Diplomski studij

Informacijska i komunikacijska tehnologija

Računarstvo

Telekomunikacije i informatika
Obradba informacija
Računalno inženjerstvo

Internet stvari

Home observation systems - HOS

Projekt

Lovro Furač

Tvrtko Ivasić

Vlaho Ljubić

Andro Malobabić

Lovro Perković

Dalijo Vorkapić

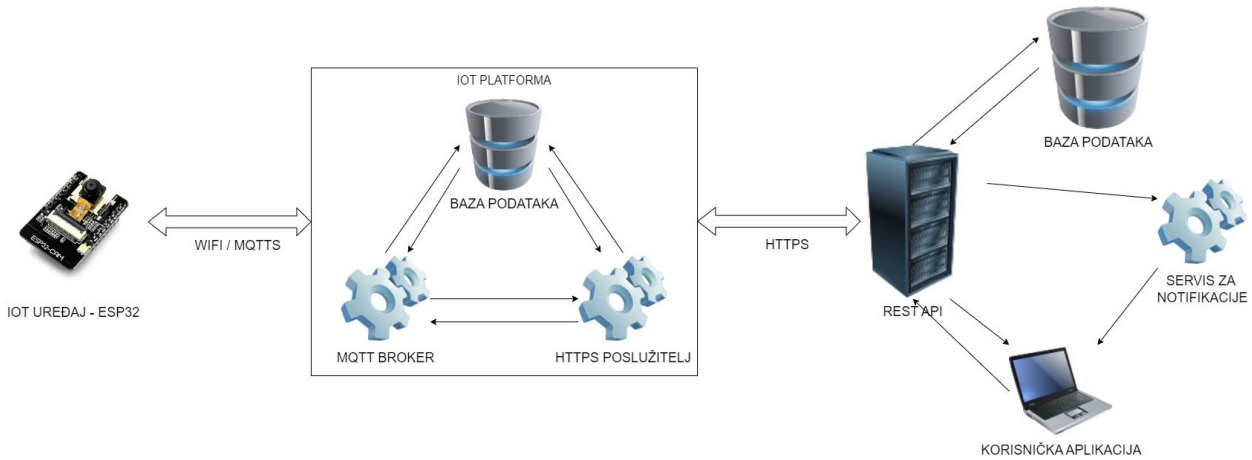
Ak.g. 2022./2023.

Sadržaj

1. Uvod

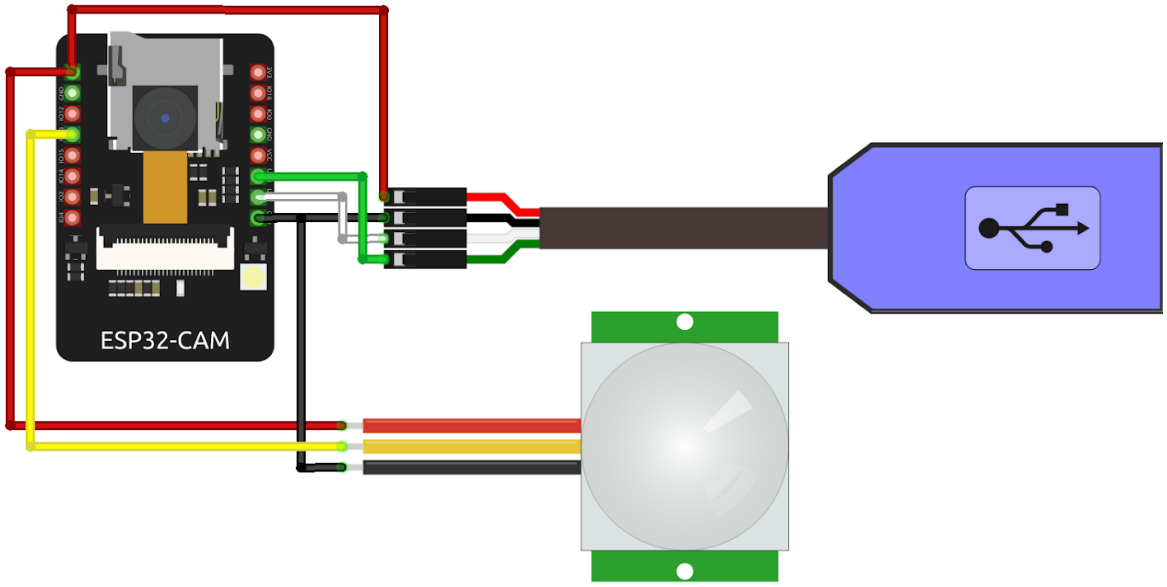
Razvijeno rješenje služi kao sustav za osiguranje imanja i okućnice korištenjem detektora pokreta i kamera. Jedinice za detekciju i nadzor postave se u okolinu vanjskog prostora i preko WiFi mreže šalju podatke na korisničku aplikaciju. U slučaju kad senzor na jedinici detektira pokret, kamera zabilježi sliku područja u kojem je došlo do aktivacije senzora i šalje sliku korisniku. Korisnik ima mogućnost pregledati slike koje su nastale u trenutku kretnje na postavljenoj lokaciji i na taj način mu je omogućen nadzor imanja u stvarnom vremenu. Pasivni infracrveni senzor (PIR) za detekciju pokreta spojen je s ESP32-CAM mikro kontrolerom koji ima ugrađeni modul za kameru. Mikro kontroler ima i WiFi modul, stoga je moguće slanje podataka direktno preko WiFi veze na udaljenog poslužitelja. Slični projekti su sustavi za nadzor unutar kuće, sustavi detekcije pokreta i obavještavanja korisnika pomoću push obavijesti ili alarmom.

2. Opis rješenja



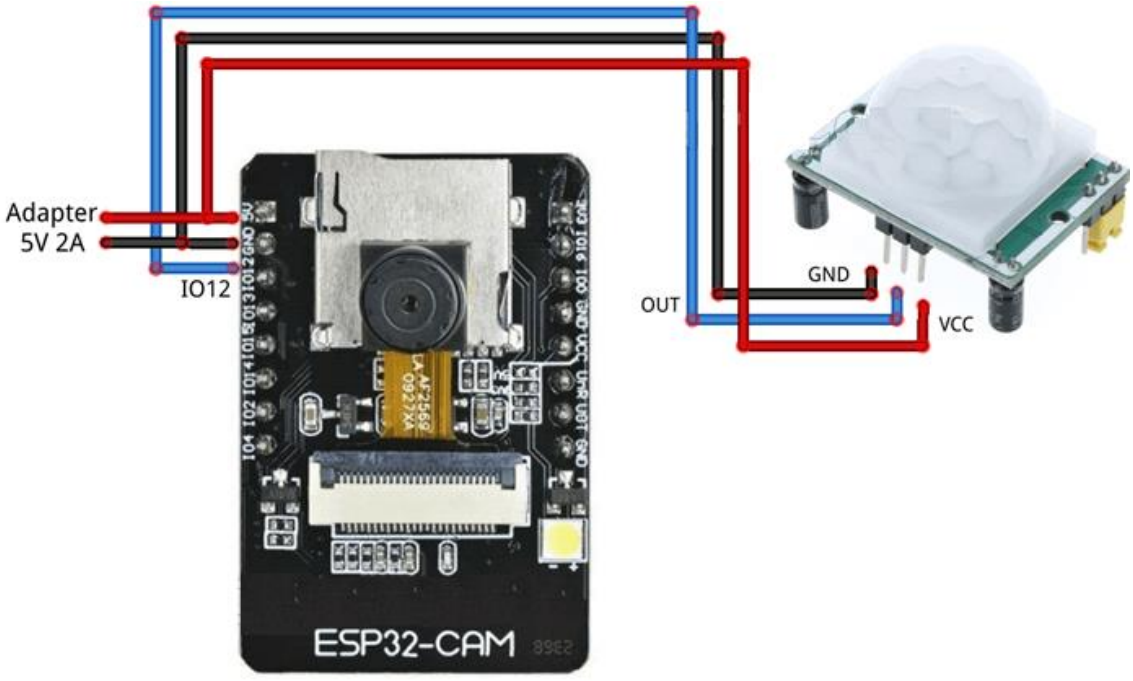
Za detekciju pokreta koristi se pasivni infracrveni senzor pokreta. Nakon detekcije pokreta, PIR senzor šalje signal na mikro kontroler ESP32-CAM, odnosno signal na pin GPIO13. Kontroler na sebi ima ugrađeni modul kamere i moguće je dohvatiti sliku kamere u bilo kojem trenutku. FTDI modul služi kao most između ESP32 i računala konvertirajući serijske podatke iz ESP32 kontrolera u USB signale prema računalu i obratno. Pomoću njega se prenosi kod i programira glavni ESP32 modul.

Nakon uspješnog spajanja na WiFi vezu, slika se šalje na platformu na topic “photo” koristeći MQTT.



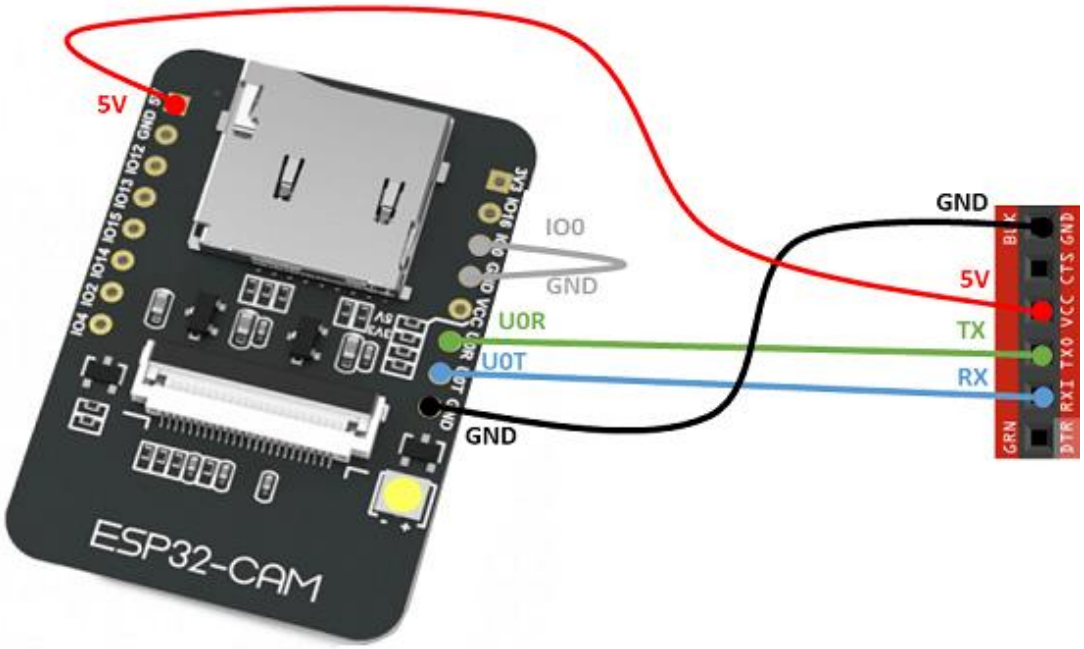
Shema spajanja PIR senzora i FTDI modula na ESP32-CAM mikrokontroler

PIR senzor povezuje se na ESP32-CAM putem 3 pina. GND za uzemljenje, OUT kao podatkovni pin za detekciju pokreta te VCC za napajanje. OUT pin na PIR senzoru spaja se na GPIO13 pin na ESP32-CAM kontroleru.



Shema povezivanja PIR senzora s ESP32-CAM kontrolerom

FTDI modul povezuje se na ESP32-CAM putem 4 pina. GND za uzemljenje, 5V za napajanje, te RX i TX kao pinovi za primanje i slanje podataka (receive / transmit) koji su na ESP32-CAM kontroleru spojeni na pinove U0R i U0T.



Shema povezivanja FTDI modula s ESP32-CAM kontrolerom

Za prikaz mogućnosti korištenja više uređaja simulirao se proces slanja slike s ESP32 uređaja postavljanjem brojača za automatsko slanje predefinirane slike umjesto fizičke detekcije pokreta PIR senzorom.

3. IoT platforma

Kao IoT platforma koristila se vlastita implementacija, s obzirom da je riječ o sigurnosnom sustavu, na ovaj način smo htjeli postići što višu razinu sigurnosti. Platforma se sastoji od tri bitne komponente, MQTTS brokera koji komunicira s IoT uređajem, HTTPS poslužitelja za komunikaciju s korisničkom aplikacijom, te MySQL baze podataka. Za potrebe ovog projekta podignut je Ubuntu server na Azure pružatelju usluge, kako bi korisnik bio u mogućnosti pristupiti svom sigurnosnom sustavu s udaljene lokacije. Inicijalna zamisao ovog sustava je bila kako će on služiti korisniku isključivo kao lokalna zaštita, te je ta njegova funkcionalnost i dalje ostala vrlo jednostavna za izvedbu ako se za time pokaže potreba.

Osnovne mogućnosti ove platforme su prikupljanje, autentifikacija i pohrana podataka sa spojenih uređaja. Ostavljeno je prostora za opcionalnu nadogradnju konačnog proizvoda s mogućnošću aktuacije jednog proizvoljnog priključka. Dodatno je moguće za sustave s pristupom internetu integrirati jednostavno prepoznavanje objekata na detektiranim slikama, korištenjem openCV biblioteke, te tako korisnika obavještavati samo u slučaju da je to prijeko potrebno.

Kako bi naš sustav bio što fleksibilniji odlučili smo se u konačnici implementirati MQTTS protokol za vezu prema IoT uređajima, glavni razlog tomu je mogućnost kasnijeg prelaska s WiFi na Zigbee tehnologiju, čime bismo postigli veću prostornu pokrivenost našim sustavom. S obzirom da je riječ o sigurnosnom sustavu odlučili smo ga implementirati što primitivnije kako bi se potencijalnom napadaču smanjila površina napada, a nama kao programerima propusti u dizajnu. Komunikaciju između IoT uređaja smo zaštitili korištenjem TLS protokola i generiranim openSSL certifikatima. Dodatno se provjerava autentičnost pristiglih poruka, na način da se za svaku pristiglu poruku provjerava pripada li pošiljatelju s popisa. Ovaj popis dodaje se u bazu podataka preko korisničke aplikacije. Nakon autentifikacije pristiglih poruka, formatirani podatci se šalju prema korisničkoj aplikaciji.

Konfiguracija senzora i aktuatora je odrađena na strani IoT uređaja, konkretno konfiguracija kamere jedno što je potrebno konfigurirati programski, piroelektrični senzor se podešava mehanički. Kamera je konfigurirana kako bi njeno korištenje bilo prikladno za Zigbee.

```
camera_config_t config;
config.ledc_channel = LEDC_CHANNEL_0;
config.ledc_timer = LEDC_TIMER_0;
config.pin_d0 = Y2_GPIO_NUM;
config.pin_d1 = Y3_GPIO_NUM;
config.pin_d2 = Y4_GPIO_NUM;
config.pin_d3 = Y5_GPIO_NUM;
config.pin_d4 = Y6_GPIO_NUM;
config.pin_d5 = Y7_GPIO_NUM;
config.pin_d6 = Y8_GPIO_NUM;
config.pin_d7 = Y9_GPIO_NUM;
config.pin_xclk = XCLK_GPIO_NUM;
config.pin_pclk = PCLK_GPIO_NUM;
config.pin_vsync = VSYNC_GPIO_NUM;
config.pin_href = HREF_GPIO_NUM;
config.pin_sscb_sda = SIOD_GPIO_NUM;
config.pin_sscb_scl = SIOC_GPIO_NUM;
config.pin_pwdn = PWDN_GPIO_NUM;
config.pin_reset = RESET_GPIO_NUM;
config.xclk_freq_hz = 20000000;
config.pixel_format = PIXFORMAT_JPEG;
config.frame_size = FRAMESIZE_SVGA;
config.jpeg_quality = 12;
config.fb_count = 1;
```

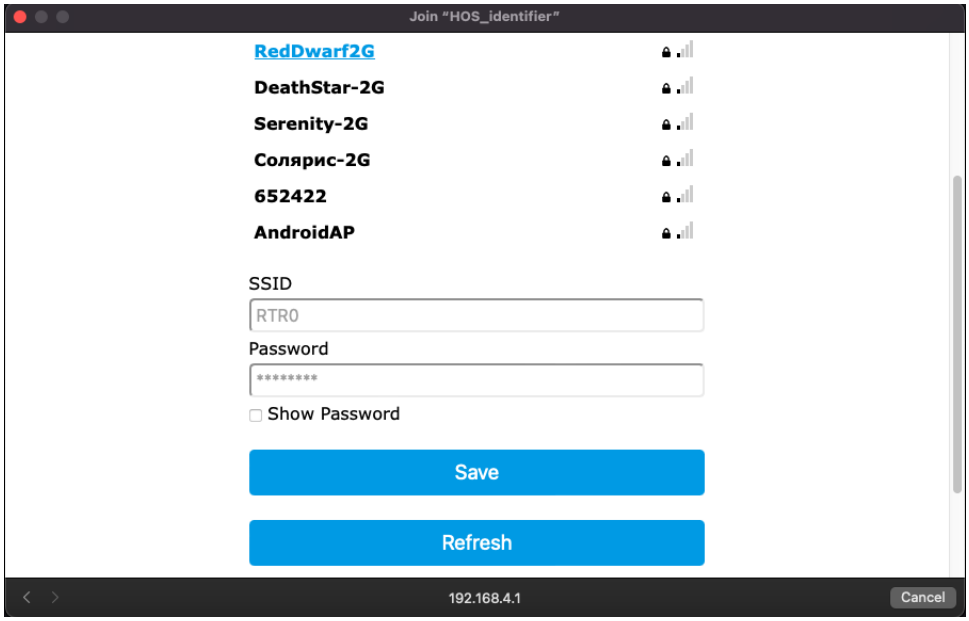
Slika 3.1 konfiguracija kamere

Kako bi se korisniku maksimalno pojednostavilo korištenje IoT uređaja, konfiguracija koja bi se u nekim jednostavnijim projektima odrađivala na strani IoT platforme prebačena je na stranu korisničke aplikacije. Međutim preostalo je još postaviti inicijalne postavke IoT uređaja koje je potrebno definirati kako bi isti bio u stanju povezati s korisničkom aplikacijom. U tu svrhu korištena je "WiFiManager" biblioteka. Riječ je o biblioteci koja implementira jednostavno korisničko sučelje za

konfiguraciju esp32 uređaja preko WIFI mreže. Prilikom prvog pokretanja uređaja, isti prelazi u način rada pristupne točke, identifikator ove pristupne točke sastoji se od zaglavlja “HOS_” nakon kojeg slijedi jedinstveni identifikator IoT uređaja. Povezivanjem na ovu otvorenu pristupnu točku otvara se sučelje za konfiguraciju uređaja.

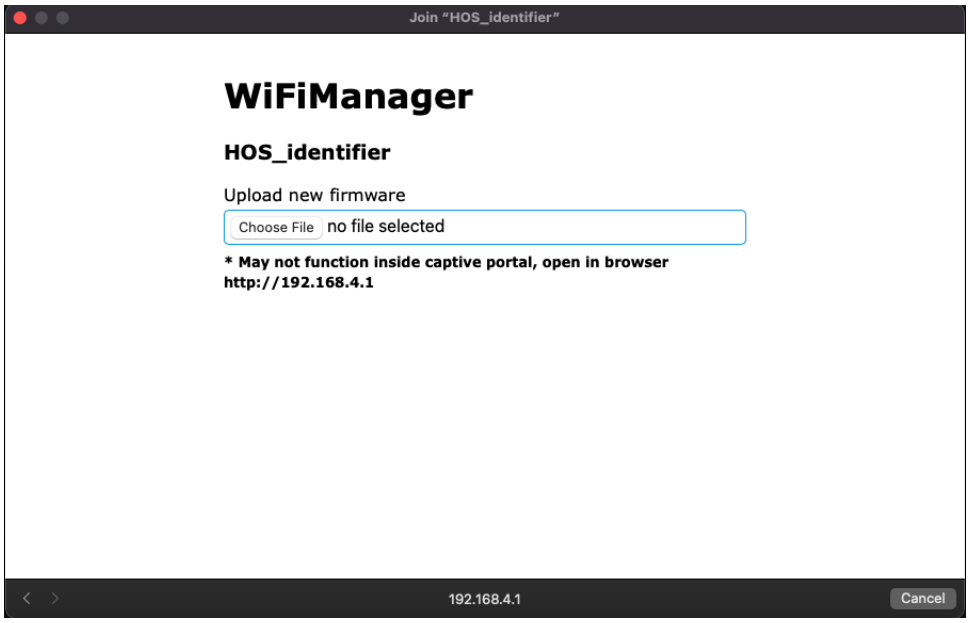
Slika 3.2 sučelje za inicijalnu konfiguraciju IoT uređaja

Prilikom konfiguracije uređaja HOS_identifier će biti zamijenjen jedinstvenim identifikatorom poput "HOS_QF4445JK75H" . Korisnik će ovaj identifikator iskoristiti prilikom registracije uređaja putem korisničke aplikacije, za dani primjer u korisničku aplikaciju je potrebno unijeti vrijednost “QF4445JK75H”. Proces registracije uređaja putem korisničke aplikacije potrebno je odraditi kako bi se uređaj bio u stanju povezati na IoT platformu. Odabirom opcije “Configure WIFI” otvara se novi prozor sa prikazom obližnjih pristupnih točaka.



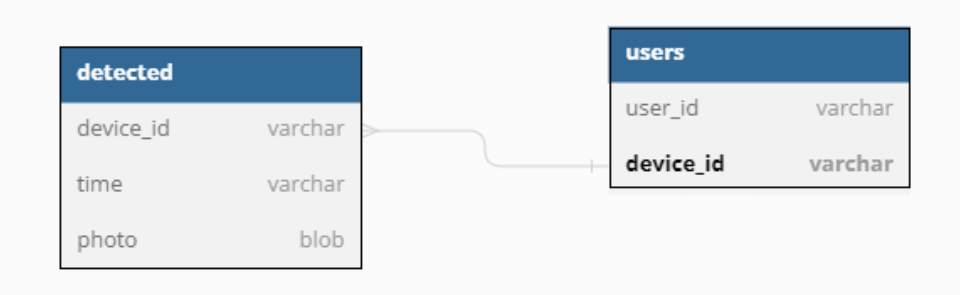
Slika 3.3 postavljanje vrijednosti za povezivanje na pristupnu točku

Osim navedenih funkcionalnosti biblioteka “WiFiManager” u standardnim postavkama pruža mogućnost pregledavanja informacija o esp32 uređaju kao i mogućnost uploada novog izvornog koda za esp32 uređaj. S obzirom da “WiFiManager” pristupa flash memoriji uređaja, moguće je dodatno konfigurirati i proizvoljne parametre koji trebaju ostati zapamćeni i nakon ponovnog pokretanja. Za potrebe našeg projekta ove funkcionalnosti nisu potrebne s obzirom da nakon inicijalnog povezivanje IoT uređaja na WIFI mrežu, te registracije identifikatora uređaja putem korisničke aplikacije, iste konfiguracije možemo odrađivati putem spomenute aplikacije.



Slika 3.4 postavljanje novog izvornog koda

Implementacija IoT platforme je odrađena tako da se prvo definiraju biblioteke koje su nam potrebne kao što su biblioteka za implementaciju MQTT brokera, biblioteka za rad sa bazom podataka te biblioteke za rad sa HTTP zahtjevima. Zatim se konfigurira MQTT broker tako da se stvori objekt koji sadrži postavke za broker kao što su port, putanje do privatnog ključa i certifikata za sigurnu komunikaciju, razina dnevnika te opcija za odbijanje veza koje ne pripadaju TLS protokolu. Nakon toga stvara se veza sa bazom podataka. Postavljaju se parametri veza kao što su maksimalni broj veza te informacije o pristupu bazi podataka, tj. host, korisnik, lozinka i naziv baze podataka. Koristeći prethodno definirane postavke stvara se instanca MQTT brokera i definira se HTTPS port na kojem će se vrtiti server. Definiraju se tri slušatelja događaja MQTT brokera. Jedan slušatelj se aktivira kada se klijent poveže s MQTT brokerom, drugi slušatelj se aktivira kada se klijent odspoji s MQTT brokera, a zadnji slušatelj se aktivira kada se objavi MQTT poruka. Ovaj slušatelj obrađuje poruke objavljene na temi “photo”. Očitaju se podaci iz dobivene poruke sa teme, provjerava se postoji li uređaj sa kojeg je poslana poruka postoji li u bazi podataka tako da ne može bilo tko slati podatke u bazu. Dobiveni podaci su slika koju je snimila kamera na IoT uređaju, ID uređaja i vremenska oznaka. Ovi podaci se spremaju u bazu podataka nakon čega se poziva funkcija “postImage” koja šalje sliku prema backendu. Funkcija “postImage” šalje sliku putem HTTP zahtjeva na vanjski API. Na kraju slijedi putanja za registraciju uređaja koja prima POST zahtjev s podacima o korisniku i uređaju. Ti podaci se zatim spremaju u bazu podataka, veza sa bazom se zatvara i stvara se HTTPS poslužitelj koristeći certifikat i ključ i pokreće se na zadanom HTTPS portu.



Slika 3.5 prikaz baze podataka na IoT platformi

4. Korisničke aplikacije

Cjelokupni izvorni kod korisničke aplikacije javno je dostupan i moguće ga je pronaći u sljedećem GitHub repozitoriju: GitHub-link. Korisnička aplikacija sastoji se od poslužiteljskog REST API-ja razvijenog pomoću Spring Boot razvojnog okvira i klijentske frontend aplikacija razvijene pomoću Angular razvojnog okvira. Također za potrebe korisničke aplikacije koristi se i PostgreSQL baza podataka i Firebase Cloud Messaging (FCM) platforma koja omogućava slanje poruka i push notifikacija korisnicima aplikacije.

Klijentski i poslužiteljski (backend i frontend) dio aplikacije su javno dostupni. REST API deployan je na Render cloud platformu, a Angular klijentska aplikacija na Azure cloud platformu. Link na javno dostupno aplikaciju: https://iot-hos.azurewebsites.net/

Tehnologije korištene pri razvoju korisničke aplikacije:

1. Spring Boot – Za razvoj backend dijela aplikacije. Gradle je korišten kao sustav izgradnje projekta, a kao programski jezik korištena je Java verzije 8. Spring Boot je popularan framework za izgradnju Java aplikacija, koji olakšava razvoj i konfiguraciju.
2. PostgreSQL – baza podataka koja se koristi za pohranu podataka aplikacije. PostgreSQL je snažan, pouzdan i skalabilan sustav upravljanja bazama podataka.
3. Firebase Cloud Messaging (FCM) - servis za slanje push obavijesti korisnicima mobilnih aplikacija. Omogućuje programerima da ciljaju određene korisnike ili grupe korisnika i šalju im obavijesti putem mobilnih uređaja.
4. Angular – za razvoj frontend dijela aplikacije. Angular je open-source web framework razvijen od strane Googlea, koji koristi TypeScript i omogućuje izgradnju modernih, skalabilnih aplikacija temeljenih na komponentnoj arhitekturi i deklarativnom pristupu.
5. Docker – za izgradnju i distribuciju aplikacija. Docker je open-source platforma koja omogućuje izgradnju, distribuciju i pokretanje aplikacija u izoliranim kontejnerima.

Poslužiteljska strana - Backend

REST (Representational State Transfer) API razvijen pomoću Spring Boot razvojnog okvira API se koristi tri glavne skupine ruta: “api/auth”, “api/main” i “api/platform”. Projekt je konfiguriran korištenjem Gradle alata za upravljanje ovisnostima i izgradnju, dok je Java 8 programski jezik korišten prilikom razvoja.

Spring Boot, kao lagani Java framework, pruža jednostavnost i brzinu razvoja REST API-ja. Konfiguracija i postavljanje projekta su pojednostavljeni, što omogućuje brzo pokretanje aplikacije bez potrebe za složenom konfiguracijom.

Opisi ruta:

1. api/auth - ova skupina ruta koristi se za prijavu i registraciju korisnika. Ove rute nisu zaštićene autentifikacijom i omogućuju korisnicima da stvore nove račune ili se prijave s postojećim računima.
2. api/main - rute unutar ove skupine dostupne su samo prijavljenim korisnicima. Ove rute omogućuju korisnicima pregledavanje, brisanje i dodavanje uređaja, kao i pregledavanje i brisanje zapisa koje je platforma dostavila aplikaciji. Autentifikacija JWT Bearer tokenom osigurava pristup samo ovlaštenim korisnicima.
3. api/platform - ova skupina ruta omogućuje komunikaciju s platformom. Rute su dostupne samo putem HTTPS veze i dopušteno je uspostavljanje veze samo s IP adresom platforme.

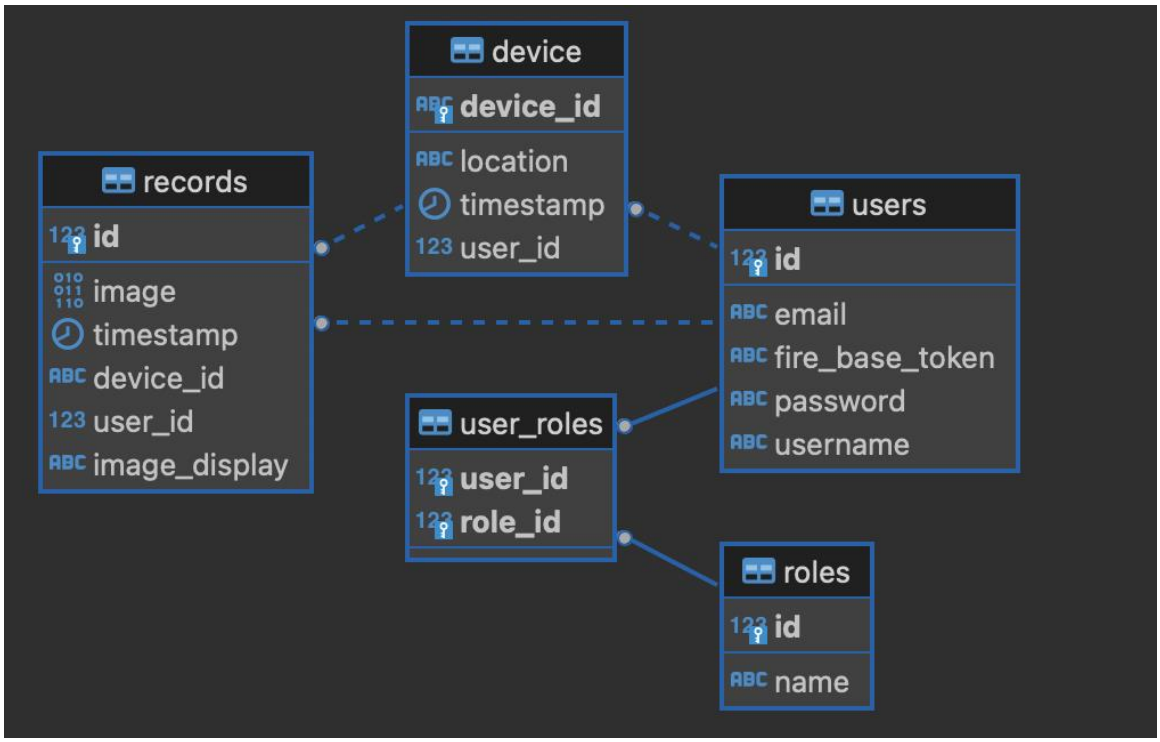
Kada je riječ o autentifikaciji, API koristi JSON Web tokene (JWT) s Bearer tipom tokena. JWT je siguran standard za autentifikaciju koji se koristi za prenošenje digitalnih potvrda. Bearer tip tokena označava da se token treba prenijeti u zaglavlju HTTP zahtjeva radi autentifikacije.

Ovako razvijen REST API pruža sigurnu i funkcionalnu backend aplikaciju koja omogućuje korisnicima prijavu, registraciju, upravljanje uređajima i slikovnim zapisima te komunikaciju s platformom, uz primjenu JWT autentifikacije radi osiguravanja zaštite podataka i pristupa.

Backend aplikacije komunicira i s Firebase servisom. Integracija Firebase servisa s korisničkom aplikacijom omogućava alarmiranje korisnika kada platforma javi da je detektirano potencijalno narušavanje sigurnosti doma korisnika (točnije kada platforma pošalje sliku aplikaciji). Firebase servis pruža mogućnosti slanja notifikacija korisnicima putem Firebase Cloud Messaging usluge, omogućujući trenutno i pouzdano obavještavanje korisnika o važnim događajima.

Za izgradnju i distribuciju aplikacije koristi se Docker, popularna platforma koja omogućuje pakiranje aplikacija i njenih ovisnosti u kontejnere. Docker kontejneri pružaju izolaciju, omogućavajući konzistentno izvršavanje aplikacije neovisno o okruženju. REST API deployan je i javno dostupan na sljedećoj poveznici: https://hos-api.onrender.com

Baza podataka korisničke aplikacije:



Shema baze podataka

Na slici iznad prikazana je shema baze podataka korisničke aplikacije. Baza podataka sastoji se od pet tablica koje aplikacija koristi kako bi funkcionirala i zadovoljila sve potrebe korisnika. Opisi tablica sadržanih u bazi dani su u tablici u nastavku teksta.

Tablica "roles"	Sadrži podatke o ulogama koje korisnik može imati u aplikaciji. Npr. "ROLE_USER", "ROLE_ADMIN" itd.
Tablica "user_roles"	Povezuje korisnika aplikacije s njegovom ulogom u aplikaciji.
Tablica "users"	Pohranjuje podatke o korisnicima. Jedinstveni email i korisničko ime korisnika. Lozinku za prijavu koja se pohranjuje u sigurnom enkriptiranom obliku (BCryptPasswordEncoder). Pohranjuje se i jedinstveni Firebase token koji omogućava slanje notifikacija korisniku.
Tablica "records"	Sadrži podatke o slikovnim zapisima koje aplikacija dobila od IoT platforme. Povezuje zapis s korisnikom i uređajem za koji je zapis zabilježen.
Tablica "devices"	Sadrži podatke o uređajima koje korisnik ima na raspolaganju. Podatke o njihovoj o lokaciji i vremenski zapis kada su dodani.

Tablice i pripadajući opisi

Klijentska strana korisničke aplikacije - Frontend

Logotip korisničke aplikacije Home Observation Systems

Korisničko sučelje aplikacije Home Observation Systems izrađeno je u obliku SPA (Single Page Application) arhitekture web aplikacije, koji se sastoji od samo jedne HTML (HyperText Markup Language) stranice i koristi programski jezik JavaScript za dinamičko ažuriranje sadržaja web stranice bez potrebe za ponovnim učitavanjem cijelog sadržaja. Time se omogućava brže i responzivnije korisničko iskustvo. U ovoj implementaciji, SPA arhitektura je postignuta korištenjem Angular TypeScript programskog okvira.

Korisničko sučelje aplikacije je prilagodljivo te se zato aplikacija može koristiti na mobilnim i ostalim uređajima.

U kontekstu funkcionalnosti, korisnička aplikacija se sastoji od sljedećih stavki:

- Autentifikacija korisnika
  - Prijava korisnika
  - Registracija korisnika
- Prikaz zapisa (snimaka sa korisnikovih uređaja) i dodatnih atributa
  - Filtriranje zapisa na temelju uređaja
  - Detaljan prikaz odabranog zapisa
  - Brisanje odabranog zapisa
- Prikaz popisa registriranih uređaja korisnika
  - Dodavanja novog uređaja
  - Brisanje odabranog uređaja
- Prikaz obavijesti na korisnikovom uređaju (alarm)

Opis funkcionalnosti

Pri inicijalnom posjetu web stranici, korisnik je usmjeren na stranicu za prijavu. Ako korisnik postoji u sustavu, unosom odgovarajućih podataka (korisničko ime i lozinka) se korisnik uspješno prijavljuje u sustav i ima pravo na pristup svim funkcionalnostima aplikacije. Ako korisnik unese krive podatke, prikazuje mu se određena poruka da se dogodila greška. Ako korisnik ne postoji u sustavu, nudi mu se opcija za registraciju sa poljima za unos korisničkog imena, e-mail adrese, lozinke i

ponovljene lozinke. Korisničko ime mora biti jedinstveno u bazi podataka korisnika, i lozinka i potvrda lozinke moraju biti jednake. E-mail adresa mora biti u odgovarajućem formatu. Ako se neko od tih pravila ne zadovolji pri registraciji, korisniku se prikazuje poruka sa greškom. Pri uspješnoj registraciji, novi korisnik je usmjeren na stranicu za prijavu, gdje se sa točnim podacima može prijaviti u sustav. Sve rute u aplikaciji su zaštićene, te im jedino može pristupiti prijavljeni korisnik. Za autentifikaciju je korišten JWT pristupni token, koji klijentska strana dobije od poslužiteljske za uspješnu prijavu korisnika. Na temelju tog pristupnog tokena klijentska strana radi zahtjeve prema poslužiteljskoj strani, koja onda provjerava token i na temelju njega šalje podatke. JWT token ima vrijeme isteka, i pri isticanju tog vremena aplikacija automatski odjavi korisnika.

Nakon uspješne prijave, korisnik je usmjeren na početnu stranicu Home Observation Systems aplikacije. Tamo mu se nude dvije opcije. Opcija za prikaz arhive zapisa snimaka sa IoT uređaja i opcija za prikaz liste registriranih uređaja trenutnog korisnika. U zaglavlju korisničkog sučelja, prijavljeni korisnik ima mogućnost za odjavu iz sustava.

Arhiva zapisa inicijalnim otvaranjem prikazuje listu sažetih zapisa snimaka sa IoT uređaja sa parametrima: vrijeme aktiviranja uređaja (“okidanja”), lokacija uređaja i snimka sa uređaja. Za svaki zapis korisnik ima mogućnost detaljnog prikaza zapisa, koji nudi opširniji prikaz snimke i detaljnije informacije. Također, korisnik može obrisati odabrani zapis sa liste, uz potvrdu na upozorenje o gubitku odabrane snimke. Korisnik također ima opciju filtriranja svih zapisa, na temelju odabira iz liste IoT uređaja koji su registrirani na tog korisnika.

Popis registriranih uređaja korisniku daje uvid u IoT uređaje koje je korisnik dodao u sustav. Svaki IoT uređaj ima svoj jedinstveni identifikator koji korisnik može upisati pri dodavanju u sustav. Na popisu uz atribut identifikator je također vidljivo i vrijeme postavljanja uređaja. Korisnik ima mogućnost brisanja odabranog IoT uređaja iz sustava. Nakon odabira opcije brisanja, korisnika sustav upozorava da će brisanjem uređaja obrisati i sve pripadajuće zapise tog uređaja. Nakon potvrđivanja, IoT uređaj i njegovi zapisi se brišu iz baze podataka.

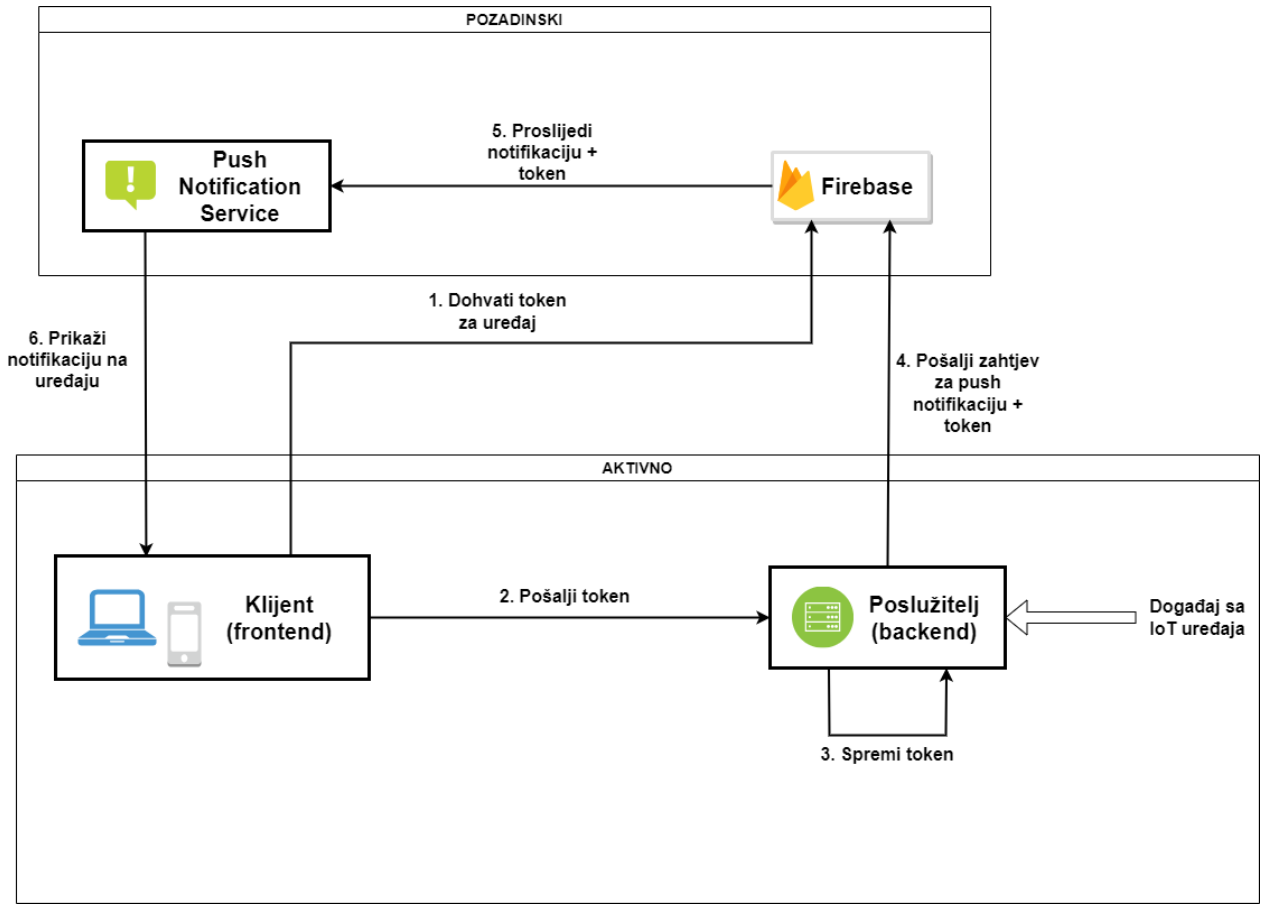
Korisnik može primiti obavijesti (alert) od sustava pri aktiviranju IoT uređaja i detekciji pokreta. Obavijest nije vezana za web aplikaciju niti za klijentski preglednik, nego na korisnikov uređaj. Pri registraciji, korisnik uz registracijske podatke također šalje i jedinstveni token za svoj uređaj, na temelju kojeg se šalju obavijesti na pripadajući uređaj. Ova funkcionalnost je izvedene korištenjem servisnog radnik u kontekstu progresivnih web aplikacija, čija će logika biti opisana u sljedećem dijelu.

Push notifikacije (obavijesti)

Uz standardno korisničko sučelje za Home Observation Systems aplikaciju, implementirana je i funkcionalnost progresivne web aplikacije, koja korisnicima omogućava izvršavanje određene logike bez potrebe za učitavanjem ili instaliranjem dodatne aplikacije. U ovom kontekstu je takva funkcionalnost korištena za stvaranje, slanje i prikaz push notifikacija (obavijesti o detektiranom pokretu sa IoT uređaja) na korisnikovom uređaju.

Aplikacijski poslužitelj (backend) sprema token uređaja koji je proslijedio korisnik pri registraciji. Kada IoT uređaj detektira pokret, na aplikacijskom poslužitelju se generira zahtjev za slanjem push notifikacije. Na temelju jedinstvenog identifikatora IoT uređaja, poslužitelj odabire pripadajući token od korisnika na kojeg je taj IoT uređaj registriran. Nakon toga, poslužitelj šalje zahtjev za slanjem push notifikacije, uz pripadajući sadržaj notifikacije i token. Taj zahtjev se šalje do Firebase

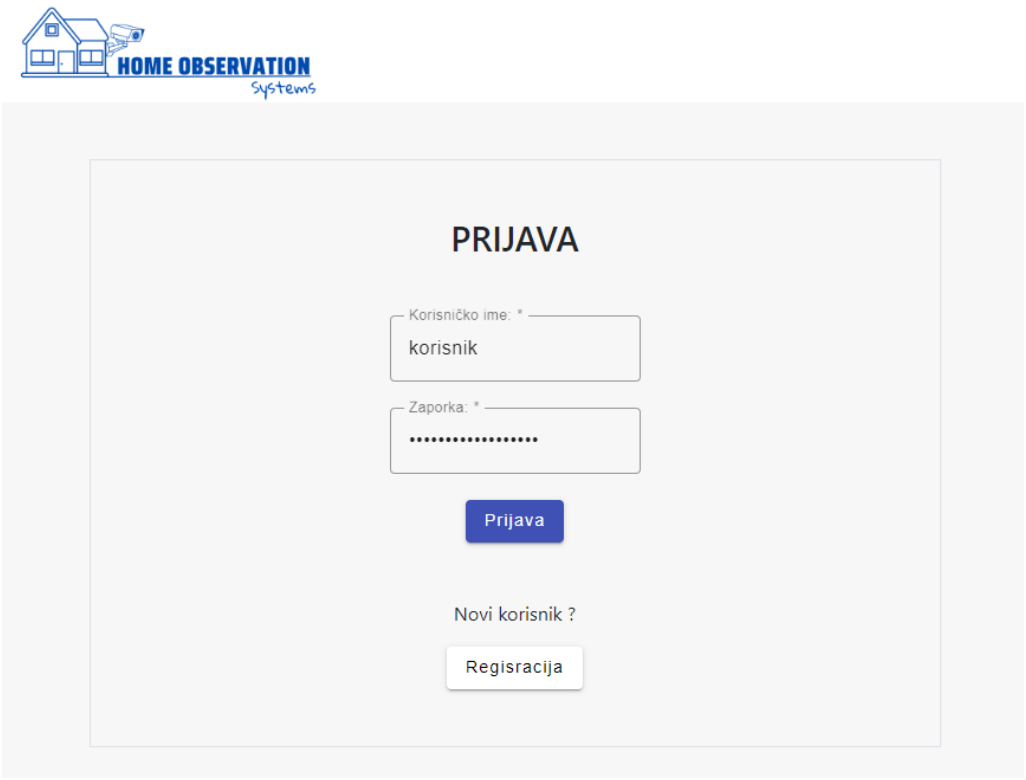
Cloud Messaging platforme. Sa druge strane, uz regularnu korisničku web aplikaciju, kreiran je servisni radnik, čija je zadaća slanje obavijesti na korisnikov uređaj. Zbog toga on cijelo vrijeme radi u pozadini, čak i kada aplikacija ne radi, i osluškuje kada će Firebase platforma poslati notifikaciju. Kada servisni radnik primi notifikaciju, na temelju primljenog tokena prosljeđuje notifikaciju na pripadajući uređaj (mobitel, laptop, računalo). Ovaj cijeli proces prikazan je dijagramom ispod.



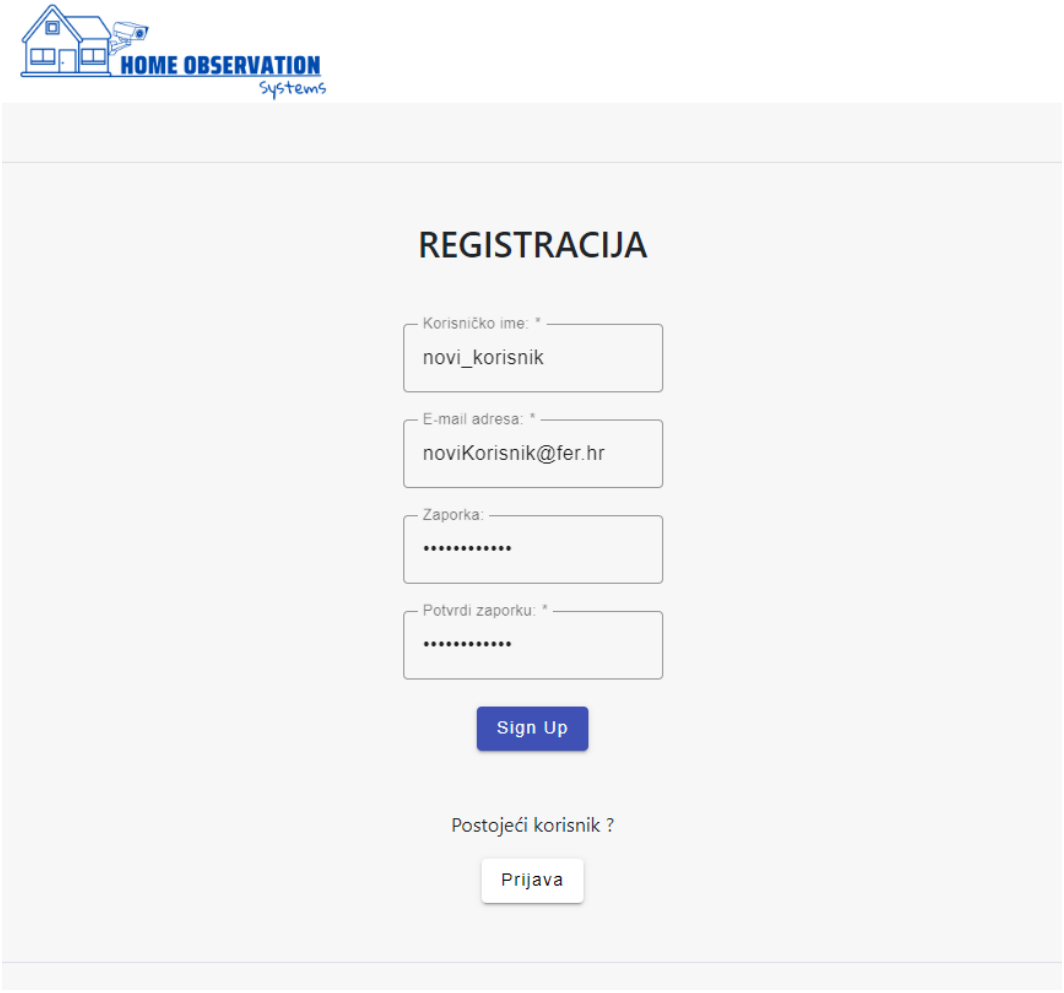
Shema push notification sistema

Korisničko sučelje aplikacije (User Interface)

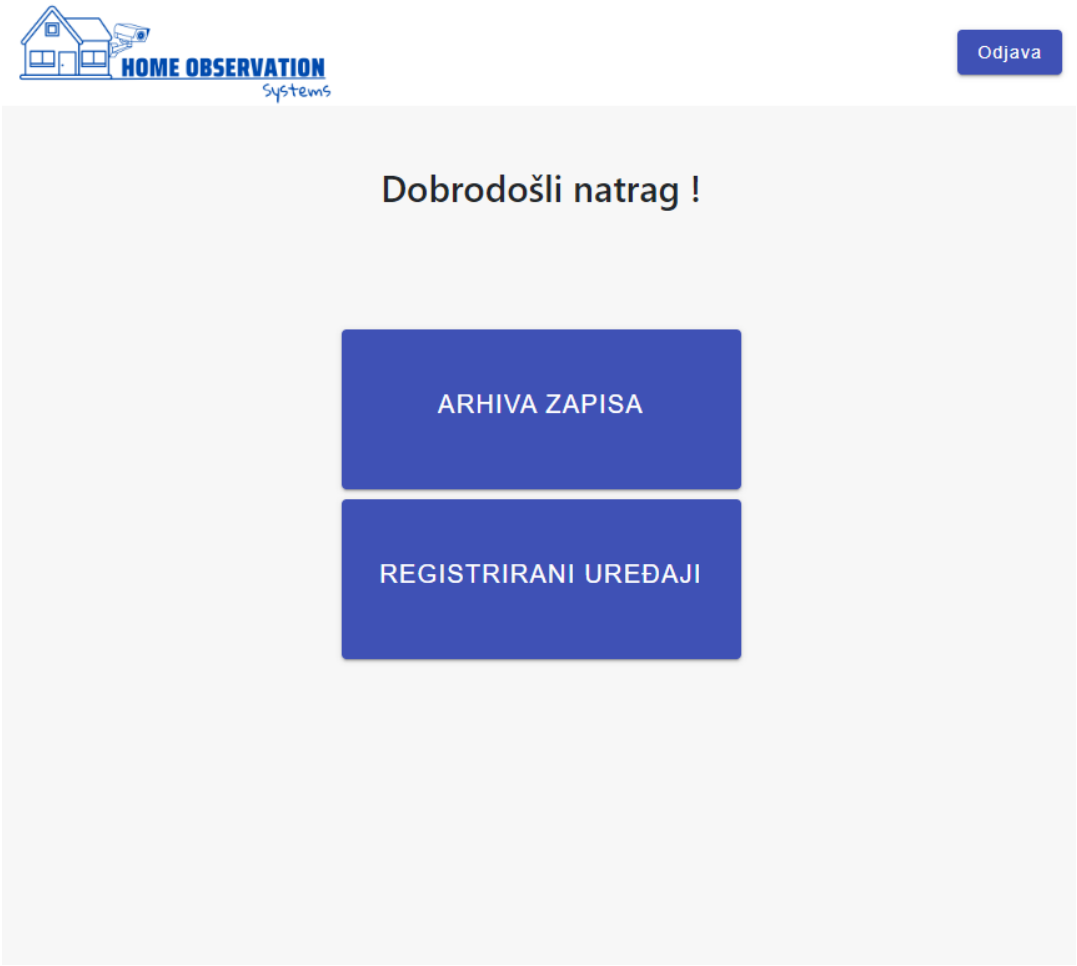
Sljedećim slikama ekrana je prikazan kompletan izgled korisničkog sučelja na temelju gore navedenih funkcionalnosti.



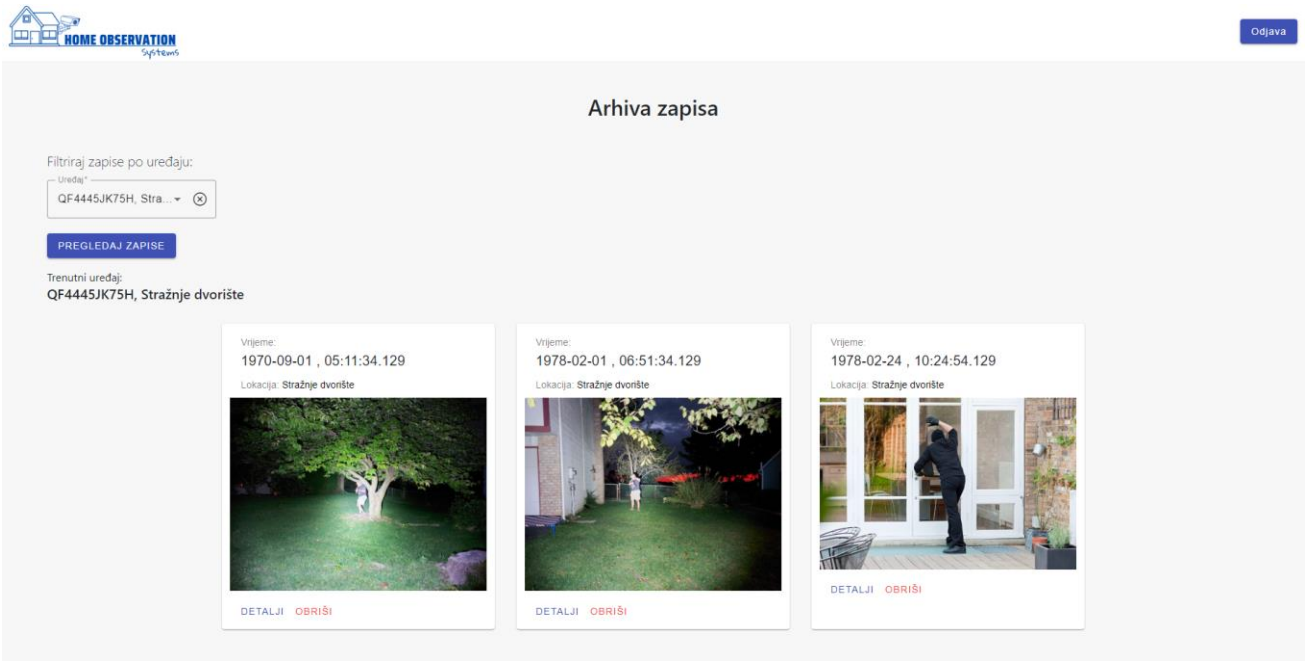
Sučelje za prijavu korisnika



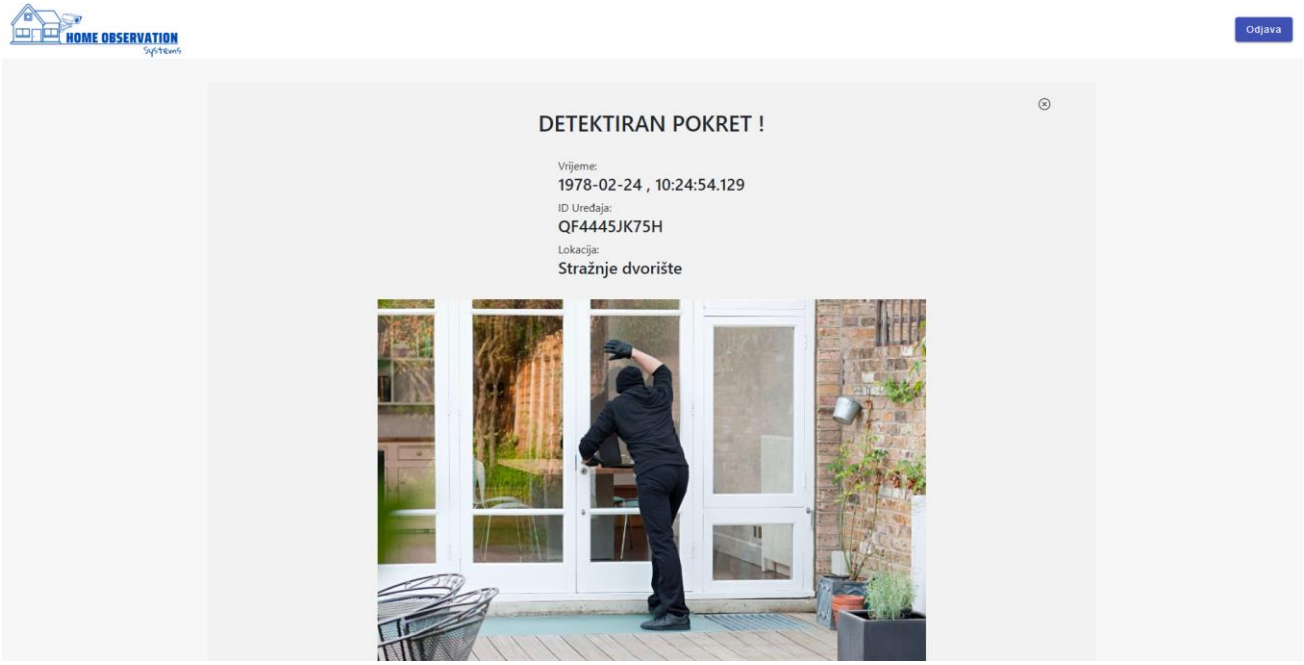
Sučelje za registraciju korisnika



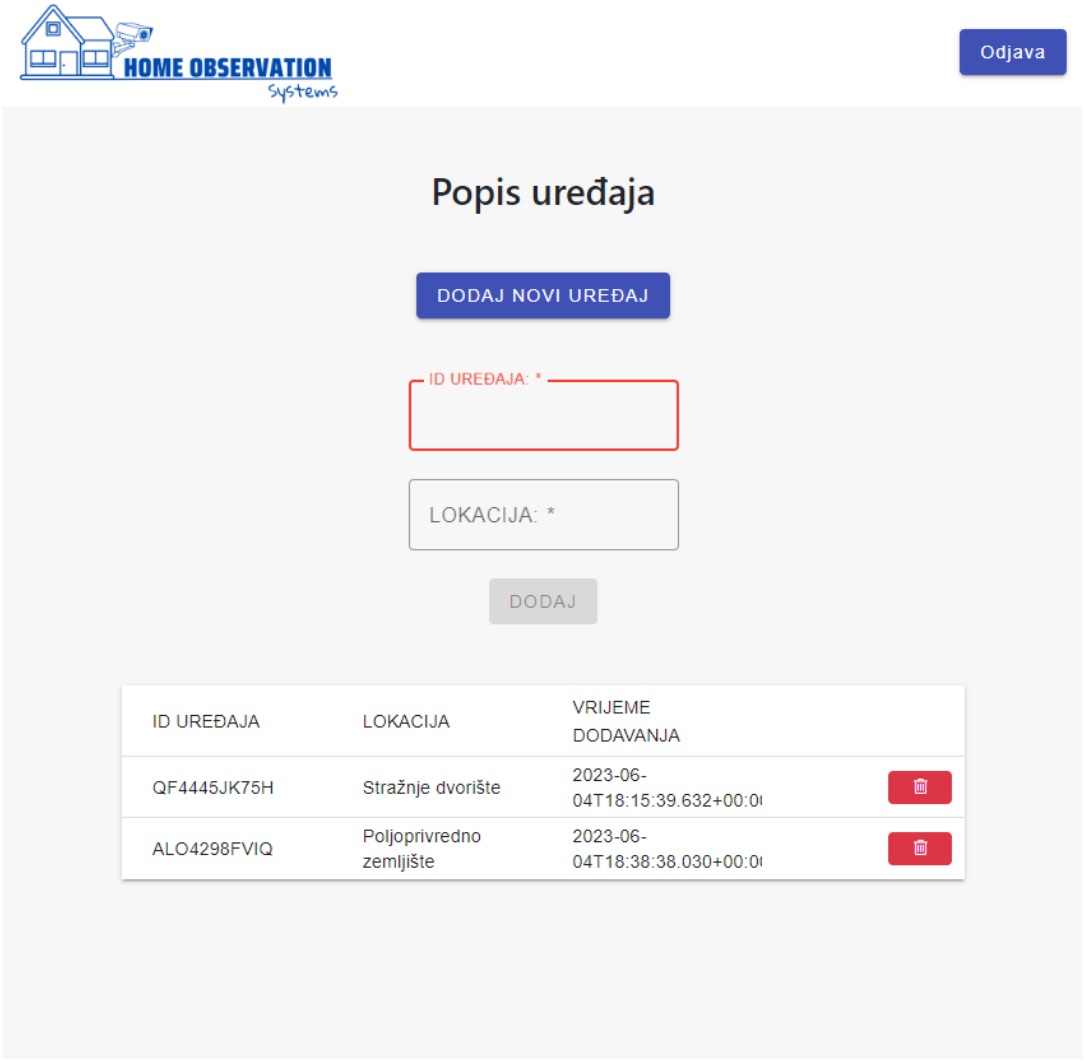
Početni ekran prijavljenog korisnika



Popis zapisa (detekcija) za odabrani IoT uređaj



Detaljan pregled zapisa



Pregled liste pripadajućih IoT uređaja uz opciju dodavanja novog

Upozorenje za brisanje IoT uređaja iz sustava

Mobilna push notifikacija pri detektiranju pokreta