Analiza i primjena tehnologija precizne navigacije u zatvorenim prostorima

Sudec, Robert

Master's thesis / Diplomski rad

2021

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **University of Zagreb, Faculty of Organization and Informatics / Sveučilište u Zagrebu, Fakultet organizacije i informatike**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:211:795220

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Robert Sudec

ANALIZA I PRIMJENA TEHNOLOGIJA PRECIZNE NAVIGACIJE U ZATVORENIM PROSTORIMA

DIPLOMSKI RAD

Varaždin, 2021.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Robert Sudec

Matični broj: 44882/16–R

Studij: Informacijsko i programsko inženjerstvo

ANALIZA I PRIMJENA TEHNOLOGIJA PRECIZNE NAVIGACIJE U ZATVORENIM PROSTORIMA

DIPLOMSKI RAD

Mentor :

Doc. dr. sc. Boris Tomaš

Varaždin, Rujan 2021.

Robert Sudec

Izjava o izvornosti

Izjavljujem da je moj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI-radovi

Sažetak

Navigacija je disciplina koja se bavi određivanjem lokacije objekta u pokretu(npr. vozila). Najveći takav sustav je GPS (Global Positioning System) koji može odrediti lokaciju/poziciju objekta sa preciznošću od 5m. Navigacija u zatvorenom prostoru je sustav pozicioniranja 'unutar zidova', a ne koristi GPS. Analizirat ćemo koje su moguće tehnologije za implementaciju IPS (Indoor Positioning System), te odabrati jednu tehnologiju koju ćemo implementirati. Pozicija korisnika određuje se putem uređaja (mobitel koji sadrži veliki broj tehnologija koje se koriste u IPS). Najpoznatije tehnologije su WiFi, Bluetooth, Infracrveni sustavi, dok se koriste i širokopojasni sustavi, akustični sustavi. Najviše korišteno sučelje, na mobilnim uređajima, je do sada bila 2D mapa (tlocrt) sa prikazom rute do odredišta. Takav sustav bi korisniku slao poruke o njegovoj poziciji ili instrukcije za praćenje rute. Ključan dio ovog rada je implementirati modernije korisničko sučelje za sustav pozicioniranja, a to je koristeći proširenu stvarnost (AR). Objekt s kojim ćemo proširiti realnost moramo precizno postaviti u prostor kako bi ispunili cilj navigacije, a takvi objekti mogu biti strelice koje govore koji je potreban smjer kretanja ili objekti koji označavaju neke točke od interesa (POI). Nakon teorijskog pregleda, odabira tehnologija, slijedi praktični dio a to je implementacija IPS pomoću proširene realnosti za Android i iOS uređaje koristeći više-platformski razvojni okvir Flutter.

Ključne riječi: IPS, navigation, tracking, Wayfinding, bluetooth, BLE beacons, AR, Flutter

Sadržaj

1. Uvod

U ovom radu definirati će se tehnologije navigacije u zatvorenim prostorima, analizirati će se postojeće tehnologije koje je moguće koristiti, te odabrati jedna. Takve tehnologije omogućuju lokalizaciju korisnika kada nije dostupan GPS, Objasniti i prikazati će se način na koji se može odrediti lokacija korisnika. Poanta je stvoriti mobilnu aplikaciju pomoću koje će se krajnji korisnik moći navigirati unutar unaprijed definiranog prostora.

Tema je značajna iz razloga što ima vrlo visoku razinu primijenjivosti u stvarnom svijetu. Za svaki prostor u koji dolazi veliki broj ljudi možemo reći da je kompleksan, odnosno da pruža veliki raspon sadržaja od interesa korisnicima. Za primjer možemo uzeti zgradu fakulteta koja ima ogroman broj dvorana na nekoliko katova u kojima se drže predavanja, svrha bi bila studentu prikazati navigacijsku rutu do željene prostorije. Još neki primjeri su knjižnice, bolnice, trgovine, šoping centri i slično.

Motivacija za ovu temu prvobitno proizlazi iz autorove ambicije za razvojem mobilnih aplikacija. Također, ovo se čini kao dobar izazov za učenje i istraživanje slabo poznatih tehnologija.

2. Analiza tehnologija pozicioniranja u zatvorenim prostorima

Svi smo već upoznati sa GPS (Global Positioning System) i većini je jasno na koji način taj sustav radi. GPS je sustav u vlasništvu SAD-a koji pruža usluge pozicioniranja, navigacije, te uslugu mjerenja/podešavanja vremena. Sastoji se od 3 sloja. Sloj u svemiru sastoji se od 31 satelita koji šalju signale s podacima o trenutnoj poziciji satelita i trenutnom vremenu. Sloj kontrole održava satelite, upravlja satelitima, kalibrira informacije satelita. Treći sloj je sloj korisnika, koji koriste uređaje s GPS prijamnicima. Korisnik će primiti signale koje šalju sateliti i pomoću tih podataka može se odrediti pozicija/lokacija korisnika na zemlji, te vrijeme. [1]

U današnje vrijeme, korisnici će najčešće koristiti svoj osobni mobilni uređaj s ugrađenim GPS prijamnikom. Takvi uređaji, u uvjetima koji ne ometaju radio signale, određuju poziciju s greškom do 4.9 metra. [2]

Sama preciznost od oko 5 metara nije dovoljno za navigaciju u zatvorenim prostorima, s time da bi i greška bila veća zbog svih prepreka koje blokiraju radio signal. Zaključujemo da GPS neće zadovoljiti naše potrebe za navigaciju u zatvorenom. Ako uzmemo u obzir da će se koristiti mobilni uređaj kao prijamnik, znamo da postoji veliki broj senzora koji nam mogu pomoći kod stvaranja IPS (Indoor Positioning System). Pomoću aplikacije ćemo ostvariti spomenuti sloj kontrole, a preostaje nam odabrati tehnologiju koja će, u ovom scenariju, zamijeniti sloj satelita. Tako ćemo pokušati izgraditi IPS na principima GPS-a.

2.1. Tehnologije

U ovom poglavlju spomenuti ćemo neke od tehnologija koje se koriste za IPS, proučiti njihov način rada, te samu primjenu tehnologije u sustavima pozicioniranja.

Općenito, svaka od ovih tehnologije će nam na neki način pružiti podatak pomoću kojega ćemo odrediti poziciju/lokaciju korisnika. Postoje nekoliko metoda, a najzanimljivije su trilateracija i fingerprinting o kojima će riječ biti kasnije.

2.1.1. WiFi sustavi

Najviše korištena tehnologija za bežičnu komunikaciju, Wi-fi (Wireless fidelity), točnije WLAN (Wireless Local Area Networks), se može koristiti u svrhu određivanja lokacije uređaja koji se nalazi na području mreže. Ovo rješenje čini se jako primamljivo s obzirom da većina ciljanih prostora već ima implementirani WLAN. Također, Wi-Fi postoji u svim modernijim mobilnim uređajima. Wi-Fi signali mogu doseći više desetaka metara, čak i do 100 metara, što ide u prilog za iskoristivost ove tehnologije. Također, za prepoznavanje signala nije potrebno vidno polje, odnosno uređaji primatelj i odašiljač mogu biti zaklonjeni.

Za određivanje lokacije, najčešće se koristi RSSI (Received Signal Strength Indicators), odnosno indikator jačine primljenog signala koji se lako može isčitati, te je ugrađen u većinu

uređaja koji se danas koriste.

2.1.2. Bluetooth odašiljači

Bluetooth tehnologija bežične komunikacije također koristi radio valove kao i Wi-Fi. Što znači da daje mogućnost korištenja RSSI. Postoje 2 načina rada s Bluetooth-om, to su Blueto-oth Classic i BLuetooh Low Energy. Bitna razlika je da BLE može smanjiti potrošnju energije za do 99% u odnosu na Classic način rada. BLE način rada stoga smanjuje mogućnosti u zamjenu za manju potrošnju energije. Što se tiče dometa, neki uređaji mogu doseći do 100 metara, no najčešće imaju domet oko 10 metara. [3]

BLE odašiljači nisu standardni Bluetooth uređaji, nego pojednostavljena verzija istih. Vjerojatno ste se sreli s uparivanjem Vašeg uređaja s nekim Bluetooth uređajem, no s odašilja-čima se veza ne uspostavlja. Oni služe samo kako bi oglašavali svoje specifikacije kao što su ime, adresa i slično. Postoji mnogo implementacija BLE odašiljača, odnosno BLE Beacon, koji se koriste u svrhe određivanja je li korisnik u blizini.

Slika 1: BLE Odašiljači različitih proizvođača (Izvor: Aislelabs, 2015)

2.1.3. Širokopojasni sustavi

Tehnologija UWB (Ultra-wide band) je tehnologija komunikacije, opet, koristeći radio valove. Specifična je po niskoj potrošnji energije za komunikaciju visoke razine propusnosti na kraćim udaljenostima. Za razliku od ostalih, UWB ne koristi specifičnu frekvenciju, nego odašilje signale na širokom rasponu (pojasu) kanala, kako navodi Scholtz.

Postoje mnogi načini primjene UWB-a, od prijenosa podataka, glasovne komunikacije, do radara. U članku za AFCEA, Kenyon je 2002. opisao da se UWB razvijao i najviše koristio u vojne svrhe, a tek kasnije, od 2002. dobiva dozvolu za korištenje u komercijalne svrhe.

2.1.4. Akustični sustavi

Sustavi komunikacije pomoću zvuka, akustični sustavi. Koriste ultrazvučne signale koje čovjek ne čuje ili pak zvučne koje čujemo. Istraživanje korisničkog iskustva je pokazalo da zvučni signali imaju jedinstvene prednosti u dijeljenju informacija od kraja do kraja (peer-to-peer). [7] Tako se može koristiti podatak TOA (Time of Arrival), odnosno koliko vremena treba signalima da prijeđu put od odašiljača do prijamnika i tako približno odrediti udaljenost.

2.1.5. Infracrveni sustavi

Infracrvena tehnologija koristi valove dulje od valova vidljivog svjetla, pa su nevidiljvi ljudskom oku. Ovi sustavi koriste infracrvene svjetlosne impulse (TV Daljinski upravljač) za prepoznavanje signala u prostoriji. Za razliku od radio valova, svjetlosni valovi ne prolaze kroz zidove, što govori da je ovo jedan od najboljih načina za lociranje korisnika na razini prostorije. Teško je stoga, odrediti gdje je korisnik točno u određenoj prostoriji, pa nam ovakvi sustavi nisu od pomoći u ovoj primjeni. [8]

2.2. Odabir tehnologije i zaključak

Nakon pregleda nekih mogućih tehnologija, potrebno je odabrati jednu uz pomoć kojeg će se razviti sustav u ovom radu.

Isključena je infracrvena tehnologija zbog samog načina rada i nedovoljne preciznosti.

Akustični, odnosno ultrazvučni sustavi su pogodni zbog rasprostranjenosti opreme. Mobilni uređaji imaju mikrofone, neki prostori već imaju zvučnike kojima bi se odašiljali signali. Neki prostori nemaju, stoga ne želimo stvoriti potencijalno velike troškove ugradnje zvučnih sustava, zvučnika i izvora struje.

Širokopojasni sustavi su već poprilično adaptirani za ovakve primjene, pokrivaju više prostora, oprema nije skupa, a i pokazuju bolje rezultate. Razlog zbog kojeg nećemo koristiti ovu tehnologiju je tek nedavna integracija u mobilne uređaje. Tek trenutna generacija "flagship" Samsung modela ima integrirani UWB što nije zadovoljivo. UWB pokazuje veliki potencijal za budućnost.

Preostale su Wi-Fi i Bluetooth tehnologije. Rade na sličan način, imaju usporedive domete, danas ih većina mobilnih uređaja koristi. Što se tiče cijene, uređaji Wi-Fi pristupne točke se mogu naći već od oko 100 HRK, te da većina ljudi već ima Wi-Fi pristupne točke u svojim prostorijama. Bluetooth odašiljači, mnogo tvrtki ih proizvodi i prodaju po različitim cijenama, otprilike po 150 HRK. Drugi faktor je potrošnja energije. Dok bluetooth odašiljači mogu raditi čak do 2 godine na jednoj standardoj CR2302 bateriji, Wi-fi uređaji bi vjerojatno trebali konstantan pristup izvoru energije, znači veći trošak i veći trud u dužem periodu. Zato će se u ovom radu nastaviti koristeći Bluetooth tehnologiju.

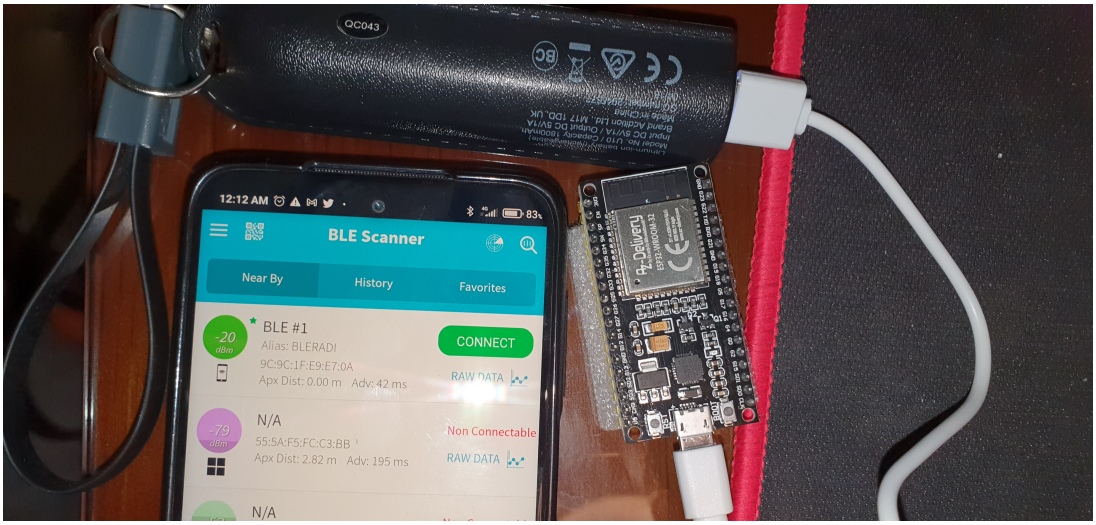
3. Improvizacija BLE odašiljača

Za provedbu ovog istraživanja koristiti će se 5 Bluetooth uređaja. Cijena komercijalnih odašiljača je dosta visoka. Komercijalni odašiljači lijepo izgledaju i tvornički su spremni za rad. Odašiljač može biti bilo koji uređaj koji podržava Bluetooth 4.0 (ili noviju) verziju, pa zašto ne smanjiti troškove. Iskoristiti ćemo ESP32 Wifi mikrokontroler koji ima spremne i integrirane Wifi i Bluetooth module. Kao prvo, jeftiniji su, oko 45 HRK po komadu, imaju više mogućnosti i omogućuju najvišu razinu kontrole. Nažalost, većina odašiljača koriste nRF51822 čipove koje je moguće napajati s CR2302 baterijom, ali ESP32 čipovi troše više energije pa će se napajati putem USB-a koristeći prijenosne punjače.

3.1. Arduino IDE

ESP32 ima velike mogućnosti, tako da iz tvornice dolazi prazan, a korisnik mora odlučiti što će s njim napraviti. Želimo od ESP32 napraviti BLE odašiljač, odnosno pokrenuti ga kao Bluetooth uređaj koji se oglašava na mreži.

Putem Arduino okruženja, potrebno je preuzeti dostupnu gotovu skriptu, te bez ikakvih izmjena poslati skriptu na svaki od ESP32 uređaja. Potrebno je samo svakom uređaju dodijeliti jedinstveno ime da se mogu razlikovati, trenutno su imenovani BLE #1, BLE #2...



Slika 2: Prikaz rada BLE odašiljača (Izvor: Autorski rad)

4. Navigacija

BOWDITCH u svojoj knjizi "The American Practical Navigator" definira navigaciju kao disciplinu koja se bavi praćenjem kretanja objekta od jednog mjesta do drugog. Postoje mnogi načini navigacije koji se već dugo koriste. S napretkom tehnologije, dolaze nam i novi načini navigacije.

Za svrhu ovog rada, navigacija je definirana kao postupak skupljanja informacija koje nam govore kako i na koji način doći, s trenutne lokacije, do željenog odredišta.

Navigacija može biti usmena; "Idi 100 metara ravno, skreni lijevo pa kad prijeđeš preko pruge, skreni desno.". Od davnih vremena izrađuju se karte ili mape koje pomažu pojedincu kod navigacije. Kod takvih mapa korisnik ima prikaz cijelog područja, mora sam sebe pronaći na mapi (lokalizacija) kako bi došao do uputa koje će pratiti prema odredištu. Zatim dolaze digitalne karte i lokacijske usluge, gdje prilikom pokretanja karte, sustav će nas automatski lokalizirati koristeći GPS. Korisnik može unijeti i odredište, a sustav će se pobrinuti da dobije upute. Upute mogu biti vizualne (istaknut je put koji treba pratiti, istaknut je smjer kretanja..), glasovne (upute će biti izrečene).

Sve je ovo moguće, danas, u vanjskim prostorima gdje je dostupan GPS. Kada promatramo "unutarnji svijet", malo je vjerojatno da se netko susreo sa modernim navigacijskim konceptima. Većina unutarnjih prostora kojima je potrebna navigacija još uvijek je ostalo na običnim kartama pomoću kojih se korisnik mora sam snaći. Na primjer, šoping centar na ulazu ima izloženu kartu kako bi se kupac mogao snaći među brojnim trgovinama.

Za sada je dovoljno reći da će se koristiti fiksni Bluetooth odašiljači kao temelj unutarnje karte, a objekt koji će se pratiti je korisnikov mobilni uređaj. Navigacija, odnosno iznošenje uputa kretanja, će se odvijati putem proširene realnosti (AR) na način da će se korisnikov pogled kroz objektiv (kamera) proširiti sa navigacijskim uputama.

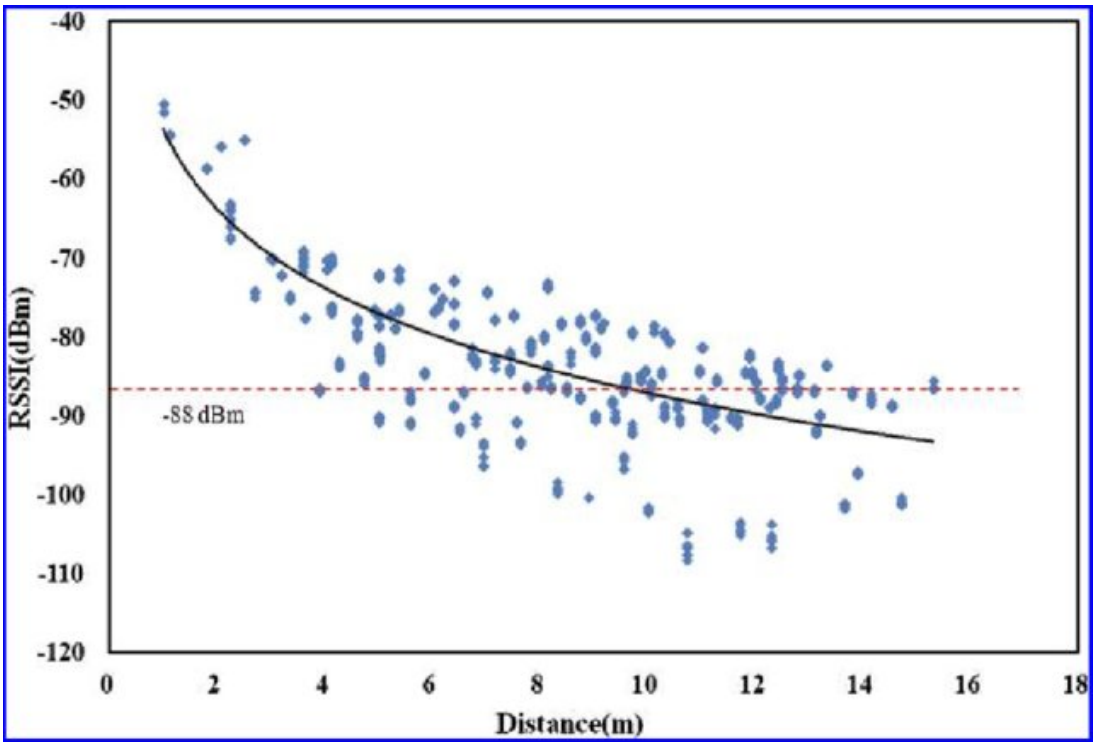
4.1. Određivanje pozicije korisnika

Prvi korak u navigaciji je lokalizacija, odnosno odrediti gdje se korisnik nalazi na toj karti. Za lokalizaciju će nam pomoći BLE odašiljači. Odašiljači su fiksni u prostoru, pa možemo odrediti relativnu lokaciju korisnika u odnosu na odašiljače.

4.1.1. Trilateracija

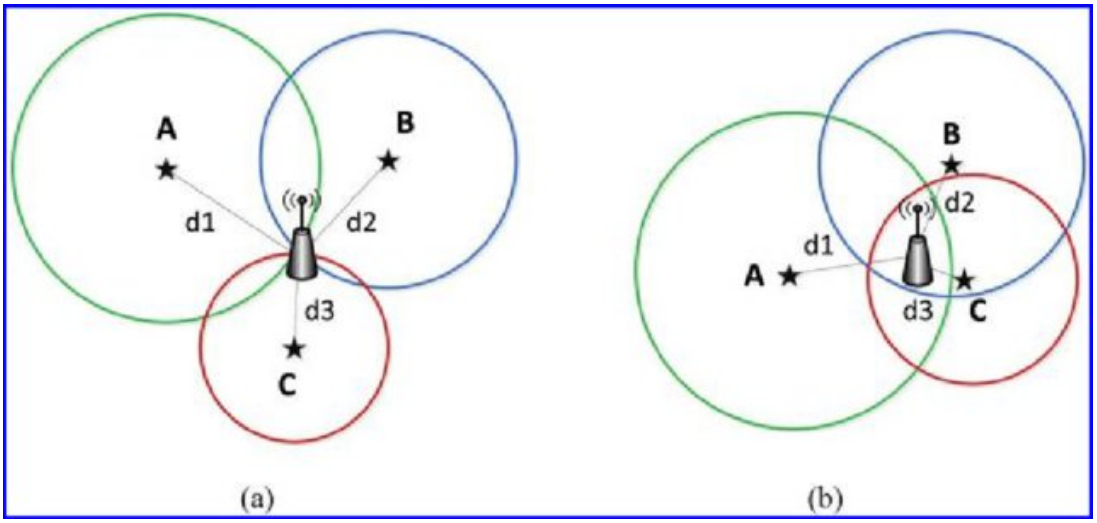
Trilateracija je osnovna metoda za računanje pozicije čvora u mreži (čvor je korisnik), tako navode Boukerche, Oliveira, Nakamura i dr. Pozicija se računa kao presjek kružnica. Ako znamo lokacije fiksnih odašiljača i udaljenost od svakog od njih, možemo odrediti lokaciju čvora.

Udaljenost od odašiljača možemo približno odrediti na temelju spomenutog RSSI, ali treba imati na umu da što je veća udaljenost, veća je i greška u kalkulaciju udaljenosti.



Slika 3: Prikaz odnosa RSSI i udaljenosti (Izvor: Soleimanifar, Shen, Lu i dr., 2014)

U 2-dimenzionalnom prostoru, potrebne su minimalno 3 fiksne točke za lokalizaciju jedne varijabilne. Zapravo promatramo 3-dimenzionalni prostor, no možemo zanemariti z-os (visinu). U geometrijskom smislu, stvaramo 3 kružnice kojima su središta odašiljači. Svaka kružnica ima svoj radijus koji odgovara izmjerenoj udaljenosti korisnika od odašiljača, te teoretski, presjek će nam dati lokaciju korisnika. Primjer idealnog i stvarnog slučaja na slici.



Slika 4: Prikaz idealnog i stvarnog slučaja trilateracije (Izvor: Soleimanifar, Shen, Lu i dr., 2014)

U stvarnom slučaju, mala je vjerojatnost da će se dogoditi idealni slučaj kao na slici. Prvi faktor je sam izračun udaljenosti, na slici odnosa vidimo da udaljenost ima visoku varijabilnost pa bi to moglo dovesti do slučaja gdje se kružnice ne sijeku i prezentiralo beskonačno mogućih

točaka lokacije. Također, bitna je i okolina u kojoj se odvija lokalizacija, gdje razni uvjeti mogu omesti radio signale i predočiti, opet, neispravne vrijednosti udaljenosti.

Multilateracija je postupak s istim značenjem kao trilateracija, samo što koristi više odašiljača. Multilateracija bi teoretski trebala ostvariti bolje rezultate,

4.1.2. Fingerprinting

Jedna od metoda lokalizacije je "fingerprinting", postavljanje otisaka očitanja signala. Pretpostavka je da svaka prostorija ima skup podataka koji sadrži očitanja jačine signala od fiksnih odašiljača.

Sastoji se od 2 faze. Prva faza je stvaranje skupa podataka, odnosno, moramo na svakoj točki gdje bi se korisnik mogao lokalizirati izmjeriti vrijednosti jačine signala i spremiti u bazu podataka, Druga faza je aktivna faza, gdje korisnik očitava trenutne jačine signala i uspoređuje ih sa zapisima u bazi podataka.

Na prvi pogled, ova metoda rješava probleme trilateracije. Kod trilateracije smo imali problem smetnje i varijabilne udaljenosti. Koristeći fingerprinting, registriramo te smetnje jer će se vjerojatno opet pojaviti. U fazi skupljanja podataka, možemo skupiti više očitanja za svaku poziciju i tako probati riješiti problem varijabilne udaljenosti.

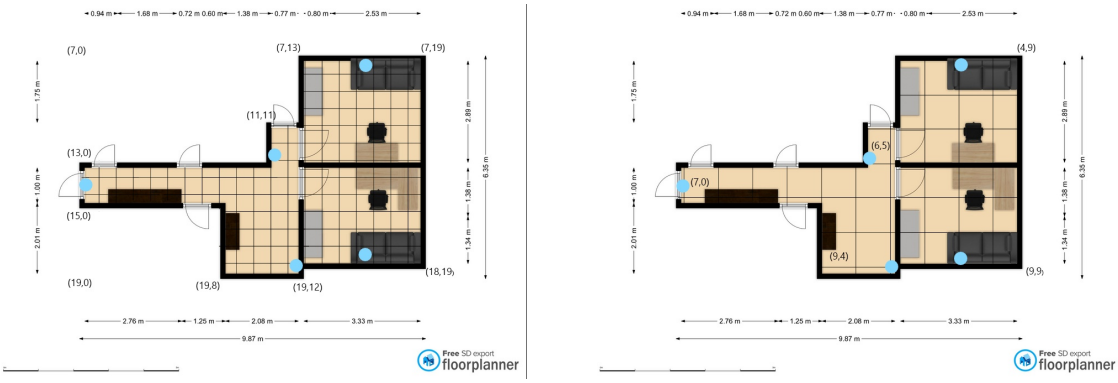
Za sada, ova metoda se čini super. No, moramo adresirati i neke nove probleme. Problem je što se skup podataka odvija u jednom vremenu, dok će se faza lokalizacije odvijati duže vremena. Znamo da zabilježavamo trenutne vrijednosti jednom, no svaka promjena bi mogla imati utjecaj na ispravnost našeg skupa. Promjene se odnose na promjene u infrastrukturi, dizajnu unutrašnjosti, rasporedu namještaja i zidova.

Veliki nedostatak je iscrpno, ručno, skupljanje podataka, koje potencijalno zahtijeva redovna ažuriranja. Svakako, u ovom radu primijeniti će se fingerprinting metodu.

4.1.3. Priprema lokalizacije - uzimanje otisaka

Za lakše snalaženje, izrađen je tlocrt prostora. Koristio sam besplatni plan na aplikaciji Floorplanner. Odašiljači su označeni plavom bojom na slici, a postavljeni su na zid na 2m visine. U ovom slučaju maksimalni razmak odašiljača je 6 metara. Po uzoru na GPS koordinate, napraviti ćemo vlastiti koordinatni sustav koji će predstavljati prostor. Cilj je napraviti dovoljno dobar i precizan skup podataka, stoga će biti razmotrene različite udaljenosti među koordinama. Što je koordinata manja, ostvaruje se veća preciznost, a povećava se i mogućnost greške. Na kraju ćemo usporediti rezultate.

Sada možemo krenuti u prvu fazu pripreme, odnosno stvaranja skupa podataka (eng. dataset) koji će se koristiti za lokalizaciju. Potrebno je u prostoru, prema definiranom tlocrtu i koordinatnom sustavu, pozicionirati se na određenu koordinatu, te spremiti očitanje BLE odašiljača u dometu (ako odašiljač nije u dometu, signal je jačine 0). Za ovaj primjer preuzima se 30 očitanja jačine signala kroz 3 sekunde. Ovaj postupak rezultira skupom podataka definiranog formata neovisno o broju BLE odašiljača.



(a) Koordinata veličine 0.5m

(b) Koordinata veličine 1m

Slika 5: Prikaz tlocrta prostorije, pozicije odašiljača i koordinatni sustav (Izvor: Autorski rad)

Pomoću dobivenog skupa podataka možemo približno prepoznati gdje se nalazi korisnik u koordinatnom sustavu.

Listing 4.1: "Format skupa podataka"

```
BLE_RSSI_1,BLE_RSSI_2,BLE_RSSI_3,...,BLE_RSSI_N,X-Y
...
BLE_RSSI_1,BLE_RSSI_2,BLE_RSSI_3,...,BLE_RSSI_N,X-Y
```

Listing 4.2: "Primjer dijela skupa podataka"

```
{
    ...
    -85,-74,-73,-73,-94,10-18
    -85,-74,-73,-73,-94,10-18
    -88,-89,-73,-80,-94,10-19
    -88,-89,-73,-80,-94,10-19
    ...
}
```

4.2. Optimalne rute

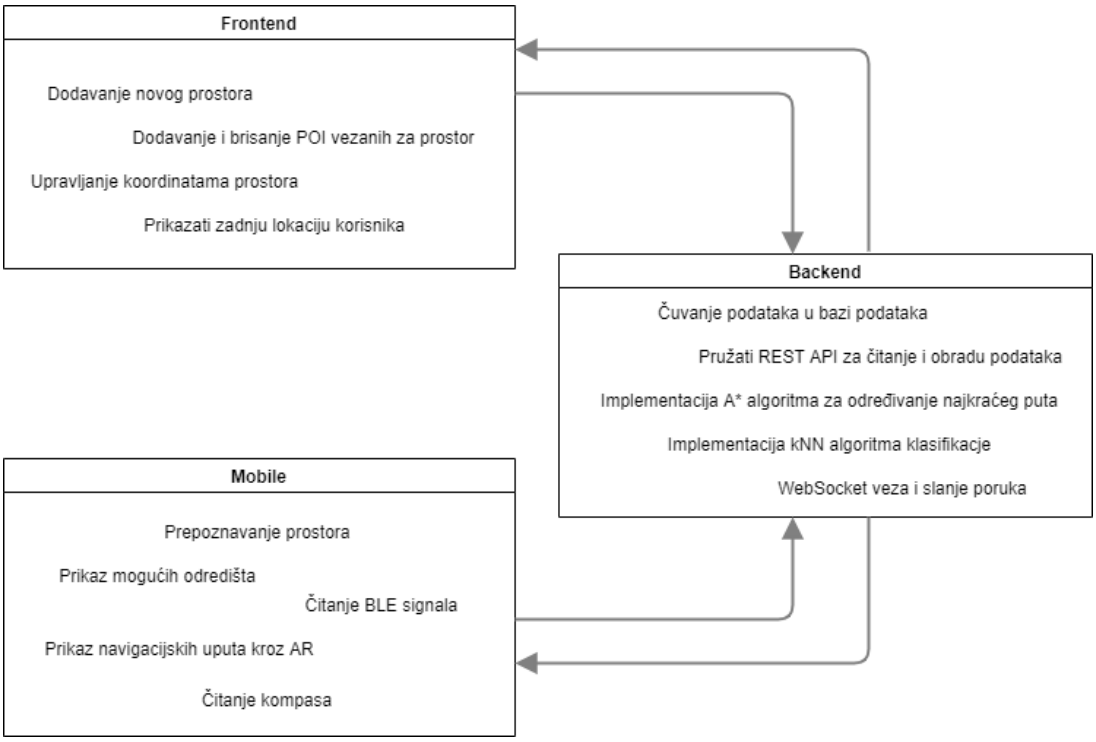
Cilj je omogućiti korisniku navigacijske upute prema nekoj točki, njemu od interesa. Koristeći koordinatni sustav, lokacija korisnika i lokacije POI su poznate.

Jedna od opcija su predefinirane rute. Pružatelj usluge će definirati rute od POI A do POI B, koja će se adaptirati relativno o lokaciji korisnika. Druga opcija je dinamičko računanje rute, svaki pokušaj navigacije će zahtijevati kalkulaciju rute od trenutne lokacije korisnika do željenog POI.

Koristiti će se dinamičko računanje puta. Vjerojatno bi bilo najbolje odrediti, ne bilo kakav put, nego najkraći mogući. Na koordinatnom sustavu, lako je primijeniti algoritme najkraćeg puta. Odabran je vrlo popularan algoritam A*. Patel je naveo da taj algoritam koristi informacije ostalih algoritama za traženje puta i tako postaje fleksibilan i spreman za široku uporabu.

5. Planiranje i dizajn sustava

Za potrebe ovog istraživanja izraditi će se sustav koji se sastoji od poslužitelja (eng. backend), web administratorskog sučelja (eng. frontend) i mobilne aplikacije (eng. mobile application).



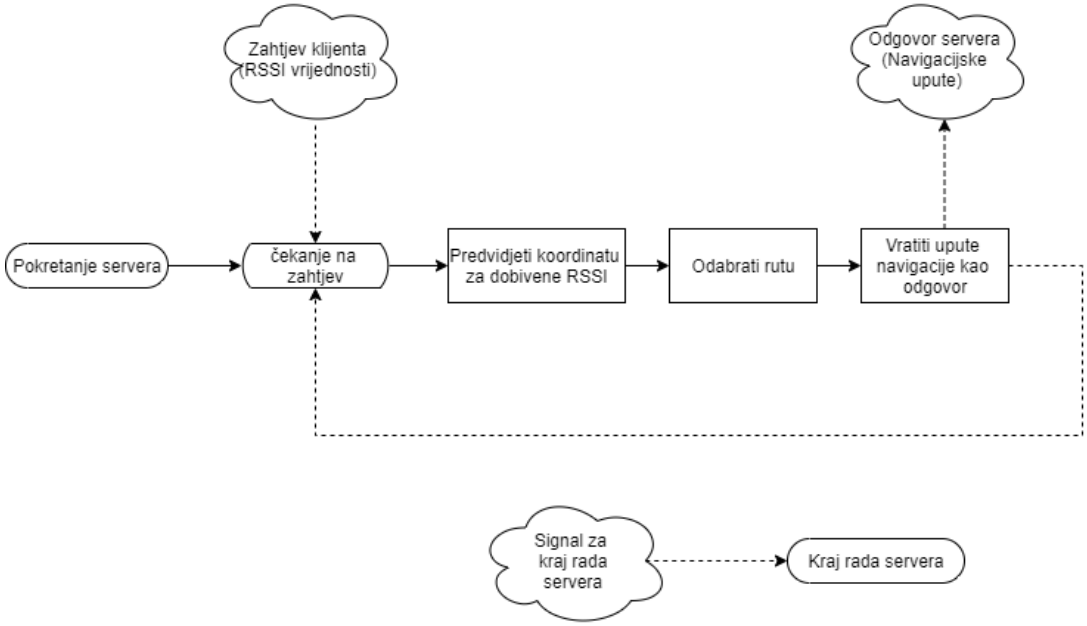
Slika 6: Prikaz arhitekture sustava (Izvor: Autorski rad)

5.1. Backend sustav

Na backend sustavu će se odvijati svaka obrada podataka. Ovaj način rada će prouzrokovati kašnjenje podataka zbog same arhitekture, svaki će zahtjev potrošiti neko vrijeme na sam prijenos podataka i na obradu.

5.1.1. Komunikacija

Za komunikaciju imamo 2 opcije, koristeći REST API ili WebSocket komunikaciju. REST API sustav je određen da klijent šalje zahtjev na server i čeka odgovor, koristi jednosmjernu komunikaciju. S druge strane, WebSocket nam omogućuje dvosmjernu komunikaciju porukama. Što se tiče performansi, API zahtjevi će svaki put uspostavljati vezu sa serverom i čekati na odgovor, a za WebSocket je potrebno jednom ostvariti vezu sa serverom. [14] Za različite potrebe koristiti će se kombinacija 2 načina.



Slika 7: Prikaz toka backend sustava na visokoj razini (Izvor: Autorski rad)

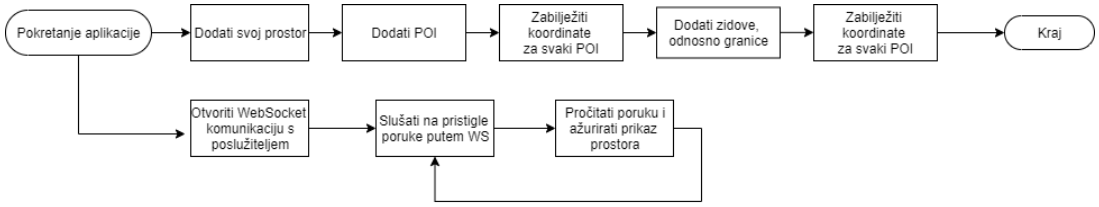
5.1.2. Alati

Za implementaciju backenda potrebni su nam prikladni alati. Za komunikaciju koristit ću Rust i njegov razvojni okvir Actix-Web za implementaciju HTTP poslužitelja. [15] PostgreSQL će nam poslužiti kao baza podataka u kombinaciji s Diesel ORM (Object-Relational Mapping) alatom.

Fingerprint skup podataka se čuva u tekstualnim datoteka, zbog toga, a i zbog potrebne statističke obrade poslužitelj će još koristiti Python i njegove alate za statističku obradu.

5.2. Frontend sustav

Frontend web aplikacija će u ovom projektu biti potpora i biti sučelje za administratora. Kroz takvo sučelje zapravo se komunicira s bazom podataka za dodavanje, brisanje, ažuriranje podataka, putem REST API krajnjih točaka. Administrator prostora će tako pružiti sve potrebne podatke za prostor kako bi sustav funkcionirao, te će imati uvid u zadnje snimljene lokacije svojih korisnika.



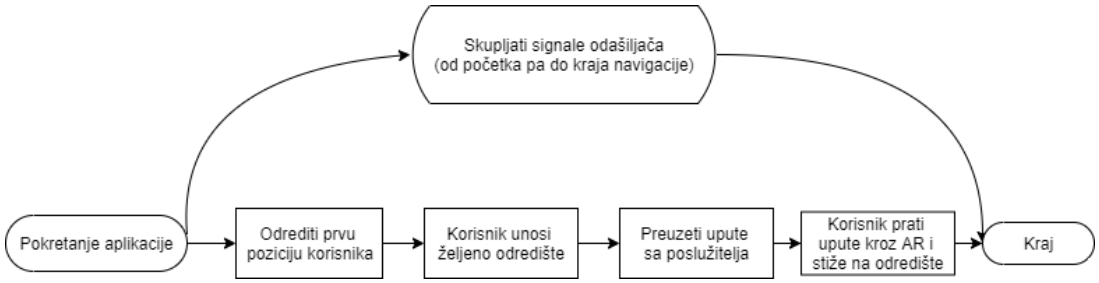
Slika 8: Prikaz rada web aplikacije na visokoj razini (Izvor: Autorski rad)

5.2.1. Alati

Web aplikacija je izrađena u Flutter razvojnom okviru. Flutter je Google-ov više-platformski razvojni okvir. Koristiti ćemo ga u kombinaciji s Get okvirom za upravljanje stanjima (eng. state management).

5.3. Mobilna aplikacija

Aplikacija predstavlja sučelje prema navigaciji. Primit će signale odašiljača, te će te informacije prenijeti na poslužitelj. Stoga će biti potrebna internetska veza. Poslužitelj čuva sve fingerprint uzorke koje smo zabilježili, te uspoređuje primljene informacije od korisnika i vraća koordinatu i upute. Na slici je prikazana logika na najvišoj razini, kasnije će se detaljnije prikazati način rada.



Slika 9: Prikaz toka mobilne aplikacije na visokoj razini (Izvor: Autorski rad)

5.3.1. Alati

Mobilna aplikacija je također izrađena u Flutter razvojnom okviru. Potrebno je pisati nativan kod kod pristupa hardware-u mobilnog uređaja jer su oni specifični za svaku platformu. Vezano za specifične mogućnosti koje su ovisne o platformi, koristiti će se: Bluetooth, kompas, lokaciju i AR. Za sve mogućnosti korisni će biti već postojeći paketi koji daju funkcionalnost na obe platforme (iOS/Android).

5.3.2. Proširena realnost

Furht (2006) definira proširenu realnost kao sustav koji poboljšava stvarni svijet tako što uključuje računalno generirane informacije. Takve informacije mogu biti u bilo kojem obliku, slike, 3D modeli, audio/video materijali, tekst i slično.

Najpoznatiji alati za razvoj su Apple-ov ARKit izdan 2017, Google-ov ARCore i Vuforia koji je spreman za razvoj na iOS, Android i UWP platformama, kako pišu Chen, Wang, Chen i dr.

U ovom radu koristiti će se ARKit i ARCore za razvoj aplikacije na iOS i Android uređajima. Funkcionalnosti će biti postavljanje AR virtualnih objekata, 3D modela u okolinu tako da pokazuju put prema odredištu.

6. Razvoj sustava

U ovom dijelu rada detaljnije će se opisati svaka funkcionalnost pojedinog dijela sustava.

6.1. Backend sustav

6.1.1. Struktura projekta

Sav izvorni kod nalazi se u src direktoriju, zajedno s početnom datotekom main.rs. Svaki modul (spaces, pois...) nalazi se u svom direktoriju sa datotekom koja opisuje model, te datotekom koja opisuje moguće zahtjeve odnosno HTTP rute. Svaki modul ima i datoteku mod.rs u kojoj se izlažu moduli za korištenje.

Listing 6.1: "Primjer datoteke mod.rs"

```
mod model;
mod route;
pub use model::*;
pub use route::init_routes;
```

U korijenskom direktoriju projekta nalazi se još i direktorij za čuvanje skupova podataka, direktorij za Diesel migracije, te direktorij s vanjskim Python skriptama.

Datoteke schema.rs i db.rs vezane su za Diesel ORM i konekciju prema bazi podataka.

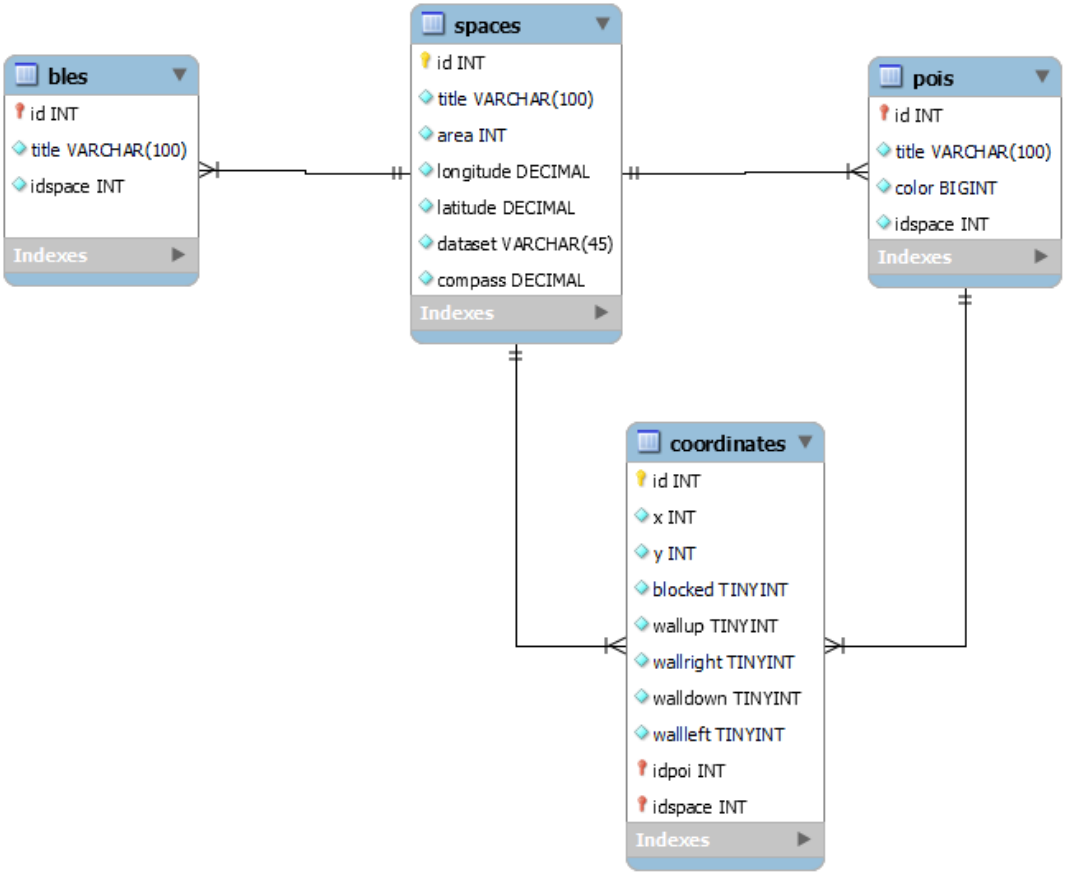
Listing 6.2: "Struktura Rust backend projekta"

```
-datasets
-migrations
-py
-src
    -astar
    -bles
    -coordinates
    -pois
    -python_knn
    -route_request
    -spaces
    -websocket
    main.rs
    schema.rs
    db.rs
```

6.1.2. Baza podataka

Baza podataka je ključan dio većine projekata, a kao što je rečeno, baza podataka koju koristimo je PostgreSQL koja je sadržana unutar Docker kontejnera.

Baza će se sastojati od 4 tablice. Te tablice će opisivati prostore, točke od interesa, koordinate i BLE uređaje.



Slika 10: ERA Baza podataka (Izvor: Autorski rad)

Tablica Spaces čuva podatke o pojedinom prostoru. Imamo naslov, površinu koja je uvijek kvadratna. Zatim, geografska širina i dužina pomoću kojih će mobilni uređaj automatski prepoznati u kojem se prostoru nalazi, naziv skupa podataka u kojem se nalaze otisci prostora, te orijentacija prostora s obzirom na sjever. Podatak o orijentaciji potreban je kako bi sustav znao izračunati navigacijske upute. Detaljnije ćemo to opisati kasnije.

Svakom prostoru potrebno je dodijeliti BLE uređaje, ta tablica čuva samo naziv BLE uređaja kako bi mobilna aplikacija opet znala koje signale treba pročitati.Postoji i veza 1:N, odnosno jedan prostor može imati više BLE uređaja.

Također, tablica pois čuva podatke o točkama od interesa, naslov za prepoznavanje, a čuva i boju kojom ćemo prikazati taj POI na web sučelju. Postoji i veza 1:N, odnosno jedan prostor može imati više POI-a.

Zadnja tablica opisuje koordinate, naravno x i y vrijednosti. Uz to i još su dodatni podaci je li koordinata blokirana, što znači da se korisnik ne može kretati po toj koordinati. Također, čuvamo podatke postoji li zid između koordinata, odnosno postoji li zid iznad, desno, ispod ili lijevo od koordinate. Ovdje naravno postoje veze 1:N iz tablica spaces i pois.

6.1.2.1. Diesel migracije

Diesel nam omogućuje jednostavniju interakciju s bazom podataka mapiranjem objekata u relacije (ORM). [18] Strukturom baze podataka upravljamo Diesel migracijama, sustavom kroz koji se baza podataka inkrementalno gradi i omogućava povratak na staro (eng. undo, revert), te tako gradi schema.rs datoteku. U toj datoteci opisane su tablice, tipovi podataka njihovih atributa prevedeni u Rust, te pravila spajanja tablica.

Listing 6.3: "Primjer: Tablica spaces u generiranoj datoteci schema.rs"

```
table! {
    spaces (id) {
        id -> Int4,
        title -> Varchar,
        area -> Int4,
        longitude -> Float8,
        latitude -> Float8,
        dataset -> Varchar,
        compass -> Float8,
    }
}
```

Potrebno je i model u kodu anotirati s atributom table_name pa možemo slati objekte unutar Diesel upita (eng. query), umjesto mukotrpnih SQL upita.

Listing 6.4: "Primjer: Struktura za tablicu spaces i upit za dohvaćanje svih prostora"

```
#[derive(Deserialize, Serialize, Debug, Queryable, Clone, Insertable)]
#[table_name = "spaces"]
pub struct Spaces {
    pub id: i32,
    pub title: String,
    pub area: i32,
    pub longitude: f64,
    pub latitude: f64,
    pub dataset: String,
    pub compass: f64
}
impl Spaces {
    pub fn find_all() -> Result<Vec<Self>, CustomError> {
        let conn = db::connection()?;
        let spaces = spaces::table.load::<Spaces>(&conn)?;
        Ok(spaces)
    }
}
```

6.1.3. WebSocket

Spomenuto je da će se koristiti WebSocket komunikacija. Prigodno je jer želimo prikazati zadnje lokacije korisnika na web sučelju. Umjesto da Web sučelje konstantno traži od poslužitelja te informacije, poslužitelj na ovaj način može poslati informaciju kada je ona spremna.

Sada ćemo prikazati implementaciju tako da se možemo pozvati kasnije u tekstu.

Ovaj način komunikacije implementiran je kao WebSocketServer struktura. Ova struktura čuva listu svih spojenih klijenata, te generator nasumičnih brojeva za dodijeljivanje identifikatora klijentima. Također postoje i metode za dodavanje i uklanjanje klijenata s veze, te nama najbitnija metoda za slanje poruke koja svim povezanim klijentima šalje tekstualnu poruku.

Listing 6.5: "Struktura WebSocketServer"

```
#[derive(Default)]
pub struct WebSocketServer {
    sessions: HashMap<usize, Recipient<Message>>,
    rng: ThreadRng,
}
impl WebSocketServer {
    pub fn add_session(&mut self, client: Recipient<Message>) -> usize {
       //...
    }
    pub fn send_message(&mut self, message: String) {
        for (_id, recipient) in &self.sessions {
            recipient
                .do_send(Message(message.to_owned()))
                .expect("Could not send message to the client.");
        }
    }
    pub fn remove_session(&mut self, session_id: usize) {
        //...
    }
}
```

Bitno je dodati da je WebSocket implementacija bazirana na Actor Model. To dozvoljava komunikaciju između neovisnih, ali kooperativnih Actor-a. Actor je objekt koji sadrži stanje (eng. state) i ponašanje (eng. behavior). Tako je WebSocketServer jedan Actor, dok je svaka sesija, odnosno veza s klijentom poseban Actor. Rade paralelno, komuniciraju porukama, te su međusobno neovisni. [19]

Klijent može poslati zahtjev za spajanje putem jedne REST krajnje točke.

Listing 6.6: "WebSocket krajnja točka"

```
#[get("/ws/")]
pub async fn websocket(
    request: HttpRequest,
    stream: Payload,
) -> Result<HttpResponse, actix_web::Error> {
    let response = ws::start(WebSocketSession::new(), &request, stream);
    response
}
```

6.1.4. REST API

REST API je sučelje pomoću kojeg definiramo moguće radnje koji upravljaju podacima. Koristeći Actix-web paket za implementaciju HTTP poslužitelja, za svaku od navedenih tablica stvorili smo 5 krajnjih točaka za osnovne zahtjeve, odnosno CRUD (Create, Read, Update, Delete) operacije.

Za definiranje HTTP putanje dovoljno je anotirati funkciju uz definiranje HTTP metode te relativnu putanju.

- GET: find_all : za dohvaćanje svih zapisa
- GET: find(id) : za dohvaćanje jednog zapisa sa zadanim ID
- POST: create(novi_zapis) : za dodavanje novog zapisa
- PUT: update(id, ažurirani_zapis : za ažuriranje zapisa sa zadanim ID
- DELETE: delete(id) : za brisanje zapisa sa zadanim ID

Uz navedene zahtjeve postoje i dodatni koji se razlikuju od tablice do tablice, ovisno o potrebi.

Listing 6.7: "Primjer: GET i POST putanje za tablicu spaces"

```
#[get("/spaces/{id}")]
async fn find(id: web::Path<i32>) -> Result<HttpResponse, CustomError> {
    let space = Spaces::find(id.into_inner())?;
    Ok(HttpResponse::Ok().json(space))
}
#[post("/spaces")]
async fn create(space: web::Json<Space>) -> Result<HttpResponse, CustomError> {
    let space = Spaces::create(space.into_inner())?;

    Ok(HttpResponse::Ok().json(space))
}
```

6.1.4.1. Prostori

Za ovu tablicu imamo jedan dodatni mogući zahtjev. To je pronalazak najbližeg prostora ovisno o geolokaciji. Pomoću toga klijent može automatski prepoznati u kojem se prostoru nalazi. Bitno je napomenuti da je pretpostavka da se 2 prostora ne nalaze u jednoj zgradi. Odgovor na ovaj zahtjev je JSON reprezentacija objekta prostora.

Listing 6.8: "Primjer zahtjeva za pronalazak najbližeg prostora"

```
{
    "x" : 14.12345,
    "y" : 25.54321
}
```

Listing 6.9: "Endpoint za pronalazak najbližeg prostora"

```
#[post("/spaces/find")]
async fn find_closest(raw: web::Json<RawLocation>) -> Result<HttpResponse,
    CustomError> {
    let space = Spaces::find_closest(raw.x, raw.y)?;
    Ok(HttpResponse::Ok().json(space))
}
```

6.1.4.2. Točke od interesa, BLE odašiljači, Koordinate

Za tablice iz podnaslova imamo jedan dodatni mogući zahtjev. To je pronalazak objekata za zadani prostor. Ovaj zahtjev putem GET metode traži da se unese ID prostora unutar samog URL-a, a kao odgovor vraća JSON reprezentaciju polja koje sadrži JSON reprezentaciju objekata.

Listing 6.10: "Endpoint za pronalazak liste POI objekata za zadani prostor"

```
#[get("/pois_space/{idspace}")]
async fn find_by_space_id(idspace: web::Path<i32>) -> Result<HttpResponse,
    CustomError> {
    let pois = Pois::find_by_space_id(idspace.into_inner())?;
    Ok(HttpResponse::Ok().json(pois))
}
```

Listing 6.11: "Primjer: Odgovor kod zahtjeva za pronalazak liste POI objekata za zadani prostor"

```
[
    {
        "id": 54,
        "title": "Robert's room",
        "idspace": 21,
        "color": 4294443011
    },
    {
        "id": 55,
        "title": "Alen's room",
        "idspace": 21,
        "color": 4278200831
    },
    ...
]
```

6.1.4.3. Skupovi podataka

Potrebno je čuvati skupove podataka za svaki prostor, tako da je implementiran endpoint za prijenos datoteka. Na poslužitelju definiran je direktorij u koji se spremaju datoteke, a u bazi podataka čuva se samo naziv vezane datoteke.

6.1.4.4. Lokalizacija

U ovom poglavlju opisati ćemo krajnju točku za određivanje lokacije korisnika. Sada će se koristiti skup podataka otisaka koji je napravljen u fazi pripreme lokalizacije. Spomenuli smo već da su signali volatilni zbog mnogo faktora, zato većina istraživanja predlažu statističke prognoze kako bi se smanjila greška. [20] Također, koristit ćemo Python skriptu tako da će se ista pozivati iz Rust koda.

U ovom slučaju skup podataka sadrži 30 očitanja za svaku koordinatu, ali što ako se niti jedno očitanje ne podudara s RSSI vrijednostima iz zahtjeva. Osoba može stajati između koordinata, može biti okrenuta u suprotnu stranu od one na kojoj smo mi uzeli otisak. Može se puno toga razlikovati stoga postoji i šansa da se vrijednosti neće točno podudarati. Jedna od statističkih metoda kNN klasifikacije (k-Nearest Neighbors).[21]

Python će olakšati stvari oko rada sa skupovima podataka i statističkom obradom istih. Poslužiti će nam paketi pandas i sklearn. Skripta će primiti 2 argumenta, a to su JSON reprezentacija BLE očitanja i datoteka u kojoj je sadržan skup podataka. Zatim funkcija odgovorna za klasifikaciju preuzima vrijednosti BLE očitanja i koristeći KNeighborsClassifier pokušava predvidjeti jednu (n_neighbors = 1) najbližu klasu za dobivene vrijednosti BLE. Klasa je u ovom slučaju koordinata "X-Y", zadnji stupac u skupu podataka.

Listing 6.12: "Funkcija klasifikacije"

```
def classify(indexes, values):
    x = dataset.iloc[:, indexes].values
    x = x.astype(int)
    y = dataset.iloc[:, column_numb-1 ].values
    neigh = KNeighborsClassifier(n_neighbors=1)
    neigh.fit(x, y)
    classification = neigh.predict([values])
    coordX , coordY = classification[0].split('-')
    result = json.dumps({"x": int(coordX), "y" : int(coordY)})
    print(result)
```

Kao odgovor gradimo JSON objekt koordinate, a rezultat se ispisuje kako bi Rust mogao to pročitati. Sada je krajnja točka spremna na zahtjev za lokalizaciju. Klijent mora poslati informacije o prostoru u kojem se nalazi, imenu koje će se prikazati na Web-u, te očitanja signala.

Listing 6.13: "Primjer tijela zahtjeva"

```
{
   "idspace" : 21,
   "name" : "iPhone␣7",
   "source" : [
        {
            "id_ble": 1,
            "rssi": -78
        },
        {
            "id_ble": 2,
            "rssi": -67
        },
        {
            "id_ble": 3,
            "rssi": -84
        },
        {
            "id_ble": 4,
            "rssi": -70
        },
        {
            "id_ble": 5,
            "rssi": -83
        }
    ]
}
```

Sama krajnja točka će korisniku vratiti odgovor koji je dobiven iz python skripte, odnosno JSON objekt s X i Y atributima, a u međuvremenu šalje se WebSocket poruka prema klijentima. WebSocket poruka je rezultirajući JSON s X i Y, te dodanim atributom 'name'.

Listing 6.14: "Krajnja točka za lokalizaciju"

```
#[post("/coordinate")]
async fn coordinate(request: web::Json<CoordinateRequest> ) -> Result<HttpResponse,
    CustomError> {
    let name = request.name.clone();
    let _response = RouteRequest::handle_coordinates(request.into_inner());
    let _response_with_name = CoordinateWSResponse {
        x: _response.x,
        y: _response.y,
        name: name
    };
    let msg = SendMessage{id: 1, name: String::from("Server"), content:serde_json::
        to_string(&_response_with_name).unwrap()};
    WebSocketServer::from_registry().do_send(msg);
    Ok(HttpResponse::Ok().json(_response))
}
```

6.1.4.5. Navigacija

Ovdje će se opisati krajnja točka pomoću koje klijent dobije navigacijske upute. Koristiti će se algoritam za traženje najkraćeg puta A*. Gotovu implementaciju algoritma omogućuje paket pathfind. A* algoritam pronalazi najkraći put u težinskom grafu koristeći heuristiku za vođenje.[22]

A* algoritam traži podatke o izvornoj koordinati, svim koordinatama prostora i odredišnoj koordinati. Treba uzeti u obzir i da je za svaku koordinatu potrebno računati polje mogućih slijedbenika. Metoda successors je implementirana na Coordinates strukturi uz pomoć spomenutih blocked, wallup, wallright, walldown i wallleft atributa. Metoda dohvaća sve 4 koordinate a zatim provjerava je li moguće prijeći na tu koordinatu.

Listing 6.15: "Metoda successors() za računanje mogućih slijedećih koordinata u navigacijskoj ruti"

```
pub fn successors(&self, coords: Vec<Coordinates>) -> Vec<(Coordinates, u32)> {
        let &Coordinates {id: _, x, y, blocked: _, idspace: _, idpoi: _, wallup: _,
            wallright: _, walldown: _, wallleft: _ } = self;
        let left = coords.iter().find(|&c| c.x == x - 1 && c.y == y).clone();
        let top = coords.iter().find(|&c| c.x == x && c.y == y + 1).clone();
        let right = coords.iter().find(|&c| c.x == x + 1 && c.y == y).clone();
        let down = coords.iter().find(|&c| c.x == x && c.y == y - 1).clone();
        let mut succ: Vec<Coordinates> = vec![];
        match left {
            Some(c) => {
                if c.blocked == false && self.wallleft == false {
                    succ.push(c.clone());
                }}
            None => {}
        }
        match top {
            Some(c) => {
                if c.blocked == false && self.wallup == false {
                    succ.push(c.clone());
                }}
            None => {}
        }
        match right {
           ...
        }
        match down {
           ...
        }
        succ.into_iter().map(|p| (p, 1)).collect()
    }
```

Funkcija astar_bag preuzima redom, izvornu koordinatu, 'closure' funkciju za računanje slijedbenika, 'closure' funkciju za heuristiku koja je u ovom slučaju udaljenost do odredišta, te 'closure' funkciju za zaustavljanje algoritma, odnosno za uspjeh. Algoritam staje kada se prvi

put dogodi da je trenutna koordinata unutar istog POI-a kao i odredišna. Kao rezultat dobije se polje koordinata kao navigacijske upute.

Listing 6.16: "Funkcija za pokretanje A*"

```
pub fn pathfind(source: Coordinates, space_coords: Vec<Coordinates>, dest: &
    Coordinates) -> Option<(Vec<Coordinates>, u32)> {
    let mut res = astar_bag(&source, |p| p.successors(space_coords.clone()), |p| p.
        distance(&dest), |p| p.idpoi == dest.idpoi).unwrap();
    return match res.0.next(){
        Some(x) => Some((x,res.1)),
        None => None,
    }
}
```

Proširena stvarnost inicijalizacijom uređaj postavlja u novi koordinatni sustav tako da je uređaj novo ishodište. To nas dovodi do novog problema konverzije iz koordinatnog sustava prostora u novi relativni sustav. Novi sustav sastoji se od 3 dimenzije (osi), ali ćemo Y os za visinu ignorirati, a koristiti ćemo samo X i Z. Z os u 3D sustavu će dobiti vrijednosti iz Y osi iz 2D sustava.

Slika 11: Ishodište AR prostora (Izvor: Autorski rad)

Možemo zaključiti da je potrebno rotirati sustav s obzirom na orijentaciju, te translatirati ishodište u koordinatu na kojoj se uređaj nalazi.

Za početak potrebno je pretvoriti stupnjeve u radijane:

$$\alpha' = \frac{\pi}{180} * \deg \alpha.$$

Zatim, rotirati točku za određeni kut:

$$X' = X * \cos \alpha + Y * \sin \alpha$$

$$Y' = -Y * \sin \alpha + X * \cos \alpha$$

Još je preostala translacija:

$$X' = X - h$$

$$Y' = Y - k$$

U ovom primjeru potrebno je koristiti kut koji zatvara orijentacija uređaja s obzirom na sjever poništen za orijentaciju prostora. Isto tako za translaciju potrebna je izvorišna koordinata. Uz sve ovo navedeno sada možemo prikazati funkciju koja radi konverziju. Funkcija rezultira s novim koordinatama. Prostori mogu imati različite veličine koordinate, a u 3D prostoru koordinata uvijek pokriva 1m, stoga je potrebno dodatno prilagoditi koordinate tako da se podijele s veličinom koordinate trenutnog prostora.

Listing 6.17: "Funkcija za rotiranje i translaciju točaka u koordinatnom sustavu"

```
pub fn convert_to_new_cartesian(coordinate: Coordinates, compass: f32, source_x: f32
    , source_y: f32, space_compass: f32, space_coord_size: f32) ->
    CoordinateResponse{
    let old_x = coordinate.x as f32;
    let old_y = coordinate.y as f32;

    let rad  = std::f32::consts::PI / 180.0 * (360.0 - (compass - space_compass));

    let rotated_source_x = source_x * rad.cos() + source_y * rad.sin();
    let rotated_source_y =  (-1.0 * source_x) * rad.sin() + source_y * rad.cos();

    let rotated_x = old_x * rad.cos() + old_y * rad.sin();
    let rotated_y = (-1.0 * old_x) * rad.sin() + old_y * rad.cos();

    let rotated_translated_x = rotated_x - rotated_source_x;
    let rotated_translated_y = rotated_y - rotated_source_y;

   CoordinateResponse{
       x: rotated_translated_x * space_coord_size,
       y: rotated_translated_y * space_coord_size
   }
}
```

Zahtjev korisnika mora pružiti sve potrebne informacije za lokalizaciju, za podatke o prostoru, za podatke o odredišnom POI, orijentaciju uređaja i naziv korisnika.

Listing 6.18: "Zahtjev za navigaciju"

```
{
    "id" : "iPhone 7",
    "compass": 198,
    "space" : 21,
    "destination_poi" :55,
    "source" : [
        ...
        {
            "id_ble": 2,
            "rssi": -78
        },
        ...
    ]
}
```

Zahtjev za navigaciju rezultirati će konvertiranim koordinatama prilagođenim za orijentaciju i poziciju mobilnog uređaja. U većini slučajeva to će izgledati kao na primjeru.

Listing 6.19: "Primjer odgovora"

```
[
    ...
    {
        "x": -0.078504086,
        "y": 1.4979444
    },
    {
        "x": 0.4208107,
        "y": 1.5241127
    },
    {
        "x": 0.39464283,
        "y": 2.0234275
    },
    ...
]
```

6.2. Frontend sustav

6.2.1. State management

Kod izgradnje web aplikacije uz pomoć nekih modernijih razvojnih okvira javlja se pitanje upravljanja stanjem. Većina takvih razvojnih okvira gradi aplikaciju pomoću jednostavnih komponenti koje sadrže način na koji će se prikazati, stanje komponente, te definirano ponašanje kojim se upravlja stanjem. Najpoznatiji web razvojni okviri su Angular, Vue i React, no mi ćemo koristiti Flutter.

Stanje jedne komponente može sadržavati npr. popis svih stavki, ili brojač, ili vrijednosti širine i dužine i slično. Iz iskustva znamo da su današnje web aplikacije vrlo kompleksne i zahtjevne, stoga su performanse u pitanju. Zato ovakvi okviri omogućuju, umjesto osvježavanja cijele stranice s novim vrijednostima stanja, osvježavanje pojedinačne komponente bez osvježavanja stranice. Ako ste se ikada susreli s ovim načinom rada, znate da upravljanje stanjem postaje eskponencijalno teže u odnosu na rast projekta. Zato se koriste paketi treće strane.

Flutter ima broj takvih paketa, od kojih su neki Provider, Bloc, Redux GetIt, GetX, Riverpod, dok je bazni način upravljanja pod nazivom setState. [23] Mi ćemo koristiti GetX jer je dosta pojednostavljen i dovoljan za potrebe projekta. GetX ima 3 ključne značajke, a to su state management visokih performansi, injekcija ovisnosti (eng. dependency injection) i upravljanje rutama. Potrebno je napomenuti da aplikacije izgrađene ovakvim okvirima koriste interno upravljanje rutama. [24]

Način rada je da se implementira Model-View-Controller arhitektura. Stanje i ponašanje komponente se prebacuje u Controller, dok u View ostaje samo definiranje izgleda. Controller sve elemente stanja izlaže prema View kao Observable, a View se tako pretplaćuje na pojedini element iz stanja. View može pokrenuti izmjenu stanja putem događaja, a definirane izmjene stanja postoje unutar Controller-a. Svaka izmjena stanja događa se u Controlleru nad Observable elementima koje onda pokreću osvježavanje View komponente koja je pretplaćena. Controller je u komunikaciji s Model-om i on poziva sve izmjene i dohvaćanje podataka.

6.2.2. Struktura projekta

Projekt je strukturiran na način da prati MVC arhitekturu. View se sastoji od stranica i komponenata. Imamo 2 stranice, početnu stranicu i stranicu za detaljni prikaz prostora. Dok komponenata ima više, a tu su komponente za prikaz liste prostora, liste POI-a, komponente za unos novih prostora, POI-a i slično. Neki od tih View-ova dijele Controller-e, ovisno o kojim se podacima radi, i Model sadrži model za svaki tip podatka.

Listing 6.20: "Struktura web projekta"

```
    -lib
        -controllers
            -beacons.dart
            -pois.dart
            ...
        -models
            -beacons.dart
            -coordinates.dart
            ...
        -pages
            -home.dart
            -spaceDetail.dart
        -widgets
            -beaconAdd.dart
            -beaconList.dart
            -poiAdd.dart
            -poiList.dart
            ...
        main.dart
```

6.2.3. Pokretanje aplikacije

U Flutter-u, uvijek se prvo pokreće main() metoda u datoteci main.dart u kojoj se inicijalizira početna stranica. Odmah možemo iskoristiti 'dependency injection' što nam pruža GetX i inicijalizirati sve Controller klase. Controller postaje dostupan u svim komponentama koji se nalaze ispod komponente unutar koje je Controller inicijaliziran. Znači da će svi Controller-i biti dostupni u cijeloj aplikaciji ako ih se odmah inicijalizira.

Listing 6.21: "Korijenska komponenta Web projekta"

```
class MyApp extends StatelessWidget {
  final spacesController = Get.put(SpacesController());
  final poisController = Get.put(PoisController());
  final beaconsController = Get.put(BeaconsController());
  final spaceGridController = Get.put(SpaceGridController());
  final locationsController = Get.put(LocationController());

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      debugShowCheckedModeBanner: false,
      title: 'NavINAR',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ),
      home: MyHomePage(title: 'Navigate Indoors using Augmented Reality'),
    );
  }
}
```

Dočekati će nas početna stranica MyHomePage koja je odgovorna za prikaz liste prostora i za dodavanje novih. Iako su to 2 različite komponente, 2 različita View-a, koriste isti tip podataka pa će i dijeliti Controller. Controller ima svoj životni ciklus, pa možemo već kod inicijalizacije Controller-a izvršiti neke radnje, a to će biti dohvaćanje podataka sa poslužitelja. Stanje se označava s .obs kao 'Observable', a stanje čuva podatke o prostorima koje je na početku prazno, no inicijalizacijom i dohvaćanjem prostora mijenjamo stanje na novo.

Listing 6.22: "Primjer Controller-a za podatke o prostorima"

```
class SpacesController extends GetxController {
  RxList<Space> spaces = List<Space>.empty().obs;
  // Other state
  // ..

  @override
  void onReady() {
    getSpaces();
    super.onReady();
  }

  Future<void> getSpaces() async {
    final response = await http.get(
        Uri.parse(
            "http://${dotenv.env['IP_ADDR']}:${dotenv.env['PORT']}/spaces"),
        headers: {
          "Accept": "application/json",
          "Access-Control-Allow-Origin": "*"
        });

    final items = json.decode(response.body).cast<Map<String, dynamic>>();
    List<Space> newSpaces = items.map<Space>((json) {
      return Space.fromJson(json);
    }).toList();

    spaces(newSpaces);
  }
  // Other behavior
  // ...
 }
```

Unutar View-a (sada promatramo komponentu SpacesList), potrebno je dohvatiti Controller koji smo injektirali u korijenskoj komponenti. Koristeći GetX to je vrlo jednostavno pozivajući Get.find() uz obavezno definiranje klase koju tražimo. Zatim, potrebno je pretplatiti se na stanje. Ako je dio komponente promjenjiv, odnosno ako joj se stanje mijenja, potrebno je omotati (eng. wrap) tu komponentu s ObX komponentom. U tijelu ObX komponente potrebno je samo iskoristiti stanje bilo kojeg Controller-a i GetX će prepoznati pretplatu i automatski osvježiti potrebne dijelove View-a.

Listing 6.23: "Primjer Controller-a za podatke o prostorima"

```
class SpacesList extends StatelessWidget {
  final SpacesController controller = Get.find();
 // Other controllers

  @override
  Widget build(BuildContext context) {
    return Center(
      child: Container(
        child: Obx(() {
          return ListView(
            children: [
              ...controller.spaces
                  .map(
                    (e) => SpaceItem(e)
                  )
                  .toList(),
              SpaceAdd(),
            ],
          );
        }),
      ),
    );
  }
}
```

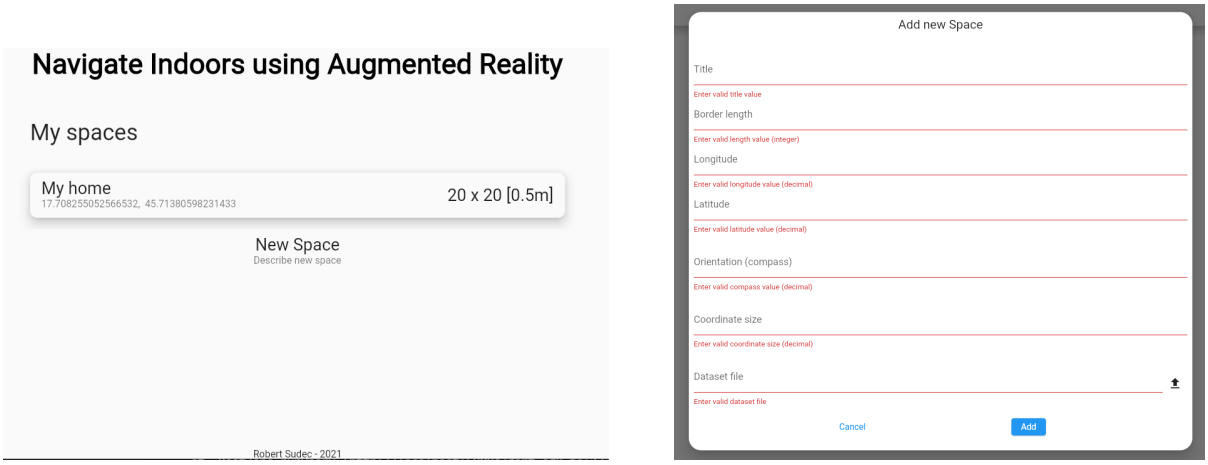
Nakon prikaza liste prostora, dolazi komponenta SpaceAdd pomoću koje možemo kreirati novi prostor. Prikazuje se forma s potrebnim poljima za naslov, dužini, geolokaciju, orijentaciju prostora i vezani skup podataka. Skup podataka potrebno je prenijeti na poslužitelj kako bi tamo bio spreman za kasnije korištenje. Korisnik će datoteku odabrati putem FilePicker-a, a otvara se nativni File Explorer ovisno o sustavu. Datoteka se prenosi putem HTTP Multipart zahtjeva. Klikom na jedan od prostora, čuvamo stanje o odabranom prostoru, te dohvaćaju se ostali podaci vezani za prostor, točke od interesa, koordinate i BLE odašiljači. Pokreće se detaljna stranica jednog prostora.

6.2.4. Upravljanje prostorom

Kod kreiranja novog prostora, automatski mu se dodaje jedan bazni POI, te se kreiraju koordinate ovisno o unesenoj dužini. Ako je dužina 20, stvara se kvadratni prostor 20x20. Zatim se svaka koordinata na početku veže za bazni POI koji ima isti naziv kao i prostor.

Ova stranica sastoji se od komponenti prikaza prostora SpaceGrid, gdje je svaka koordinata zasebna komponena CoordinateTile, prikaza popisa BLE odašiljača, dodavanje BLE odašiljača, komponenta za odabir strane zida, komponenta za prikaz popisa POI-a i komponenta za dodavanje POI.

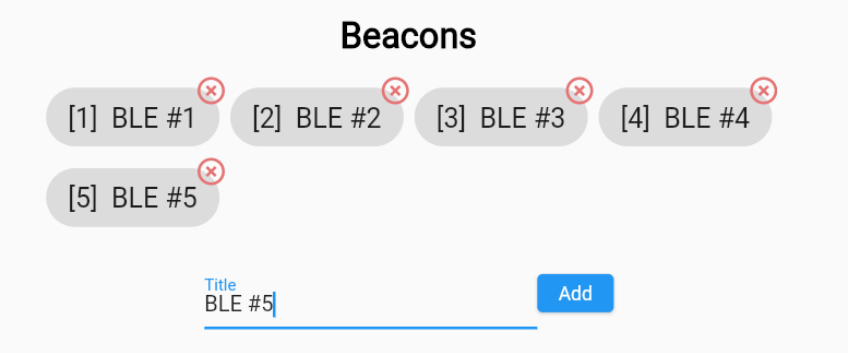
BLE odašiljači dodaju se dinamički i vežu za prostor kako bi mobilna aplikacija mogla znati koje BLE uređaje treba slušati. Redoslijed imena BLE uređaja mora biti jednak kao u skupu



(a) Popis prostora (b) Dodavanje novog prostora

Slika 12: Početna stranica (Izvor: Autorski rad)

podataka kako ne bi došlo do krivih rezultata. Ova komponenta ima jednostavan Controller koji čuva samo listu objekata, a objekte možemo dohvatiti, dodati i brisati.



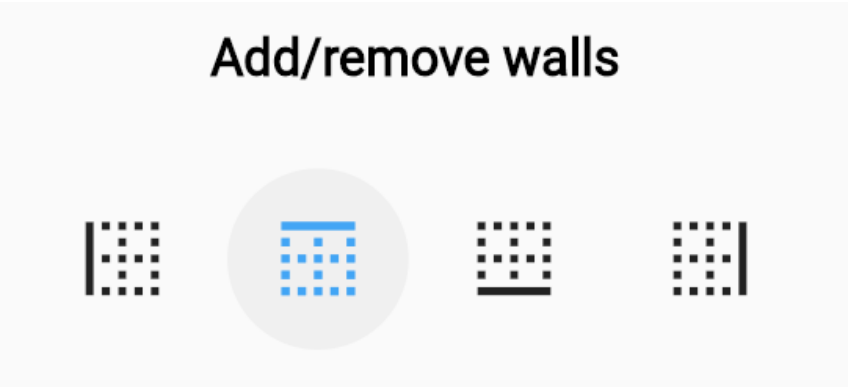
Slika 13: Komponente vezane uz BLE uređaje (Izvor: Autorski rad)

View za prikaz jednog BLE odašiljača sastoji se od Stack komponente koja omogućuje da se njena djeca mogu preklapati i apsolutno pozicionirati. Tako se Chip komponenta koja sadrži naziv preklapa s IconButton komponentom za brisanje. Također, lista svih BLE naziva omotana je s komponentom Wrap, koja će se pobrinuti da djeca te komponente budu pravilno prikazana tako što im dozvoljava da prijeđu u novi red ako nema dovoljno mjesta u prvom redu što nam pomaže kod responzivnosti.

Listing 6.24: "Komponenta prikaza naziva BLE uređaja"

```
Obx(() {
    return Wrap(
        children: [
            ...beaconsController.beacons
                .map((
                    e,
                ) =>
                    Stack(
                        children: [
                            BeaconItem(e)
                        ],
                    ),
            ),
            .toList(),
        ],
    );
}
```

Odabir zidova je komponenta koja sadrži 4 ikone s mogućnošću odabira. To su lijevi, gornji, donji i desni zid. Koristi se za dodijeljivanje zidova pojedinačnoj koordinati što onda zabranjuje da algoritam za računanje puta prolazi kroz zidove.



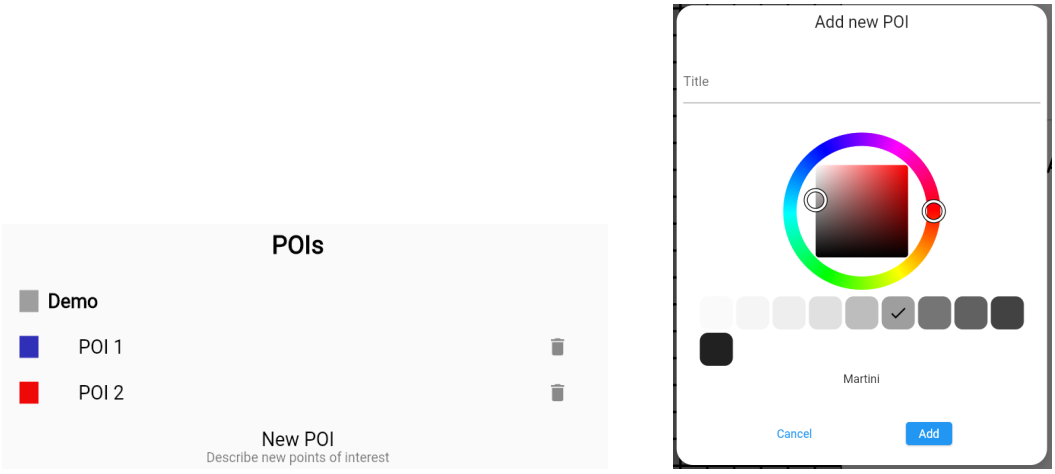
Slika 14: Komponente vezane za upravljanje zidovima (Izvor: Autorski rad)

Kod ove komponente potrebno je jedino čuvati stanje o odabranom zidu. Stranu zida opisivati ćemo enumeracijom WallSide.

```
enum WallSide { left, top, right, bottom, none }
```

Popis POI sastoji se od 2 komponente, same liste točaka od interesa i komponente za dodavanje. Kod dodavanja novog POI potrebno je samo odabrati naslov i boju za prikaz. Nema se što previše reći, potrebno je čuvati stanje o odabranom POI-u, a implementaciju liste i forme za dodavanje smo već vidjeli.

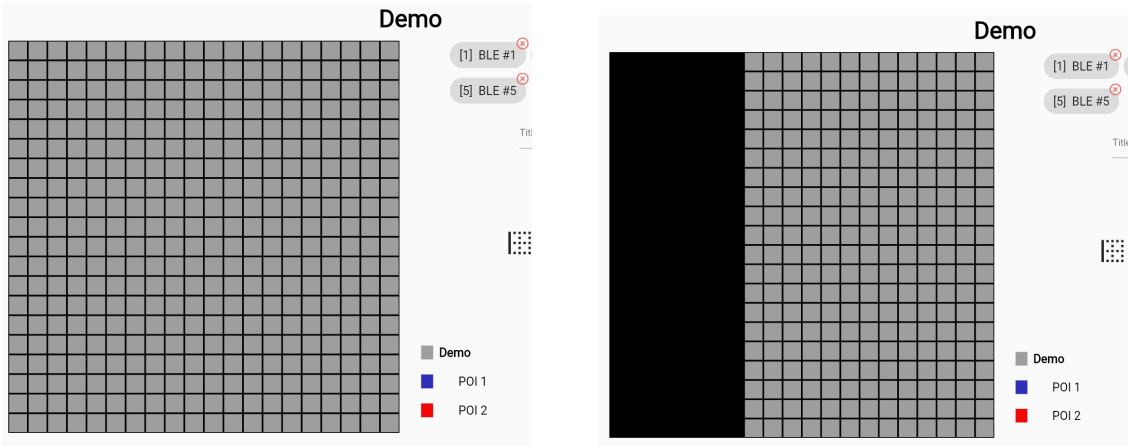
Mreža prostora (SpaceGrid) zahtijeva slijedeće konfiguriranje. Sustav će prostor definirati kao kvadratni, tako da administrator prostora mora oblikovati svoj prostor (koji vjerojatno nije



(a) Popis POI (b) Dodavanje novog POI

Slika 15: Komponente vezane za POI (Izvor: Autorski rad)

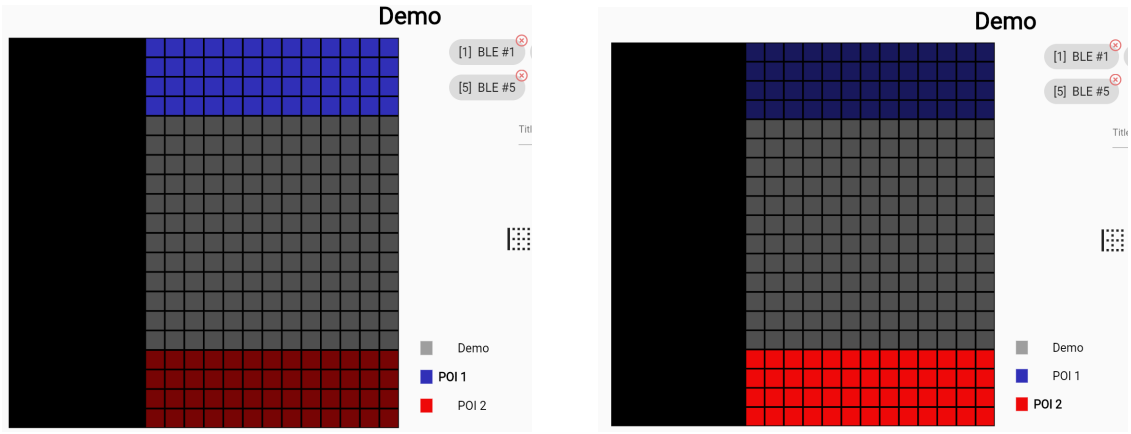
kvadratni) na način da blokira koordinate. Koordinate se blokiraju na način da se odabere bazni POI, te klikom na koordinatu šalje se zahtjev na server za ažuriranje te koordinate, postavljanjem atributa blocked na true vrijednost. Isto tako se koordinata može odblokirati. Prikazani su koraci konfiguracije na Demo prostoru. Uzeti u obzir da je ishodište (0,0) donji lijevi kut.



(a) Početno stanje SpaceGrid (b) Stanje nakon blokiranja koordinata

Slika 16: Adaptacija kvadratnog prostora u željeni pravokutni(Izvor: Autorski rad)

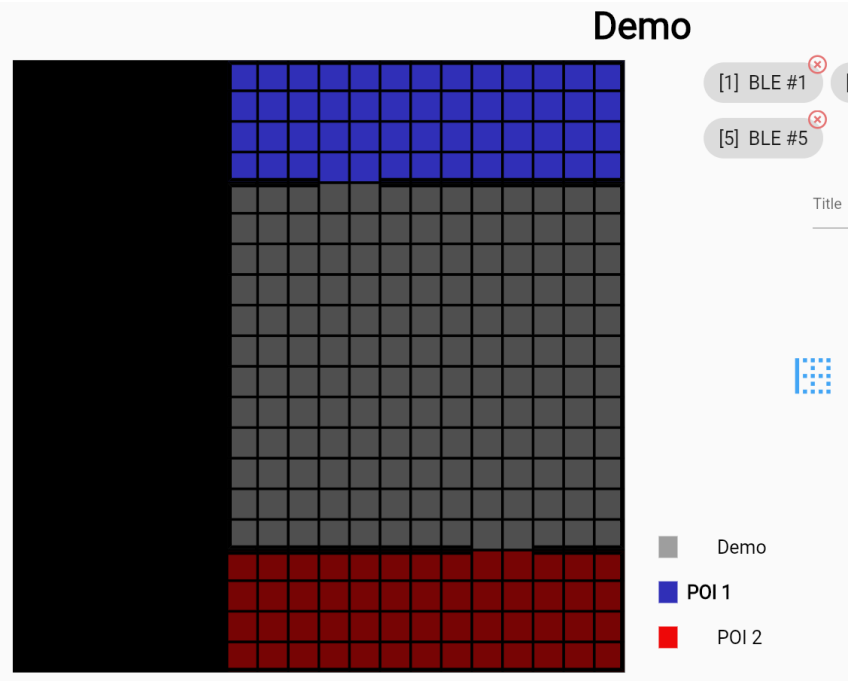
Zatim, potrebno je dodijeliti koordinate za svaki željeni POI. Odvija se na sličan način kao i blokiranje samo je potrebno označiti željeni POI iz popisa i označiti koordinate koje želimo. Klikom na koordinatu šalje se zahtjev na server za ažuriranje koordinate, odnosno ažuriranje polja idpoi na željeni POI. Ako želimo odvojiti koordinatu od trenutnog POI-a, klikom se koordinata vraća na bazni POI. Boja odabranog POI-a će se istaknuti, dok će se ostale boje prigušiti.



(a) Definiranje POI 1 (b) Definiranje POI 2

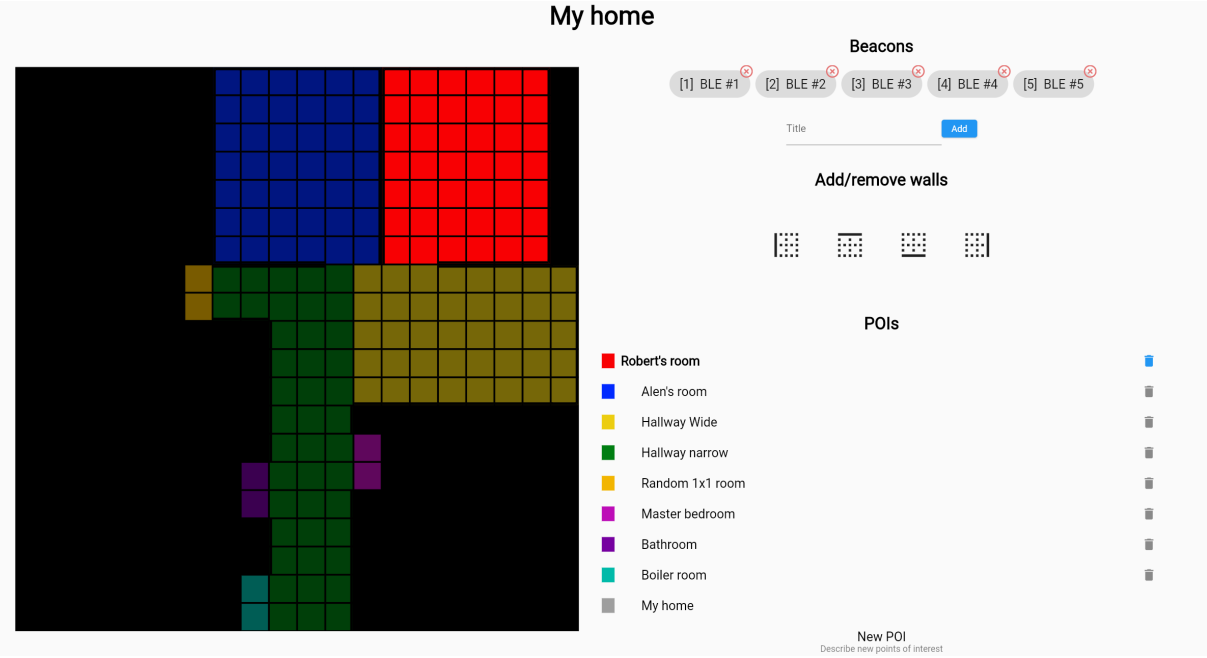
Slika 17: Definiranje POI 1 i POI 2(Izvor: Autorski rad)

Potrebno je još 'nacrtati' zidove kako bi algoritam dao ispravan put. Radi se na isti način kao i prethodne konfiguracije. Odabire se strana zida i klikom na koordinatu se dodaje isti. Potrebno je dodati zidove objema koordinatama između kojih se zid nalazi. Zid u prikazu je implementiran kao Border, rub kontejnera.



Slika 18: Dodani zidovi na prostor (Izvor: Autorski rad)

My Home My Home je prostor koji je primjer u ovom radu, a sastoji se od 8 točaka od interesa.



Slika 19: Konfiguracija prostora My Home (Izvor: Autorski rad)

6.2.5. Live location

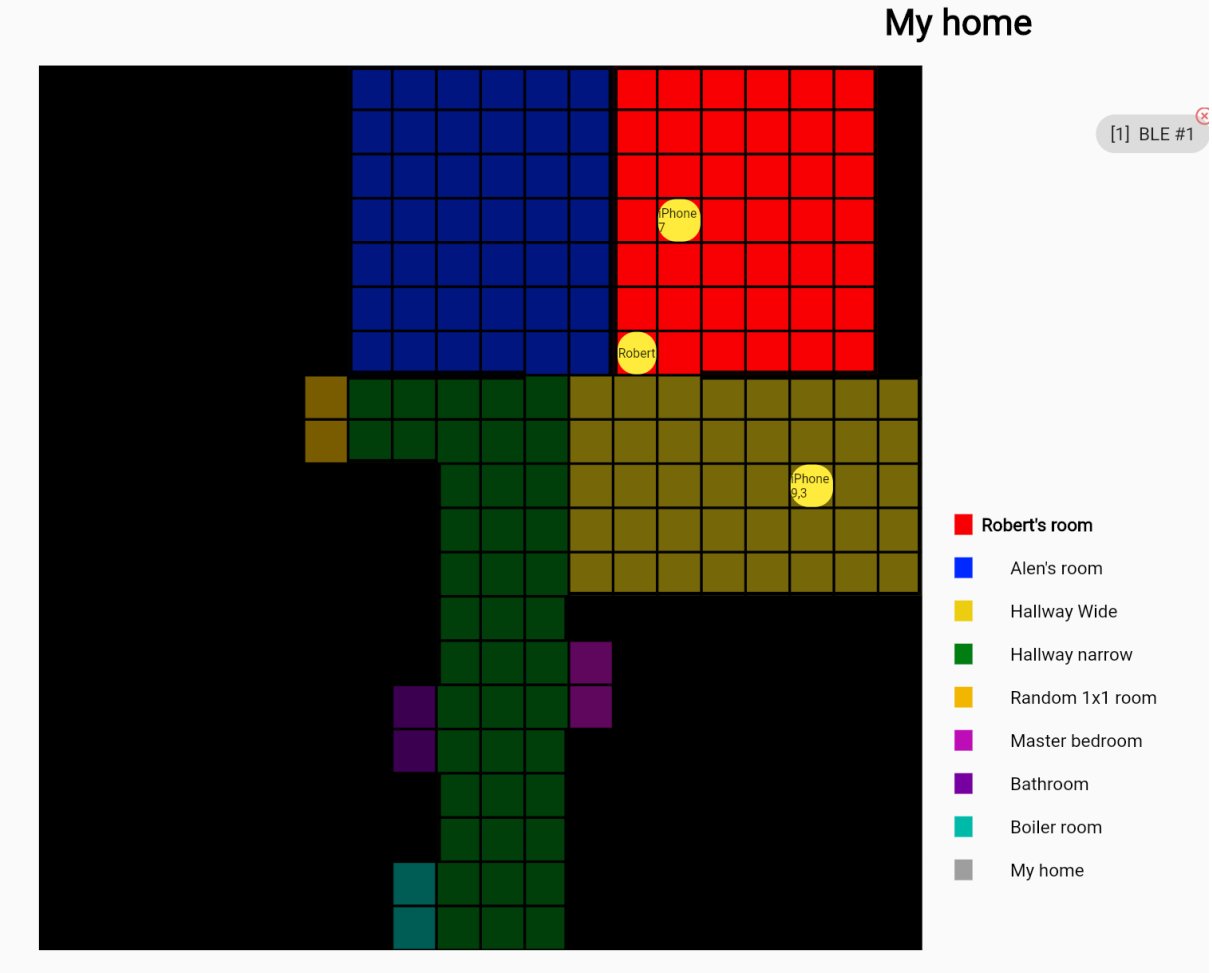
U zadnjem dijelu Frontend sustava dolazi WebSocket komunikacija i prikaz lokacije korisnika. Potreban nam je poseban Controller koji čuva listu lokacija svih spojenih korisnika kao svoje stanje. Kao i ostali modeli potrebna je deserijalizacija iz JSON oblika u Dart objekte.

Listing 6.25: "Prikaz Dart klase s (de)serijalizacijom"

```
class Location {
  final int? x;
  final int? y;
  final String? name;

  Location({this.x, this.y, this.name});

  factory Location.fromJson(Map<String, dynamic> json) {
    return Location(
      x: json['x'],
      y: json['y'],
      name: json['name'],
    );
  }

  Map<String, dynamic> toJson() => {
        'x': x,
        'y': y,
        'name': name,
      };
}
```

LocationController se inicijalizira pokretanjem aplikacije te se spaja na WebSocket endpoint koji smo definirali. Odmah se i registrira callback funkcija za prijem poruke u kojoj stvaramo objekt iz JSON reporezentacije. Ako već postoji lokacija s istim imenom, odnosno od istog korisnika, tu lokaciju ažuriramo, a ako ne postoji onda ju samo dodajemo. Funkciju definiramo kroz metodu onInit() životnog ciklusa GetXController-a.

```
@override
void onInit() {
    webSocket.onMessage.listen((MessageEvent e) {
        var location = Location.fromJson(jsonDecode(e.data));
        locations.removeWhere((element) => element.name == location.name);
        locations.add(location);
    });
    super.onInit();
}
```



Slika 20: Prikaz lokacije korisnika (Izvor: Autorski rad)

6.3. Mobilna aplikacija

Mobilna aplikacija usmjerena je na krajnjeg korisnika koji će korištenjem aplikacije doći do željenog odredišta. Izgrađena je za korisnike iOS i Android uređaja.

6.3.1. Struktura projekta

Korišten je Flutter, zajedno s GetX state management-om što je opisano u dijelu razvoja frontend sustava. Aplikacija ima 2 stranice, početnu stranicu i AR prikaz.Koristi se mali broj komponenata, dok se više koristi programska logika.

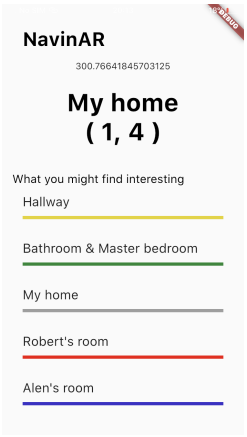
Listing 6.26: "Struktura mobilne aplikacije"

```
-models
-lib
    -controllers
        -ARController.dart
        .BLEController.dart
        -CompassController.dart
        -HomePageController.dart
    -models
        -ARCoordinate.dart
        -BLEReading.dart
        -Ble.dart
        ...
    -pages
        -HomePage.dart
        -ARPage.dart
    main.dart
```

6.3.2. Početne aktivnosti

Proučili smo način na koji GetXController funkcionira, pa znamo da se kontroleri injektiraju već pri otvaranju aplikacije. U ovoj situaciji, ARController nećemo odmah inicijalizirati jer su nam potrebni prvo podaci koji ovise o odabiru korisnika.

Učitava se HomePage na kojemu su prikazane trenutne koordinate uređaja u prostoru, orijentacija uređaja (kompas) i popis POI-a, ali prvo podaci se trebaju dohvatiti.



Slika 21: Početna stranica mobilne aplikacije (Izvor: Autorski rad)

HomePageController čuva stanje o prostoru u kojemu se korisnik nalazi, popisom POI tog prostora, te koji je POI odabran. Ranije je definirana potreba za geopodacima o svakom

prostoru. Dohvaćaju se podaci geografske širine i dužine pomoću paketa Geolocator, te se šalju na poslužitelj koji vraća najbliži prostor. Nastavlja se s dohvaćanjem popisa POI-a, a i BLEController kreće s radom tek kada je poznat prostor u kojem se korisnik nalazi.

Paralaleno tomu, CompassController osvježava vrijednost kompasa te čuva trenutnu orijentaciju. Jednostavnu implementaciju kompasa pruža paket flutter_compass. Potrebno je samo registrirati callback funkciju koja se poziva svakom promjenom orijentacije uređaja.

Listing 6.27: "CompassController"

```
class CompassController extends GetxController {
  RxDouble compassValue = 0.0.obs;

  @override
  void onReady() {
    FlutterCompass.events!.listen((event) {
      compassValue(event.heading);
    });
    super.onReady();
  }
}
```

6.3.3. BLE Logika

BLEController na početku ne poznaje koje odašiljače treba slušati, stoga BLEController ne kreće s radom kod inicijalizacije, nego kreće u trenutku kada su dostupni podaci o prostoru.

Metoda _getBeaconInfo() dohvaća podatke o BLE odašiljačima koji su vezani za trenutni prostor te za svaki odašiljač čuva listu vrijednosti signala unutar HashMap.

Listing 6.28: "Metoda za ručno pokretanje rada BLEController-a"

```
void start() async {
    await _getBeaconInfo();
    _startBleScan();
    _setupService();
  }
```

Koristeći flutter_reactive_ble paket za upravljanje BLE komunikacijom pokreće se skeniranje BLE uređaja. To se odvija u metodi _startBleScan() Za svaki skenirani uređaj provjerava se je li uređaj vezan za ovaj prostor i sprema se jačina signala u listu vezanu za taj uređaj. BLE uređaji oglašavaju se svakih 200ms u prosjeku a čuvamo zadnjih 10 pročitanih RSSI vrijednosti za svaki od tih uređaja. To nam pruža dodatne mogućnosti prilagođavanja odabranih RSSI vrijednosti. Zbog mogućih velikih odstupanja BLE signala iz bilo kojeg razloga, može se npr. izračunati prosječni RSSI u zadnjih 10 kako bi se to odstupanje minimiziralo.

Zadnja metoda kod pokretanja BLEControllera je _setupService() koja nam registrira funkciju koja se izvodi periodično. Ta funkcija će preuzeti vrijednosti iz HashMape za svaki BLE uređaj, preuzeti ime uređaja te slati zahtjev na poslužitelj. Kada poslužitelj vrati odgovor ažurira se stanje BLEControllera, odnosno stanje trenutne koordinate koja se onda osvježava na HomePage.

6.3.3.1. Analiza preciznosti

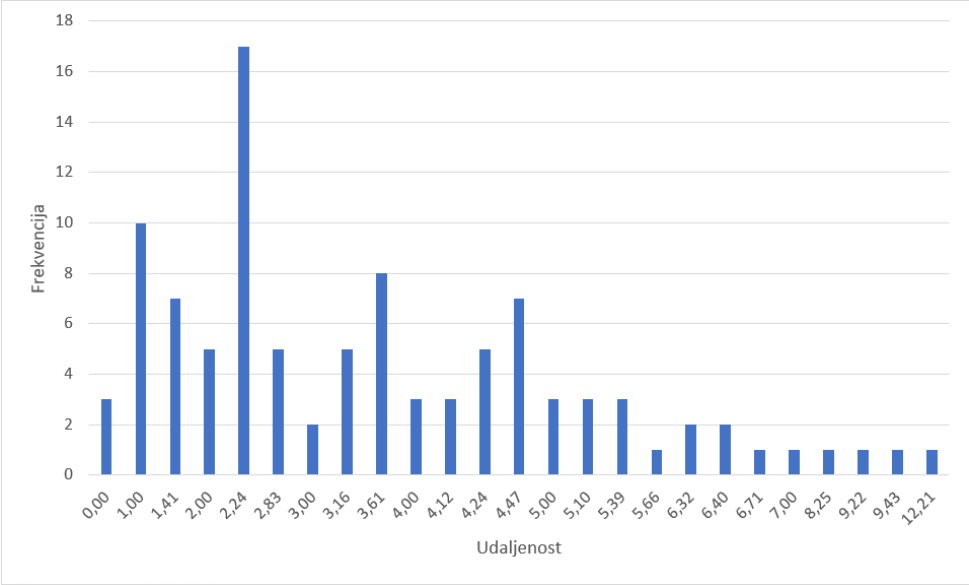
Provedena je analiza preciznosti implementirane lokalizacije. Kao što je rečeno, usporediti ćemo rezultate lokalizacije na 3 prostora s koordinatama različite širine. Osoba koja testira posjetiti će nasumično određenu koordinatu te započeti test lokalizacije. Test je osmišljen na način da se na svakoj koordinati provede lokalizacija 6 puta. Skeniranje RSSI BLE uređaja odvijati će se konstantno dok koristimo 2 načina uzimanja relevantnih RSSI vrijednosti. Prvi način je uzimanje najskorijeg, najnovijeg očitanja za svaki BLE uređaj, a drugi način je uzimanje prosječnog RSSI u zadnjih 10 očitanja. Za svaki od 2 načina mjerenje lokalizacija se provodi 3 puta, odnosno u trenutku dolaska osobe na koordinatu, 1 sekunda stajanja u koordinati i 2 sekunde stajanja u koordinati. Uzorak čini 100 koordinata za svaki prostor.

Rezultati će prikazivati prosječnu i maksimalnu udaljenost od stvarne do lokalizirane koordinate. Broj koordinata na kojima je lokalizacija bila uspješna (udaljenost je jednaka 0), te broj koordinata u kojima je udaljenost bila manja ili jednaka 1, 2 ili 4 koordinate. Zadnji podatak je standardna devijacija udaljenosti, odnosno prosječno odstupanje od prosjeka.

Koordinata 0.5m x 0.5m. Prosječna udaljenost od stvarne do lokalizirane koordinate nalazi se u rasponu [3.4, 3.94], odnosno [1.7m, 1.97m], dok je maksimalna udaljenost u rasponu [10.7, 18.9], odnosno [5.35m, 9.45m]. Vidimo da su rezultati u slučaju promatranja prosječnog RSSI-a lošiji u svim vremenima mjerenja. Rezultati pokazuju da se u 3-8% slučajeva dogodi ispravna lokalizacija, a samo 60-65% slučajeva ostvaruje lokalizaciju unutar 4 koordinate, odnosno 2 metra. U najboljem slučaju (Last known RSSI / 1 sec) std. devijacija iznosi 2.1, odnosno većina koordinata nalazi se u rasponu udaljenosti [1.3, 5.5] od stvarne koordinate.

	Last known RSSI			Average RSSI		
distance/t	0 sec	1 sec	2 sec	0 sec	1 sec	2 sec
Average(d)	3,51	3,4	3,52	3,63	3,53	3,94
Maximum(d)	10,7	12,21	11	11,18	15	18,9
N(d = 0)	6	3	8	4	5	4
N(d<= 1)	15	13	15	18	20	19
N(d<=2)	31	25	27	30	33	28
N(d<=4)	63	65	64	60	65	60
Standard deviation	2,37	2,1	2,35	2,48	4,28	3,02

Tablica 1: Analiza prostora s koordinatama 0.5m x 0.5m (Izvor: Autorski rad)

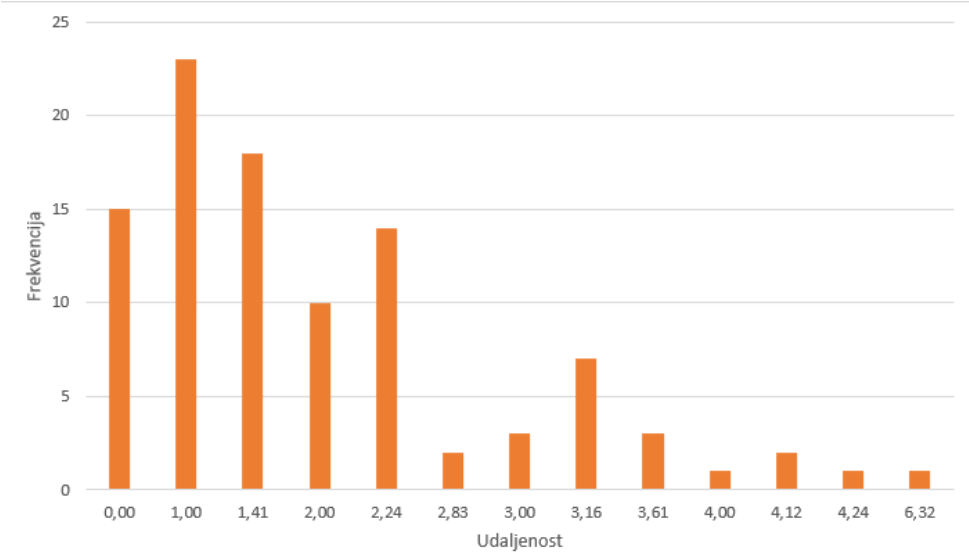


Slika 22: Prikaz frekvencije udaljenosti stvarne koordinate od lokalizirane [0.5m] (Izvor: Autorski rad)

Koordinata 1m x 1m. Povećanje širine koordinate donijelo je bolje rezultate, sada je prosječna udaljenost stvarne od lokalizirane koordinate u rasponu [1.57, 2.13], odnosno [1.57m, 2.13m]. Prosjek udaljenosti opet je lošiji u slučaju korištenja prosječnog RSSI. U ovom slučaju dolazi do ispravne lokalizacije u 7-22% pokušaja, što je veliki pomak u odnosu na prvi slučaj, no još uvijek nije dovoljno za korištenje. Za najbolji rezultat (Last known RSSI / 2 sec), standardna devijacija iznosi 1.19, što nam govori da su rezultati manje raspršeniji nego u prošlom slučaju, odnosno da su vrijednosti bliže prosjeku.

	Last known RSSI			Average RSSI		
d/t	0 sec	1 sec	2 sec	0 sec	1 sec	2 sec
Average(d)	1,78	1,57	1,7	2,13	2,09	2
Maximum(d)	5,83	6	6,32	5,39	6,32	6,3
N(d = 0)	14	22	15	7	14	16
N(d<= 1)	39	45	38	24	30	35
N(d<=2)	60	66	66	52	49	50
N(d<=4)	98	98	96	90	91	91
Standard deviation	1,19	1,23	1,19	1,22	1,36	1,39

Tablica 2: Analiza prostora s koordinatama 1m x 1m (Izvor: Autorski rad)

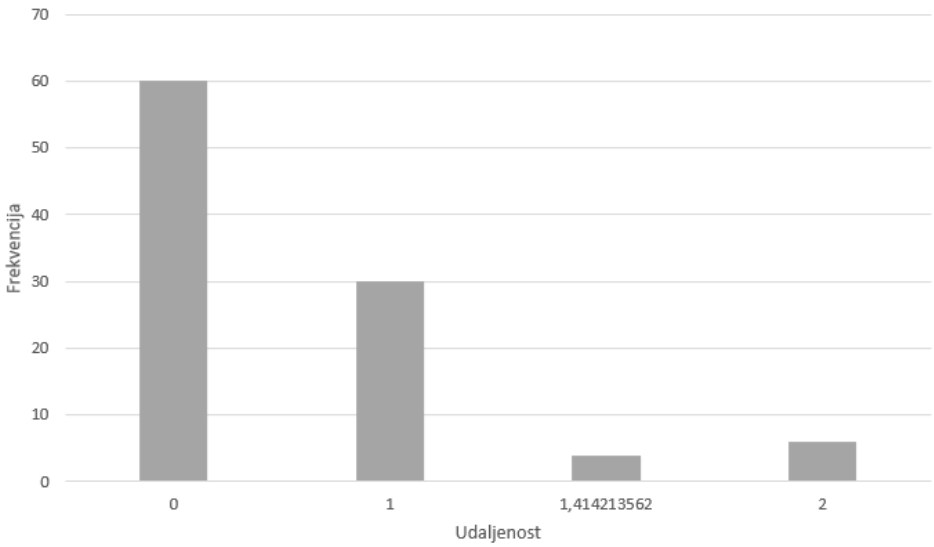


Slika 23: Prikaz frekvencije udaljenosti stvarne koordinate od lokalizirane [1m] (Izvor: Autorski rad)

Koordinata 2m x 2m Najbolji i zadovoljivi rezultati ostvareni su koordinatom širine 2m. Prosjek udaljenosti je [0.46, 0.63], odnosno [0.92m, 1.26m] što je manje od jedne koordinate. Vidimo da je i moguća udaljenost od 2+ koordinate, no vrlo rijetko iz razloga što 97-100% slučajeva se lokalizira s udaljenošću od 2 ili manje koordinate, dok u 53-65% slučajeva lokalizacija je ispravna. Prosječno odstupanje od prosječne udaljenosti smanjila se na [0.63, 0.74], što daje najmanju raspršenost rezultata do sada.

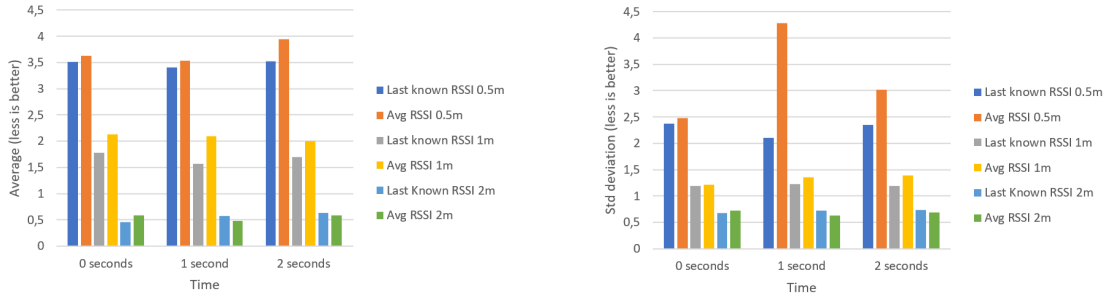
	Last known RSSI			Average RSSI		
d/t	0 sec	1 sec	2 sec	0 sec	1 sec	2 sec
Average(d)	0,46	0,57	0,63	0,59	0,48	0,59
Maximum(d)	2,24	2,24	2,24	2,24	2	2,24
N(d = 0)	65	57	53	56	60	55
N(d<= 1)	86	84	82	81	90	86
N(d<=2)	98	97	98	99	100	99
N(d<=4)	100	100	100	100	100	100
Standard deviation	0,68	0,73	0,74	0,72	0,63	0,69

Tablica 3: Analiza prostora s koordinatama 2m x 2m (Izvor: Autorski rad)



Slika 24: Prikaz frekvencije udaljenosti stvarne koordinate od lokalizirane [2m] (Izvor: Autorski rad)

Što se tiče usporedbe načina preuzimanja RSSI, najskorijeg ili prosječnog. Iz podataka se lako vidi da su rezultati bolji kada je korišten zadnji poznati RSSI. Za 14 slučajeva vrijednosti LastKnown RSSI, dok u samo 4 slučaja bolje rezultiraju vrijednosti Average RSSI.



(a) Vrijednosti varijable Average

(b) Vrijednosti varijable Std deviation

Slika 25: Usporedba LastKnown RSSI i Average RSSI (Izvor: Autorski rad)

Što se tiče vremena čitanja RSSI vrijednosti, iz priloženog se vidi da ta varijabla nije imala veliki značaj za ostvarenje najboljeg rezultata. Gledano striktno na vremena, vrijednosti dobivene u trenutku dolaska na stvarnu koordinatu su bile najbolje u 35% slučajeva, vrijednosti dobivene 1 sekundu nakon dolaska najbolje u 42.5% slučajeva, te vrijednosti 2 sekunde nakon dolaska najbolje su bile u 22.5% slučajeva.

6.3.4. Proširena realnost

Apple i Google za svoje mobilne operativne sustave koriste različite AR platforme, respektivno, ARKit i ARCore. Iz tog razloga potrebno je razviti aplikaciju za obe platforme i tu će nam pomoći paket ar_flutter_plugin. Taj paket povezuje obe platforme i pruža jedno zajedničko sučelje koje se koristi.

ARPage sastoji se od već spomenute Stack komponente tako da je pri dnu komponetna ARView, a iznad nje možemo postaviti dodatne komponente kao UI. Kod inicijalizacije ARView, kreiraju se dodatni objekti za upravljanje AR scenom. Od tih objekata najbitniji, za ovaj primjer, ARObjectManager.

Listing 6.29: "Metoda za pokretanje rada ARView komponente"

```
void onARViewCreated(
      ARSessionManager arSessionManager,
      ARObjectManager arObjectManager,
      ARAnchorManager arAnchorManager,
      ARLocationManager arLocationManager) async {
    this.arSessionManager = arSessionManager;
    this.arObjectManager = arObjectManager;

    this.arSessionManager.onInitialize(
          showFeaturePoints: false,
          showPlanes: true,
          customPlaneTexturePath: "Images/triangle.png",
          showWorldOrigin: true,
          handleTaps: false,
        );
    this.arObjectManager.onInitialize();
  }
```

Kod pokretanja ARView i nakon postupka inicijalizacije, stvara se novi koordinatni sustav (x,y,z) s ishodištem na trenutnoj poziciji uređaja. Bitno je sinkronizirati kompas i lokaciju uređaja sa stvaranjem AR koordinatnog sustava, zato se u tom trenutku inicijalizira i ARController te šalje zahtjev na poslužitelj za dohvaćanje rute. Poslužitelj nam vraća put pomoću koordinata koje su prilagođene za novi koordinatni sustav.

ARController kao stanje sadrži listu koordinata za prikaz, te listu već prikazanih objekata ARNode za lakše upravljanje.

Potrebno je još samo prikazati objekte na dobivenim koordinatama. Stvara se novi objekt ARNode s 3D modelom koji će se prikazati u prostoru. Možemo zadati proporcije i rotaciju modela, a za poziciju koristimo dobivene koordinate. Pozicija se definira kao Vector3(X,Y,Z). Vrijednost Y određuje visinu, no ona će uvijek ostati 0 što znači da će objekti biti na istoj visini kao i uređaj. Vrijednost Y iz 2D prostora postaje vrijednost (-Z) u 3D prostoru. ARNode se dodaje putem ARObjectManagera dobivenog kod inicijalizacije ARView metodom addNode(), a briše se metodom removeNode().

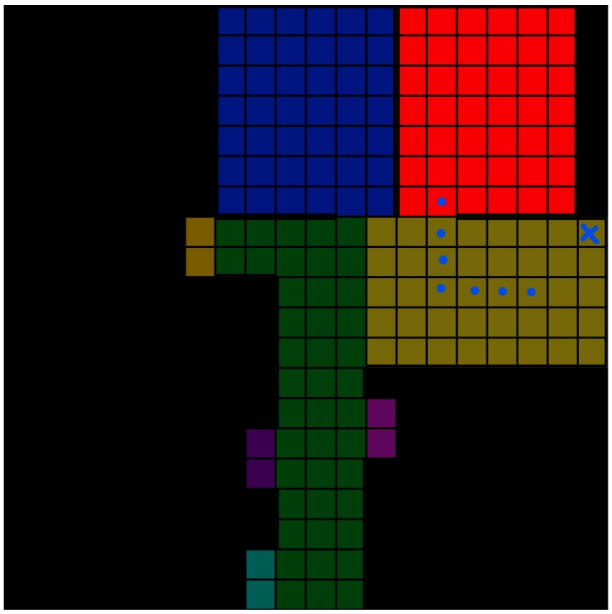
Listing 6.30: "Metode za dodavanje/brisanje ARNode objekata"

```
Future<void> attachObjects() async {
    Future.forEach<ARCoordinate>(arController.coordToShow, (element) async {
      var newCoord = ARNode(
          type: NodeType.localGLTF2,
          uri: "Models/Chicken_01/Chicken_01.gltf",
          scale: Vector3(0.2, 0.2, 0.2),
          position: Vector3(element.x!, 0.0, - element.y!),
          rotation: Vector4(1.0, 0.0, 0.0, 0.0));
      await this.arObjectManager.addNode(newCoord);
      arController.attachedNodes.add(newCoord);
    });
}
Future<void> clearObjects() async {
    for (int i = 0; i < arController.attachedNodes.length; i + 1) {
        this.arObjectManager.removeNode(arController.attachedNodes[i]);
        arController.attachedNodes.removeAt(i);
    }
}
```

6.3.4.1. Primjeri

Prikazati ćemo nekoliko primjera navigacije. Primjeri će obuhvatiti prostore sukladno s analizom, odnosno 3 prostora sa različitim veličinama koordinata (0.5m, 1m i 2m) i to na istom prostoru stvarnog svijeta (My Home).

U analizi preciznosti, doznali smo da lokalizacija na koordinatama širine 0.5 metara nije zadovoljiva, pa s time dolaze i takvi rezultati. U prvom primjeru odredište je Robert's room (crveno). Vidimo da je dobivena ruta ispravnog oblika, ali došlo je do greške u lokalizaciji. Korisnik je stajao na koordinati s oznakom X, stoga dolazi do lošeg korisničkog iskustva.

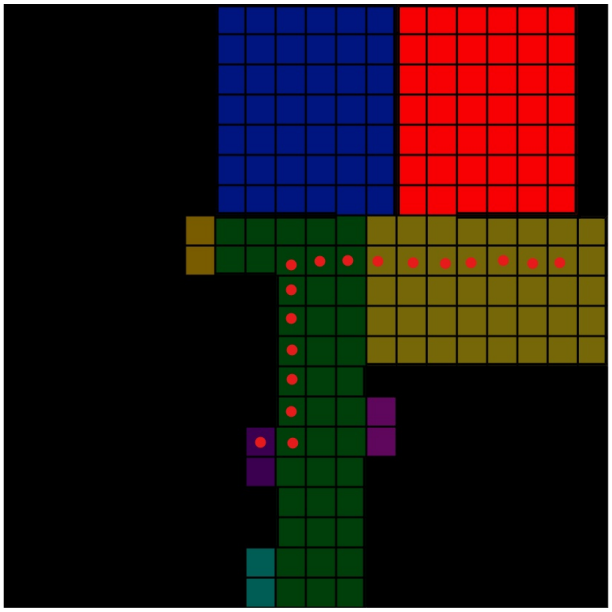


(a) Put do odredišta na tlocrtu

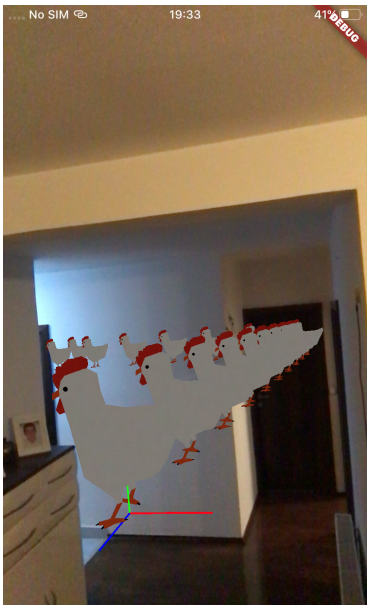
(b) Put do odredišta u AR

Slika 26: Greška lokalizacije [0.5m] (Izvor: Autorski rad)

Lokalizacija u ovom prostoru ponekad i uspije, ali rijetko. Na ovom primjeru odredište je Bathroom (ljubičasto). Postoji zanemariva greška.



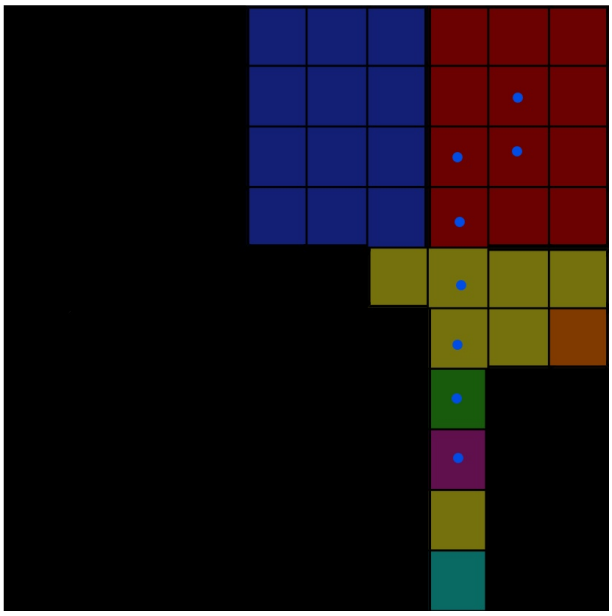
(a) Put do odredišta na tlocrtu



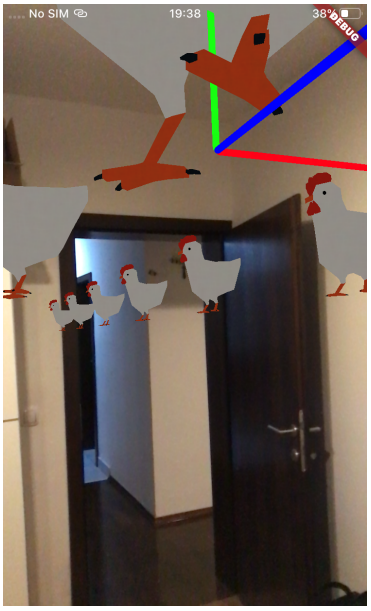
(b) Put do odredišta u AR

Slika 27: Ispravna lokalizacija [0.5m] (Izvor: Autorski rad)

Slijedi primjer za prostor s koordinatom širine 1m koji daje bolje rezultate, no šanse za grešku nisu male. Korisnik stoji unutar POI Robert's room (crveno) i kreće se prema Bathroom (ljubičasto)



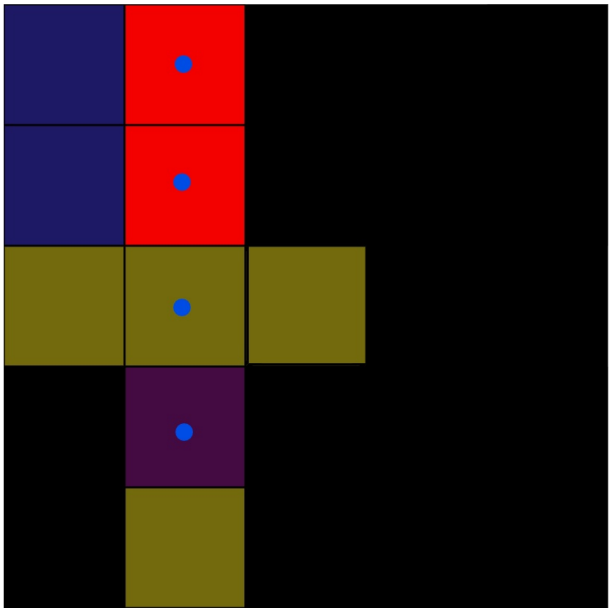
(a) Put do odredišta na tlocrtu



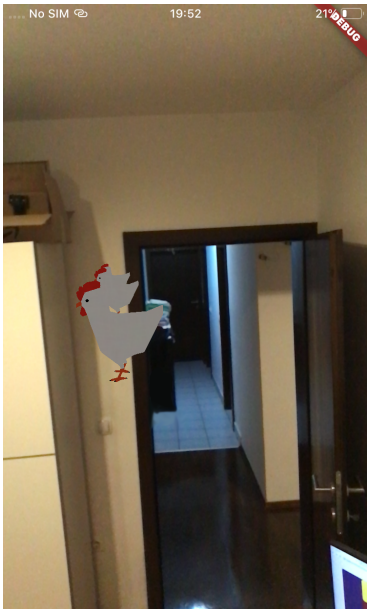
(b) Put do odredišta u AR

Slika 28: Ispravna lokalizacija [1m](Izvor: Autorski rad)

Treći prostor s koordinatom širine 2m ima najbolje rezultate što se tiče lokalizacije, no javljaju se novi problemi. Rute su ispravne, ali može doći do greške u prikazu modela. Do toga dolazi u ovom slučaju, kada su prostori uski a koordinate široke. Opet promatramo rutu od Robert's room (crveno), prema Bathroom (ljubičasto). Kako jedna koordinata zauzima 4 kvadratna metra, ne možemo sa sigurnošću znati gdje je točno korisnik unutar koordinate, pa dolazi do lošijeg korisničkog iskustva u uskim prostorima.



(a) Put do odredišta na tlocrtu



(b) Put do odredišta u AR

Slika 29: Ispravna lokalizacija [2m], greška zbog preuskog prostora (Izvor: Autorski rad)

7. Zaključak

U ovom radu odabrali smo BLE odašiljače kao trenutno najbolju/najisplativiju tehnologiju za unutarnje navigacijske sustave. Kombinacijom BLE uređaja i lokalizacijske tehnike spremanja otisaka stvorili smo lokalizacijski sustav. Sustav je započet s pretpostavkom da će sustav imati visoku razinu točnosti (>80%) s visokom razinom preciznosti na koordinatama širine 0.5m. Međutim, pretpostavka je opovrgnuta i zaključeno je da BLE lokalizacijski sustavi, za zadovoljive rezultate, moraju biti implementirani na prostorima povećih koordinata, odnosno oko 2 metra. Također se preporučuje ovakav sustav implementirati na širim prostorijama zbog pogrešaka širokih koordinata.

Kao sučelje za korištenje navigacijskog sustava, u radu je opisan način rada mobilne aplikacije koja koristi proširenu realnost za stvaranje digitalnih objekata u stvarnom svijetu kao navigacijske upute. Ovaj način sučelja je pokazao vrlo dobre rezultate ako uzmemo u obzir da se baziraju na prethodno opisanom lokalizacijskom sustavu u kojemu može doći do greške.

Što se tiče trenutne primjenjivosti, ovakav sustav bi dobro funkcionirao u prostorima poput šoping centra, velikih trgovina.

7.1. Budući razvoj

Ponekad i na prostoru koordinata širine 2 metra dolazi do pogreške. Definitivno bi trebalo pokušati smanjiti šansu greške, odnosno povećati preciznost. Kod uzimanja otisaka, osoba koja čita vrijednosti postaje prepreka između uređaja i odašiljača. Tako bi se mogla iskoristiti informacija o orijentaciji uređaja. ISto tako, u smislu lokalizacije, ako su otisci vrlo slični, sustav bi mogao prepoznati ispravnu koordinatu na temelju orijentacije.

Vezano za AR i 3D modele, optimizacija algoritma dodavanja 3D modela na način da se ne prikazuju instantno sve točke navigacije, nego da se prikazuju slijedećih nekoliko točki i ažuriraju kretanjem korisnika. S druge strane, pomoglo bi kod korisničkog iskustva, prikazati samo navigacijske točke u trenutnoj prostoriji, pa prilikom prijelaza u drugu prostoriju, prikazati nove. Tako bi se rasteretili uređaji od zahtjevnih operacija, i 3D modeli ne bi bili prikazivani kroz zidove. Zbog velikih dimenzija, u koordinati može biti velika količina korisniku zanimljivog. Korisniku bi olakšali ako bi dodali detaljniji opis i metapodatke za svaku koordinatu, pa te iste podatke prikazali u AR.

Za širenje primjenjivosti ovakvog sustava, u budućem razvoju, podržati višeetažne prostore. Takvi prostori bi sadržavali više tlocrta što bi bilo lako upravljati putem web sučelja.

Popis literature

[1] N. National Coordination Office for Space-Based Positioning i Timing, *GPS overview*. adresa: https://www.gps.gov/systems/gps/.

[2] ——, *GPS Accuracy*. adresa: https://www.gps.gov/systems/gps/performance/accuracy/.

[3] B. SIG, *Bluetooth Radio Versions*, 2021. adresa: https://www.bluetooth.com/learn-about-bluetooth/radio-versions/.

[4] N. of Aislelabs, „The Hitchhikers Guide to iBeacon Hardware: A Comprehensive Report by Aislelabs (2015),” 2015. adresa: https://www.aislelabs.com/reports/beacon-guide/.

[5] R. A. Scholtz, *USC ELECTRICAL ENGINEERING: INNOVATION AND EXCELLENCE*, 2021. adresa: https://viterbi.usc.edu/news/news/2006/usc-electrical-engineering.htm.

[6] H. S. Kenyon, *Ultrawideband, Free But Not Clear*, 2002. adresa: https://www.afcea.org/content/ultrawideband-free-not-clear.

[7] A. Mehrabi, A. Mazzoni i D. Jones, *Evaluating the user experience of acoustic data transmission*, 2019. adresa: https://link.springer.com/article/10.1007/s00779-019-01345-7.

[8] L. Mainetti, L. Patrono i I. Sergi, *A survey on indoor positioning systems*, 2014. adresa: https://ieeexplore.ieee.org/abstract/document/7039067.

[9] N. BOWDITCH, *THE AMERICAN PRACTICAL NAVIGATOR*. Bethesda, Maryland: NATIONAL IMAGERY i MAPPING AGENCY, 1995.

[10] A. Boukerche, H. A. Oliveira, E. F. Nakamura i A. A. Loureiro, „Localization systems for wireless sensor networks,” *IEEE Wireless Communications*, sv. 14, br. 6, str. 6–12, 2007. DOI: 10.1109/MWC.2007.4407221.

[11] M. Soleimanifar, X. Shen, M. Lu i I. Nikolaidis, „Applying received signal strength based methods for indoor positioning and tracking in construction applications,” 2014. adresa: https://www.researchgate.net/publication/273531227_Applying_received_signal_strength_based_methods_for_indoor_positioning_and_tracking_in_construction_applications.

[12] Floorplanner, *2D & 3D floorplans fast and easy!* Adresa: https://floorplanner.com.

[13] A. Patel, *Amit's A* Pages*. adresa: http://theory.stanford.edu/~amitp/GameProgramming/index.html.

[14] A. Lombardi, *WebSocket: Lightweight Client-Server Communications*. 1005 Gravenstein Highway North, Sebastopol, CA 89472: O'Reilly Media, Inc., 2015.

[15] T. A. Team, *Websockets*. adresa: https://actix.rs/docs/websockets/.

[16] „Augmented Reality," *Encyclopedia of Multimedia*, B. Furht, ur. Boston, MA: Springer US, 2006., str. 29–31, ISBN: 978-0-387-30038-2. DOI: 10.1007/0-387-30038-4_10. adresa: https://doi.org/10.1007/0-387-30038-4_10.

[17] Y. Chen, Q. Wang, H. Chen, X. Song, H. Tang i M. Tian, „An overview of augmented reality technology," *Journal of Physics: Conference Series*, sv. 1237, str. 022 082, lipanj 2019. DOI: 10.1088/1742-6596/1237/2/022082. adresa: https://doi.org/10.1088/1742-6596/1237/2/022082.

[18] T. D. C. Team, *Diesel*. adresa: https://diesel.rs.

[19] T. A. Team, *Actor*. adresa: https://actix.rs/book/actix/sec-2-actor.html.

[20] Y.-C. Pu i P.-C. You, „Indoor positioning system based on BLE location fingerprinting with classification approach," *Applied Mathematical Modelling*, sv. 62, str. 654–663, 2018., ISSN: 0307-904X. DOI: https://doi.org/10.1016/j.apm.2018.06.031. adresa: https://www.sciencedirect.com/science/article/pii/S0307904X18302841.

[21] Z. Zhang, „Introduction to machine learning: k-nearest neighbors," eng, *Annals of translational medicine*, sv. 4, br. 11, str. 218–218, lipanj 2016., atm-04-11-218[PII], ISSN: 2305-5839. adresa: https://pubmed.ncbi.nlm.nih.gov/27386492.

[22] samueltardieu, *pathfinding*. adresa: https://crates.io/crates/pathfinding.

[23] flutter-dev@, *List of state management approaches*. adresa: https://flutter.dev/docs/development/data-and-backend/state-mgmt/options.

[24] getx.site, *get*. adresa: https://pub.dev/packages/get.

Popis slika

Popis tablica