFrGU~DWlBIXo@p@kACwAD]A"
    },
    "summary": "Zagreba\u010dka ul.",
    "warnings": [],
    "waypoint_order": []
  }
 ],
 "status": "OK"
}
```


Slika 25. Prikaz četvrtog dijela API poziva

Slike 22, 23, 24, 25 predstavljaju Google Directions API poziv, te API vrati podatke u obliku JSON formata. U formatu se nalaze svakakvi parametri, međutim za izradu aplikacije potrebni su sljedeći podaci: u podacima koji su vraćeni nalazi se parametar „steps" koji zapravo predstavlja korake od početne točke do završne točke. Cilj ovog JSON formata je sljedeći: broj koraka može biti n broj koraka – sve ovisno o tome kolika je udaljenost početne točke i završne i ovisnost o tome koliko su ceste komplicirane – npr. kružni tokovi su najviše komplicirani jer je potrebno puno više točaka kako bi se ocrtala kvalitetna linija. Prikaz jednog koraka se vidi u nastavku na slici 26.

Slika 26. Prikaz jednog koraka

Kako bi dobili koordinate napravljeno je sljedeće rješenje: program koji dekodira JSON format na sljedeći način:

Za svaki korak (step) napravi sljedeće:

1. uzmi početnu točku (start_location) i ubaci na početak
2. uzmi polilinju i dekodiraj je pomoću funkcije koja je navedena ispod teksta (time se dobiva dodatne točke između početne i završne točke)
3. na kraj postavi završnu točku (end_location)

Program, odnosno funkcije su oformljene koristeći dokumentaciju platforme Google Maps i koristeći internet resurse. Programski kôd je prikazan u nastavku na slici 27.

```
export const decodePolyline = (polylineStr) => {
  let index = 0;
  const points = [];
  let lat = 0;
  let lng = 0;

  while (index < polylineStr.length) {
    let shift = 0;
    let result = 0;
    let byte;

    do {
      byte = polylineStr.charCodeAt(index++) - 63;
      result |= (byte & 0x1f) << shift;
      shift += 5;
    } while (byte >= 0x20);

    const deltaLat = result & 1 ? ~(result >> 1) : result >> 1;
    lat += deltaLat;

    shift = 0;
    result = 0;

    do {
      byte = polylineStr.charCodeAt(index++) - 63;
      result |= (byte & 0x1f) << shift;
      shift += 5;
    } while (byte >= 0x20);

    const deltaLng = result & 1 ? ~(result >> 1) : result >> 1;
    lng += deltaLng;

    points.push([lat * 1e-5, lng * 1e-5]);
  }

  return points;
};
```

Slika 27. Prikaz kôda za dekodiranje jedne polilinije

Kôd je napravljen pomoću navedene literature [6][7][8]. Ova funkcija služi za pretvaranje polilinije u listu stvarnih geografskih koordinata.

```javascript
export const parseJSONRoute = () => {
  const data = jsonFile;
  const route = [];
  data.routes[0].legs[0].steps.forEach((step) => {
    const decodedPolylineCoords = decodePolyline(step.polyline.points).map(
      (polyline) => {
        return { latitude: polyline[0], longitude: polyline[1] };
      }
    );
    route.push(
      {
        latitude: step.start_location.lat,
        longitude: step.start_location.lng,
      },
      ...decodedPolylineCoords,
      {
        latitude: step.end_location.lat,
        longitude: step.end_location.lng,
      }
    );
  });

  console.log(
    "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  );
  console.log(
    "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  );
  console.log(
    "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  );
  console.log(
    "------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------
  );
  console.log(route);
};
```

Slika 28. Prikaz dodatnog programa koji uzima JSON format koordinata i pretvara u točke


```
LOG ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------
LOG ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------
LOG ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------
LOG ----------------------------------------------------------------------------------------------------------------------------------------------------------------------------
----------------------------------
LOG [{"latitude": 44.86842009999999, "longitude": 13.852336}, {"latitude": 44.86842, "longitude": 13.852340000000002}, {"latitude": 44.868480000000005, "longitude": 13.852360000000001}, {"latitude": 44.8684849, "longitude": 13.852
3644}, {"latitude": 44.8684849, "longitude": 13.8523644}, {"latitude": 44.868480000000005, "longitude": 13.852360000000001}, {"latitude": 44.86862000000001, "longitude": 13.8518}, {"latitude": 44.86858, "longitude": 13.850800000000000
01}, {"latitude": 44.86858, "longitude": 13.850420000000002}, {"latitude": 44.868660000000006, "longitude": 13.849710000000002}, {"latitude": 44.868660000000006, "longitude": 13.84969}, {"latitude": 44.86867, "longitude": 13.8495800
00000001}, {"latitude": 44.86869, "longitude": 13.84946}, {"latitude": 44.86871000000001, "longitude": 13.849400000000001}, {"latitude": 44.86881, "longitude": 13.848910000000002}, {"latitude": 44.868860000000005, "longitude": 13.84
8780000000001}, {"latitude": 44.8691, "longitude": 13.848530000000002}, {"latitude": 44.8690951, "longitude": 13.8485257}, {"latitude": 44.8690951, "longitude": 13.8485257}, {"latitude": 44.8691, "longitude": 13.848530000000002}, {"
latitude": 44.86948, "longitude": 13.848550000000001}, {"latitude": 44.86992, "longitude": 13.84852}, {"latitude": 44.869980000000005, "longitude": 13.848530000000002}, {"latitude": 44.8699843, "longitude": 13.8485275}, {"latitude":
 44.8699843, "longitude": 13.8485275}, {"latitude": 44.869980000000005, "longitude": 13.848530000000002}, {"latitude": 44.870070000000005, "longitude": 13.848530000000002}, {"latitude": 44.8700678, "longitude": 13.8485338}]
```


Slika 29. Rezultat navedenih programa

Rezultat dobivenih programa izgleda kao niz koordinata koje su ručno kopirane u svaku liniju. Napomena je da je ovo primjer samo djelića autobusne linije, za svaku liniju je bilo potrebno n broj API poziva i n broj ovakvih rezultata koje je bilo potrebno obraditi, testirati i slično. Broj točaka svake linije ima ovisnost o kompleksnosti same linije, odnosno ako linija ima puno zavoja, kružnih tokova, raskrižja i slično, točaka će biti puno više nego ako linija ide ravnim pravcem.

Nakon transformacije svih podataka i načina transformacije podataka koji su navedeni u radu kreće manipuliranje podacima i izrada sučelja aplikacije.

5. Opis aplikacije i funkcionalnosti

Aplikacija je izrađena pomoću nekoliko paketa:

- react
- react native
- react native maps
- expo location
- expo vector icons

Svaki paket ima svoju funkciju u aplikaciji, npr. pomoću react paketa kreiraju se stanja u aplikaciji (states). Pomoću manipuliranja stanja aplikacija se ažurira. React native paket sadrži elemente poput pogleda (View), teksta (Text), slike (Image) i slično. React native maps omogućava izradu karte (MapView). Expo location pomaže u identifikaciji korisnikove lokacije i slično. Expo vector icons paket pruža širok spektar različitih ikona. Svaki paket naravno ima još puno toga što može ponuditi, međutim navedene su samo neke pojedinosti koje su korištene u radu. U sljedećem dijelu diplomskog rada bit će prezentirani važniji dijelovi kôda i slike ekrana. Cijeli kôd bit će na poveznici github-a koja je postavljena u uvodu ovog rada.

Kroz cijelu aplikaciju koristi se useContext koji ima svrhu globalne varijable. U useContext spremljeni su podaci vezano za jezik i za radijus koji će biti kasnije objašnjeni u radu. U nastavku na slici 30 je prikazan programski kôd koji sadrži implementaciju useContext-a u aplikaciji.

```
import React, { useEffect, useState } from "react";
import AsyncStorage from "@react-native-async-storage/async-storage";

export const UserContext = React.createContext({
  userData: "",
  setUserData: (userData) => {},
});

const UserContextProvider = ({ children }) => {
  const [userData, setUserData] = useState("");

  const getData = async () => {
    try {
      const value = await AsyncStorage.getItem("userData");
      if (value !== null) {
        setUserData(JSON.parse(value));
        return value;
      } else {
      }
    } catch (error) {}
  };

  useEffect(() => {
    getData();
  }, []);

  const setData = (userData) => {
    try {
      AsyncStorage.setItem("userData", JSON.stringify(userData));
      setUserData(userData);
    } catch (error) {}
  };

  const contextValue = { userData, setUserData, setData };

  return (
    <UserContext.Provider value={contextValue}>{children}</UserContext.Provider>
  );
};

export default UserContextProvider;
```

Slika 30. Prikaz useContext-a u kôdu

U radu je korištena Haversina formula za izračun udaljenosti između dvije koordinate. Napravljen je dodatan dio uz formulu koji pretvara izračun u metre. Izgled kôda je prikazan na slici 31 u nastavku:

```
export const calculateDistance = (lat1, lon1, lat2, lon2, toString) => {
  const R = 6371;
  const dLat = toRadians(lat2 - lat1);
  const dLon = toRadians(lon2 - lon1);
  const a =
    Math.sin(dLat / 2) * Math.sin(dLat / 2) +
    Math.cos(toRadians(lat1)) *
      Math.cos(toRadians(lat2)) *
      Math.sin(dLon / 2) *
      Math.sin(dLon / 2);
  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));
  const distance = R * c;

  if (toString) {
    const stringNumber = "~" + (distance * 1000).toFixed(1).toString() + "m";
    return stringNumber;
  }

  return distance;
};

const toRadians = (degrees) => {
  return (degrees * Math.PI) / 180;
};
```

Slika 31. Prikaz Haversine formule u JavaScript kôdu

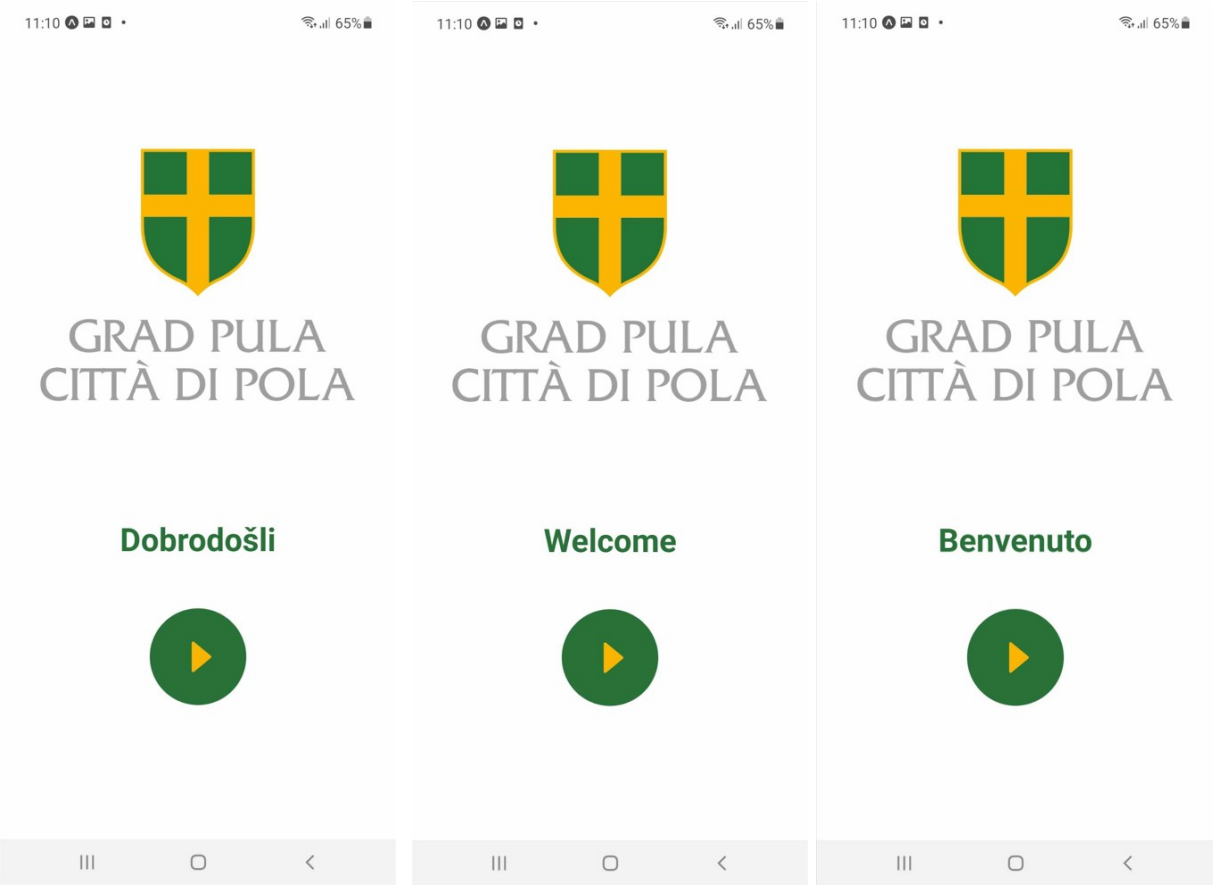
5.1 Početni prozori

Prilikom ulaska u aplikaciju, korisnike će dočekati početni zaslon aplikacije koji će pozdraviti korisnike na nekoliko različitih jezika (npr. na hrvatskom, engleskom, njemačkom, talijanskom…). Pozdravi se izmjenjuju svakih pola sekunde kako bi korisnici dobili iluziju animacije. U nastavku na slici 32 je prikazan programski kôd početnog prozora, te na slikama 33, 34, 35 je prikazan izgled.

```
const FirstScreen = ({ setAppLoad }) => {
  const insets = useSafeAreaInsets();
  const greetings = [
    "Dobrodošli",
    "Welcome",
    "Bienvenido",
    "Willkommen",
    "Bienvenue",
    "Benvenuto",
  ];

  const [currentGreetingIndex, setCurrentGreetingIndex] = useState(0);

  useEffect(() => {
    const interval = setInterval(() => {
      setCurrentGreetingIndex(
        (prevIndex) => (prevIndex + 1) % greetings.length
      );
    }, 500);

    return () => clearInterval(interval);
  }, [greetings.length]);

  return (
    <View
      style={{
        top: insets.top,
        justifyContent: "center",
        alignItems: "center",
        flex: 1,
        width: "100%",
      }}
    >
      <View style={firstScreenStyle.imageView}>
        <Image source={pula} style={firstScreenStyle.imageStyle} />
      </View>
      <View style={firstScreenStyle.textIconView}>
        <Text style={firstScreenStyle.greetingsText}>
          {greetings[currentGreetingIndex]}
        </Text>
        <Pressable
          style={firstScreenStyle.pressableContinue}
          onPress={() => setAppLoad(false)}
        >
          <AntDesign name="caretright" size={36} color="#FAB500" />
        </Pressable>
      </View>
    </View>
  );
};

export default FirstScreen;
```

Slika 32. Prikaz kôda početnog prozora

Slike 33, 34 i 35 prikazuju par početnih prozora sa različitim pozdravim, korisnik može nastaviti u aplikaciju pritiskom na zeleni gumb.



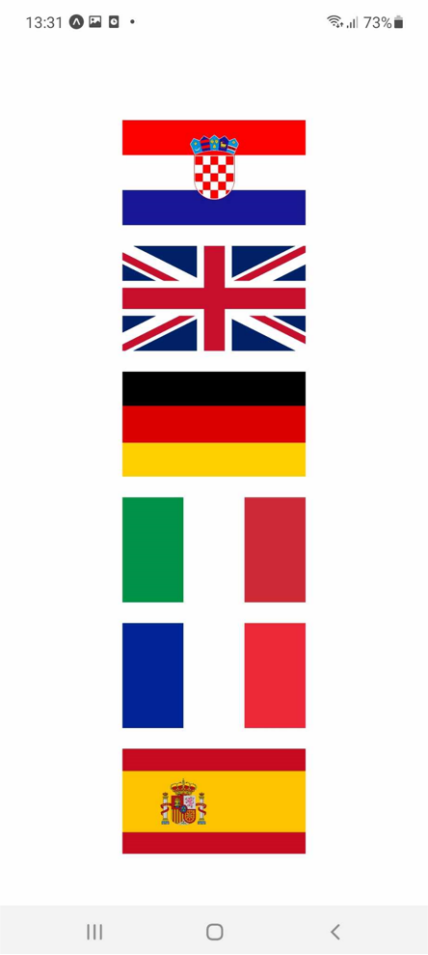
Slika 33. (lijevo) Početni prozor sa hrvatskim pozdravom
Slika 34. (sredina) Početni prozor sa engleskim pozdravom
Slika 35. (desno) Početni prozor sa talijanskim pozdravom

5.2 Postavljanje jezika

Korisnik prilikom prvog susreta s aplikacijom mora postaviti jezik kako bi nastavio do glavnog dijela aplikacije. Postavljanje jezika vrši se klikom na jednu od ponuđenih zastava. U nastavku na slikama 36 i 37 je prikazan programski kôd i izgled.

```
import { Pressable } from "react-native";
import { View } from "react-native";
import { useSafeAreaInsets } from "react-native-safe-area-context";
import { Image } from "react-native";
import langs from "../lang-data/langs";
import { useContext } from "react";
import { UserContext } from "../context/context";

const SetLang = ({ changeLang = false }) => {
  const context = useContext(UserContext);
  const insets = useSafeAreaInsets();
  return (
    <View
      style={{
        flex: 1,
        width: "100%",
        top: insets.top,
      }}
    >
      <View
        style={{
          flex: 1,
          width: "100%",
          alignItems: "center",
          justifyContent: "center",
          gap: 20,
        }}
      >
        {langs.map((lang) => {
          return (
            <Pressable
              key={lang.name}
              style={{ width: 175, height: 100 }}
              onPress={() => {
                context.setData({ data: { lang: lang.name, radius: "10" } });
              }}
            >
              <Image
                source={lang.image}
                style={{ height: "100%", width: "100%" }}
              ></Image>
            </Pressable>
          );
        })}
      </View>
    </View>
  );
};

export default SetLang;
```

Slika 36. Prikaz kôda za postavljanje jezika prilikom prve uporabe aplikacije



Slika 37. Prozor za postavljanje jezika

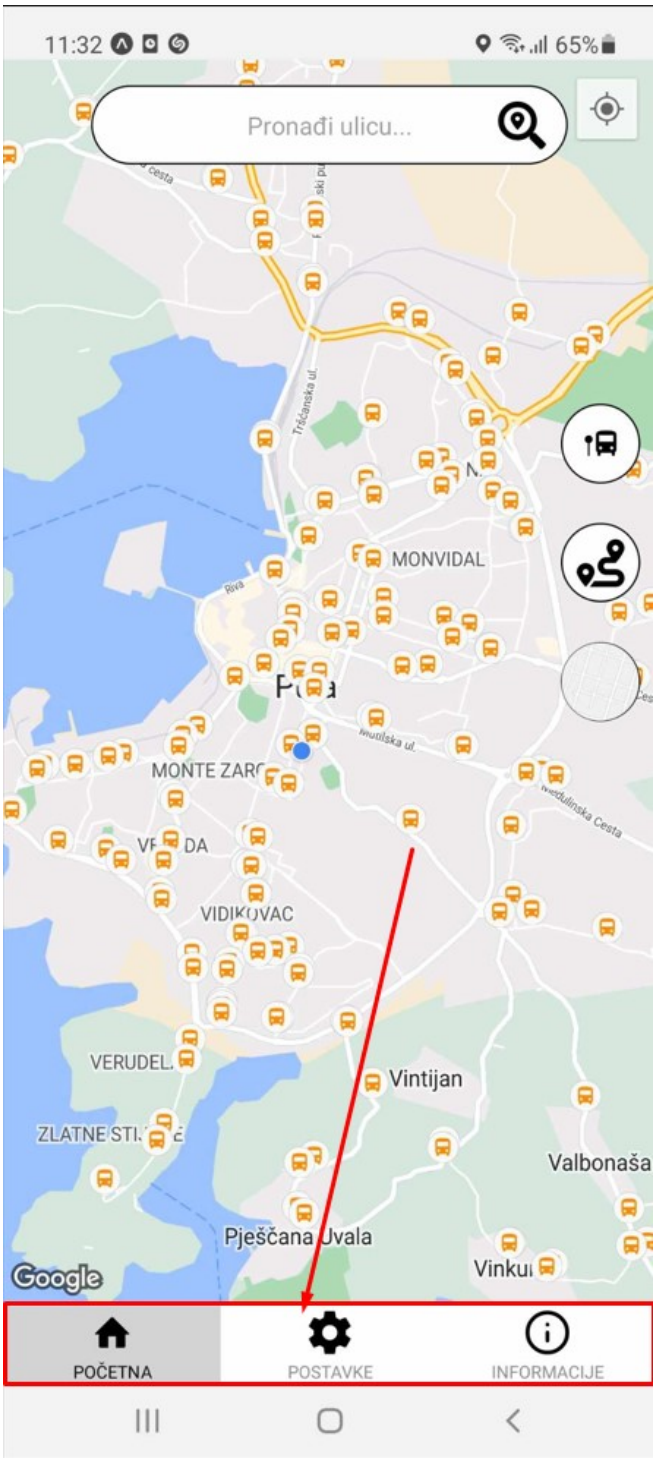
Slika 37 prikazuje prozor za postavljanja jezike , te klikom na bilo koju zastavu vodi korisnika na glavni dio aplikacije – pregled karte i funkcionalnosti.

5.3 Glavni dio aplikacije

Glavni dio aplikacije sastoji se od 3 kartice (eng. tabs). Najvažnija i najbitnija kartica je prva kartica - „POČETNA'' - odnosno pregled karte s funkcionalnostima. Druga kartica je kartica „POSTAVKE" u kojoj možemo mijenjati postavljeni jezik i smanjiti radijus prikazivanja autobusnih ikonica. Treća kartica je kartica „INFORMACIJE'' koja služi kao upute korisnicima za pojedine funkcionalnost aplikacije. U nastavku su prikazane slike vezane za programski kôd i izgled kartica.

```
  return (
    <NavigationContainer>
      <StatusBar style="dark" />
      <Tab.Navigator screenOptions={{ headerShown: false }}>
        <Tab.Screen
          name="Home"
          component={TabHomeScreen}
          options={{
            tabBarActiveBackgroundColor: "lightgrey",
            tabBarActiveTintColor: "black",
            tabBarIcon: ({ color, size }) => (
              <Entypo name="home" size={24} color="black" />
            ),
            tabBarLabel: currentLang.homeText,
          }}
        />

        <Tab.Screen
          name="Settings"
          component={TabSettings}
          options={{
            tabBarActiveBackgroundColor: "lightgrey",
            tabBarActiveTintColor: "black",
            tabBarIcon: ({ color, size }) => (
              <Ionicons name="settings-sharp" size={30} color="black" />
            ),
            tabBarLabel: currentLang.settingsText,
          }}
        />

        <Tab.Screen
          name="Help"
          component={TabHelp}
          options={{
            tabBarActiveBackgroundColor: "lightgrey",
            tabBarActiveTintColor: "black",
            tabBarIcon: ({ color, size }) => (
              <Feather name="info" size={30} color="black" />
            ),
            tabBarLabel: currentLang.infoText,
          }}
        />
      </Tab.Navigator>
    </NavigationContainer>
  );
};
```

Slika 38. Prikaz kôda za kartice (tabs)



Slika 39. Prikaz kartica u aplikaciji

5.3.1 Kartica - „POČETNA”

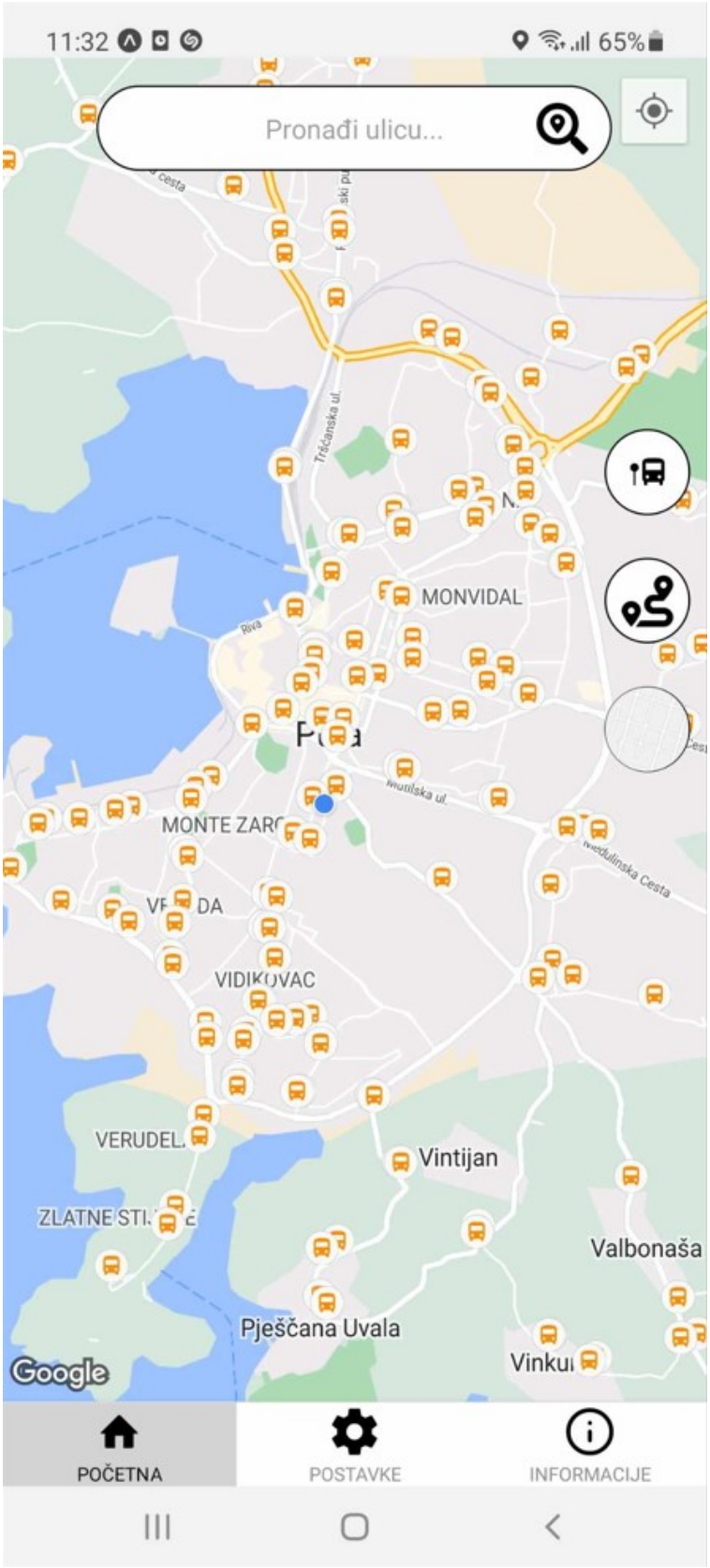
Kartica „Početna'' sadržava interaktivnu kartu koja ima nekoliko funkcionalnosti. Neke od funkcionalnosti su:

- Pomak karte na lokaciju korisnika
- Pretraživanje ulica
- Pronalazak najbližih linija za pronađenu ulicu
- Pronalazak najbliže autobusne stanice uzimaju u obzir lokaciju korisnika
- Prikaz prozora svih autobusnih linija
- Promjena izgleda karte
- Prikaz informacija o određenoj autobusnoj stanici
- Pronalazak puta po želji

Na slikama 40 i 41 su prikazani programski kôd i izgled kartice.

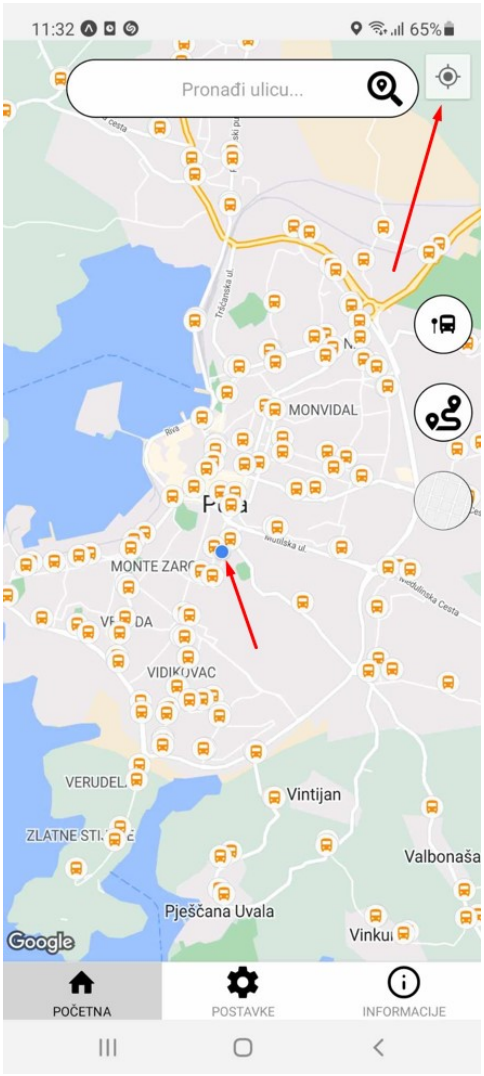
```
  return (
    <View style={{ flex: 1, paddingTop: insets.top }}>
      <SearchBar findLocation={findLocation} />
      {renderStreetInstructions()}
      {renderModalSelectedStop()}
      {renderAllRoutesModal()}
      {renderActiveRouteModal()}
      {renderSearchBarError()}
      {renderButtons()}
      {renderStreetInstructionsError()}
      <MapView
        style={styles.map}
        mapType={mapType}
        initialRegion={region}
        onRegionChangeComplete={(region) => {
          if (regionChangeTimeout) {
            clearTimeout(regionChangeTimeout);
          }

          regionChangeTimeout = setTimeout(() => {
            handleCheckRegion(region);
            setRegionChange(region);
          }, 1000);
        }}
        ref={mapRef}
        customMapStyle={mapStyle}
        showsUserLocation={true}
        onUserLocationChange={(event) => {
          const currentLocation = event.nativeEvent.coordinate;
          if (!currentLocation) {
            return;
          }
          const { latitude, longitude } = currentLocation;
          setLocation({
            coords: {
              latitude,
              longitude,
            },
          });
        }}
        onPress={() => {
          setSelectedRouteId(null);
          setSelectedStopId(null);
        }}
      >
        {renderSearchedStreetMarker()}
        {renderPolylineClosestStop()}
        {renderRoutePath()}
        {renderBusStopMarkers()}
        {renderPolylineClosestStreetStop()}
      </MapView>
    </View>
  );
};

export default Overview;
```

Slika 40. Prikaz kôda za glavni pregled - „POČETNA'' kartica



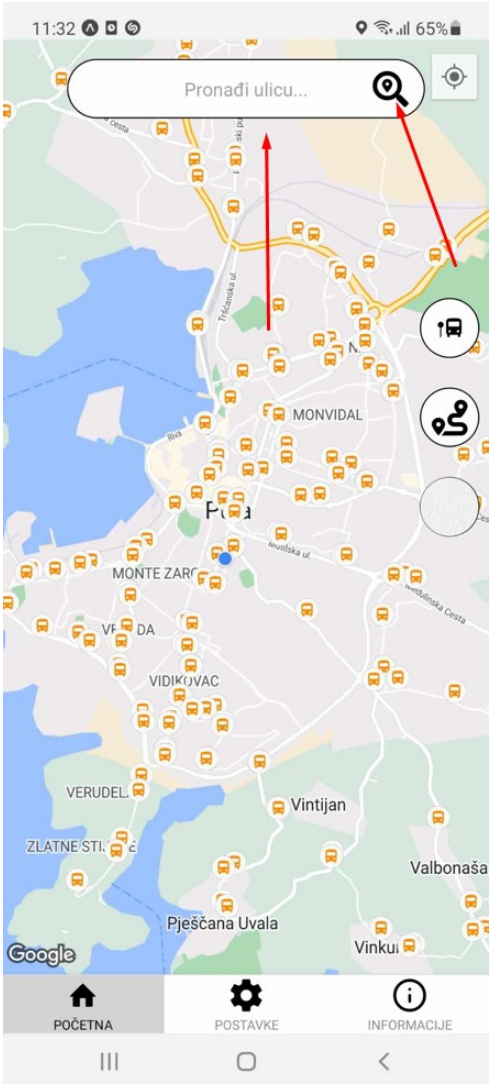
Slika 41. Prikaz kartice „POČETNA''

- Pomak karte na lokaciju korisnika možemo postići klikom na gumb koji je prikazan na slici ispod. Funkcija koja pomiče je integrirana u MapView. Na slici 42 je prikazana funkcionalnost i lokacija korisnika



Slika 42. Prikaz gumba i lokacije korisnika

• Pretraživanje ulica možemo postići upisom željene ulice u navedenu tražilicu i pritiskom na gumb. Funkcionalnost je prikazana na slici 43, te programski kôd na slici 44, 45 i 46.



Slika 43. Prikaz tražilice i gumba pretraživanja

```
import { TextInput, View, Pressable } from "react-native";
import { FontAwesome5 } from "@expo/vector-icons";
import { searchBar } from "../styles";
import { useContext, useState } from "react";
import { useSafeAreaInsets } from "react-native-safe-area-context";
import { UserContext } from "../context/context";
import langs from "../lang-data/langs";

const SearchBar = ({ findLocation }) => {
  const [inputText, setInputText] = useState(null);
  const insets = useSafeAreaInsets();
  const userCtx = useContext(UserContext);

  const currentLang = langs.find(
    (lang) => lang.name === userCtx.userData.data.lang
  );

  return (
    <View
      style={{
        width: "100%",
        position: "absolute",
        zIndex: 999,
        top: insets.top + 16,
        justifyContent: "center",
        alignItems: "center",
      }}
    >
      <View style={searchBar.searchBar}>
        <TextInput
          style={searchBar.searchBarInput}
          onChangeText={(text) => setInputText(text)}
          placeholder={currentLang.search}
        />
        <View style={searchBar.buttonSearchLocation}>
          <Pressable onPress={() => findLocation(inputText)}>
            <FontAwesome5 name="search-location" size={30} color="black" />
          </Pressable>
        </View>
      </View>
    </View>
  );
};

export default SearchBar;
```

Slika 44. Prikaz kôda tražilice

```
const findLocation = async (searchText) => {
  setSelectedRouteId(null);
  setShowAllRoutes(false);
  setSelectedStopId(null);
  const foundLocation = await searchLocation(searchText);

  if (foundLocation.status === "fail") {
    setShowSearchedStreet("error");
    return;
  }
  const selectedRegion = {
    latitude: foundLocation.latitude,
    longitude: foundLocation.longitude,
    latitudeDelta: 0.0025,
    longitudeDelta: 0.0025,
  };
  setShowSearchedStreet({
    streetName: searchText,
    latitude: selectedRegion.latitude,
    longitude: selectedRegion.longitude,
  });

  const nearStop = findNearestBusStopAtDesiredLocation({
    latitude: selectedRegion.latitude,
    longitude: selectedRegion.longitude,
  });

  mapRef.current?.animateToRegion(selectedRegion, 1000);

  setTimeout(() => {
    mapRef.current?.animateToRegion(
      {
        latitude: nearStop.latitude,
        longitude: nearStop.longitude,
        latitudeDelta: 0.0025,
        longitudeDelta: 0.0025,
      },
      1500
    );
  }, 2500);
};
```

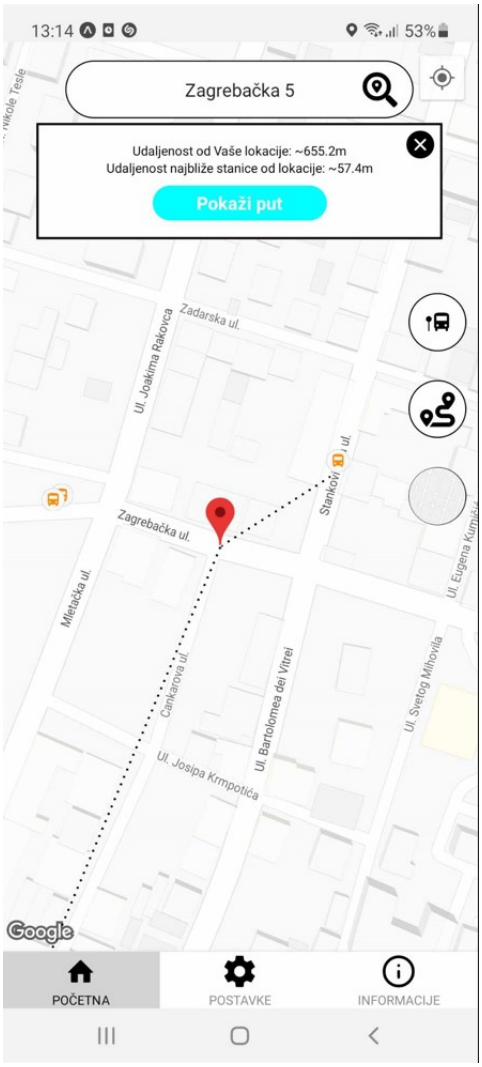
Slika 45. Prikaz kôda za traženje unesene lokacije

```
export const searchLocation = async (searchText) => {
  try {
    const response = await fetch(
      `https://nominatim.openstreetmap.org/search?q=${searchText}+Pula&format=json`
    );
    const data = await response.json();
    if (data && data.length > 0) {
      const { lat, lon } = data[0];
      return {
        status: "success",
        latitude: +lat,
        longitude: +lon,
      };
    } else {
      return { status: "fail" };
    }
  } catch (error) {
    return { status: "fail" };
  }
};
```

Slika 46. Prikaz kôda za API poziv prema OpenStreetMap

Korišten je OpenStreetMap iz razloga što je korištenje besplatno.

Nakon pritiska gumba, aplikacija će nam uvećati mapi ako je potrebno te prikazati ulicu. Primjer odabira ulice je prikazan na slici 47.



Slika 47. Prikaz pretraživanja ulice

Ukoliko korisnik klikne na gumb „Pokaži put'' aplikacija će pokušati naći najidealnije linije za dolazak na željenu destinaciju. Linije će biti poredane po broju stanica koje autobus mora proći do dolaska na destinaciju. Slike 48, 49 i 50 prikazuju programski kôd za manipuliranje sa podacima.

```
const handleClosestPath = () => {
  const markerStops = [];
  const personStops = [];
  const possibleRoutesIDs = [];
  const correctRoutesData = [];

  busStopDataInUse.forEach((busStop) => {
    const distanceToMarker = calculateDistance(
      showSearchedStreet.latitude,
      showSearchedStreet.longitude,
      busStop.latitude,
      busStop.longitude
    );
    const distanceToPerson = calculateDistance(
      location.coords.latitude,
      location.coords.longitude,
      busStop.latitude,
      busStop.longitude
    );
    if (distanceToMarker <= 0.25 && !markerStops.includes(busStop.id)) {
      markerStops.push(busStop.id);
    }
    if (distanceToPerson <= 0.25 && !personStops.includes(busStop.id)) {
      personStops.push(busStop.id);
    }
  });

  routesData.forEach((route) => {
    const passes =
      route.stops.some((id) => markerStops.includes(id)) &&
      route.stops.some((id) => personStops.includes(id));
    if (passes && !possibleRoutesIDs.includes(route.id)) {
      possibleRoutesIDs.push(route.id);
    }
  });

  possibleRoutesIDs.forEach((possibleRouteID) => {
```

Slika 48. Prikaz kôda za prikaz najoptimalnijih linija – prvi dio

```
possibleRoutesIDs.forEach((possibleRouteID) => {
  let personStopID;
  let wayPointStopID;
  const singleRoute = routesData.find(
    (route) => route.id === possibleRouteID
  );
  const distancesToPerson = singleRoute.stops
    .map((busStop) => {
      const singleStop = busStopData.find((stop) => stop.id === busStop);
      return {
        stopId: singleStop.id,
        distance: calculateDistance(
          location.coords.latitude,
          location.coords.longitude,
          singleStop.latitude,
          singleStop.longitude
        ),
      };
    })
    .sort((a, b) => {
      return a.distance < b.distance ? -1 : 1;
    });

  const distancesToMarker = singleRoute.stops
    .map((busStop) => {
      const singleStop = busStopData.find((stop) => stop.id === busStop);
      return {
        stopId: singleStop.id,
        distance: calculateDistance(
          showSearchedStreet.latitude,
          showSearchedStreet.longitude,
          singleStop.latitude,
          singleStop.longitude
        ),
      };
    })
    .sort((a, b) => {
      return a.distance < b.distance ? -1 : 1;
    });
  wayPointStopID = distancesToMarker[0].stopId;
  personStopID = distancesToPerson[0].stopId;
  const wayPointStopIndex = singleRoute.stops.indexOf(wayPointStopID);
  const personStopIndex = singleRoute.stops.indexOf(personStopID);
```

Slika 49. Prikaz kôda za prikaz najoptimalnijih linija – drugi dio

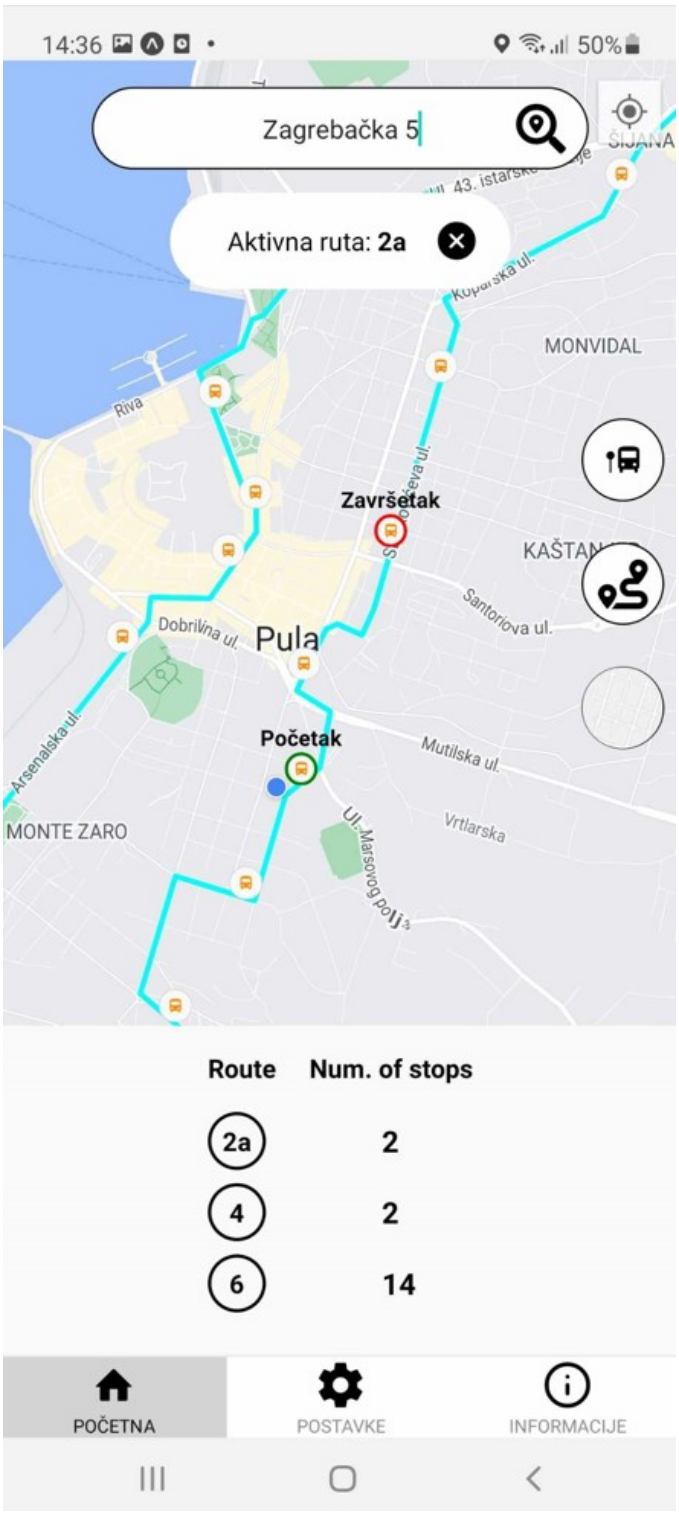
```
    if (personStopIndex < wayPointStopIndex) {
      correctRoutesData.push({
        id: singleRoute.id,
        numOfStops: wayPointStopIndex - personStopIndex,
        startStopID: personStopID,
        endStopID: wayPointStopID,
      });
    }
  });

  correctRoutesData.sort((a, b) => (a.numOfStops < b.numOfStops ? -1 : 1));
  if (correctRoutesData.length > 0) {
    setRoutesToMarker(correctRoutesData);
  } else {
    setErrorShowPath(true);
    setRoutesToMarker(null);
  }
};
```

Slika 50. Prikaz kôda za prikaz najoptimalnijih linija – treći dio

Funkcija izračunava najbliže stanice u radijusu 250 metara od tražene ulice i od korisnika. Postoje markerStops i personStops varijable koje služe da pohranjuju sve moguće stanice koje zadovoljavaju uvjet, odnosno stanice koje su u radijusu od 250 metara od tražene ulice. Sve takve stanice biti će dodane u markerStops i isto tako vrijedi i za personStops. Na kraju se dobiva niz stanica koje su u blizini tražene ulice i u blizini korisnika. Nakon toga za svaku autobusnu liniju se gleda postoji li barem 1 linija koja sadrži 2 stanice koje su sadržane u istoj liniji i koje su sadržane u tim dvaju niza. Nakon toga za svaku moguću liniju napravljen je izračun najbliže stanice korisnika i tražene stanice. Nakon toga gleda se indeks stanice blizu korisnika i blizu tražene ulice, ukoliko indeks korisnika je prije indeksa tražene ulice onda je pronađen adekvatan put, ukoliko bi indeks korisnika bio nakon indeksa tražene ulice to bi značilo da autobus prođe traženu ulicu i ide prema korisniku, međutim treba biti obrnuto - korisnik mora ići prema traženoj ulici. Tako se dobije correctRoutesData koja sadrži sve adekvatne linije, te je još sortirana konstanta prema broj stanica koju autobus mora proći da dođe do odredišta, odnosno tražene ulice. Algoritam ne uzima u obzir početak polaska autobusa i samim time to predstavlja jedan od nedostataka algoritma. Prikaz rezultata se može vidjeti na slici 51.

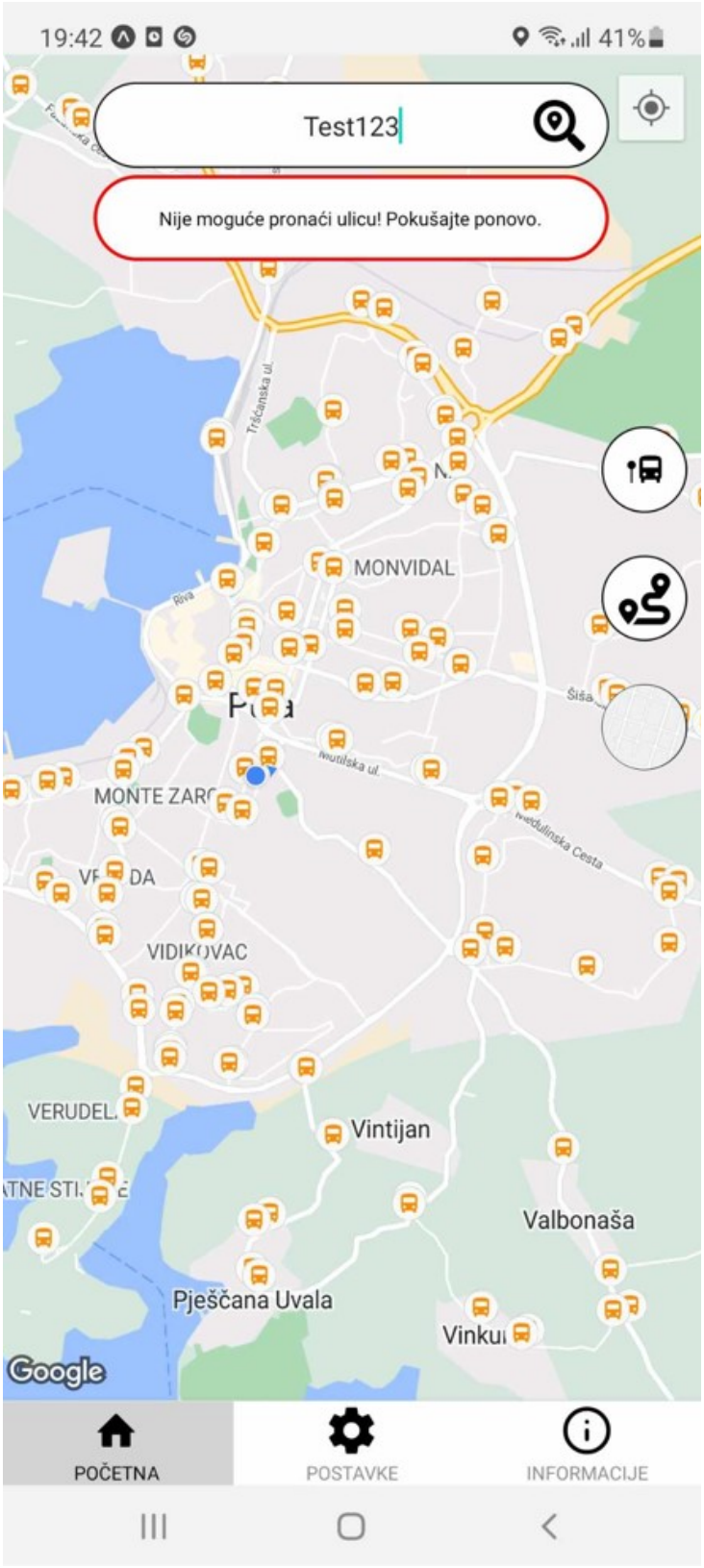
Slika 51. Prikaz prozora sa autobusnim linijama

Klikom na jednu od autobusnih linija korisnik dobiva sljedeći pregled koji se vidi na slici 52.

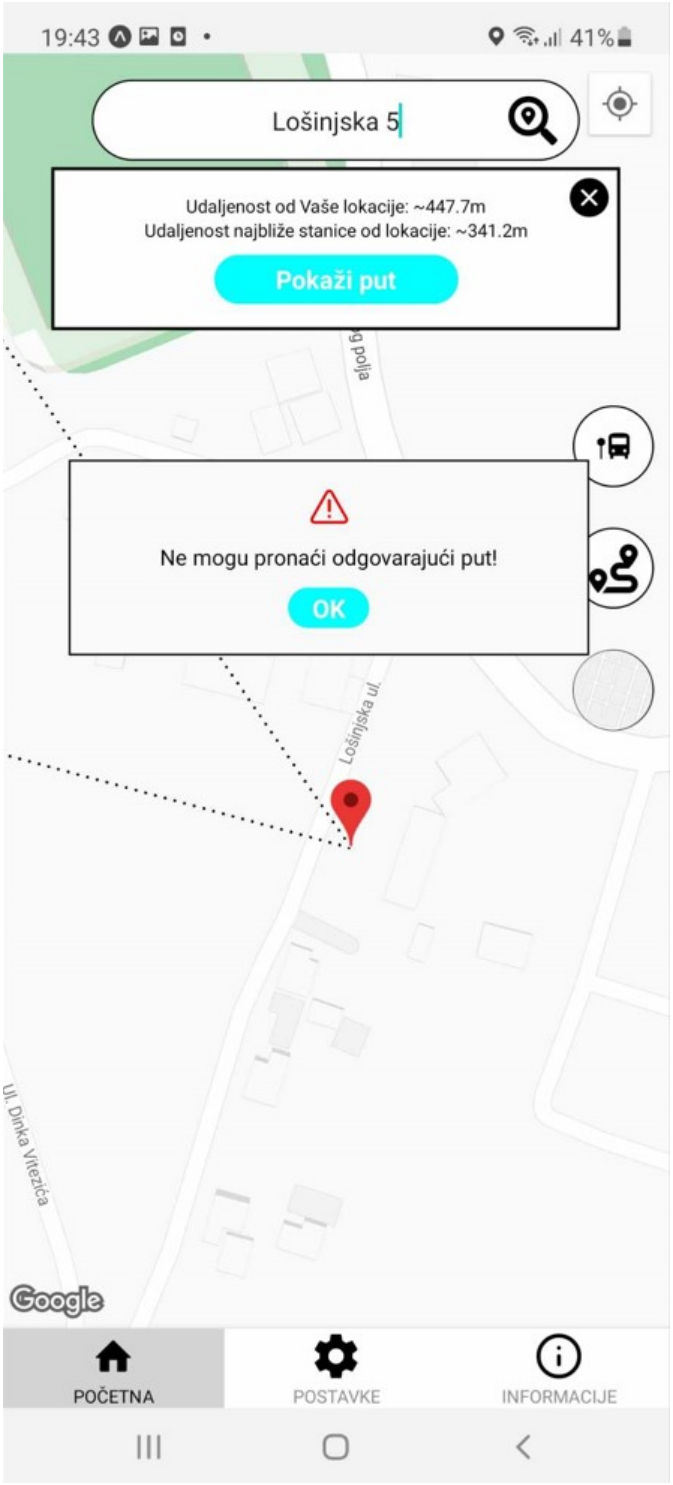


Slika 52. Prikaz autobusne linije

Ukoliko u procesu pretraživanja ulice i potencijalnog puta se dogodi nekakva greška, korisnik dobije upozorenje koje je prikazano na slikama 53 i 54.



Slika 53. Prikaz greške prilikom pretraživanja

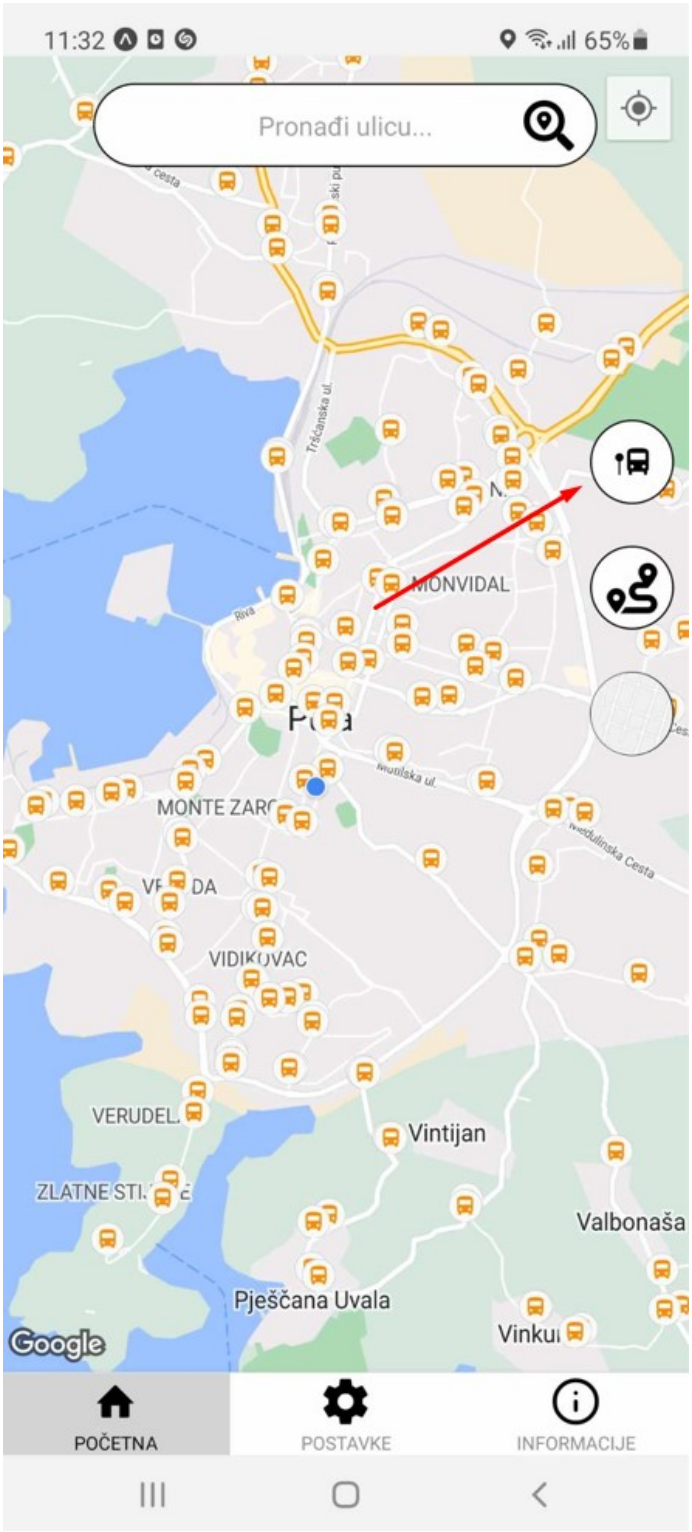


Slika 54. Prikaz greške u pronalasku odgovarajućeg puta

Slika 55 prikazuje funkciju handleClosestStop() koja pronalazi najbližu stanicu u usporedbi sa trenutnom lokacijom korisnika, te slika 56 prikazuje gumb koji izvršava tu funkciju.

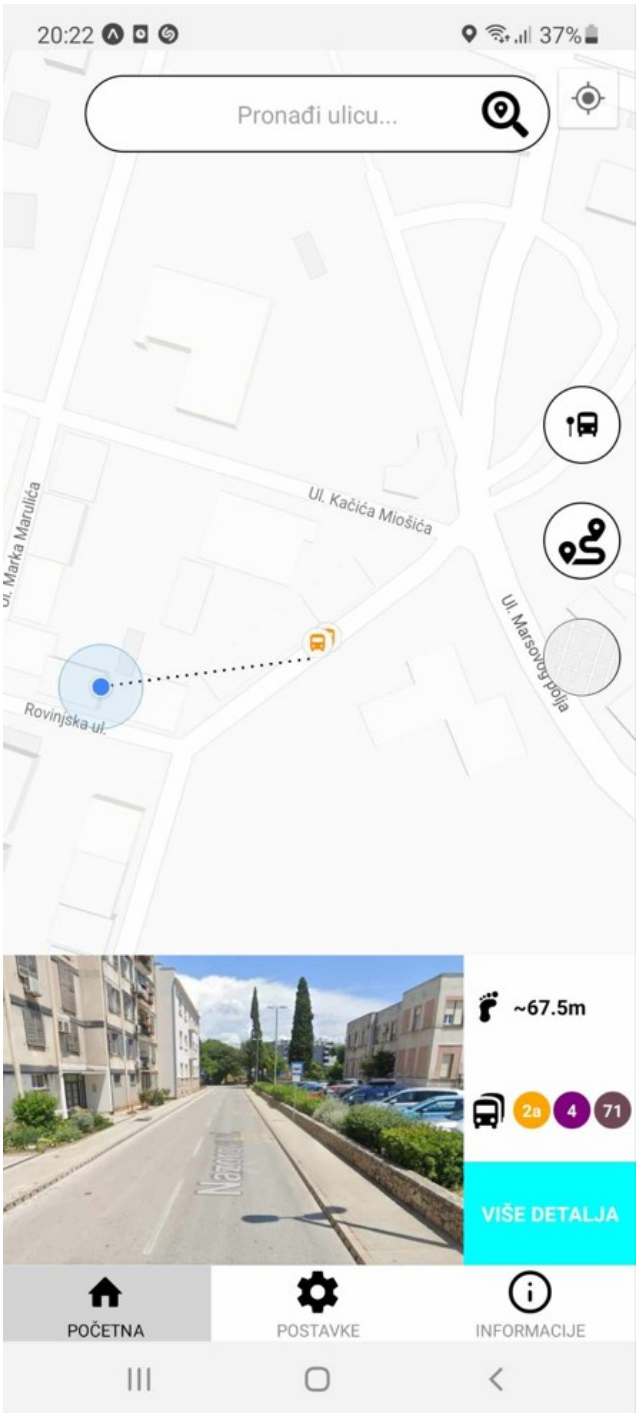
```
const handleClosestStop = () => {
  setSelectedRouteId(null);
  setShowSearchedStreet(null);
  setShowAllRoutes(false);
  if (selectedStopId !== null) {
    setSelectedStopId(null);
    return;
  }

  const startLocation = {
    latitude: location.coords.latitude,
    longitude: location.coords.longitude,
  };

  const allDistances = busStopDataInUse.map((busStop) => {
    return {
      id: busStop.id,
      distance: calculateDistance(
        startLocation.latitude,
        startLocation.longitude,
        busStop.latitude,
        busStop.longitude
      ),
      latitude: busStop.latitude,
      longitude: busStop.longitude,
    };
  });
  const smallestDistances = allDistances.filter(
    (obj) =>
      obj.distance === Math.min(...allDistances.map((obj) => obj.distance))
  );
  const region = {
    latitude: smallestDistances[0].latitude,
    longitude: smallestDistances[0].longitude,
    latitudeDelta: 0.0025,
    longitudeDelta: 0.0025,
  };

  mapRef.current?.animateToRegion(region, 500);
  setSelectedStopId(smallestDistances[0].id);
};
```

Slika 55. Prikaz kôda za pronalazak najbliže stanice korisnika



Slika 56. Prikaz gumba za pronalazak najbliže stanice

Slika 57 prikazuje pregled nakon pritiska na gumb.

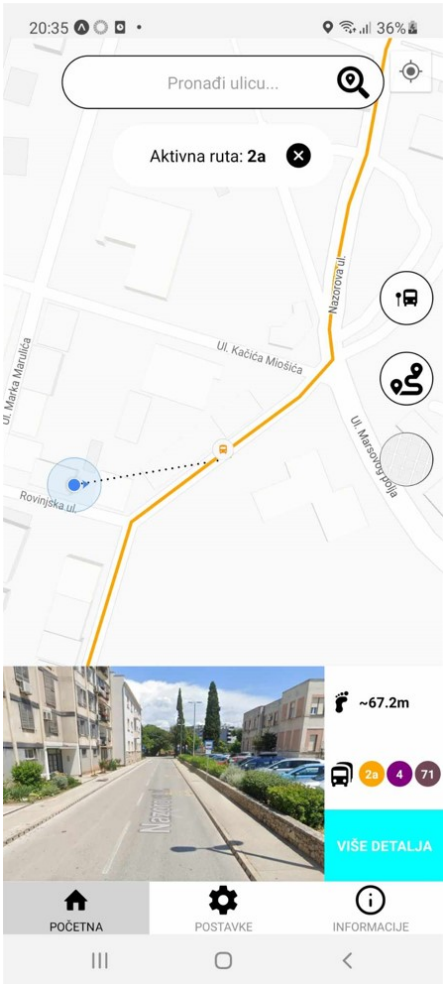


Slika 57. Prikaz najbliže stanice

Na ovom prikazu korisnik može vidjeti kako najbliža autobusna stanica izgleda, njezinu udaljenost od korisnika, njezine linije koje korisnik može kliknuti te klikom na liniju će dobiti grafički prikaz linije. Slika 58 prikazuje programski kôd, te slika 59 prikazuje izgled linije.

```
const renderRoutePath = () => {
  if (selectedRouteId === null) {
    return;
  }
  let route = routesData.find((route) => route.id === selectedRouteId);

  return (
    <Polyline
      key={route.id}
      coordinates={route.pathCoords}
      strokeWidth={3}
      strokeColor={routesToMarker === null ? route.color : "aqua"}
    />
  );
};
```

Slika 58. Prikaz kôda za iscrtavanje polilinije



Slika 59. Prikaz autobusne linije 2a

Klikom na više detalja korisniku će se otvoriti novi prozor sa informacijama o rasporedu. Slika 60, 61, 62 prikazuju programski kôd, te slika 63 prikazuje izgled.

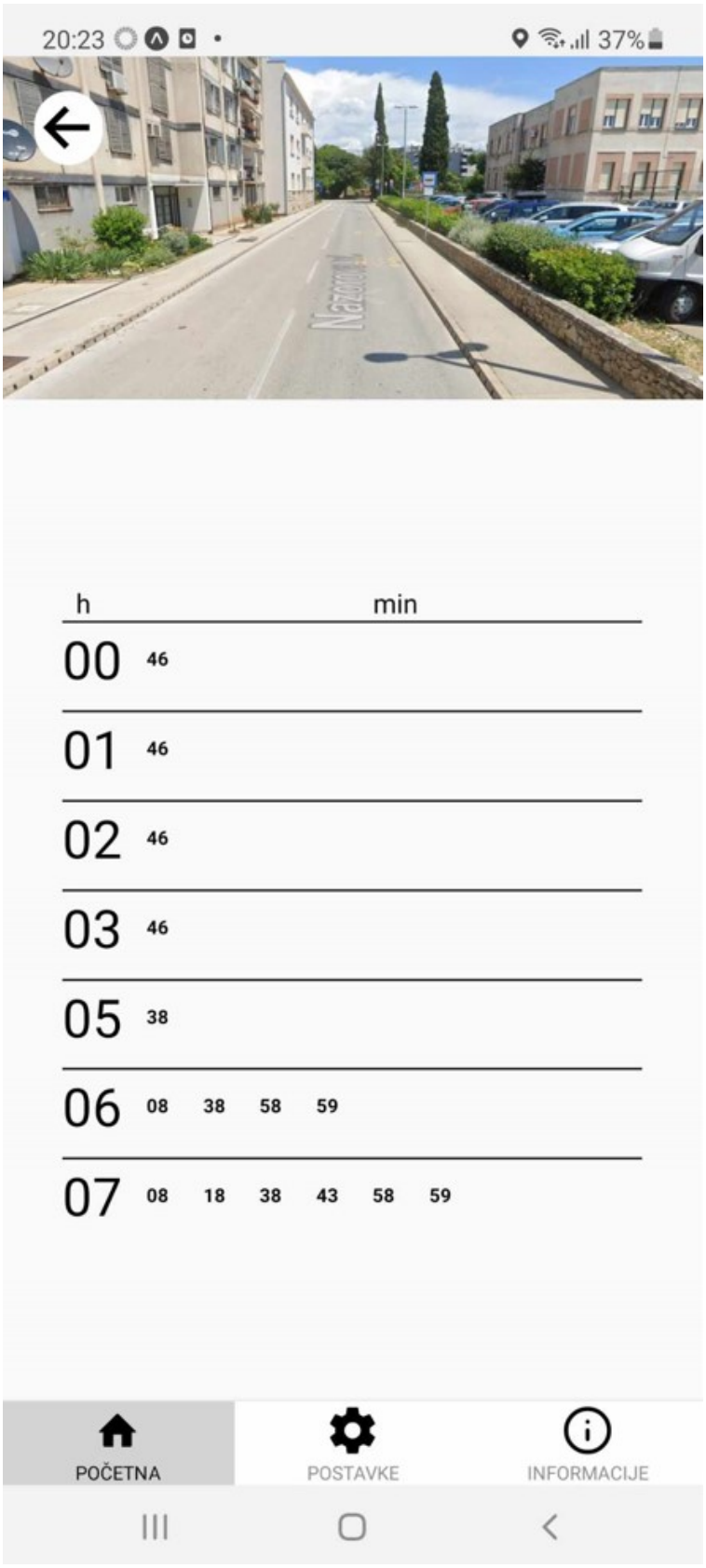
```
import { Image, ScrollView, StyleSheet, Text, View } from "react-native";
import { useSafeAreaInsets } from "react-native-safe-area-context";
import { busStopArrivalTime, busStopData } from "../bus-data/bus-stop";
import { Ionicons } from "@expo/vector-icons";

const BusStopDetails = ({ navigation, route }) => {
  const insets = useSafeAreaInsets();
  const { busStopId } = route.params;
  const currentBusStop = busStopData.find(
    (busStop) => busStop.id === busStopId
  );

  const busStopTimes = busStopArrivalTime.find(
    (busStop) => busStop.id === busStopId
  );

  if (busStopTimes === undefined) {
    return (
      <View
        style={{
          flex: 1,
          paddingTop: insets.top,
          justifyContent: "center",
          alignItems: "center",
          gap: 20,
        }}
      >
        <Text>Bus stop is not in use</Text>
        <Ionicons
          name="arrow-back"
          size={40}
          color="black"
          onPress={() => navigation.goBack()}
        />
      </View>
    );
  }

  const groupTimesByHour = (times) => {
    const groupedTimes = {};
    times.forEach((time) => {
      let hour = time.split(":")[0];

      if (hour[0] === "0") {
        hour = hour[1];
      }
      if (parseInt(hour) >= 24) {
        hour = parseInt(hour) - 24;
        hour = "0" + hour;
```

Slika 60. Prikaz kôda za prikaz redoslijeda autobusne stanice - prvi dio

```
      if (!groupedTimes[hour]) {
        groupedTimes[hour] = [];
      }

      groupedTimes[hour].push(time);
    });

    const sortedHours = Object.keys(groupedTimes).sort(
      (a, b) => parseInt(a) - parseInt(b)
    );

    const sortedGroupedTimes = {};
    sortedHours.forEach((hour) => {
      sortedGroupedTimes[hour] = groupedTimes[hour];
    });

    return sortedGroupedTimes;
  };

  const groupedTimes = groupTimesByHour(busStopTimes.times);

  return (
    <View style={{ flex: 1, paddingTop: insets.top }}>
      <View style={{ width: "100%", height: 200 }}>
        <Ionicons
          name="arrow-back"
          size={40}
          color="black"
          style={styles.iconBack}
          onPress={() => navigation.goBack()}
        />
        <Image source={currentBusStop?.image} style={styles.imageStyle} />
      </View>
      <View style={{ flex: 1 }}>
        <View style={styles.bottomContent}>
          <View style={styles.timeDiv}>
            <ScrollView contentContainerStyle={styles.container}>
              <View style={{ flexDirection: "row", width: "100%" }}>
                <View
                  style={{
                    width: 25,
                    justifyContent: "center",
                    alignItems: "center",
                  }}
                >
                  <Text>h</Text>
```

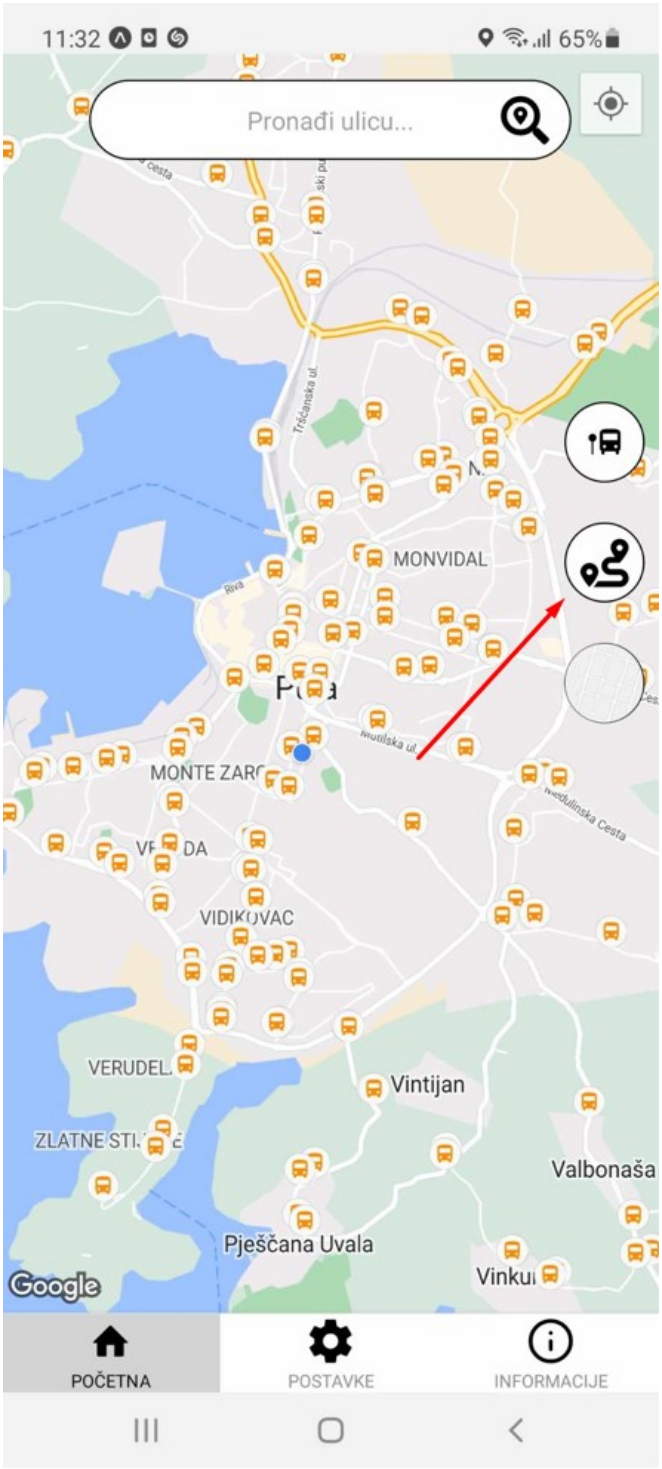
Slika 61. Prikaz kôda za prikaz redoslijeda autobusne stanice - drugi dio

Slika 62. Prikaz kôda za prikaz redoslijeda autobusne stanice – treći dio



Slika 63. Prikaz rasporeda autobusne stanice

Korisnik prilikom pritiska gumba koji je prikazan na slici 64 može vidjeti prozor sa svim autobusnim linija. Pritiskom na neku liniju aplikacija grafički ocrta liniju na karti. Na slici 65 i 66 je prikazan programski kôd te na slici 67 je prikazan izgled modalnog prozora nakon klika.



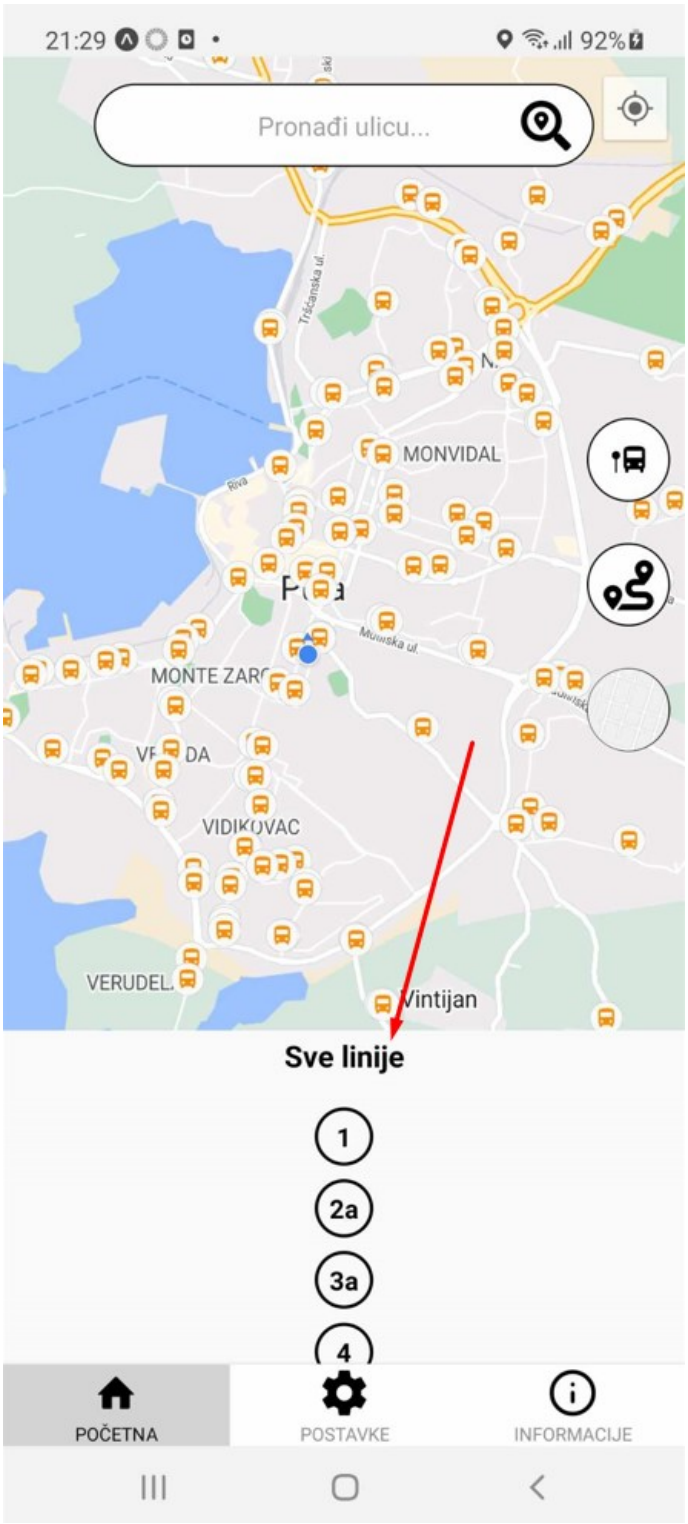
Slika 64. Prikaz gumba svih autobusnih linija

```
const renderAllRoutesModal = () => {
  if (showAllRoutes === false) {
    return;
  }

  let routesForRender = routesData;

  if (routesToMarker) {
    const tempArray = routesToMarker.map((route) => route.id);
    routesForRender = routesData
      .filter((route) =>
        routesToMarker.some((markerRoute) => markerRoute.id === route.id)
      )
      .sort((a, b) => {
        const indexA = tempArray.indexOf(a.id);
        const indexB = tempArray.indexOf(b.id);
        return indexA - indexB;
      });

    return (
      <View style={styles.renderAllRoutesModal}>
        <View
          style={{
            flexDirection: "row",
            marginTop: 5,
            gap: 50,
          }}
        ></View>
        <View style={{ flexDirection: "row", gap: 20, marginTop: 10 }}>
          <Text style={{ fontWeight: 700 }}>{currentLang.busRoute}</Text>
          <Text style={{ fontWeight: 700 }}>{currentLang.numStops}</Text>
        </View>
        <ScrollView
          contentContainerStyle={styles.renderAllRoutesViewContainer}
        >
          {routesForRender.map((route) => (
            <View key={route.id} style={styles.renderAllRoutesSingleRoute}>
              <Pressable
                style={styles.pressableSingleRoute}
                onPress={() => {
                  setShowSearchedStreet(null);
                  setSelectedRouteId(route.id);
                }}
              >
                <Text style={styles.renderAllRoutesSingleRouteName}>
                  {route.name}
                </Text>
```

Slika 65. Prikaz kôda za prikaz modalnog prozora – prvi dio

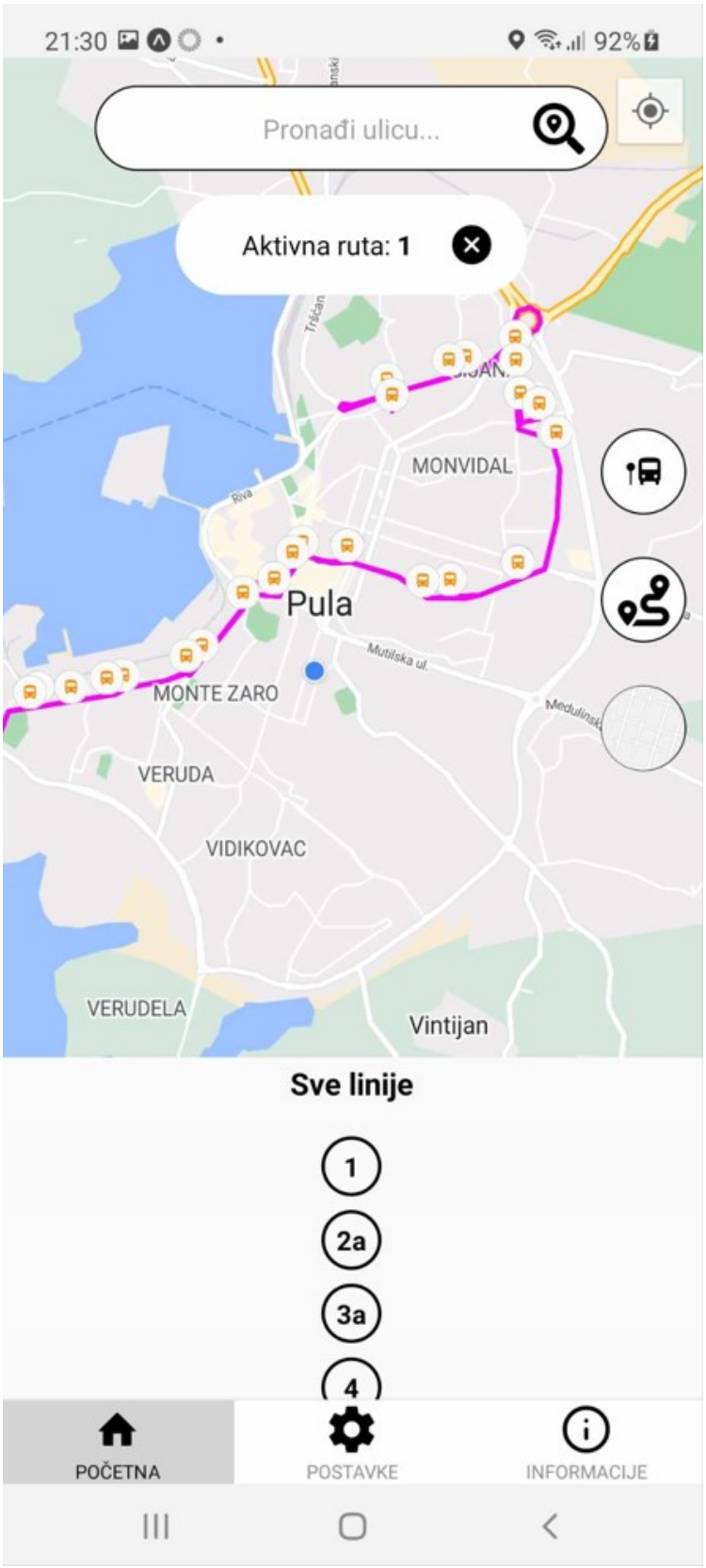
```
                <Text style={{ fontWeight: 700, fontSize: 16 }}>
                  {
                    routesToMarker.find((element) => element.id === route.id)
                      .numOfStops
                  }
                </Text>
              </View>
            ))}
          </ScrollView>
        </View>
      );
    }

    return (
      <View style={styles.renderAllRoutesModal}>
        <View style={styles.renderAllRoutesModalTitleView}>
          <Text style={styles.renderAllRoutesModalTitleText}>
            {currentLang.allRoutes}
          </Text>
        </View>
        <ScrollView contentContainerStyle={styles.renderAllRoutesViewContainer}>
          {routesForRender.map((route) => (
            <View
              key={route.id}
              style={[
                styles.renderAllRoutesSingleRoute,
                { justifyContent: "center" },
              ]}
            >
              <Pressable
                style={styles.pressableSingleRoute}
                onPress={() => {
                  setShowSearchedStreet(null);
                  setSelectedRouteId(route.id);
                }}
              >
                <Text style={styles.renderAllRoutesSingleRouteName}>
                  {route.name}
                </Text>
              </Pressable>
            </View>
          ))}
        </ScrollView>
      </View>
    );
  };
```

Slika 66. Prikaz kôda za prikaz modalnog prozora – drugi dio

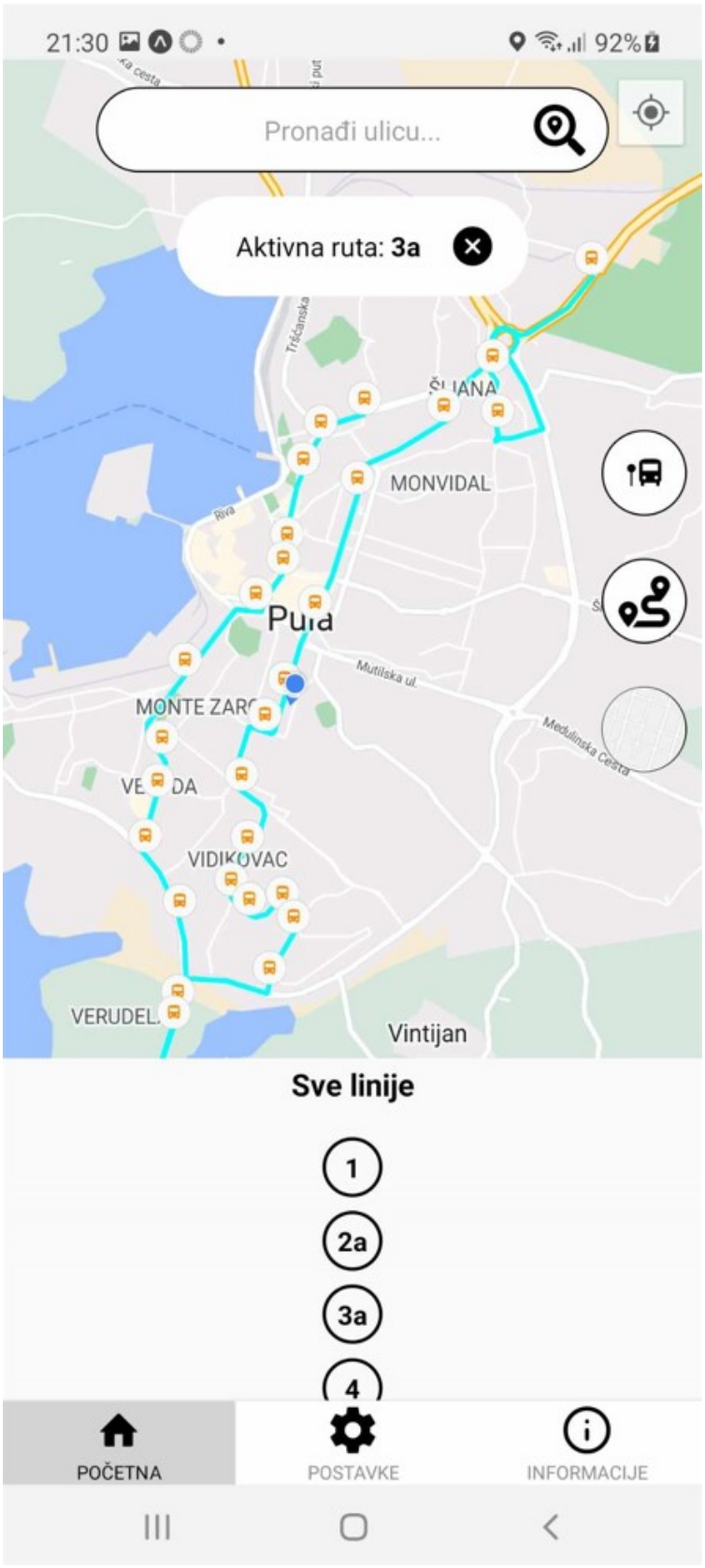


Slika 67. Prikaz svih linija

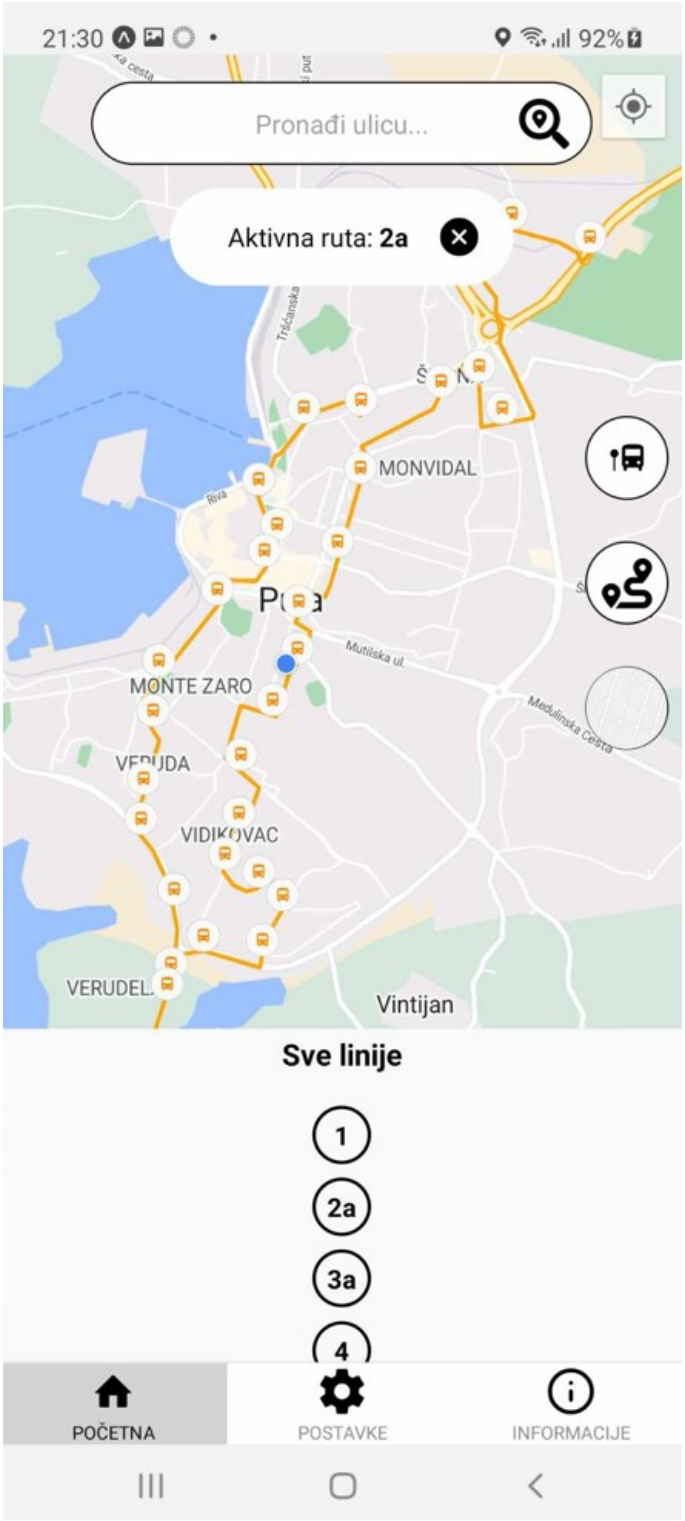
Korisnik može listati kroz popis linija te prilikom klika na liniju dobiva rezultate koji su prikazani na slici 68, 69 i 70.



Slika 68. Prikaz autobusne linije 1

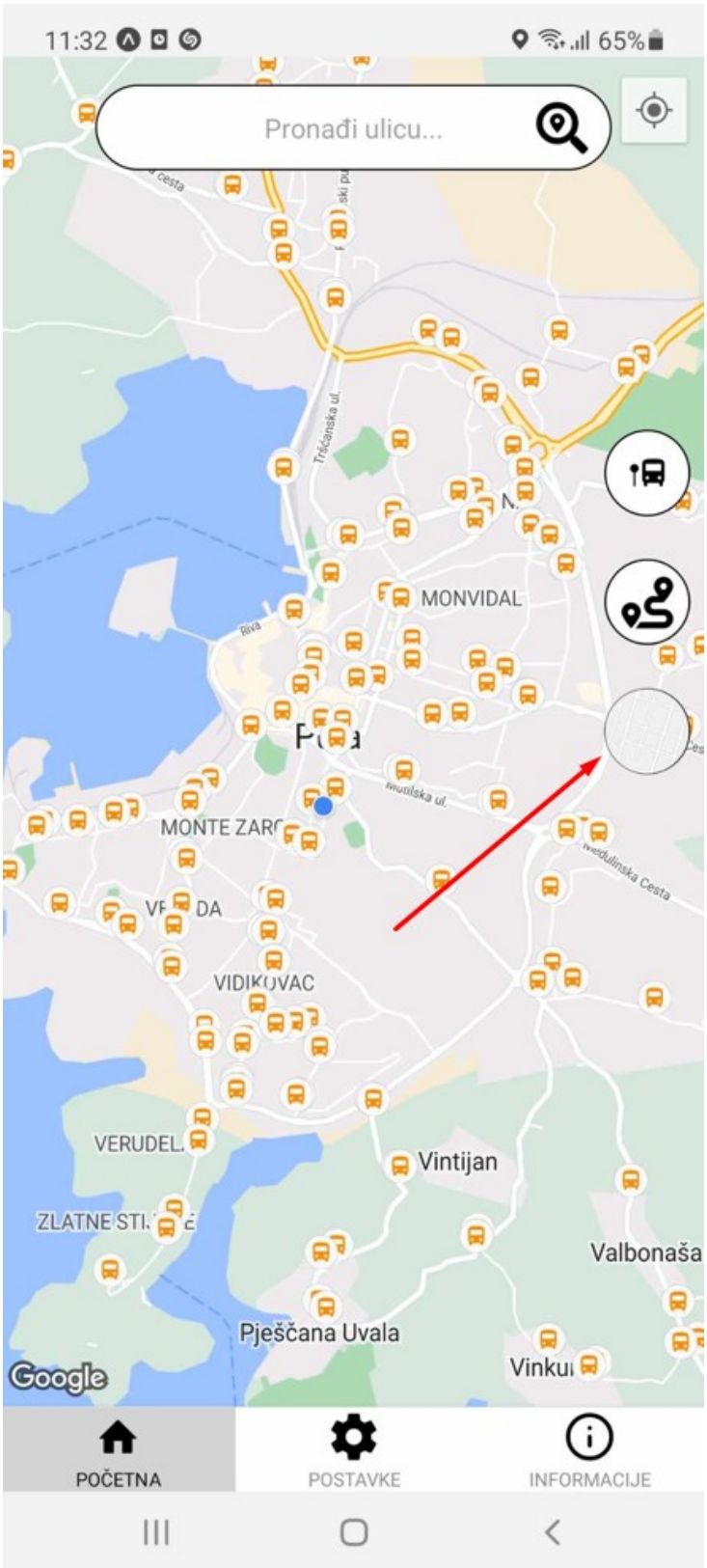


Slika 69. Prikaz autobusne linije 3a

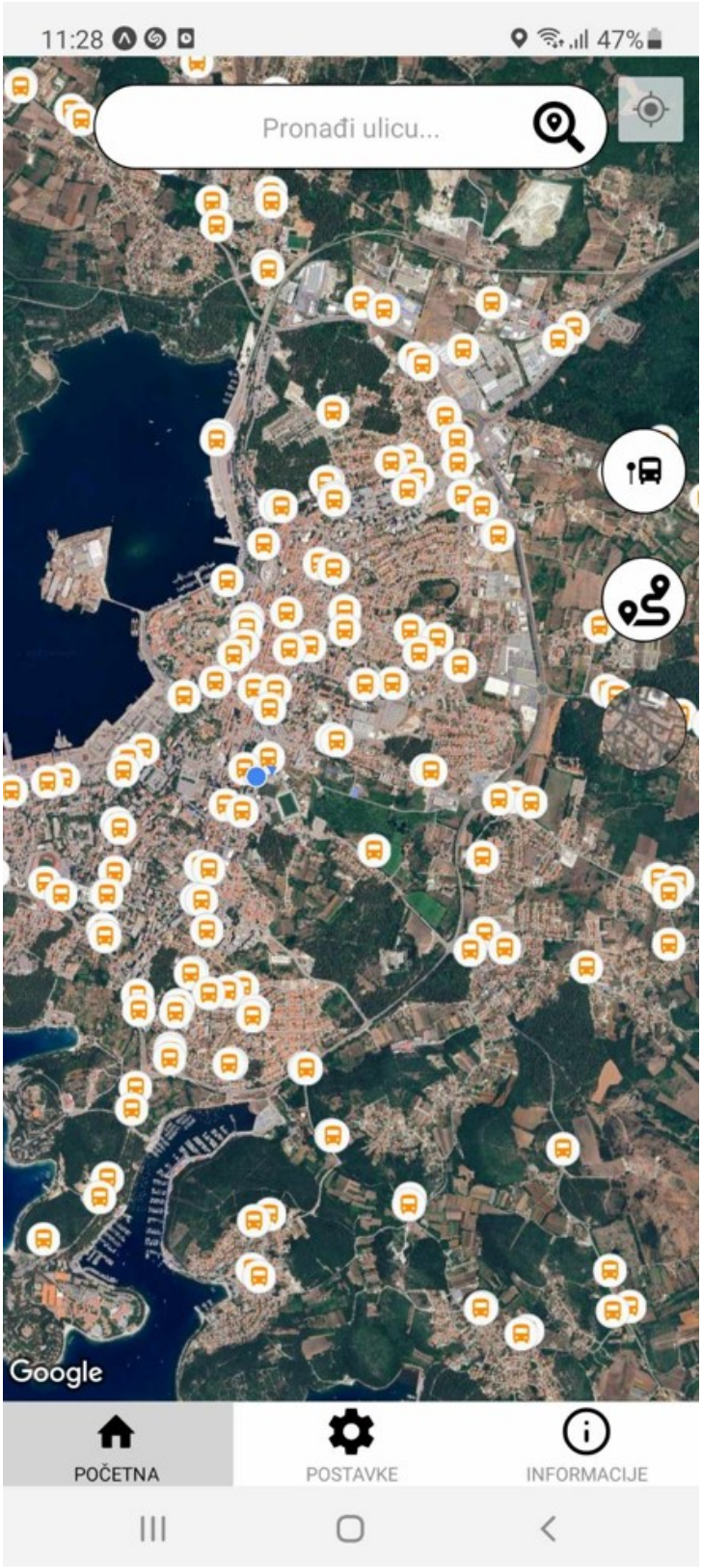


Slika 70. Prikaz autobusne linije 2a

Korisnik može promjeniti izgled karte ukoliko klikne na gumb koji je prikazan na slici 71. Promjena je prikazana na slici 72.

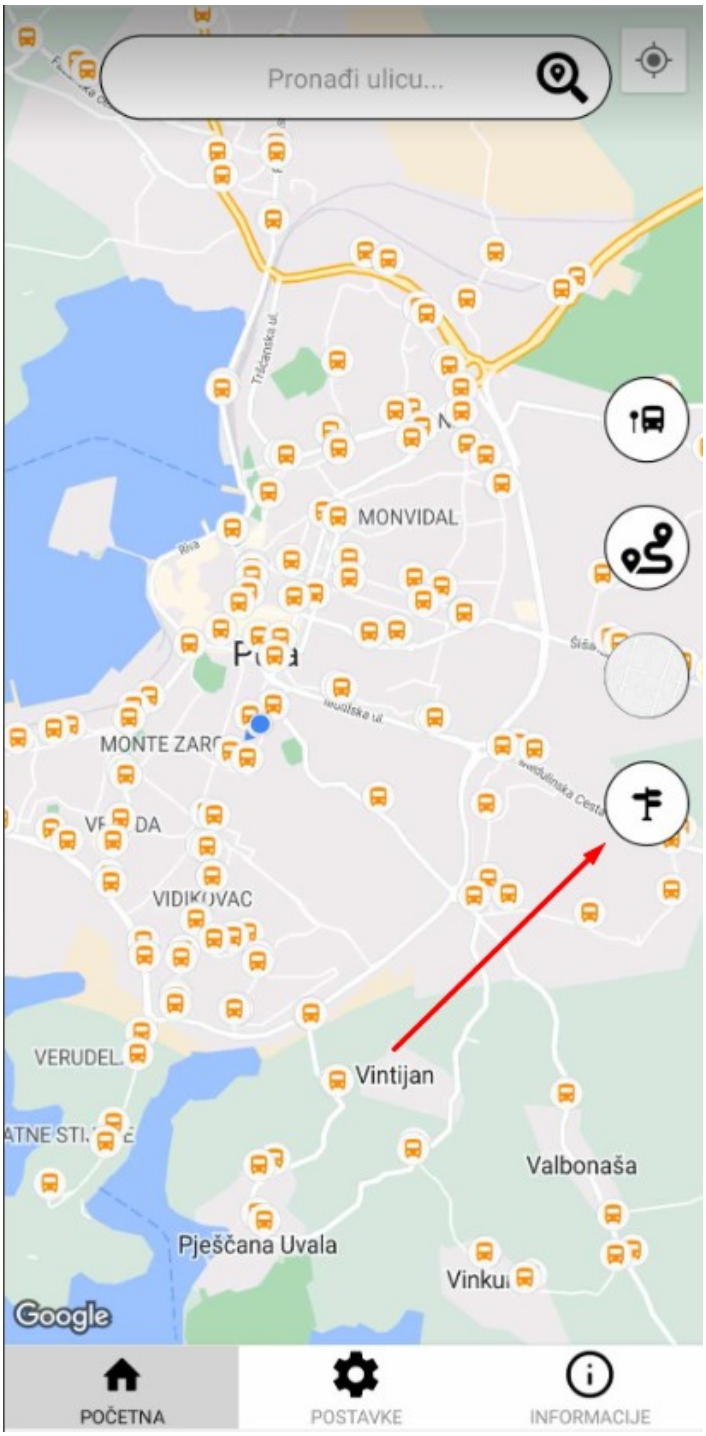


Slika 71. Prikaz gumba za promjenu izgleda karte



Slika 72. Pregled karte nakon promjene izgleda

Korisnik putem sljedeće tipke može planirati svoj put prema nekome odredištu, slika u nastavku prikazuje gumb:



Slika 73. Prikaz gumba za planiranje svog puta

Nakon pritiska na gumb korisnik dobiva sljedeći prozor koji je prikazan na slici 74 i 75.

Slika 74. Prikaz modalnog prozora

Slika 75. Prikaz upisanih vrijednosti

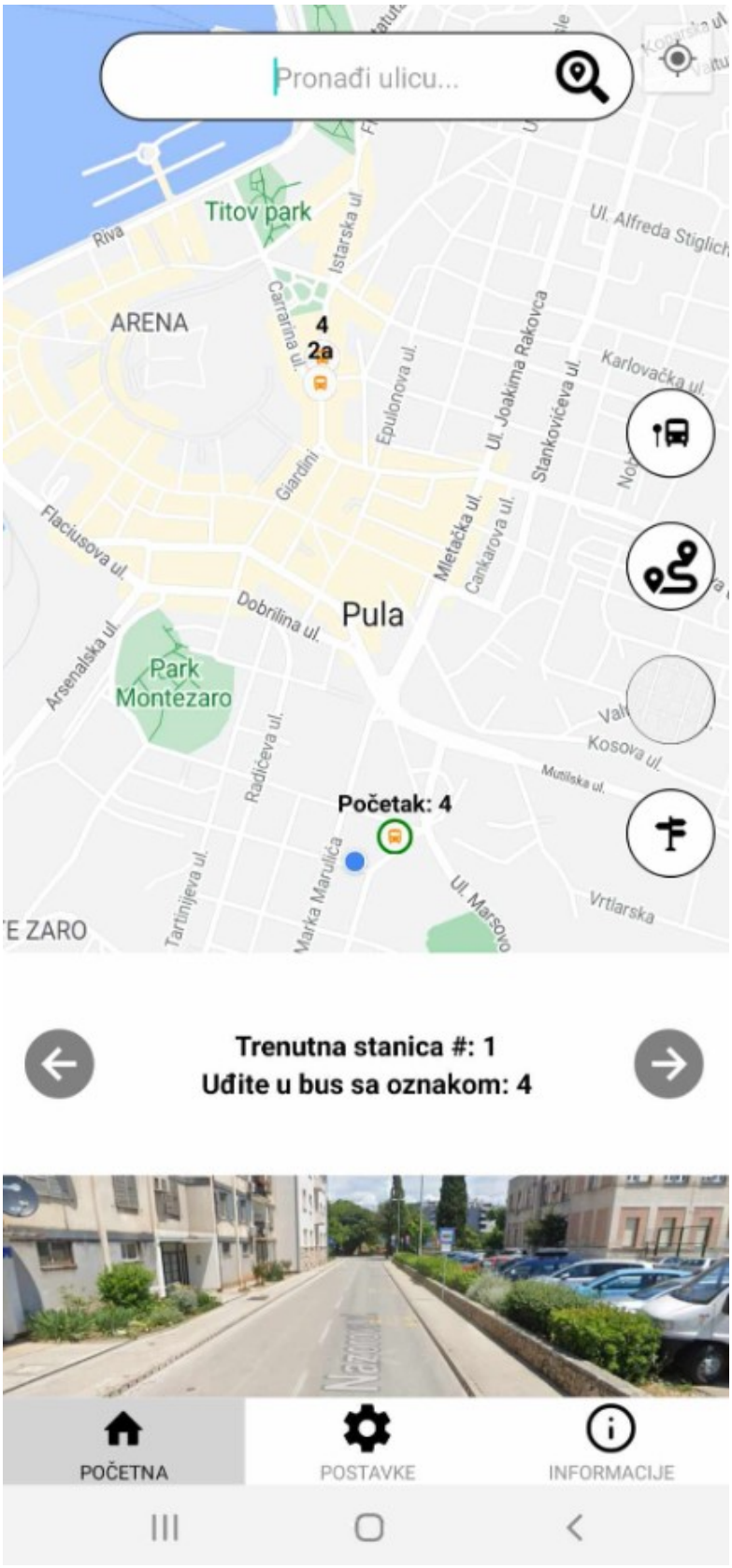
Korisnik može dodavati koliko ulica želi te će program pokušati pronaći adekvatan put od svake do svake ulice te vratiti rezultat. Ako ne pronađe adekvatan rezultat, korisnik će biti obaviješten. Funkcija djeluje slično kao i kod pretraživanja ulica i traženja puta, samo u ovome slučaju pretražuje od točke A do točke B, dok kod pretraživanja ulica imamo od lokacije korisnika do pretražene ulice. Algoritam će za svaki par ulica naći određenu liniju te će na kraju spojiti sve podatke. Za primjer korisnik upiše 4 ulice. Algoritam uzima prvu i drugu i pronalazi odgovarajuću liniju, pa uzima drugu i treću i isto napravi i na kraju uzima treću i četvrtu te spoji sve podatke i prikažemo. Za svaki par algoritam radi na sljedeći način: u varijable pointAStops i pointBstops se spremaju sve autobusne stanice unutar 250 metara od točke A i točke B. Nakon toga provjerava se da li postoje linije koje prolaze kroz stanice blizu točke A i blizu točke B. Sve potencijalne linije koje ispunjavaju uvjet se spremaju u zasebnu varijablu. Nakon toga za svaku moguću liniju se provjerava najbliža stanica za takvu liniju u odnosu na točku A i u odnosu na točku B. Za kraj se provjerava redoslijed stanica odnosno početna stanica mora biti prije završne kako bi aplikacija prikazala pravilne podatke. Ako ne bi bilo provjere, aplikacija bi izbacila stanice koje bi kršile pravila autobusne linije odnosno korisnik bi imao kontraproduktivne podatke. Ograničenje algoritma je to da algoritam ne računa vremena polaska autobusa i ne uzima ih u obzir, već uzima u obzir samo ako linije postoje. Slika 75 prikazuje upis vrijednosti, slike 76 i 77 programski kôd i slike 78, 79 i 80 prikazuju rezultat.

```
const renderRouteModal = () => {
  if (
    routePlannerData === null ||
    routePlannerData === "ErrorLocation" ||
    routePlannerData === "ErrorRoute"
  ) {
    return;
  }

  let allBusStopsOnPath = [];
  routePlannerData.forEach((path) => {
    if (!allBusStopsOnPath.includes(path.startStopID)) {
      allBusStopsOnPath.push(path.startStopID);
    }
  });

  const routeID = routePlannerData[routeStopNumber].id;
  const routeName = routesData.find((route) => route.id === routeID).name;
  let currentStopID = [allBusStopsOnPath[routeStopNumber]];
  let currentStop = busStopDataInUse.find(
    (busStop) => busStop.id === currentStopID[0]
  );

  return (
    <View style={styles.routePlanResultModal}>
      <View style={{ flex: 1, flexDirection: "row", alignItems: "center" }}>
        <Pressable
          style={{
            flex: 0.25,
            justifyContent: "center",
            alignItems: "center",
          }}
          onPress={() => {
            if (routeStopNumber - 1 >= 0) {
              setRouteStopNumber(routeStopNumber - 1);
            }
          }}
        >
          <Ionicons name="arrow-back-circle-sharp" size={48} color="grey" />
        </Pressable>
```

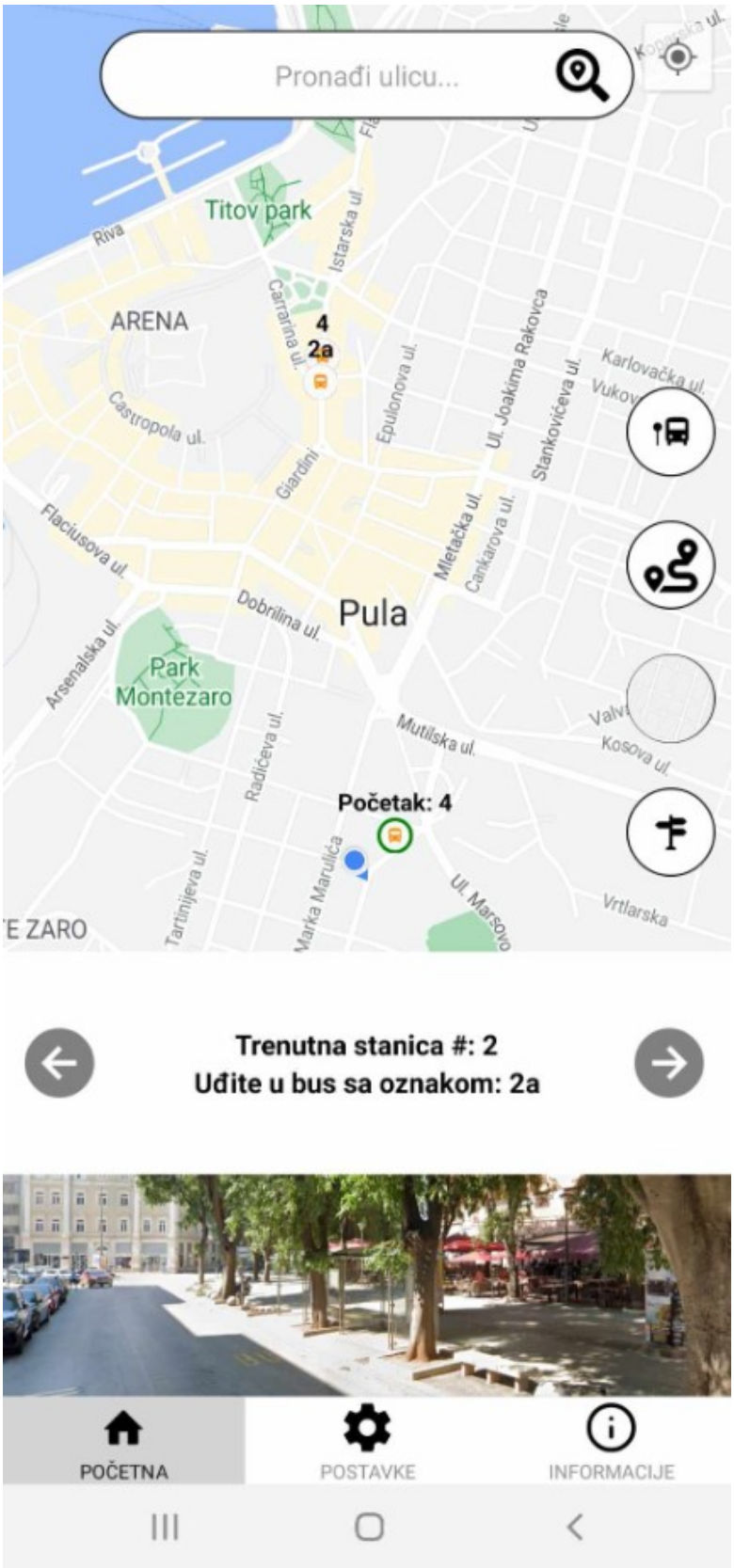
Slika 76. Prikaz kôda – prvi dio

```
            <View
              style={{ flex: 1, justifyContent: "center", alignItems: "center" }}
            >
              <Text style={{ fontWeight: 700 }}>
                {currentLang.curStation} {routeStopNumber + 1}
              </Text>
              <Text style={{ fontWeight: 700 }}>
                {currentLang.enter} {routeName}
              </Text>
            </View>
            <Pressable
              style={{
                flex: 0.25,
                justifyContent: "center",
                alignItems: "center",
              }}
            >
              <Ionicons
                name="arrow-forward-circle-sharp"
                size={48}
                color="grey"
                onPress={() => {
                  if (routeStopNumber + 2 <= routePlannerData.length) {
                    setRouteStopNumber(routeStopNumber + 1);
                  }
                }}
              />
            </Pressable>
          </View>
          <View style={{ flex: 1 }}>
            <Image
              source={currentStop.image}
              style={{ width: "100%", flex: 1 }}
            />
          </View>
        </View>
      );
    };
```

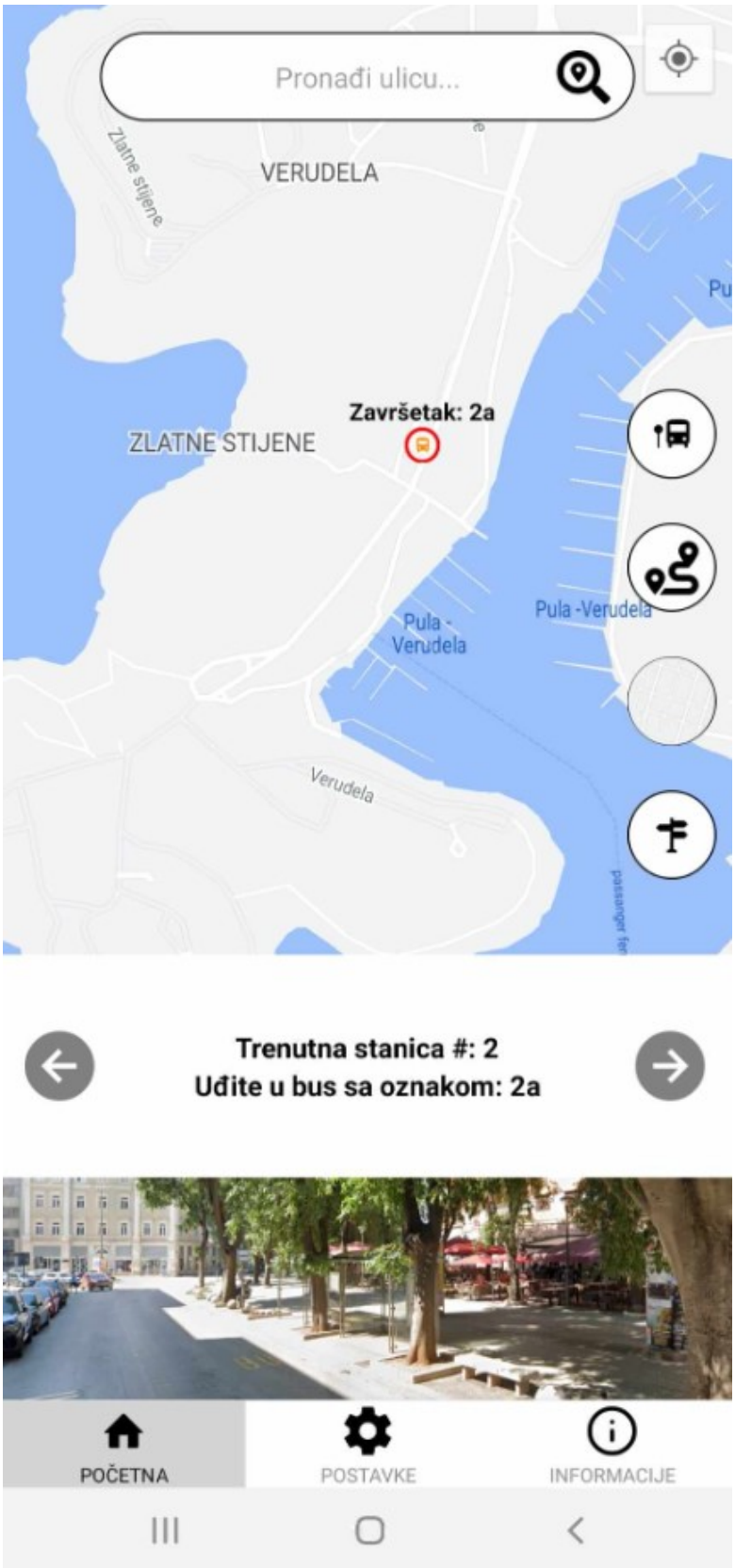
Slika 77. Prikaz kôda – drugi dio



Slika 78. Prikaz rezultata – prvi dio



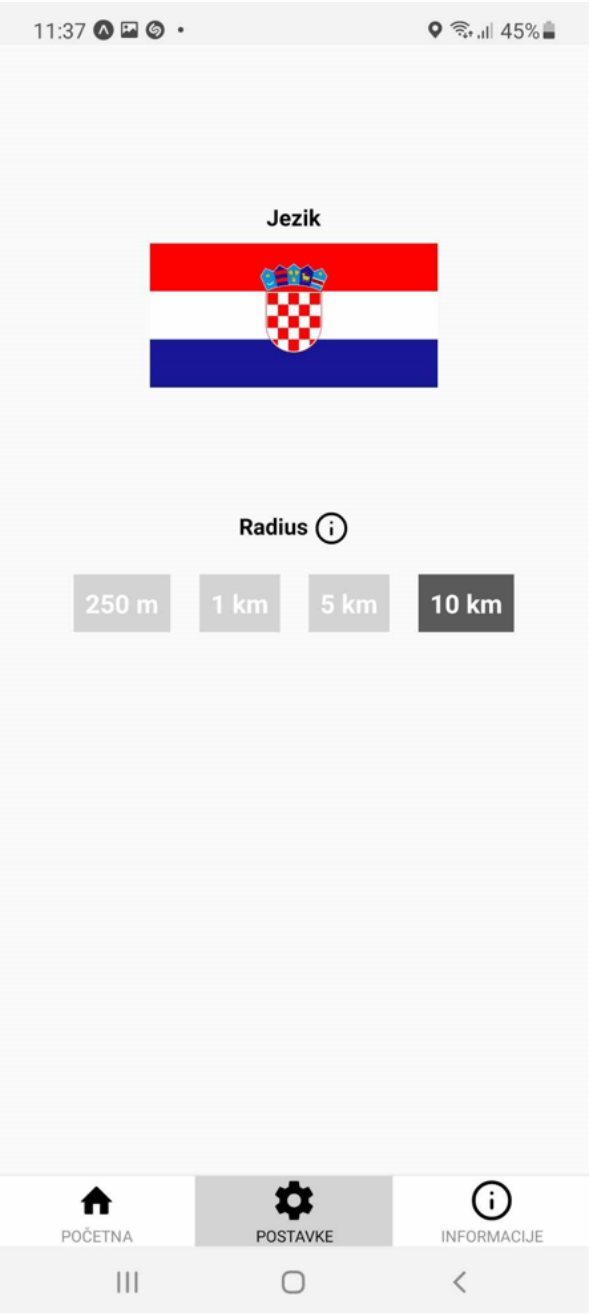
Slika 79. Prikaz rezultata – drugi dio



Slika 80. Prikaz rezultata – treći dio

5.3.2 Kartica - „POSTAVKE”

Kartica „POSTAVKE'' ima funkcionalnosti promjene jezika i radijusa autobusnih stanica. Slika 81 prikazuje izgled kartice.

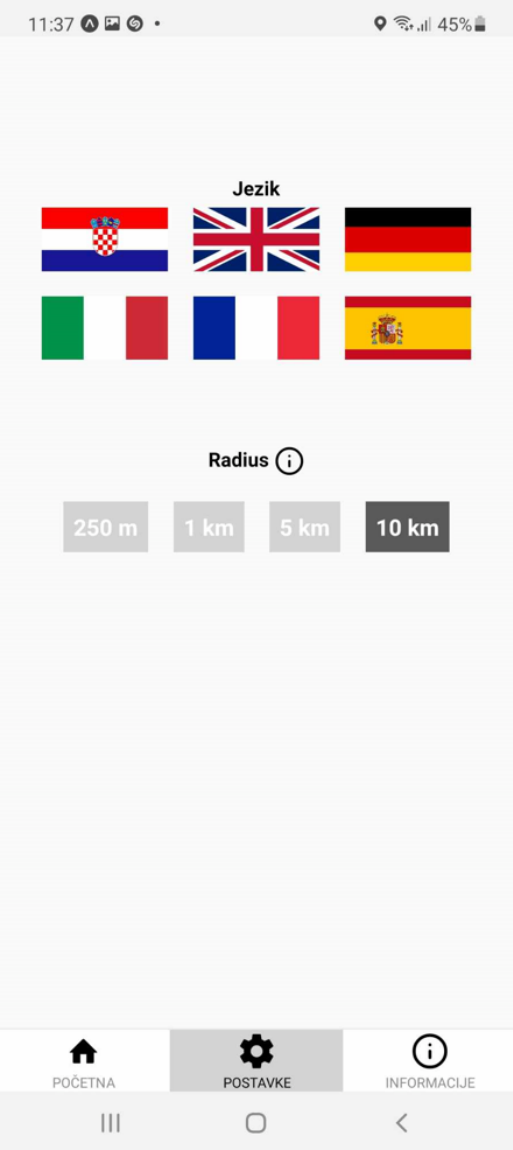


Slika 81. Prikaz kartice „POSTAVKE''

Kartica „POSTAVKE'' sadrži sljedeće funkcionalnosti:

- promjenu jezika
- promjenu radijusa

Promjena jezika označava promjenu koje se odražava na useContext, te samim time korisnik promjenom useContext-a mijenja i dijelove aplikacije što se tiče jezika. Klikom na zastavu imam sljedeći prikaz koji se vidi na slici 82, te programski kôd se vidi na slici 83.

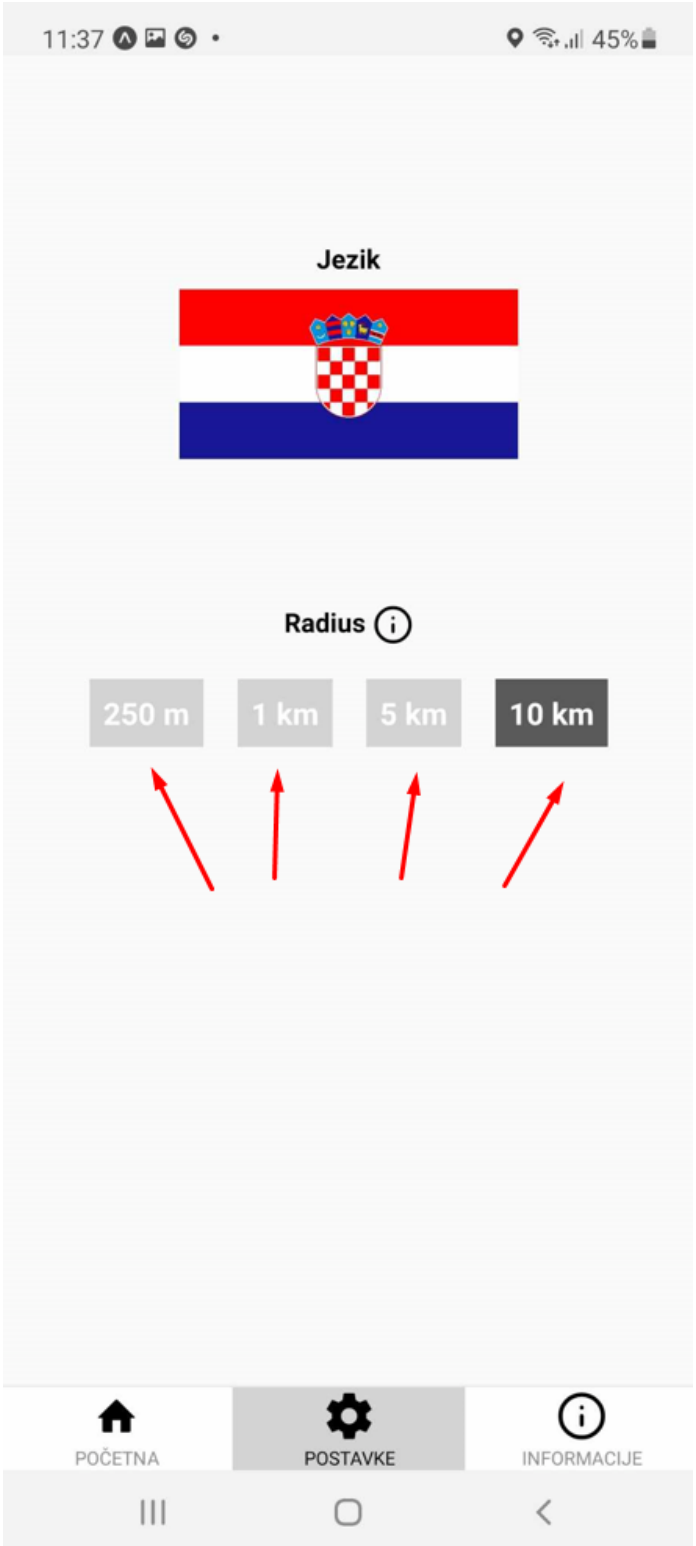


Slika 82. Prikaz kartice nakon klika na zastavu

```
{langModal === true ? (
  <View style={settingsStyle.langModal}>
    {langs.map((lang) => {
      return (
        <Pressable
          key={lang.name}
          style={{ width: 100, height: 50 }}
          onPress={() => {
            userCtx.setData({
              data: { lang: lang.name, radius: userDataRadius },
            });
            setLangModal(false);
          }}
        >
          <Image
            source={lang.image}
            style={{ height: "100%", width: "100%" }}
          />
        </Pressable>
      );
    })}
  </View>
) : null}
```

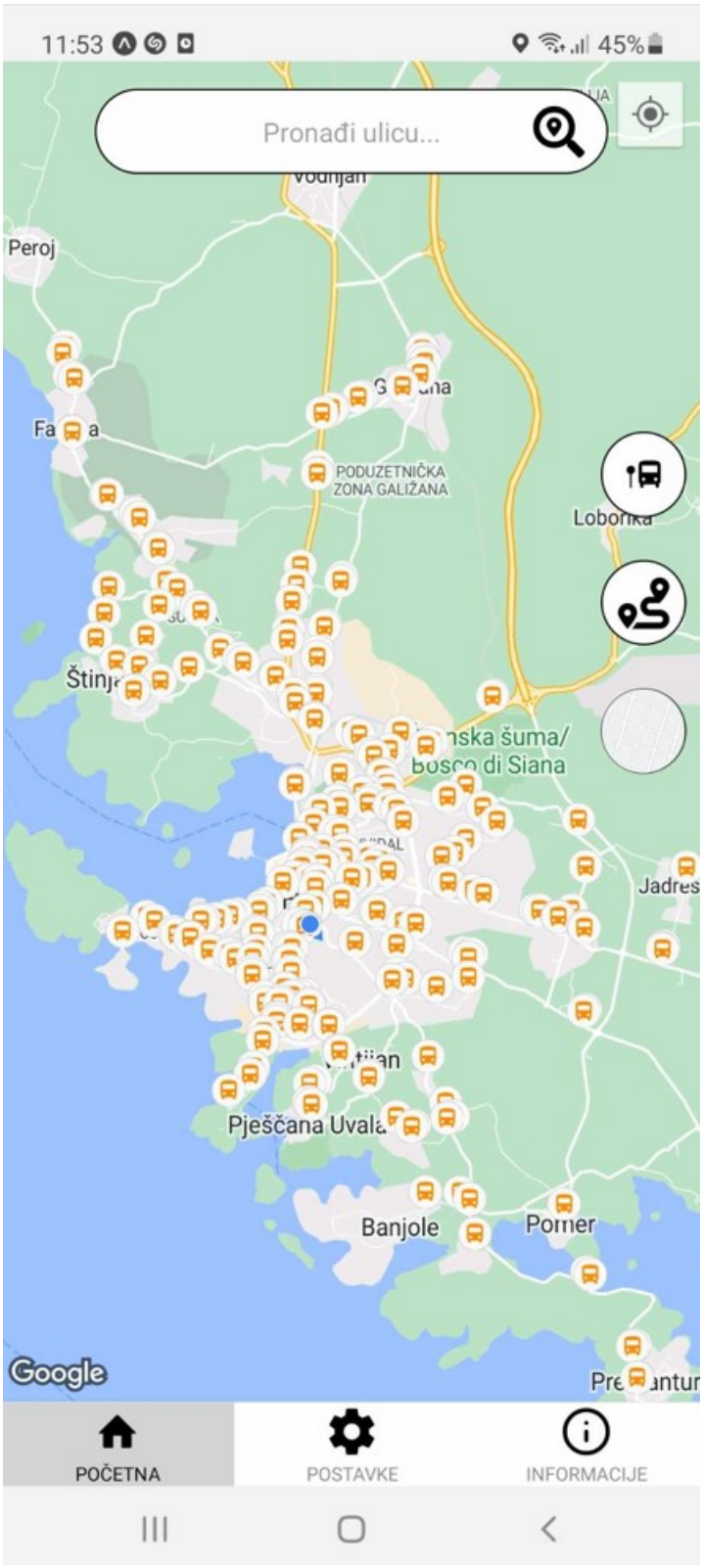
Slika 83. Prikaz kôda modalnog prozora za izmjenu jezika

Promjena radijusa označava u kojem radijusu će se prikazivati autobusne stanice u odnosu na korisnika. Npr. ukoliko korisnik odabere 1 kilometar (1 KM), samo stanice koje su najviše udaljene 1 kilometar od korisnika će biti prikazane. Promjena radijusa dobiva se klikom na jedan od ovih gumbova koji su prikazani na slici 84.

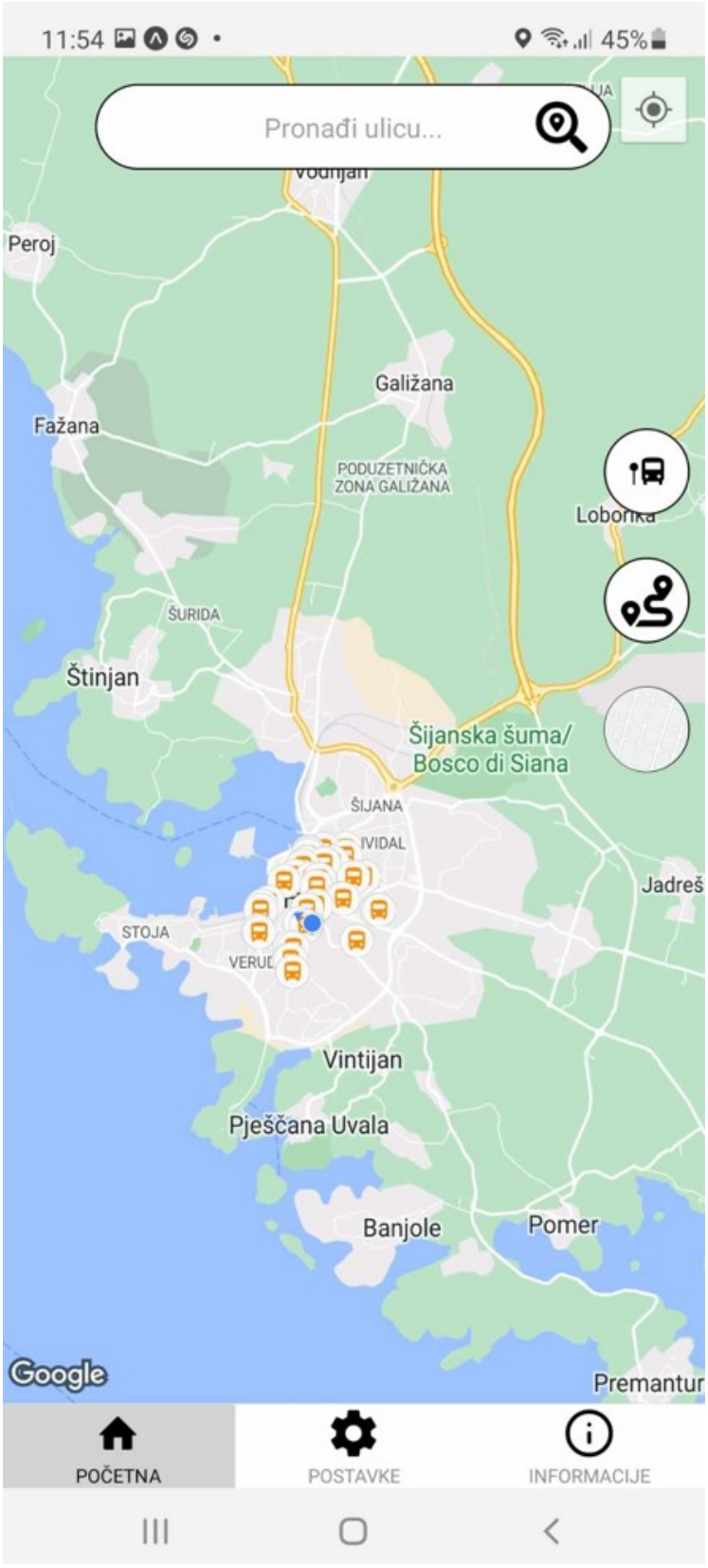


Slika 84. Prikaz gumbova za promjenu radijusa

Na slikama u nastavku prikazane su razlike u radijusu.



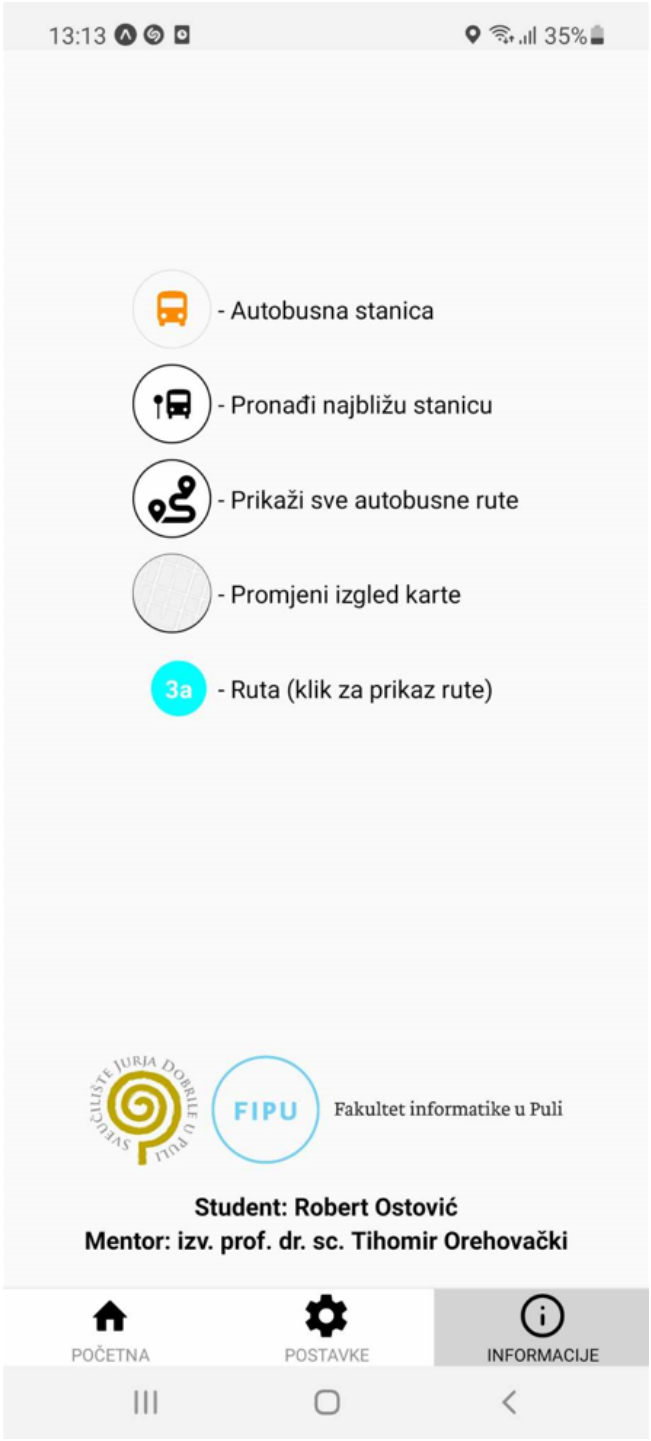
Slika 85. Prikaz radijusa – 10 kilometara



Slika 86. Prikaz radijusa – 1 kilometar

5.3.3 Kartica - „INFORMACIJE”

Kartica „INFORMACIJE’’ ne sadrži nikakve funkcionalnosti, već postoji da pojasni neke funkcionalnosti aplikacije. Izgled kartice se vidi na slici 87.



Slika 87. Prikaz kartice „INFORMACIJE’’

6. Zaključak

U zaključku ovog diplomskog rada, duboko se naglašava ključna uloga mobilnih aplikacija u suvremenom društvu, a posebno njihovu neospornu doprinosu unapređenju javnog prijevoza. Ovaj diplomski rad prezentira razvoj mobilne aplikacije koja se profilira kao informativni centar za autobusni prijevoz u gradu Puli. Upravo kroz ovu aplikaciju korisnicima se pruža mogućnost da na jednom mjestu pristupe širokom spektru informacija, uključujući pretragu ulica, pregled autobusnih stanica, raspored dolazaka autobusa te cjelokupnu mrežu gradskih linija. No, inovativnost proizlazi iz integracije interaktivne mape unutar same aplikacije. Kroz ovu naprednu funkcionalnost, korisnicima je omogućeno vizualno shvaćanje autobusnih linija i stanica, što dodatno potiče korisničku interakciju i efikasno planiranje putovanja. Rezultat toga su brže, bolje informirane odluke koje donose smanjenje neizvjesnosti i uštedu vremena na čekanju. S dubokom preciznošću pružene informacije o stanicama i vremenima dolazaka pridonose personaliziranom doživljaju svakog korisnika. Prednosti aplikacije su korištenje na više jezika, interakcija između korisnika i aplikacije, jednostavno korištenje i moderan dizajn. Ograničenja aplikacije su ograničenost na grad Pulu što se tiče pretraživanje ulica, postojanje samo gradskih linija i ograničenost algoritama u nekim segmentima što se tiče pronalaska najboljeg puta. Nove nadogradnje i funkcionalnosti koje bi se mogle implementirati su dodavanje prigradskih linija, proširenje pretraživanja ulica i na okolna mjesta (npr. Loborika), kupnja autobusnih karata pomoću aplikacije, izrada QR kôda karata, dodavanje i prikazivanje lokacije autobusa. Upotreba modernih tehnologija poput JavaScript-a, React Native-a i NodeJS-a doveli su do kreiranja aplikacije s atraktivnim dizajnom i intuitivnim korisničkim sučeljem. Nezanemariv doprinos je i fleksibilnost koju omogućava odabir između šest različitih jezika, čime se aplikacija prilagođava korisnicima i olakšava im interakciju s njom. U samoj srži diplomskog rada, svjedočimo detaljnom procesu izrade aplikacije – od konverzije podataka iz Pula Prometa u funkcionalno sučelje pa sve do implementacije složenih funkcionalnosti, predočene putem kôda i vizualnih prikaza na ekranu.

SAŽETAK

Diplomski rad sastoji se od uvoda, kritične analize postojećih rješenja, tehnologija, transformacija podataka i opisa funkcionalnosti aplikacije. U uvodu je napravljen kratak osvrt na današnje tehnologije i potrebe za izradom aplikacije. Kritična analiza postojećih rješenja prikazuje slične aplikacije. Pod tehnologijom su pokrivene tehnologije koje su korištene prilikom izrade same aplikacije. Transformacija podataka obuhvaća kreiranja formata podataka koji su dani kako bi aplikacija korektno radila. Opis funkcionalnost aplikacije opisuje funkcionalnost kroz slike kôda i slike ekrana kako bi se dočarala kompleksnost izvedbe. Cilj diplomskog rada bio je razviti novu i modernu mobilnu aplikaciju koja će uz pomoć interaktivne karte pomoći korisnicima u snalaženju u gradskom prijevozu grada Pule. Prednosti aplikacije su jednostavno korištenje, mogućnost mijenjanja jezika i precizne informacije. Tehnologije koje su korištene prilikom izrade aplikacije su: JavaScript, React, React Native, NodeJS.

KLJUČNE RIJEČI: JavaScript, React, React Native, NodeJS, Mobile App, Components

ABSTRACT

The master's thesis consists of an introduction, a critical analysis of existing solutions, technologies, data transformation, and a description of the application's functionality. The introduction provides a brief overview of current technologies and the need for developing the application. The critical analysis of existing solutions presents similar applications. Under the technology section, the technologies used in building the application are covered. Data transformation involves creating data formats necessary for the correct functioning of the application. The description of the application's functionality illustrates its features through code snippets and screen images to convey the complexity of implementation. The aim of the master's thesis was to develop a new and modern mobile application that, using an interactive map, assists users in navigating the public transportation system of the city of Pula. The advantages of the application include ease of use, language customization, and precise information. The technologies used in developing the application are JavaScript, React, React Native, and NodeJS.

KEYWORDS: JavaScript, React, React Native, NodeJS, Mobile App, Components

7. Literatura

[1] JavaScript: The Definitive Guide, 7th Edition (2020.) – Dohvaćeno 28.8. 2023. iz https://pepa.holla.cz/wp-content/uploads/2016/08/JavaScript-The-Definitive-Guide-6th-Edition.pdf

[2] JavaScript - https://developer.mozilla.org/en-US/docs/Web/JavaScript - Dohvaćeno 28.8.2023.

[3] Learning React: Modern Patterns for Developing React Apps (2020.) - Dohvaćeno 28.8.2023. iz https://sd.blackball.lv/library/Learning_React_(2020).pdf

[4] React Native in Action (2019.) - https://manning-content.s3.amazonaws.com/download/a/b410a16-4e64-4168-8b2c-909b3890129a/Dabit_RNiA_MEAP_v16_ch1.pdf - Dohvaćeno 28.8.2023.

[5] Node.js Design Patterns (2020.) - https://ia801309.us.archive.org/5/items/HandbookOfNeuralComputingApplicationsPDFStormRG/Node.js%20Design%20Patterns%20-%20Casciaro,%20Mario%20%5BPDF%5D%5BStormRG%5D.pdf – Dohvaćeno 28.8.2023.

[6] Dekodiranje polilinije - https://github.com/mapbox/polyline/blob/master/src/polyline.js - Dohvaćeno 31.8.2023.

[7] Encoded Polyline Algorithm Format – https://developers.google.com/maps/documentation/utilities/polylinealgorithm - Dohvaćeno 31.8.2023.

[8] Dekodiranje polilinije - https://jsfiddle.net/welesley/tw7qLvh4/2/ - Dohvaćeno 31.8.2023.

Popis slika