

Diplomski studij

**Informacijska i komunikacijska
tehnologija**

Računarstvo

Telekomunikacije i informatika

Obработка informacija

Računalno inženjerstvo

Internet stvari

<Gatekeeper>

Projekt

<Leonarda Gjenero>

<Neda Kušurin>

<Maksimilijan Marošević>

<Marija Papić>

<Dora Pavelić>

<Andi Škrgat>

Ak.g. 2021./2022.

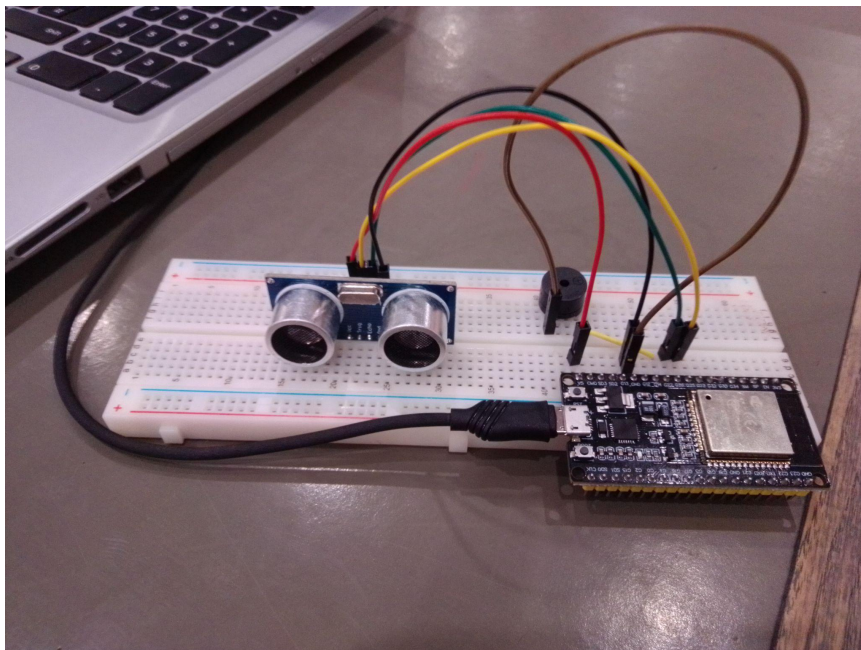
Sadržaj

1. Uvod	2
2. Opis rješenja	3
3. IoT platforma	6
4. Korisničke aplikacije	8

1. Uvod

U zadatku projekta se radi sustav za sigurnost doma, tj. alarm za detekciju potencijalne neodređene opasnosti. Uređaj bi se nalazio na ulazu u dom i detektira svaku promjenu pri bliskoj udaljenosti na vrata/prozor.

Za izradu projekta se koristi mikrokontroler, senzor i aktuator, prikazano na slici ispod. Mikrokontroler je ESP32 s integriranom WiFi povezoivošću, senzor za udaljenost je *Ultrasonic Ranging Module HC - SR04*, i aktuator je *Active Passive Buzzer*. Također, mobilna aplikacija ima ulogu virtualnog uređaja i može davati naredbe kontroleru.



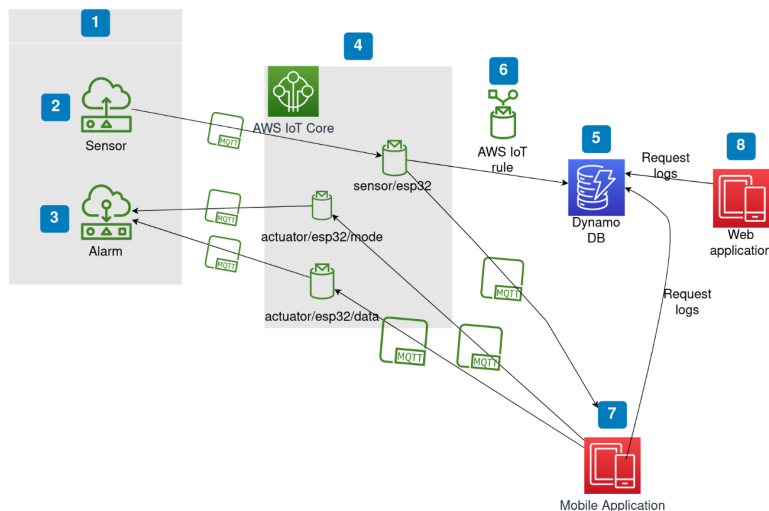
Slika 4.1: ESP32 mikrokontroler

Ovakav sustav se može koristiti za osnovnu razinu zaštite doma, ili za jednostavnu analizu promjena na ulaznim vratima. Za veću razinu sigurnosti mogu se koristiti uređaji koji rade na sličnom principu, ali sa ugrađenim kamerama i dodatnim senzorima. Sličan projekt je npr. ovaj [Ring proizvod](#)

2. Opis rješenja

Gatekeeper

Solution

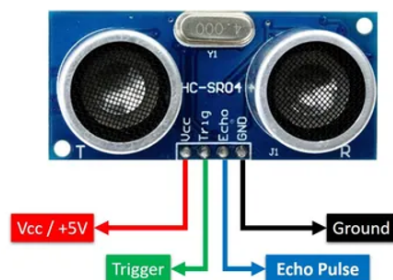


- 1 ESP32 microcontroller with WiFi support is used for connecting device to Internet and consequently, AWS IoT platform.
- 2 Detection movement sensor is constantly listening for every change and when it detects movement close enough sends message to topic sensor/esp32 on AWS IoT platform.
- 3 Alarm has its own state that saves information whether it is enabled or disabled. If sensor detects movement, alarm is automatically triggered. User subscribed to topic sensor/esp32 can turn off the alarm by sending acknowledge message to topic actuator/esp32/data.
- 4 AWS IoT core has integrated gateway in itself, so there is no need to have separate device for gateway purpose. IoT core has also integrated message broker for secure transmission of messages from IoT devices to applications and reverse while achieving low latency.
- 5 Data sent from device to AWS IoT core is forwarded to Dynamo DB database so user applications can very easily obtain historical data about movement detection and generate various analytical data. Although here could be implemented different solutions such as Amplify, we decided to use Dynamo DB so there are no additional proxies between mobile app and data storage.
- 6 AWS IoT rule is deployed and its purpose is forwarding data received at topic sensor/esp32 to Dynamo DB. It is written as SQL statement and it works perfectly.
- 7 Mobile application is used as a virtual device and it can enable/disable alarm, turn the alarm off, notify user that the sensor detected movement and show historical data.
- 8 Web application is limited to visualizing the data but without the capability of managing alarm state,

Slika 4.2: Skica rješenja Gatekeeper sustava

Kao senzor koristi se ultrazvučni modul *HC - SR04*, povezan na napajanje mikrokontrolera od 5V, i dva pina (trig i echo) koji ima doomet od 2cm - 400cm.

Buzzer je mala i jednostavna komponenta, ima dva pina i ulogu alarma tako što stvara zvuk željene frekvencije.



Slika 4.3: Senzor i alarm

Implementacija se sustava velikim dijelom temelji na funkcionalnostima AWS IoT platforme. Tako primjerice nije bilo nužno imati gateway i implementirati message broker jer su isti već ostvareni u sklopu AWS-a. Dvosmjerna komunikacija između uređaja i platforme te korisničkih aplikacija i AWS platforme obavlja se MQTT protokolom. Taj smo aplikacijski protokol primarno odabrali jer podržava objavi/pretplati način rada. MQTT protokol koristi TCP/IP internetski složaj, a iznimno je efikasan zbog svoje implementacije koja dodaje samo 2B transportnog overheada. Koristili smo QoS razinu 1 za pretplaćivanje i objavljivanje poruka od strane mobilne aplikacije i vrlo je praktično za situacije u kojoj na mreži ne vladaju idealni uvjeti. Na taj način poruka se šalje barem jedanput, a ne smatra se potpunom sve dok pošiljatelj ne primi PUBACK odgovor kako je primanje uspješno završeno.

Uređaj ESP32 pretplaćuje se na teme `actuator/esp32/data` i `actuator/esp32/mode` kako bi dobio poruke objavljene od strane mobilne aplikacije, a istovremeno, pri senzorskoj detekciji pokreta, objavljuje poruku na temu `sensor/esp32` na kojoj korisnik preko mobilne aplikacije dobiva obavijest o aktiviranju senzora. Uz to, na AWS IoT platformi definirano je pravilo koje prosljeđuje informacije o detekciji pokreta, dobivene od strane uređaja, u bazu podataka. Odabrana je Dynamo DB baza podataka zbog svoje skalabilnosti i performansi koje nudi, a i iznimno nam je jednostavno bilo implementirati rješenje koje je dio AWS-ovog tehnološkog stacka. Ono što nam je također vrlo bitno jest da u potencijalnom produkcijskom sustavu s milijunima korisnika, ovakva implementacija NoSQL baze podataka ne bi trebala uzrokovati nikakvo pogoršanje u performansama.

SQL statement	
SQL statement	SQL version
<code>SELECT value FROM 'sensor/esp32' WHERE value <> ""</code>	2016-03-23

Slika 4.4: Pravilo koje prosljeđuje podatke u Dynamo DB bazu podataka

Mobilna aplikacija nudi i mogućnost pregleda povijesnih podataka HTTP dohvatom iz baze, ali se takav dohvat, koji je skup, ne odvija stalno već samo pri dolasku na zaslon glavne aktivnosti. Također, vrlo je bitno naglasiti da se dohvat iz baze podataka ne radi kako bi aplikacija prepoznala da je došlo do aktiviranja senzora, već se za to koristi puno efikasniji pristup u vidu objavi/pretplati načina rada. Dodatno smo implementirali i web aplikaciju koja je u ovom projektu zamišljena kao pasivna aplikacija čiji je jedini cilj prezentacija podataka i statistika na temelju podataka iz baze. Smatramo da smo ovakvom implementacijom postigli vrlo efikasan sustav u kojem nema dodatnih proxya/servera za dohvaćanje informacija na mobilnu aplikaciju, a to je vrlo bitan zahtjev u izgradnji produkcijskih IoT sustava.

Svaki IoT sustav mora imati vrlo jasna definirana sigurnosna pravila. U tu smo svrhu koristili AWS Identity and Access Management te Cognito sustav koji omogućuju vrlo precizno definiranje polica (eng. policy) nužnih za definiranje pristupa pojedinog korisnika. Prvo smo dodali pravila koja omogućuju uređaju ESP32 objavljivanje podataka na temu `sensor/esp32` te pretplaćivanje na teme `actuator/esp32/data` i `actuator/esp32/mode`. Mobilna aplikacija ima obrnutu logiku pa smo definirali prava koja joj omogućuju pretplaćivanje na `sensor/esp32`, a objavljivanje na `actuator/esp32/data` i `actuator/esp32/mode`.

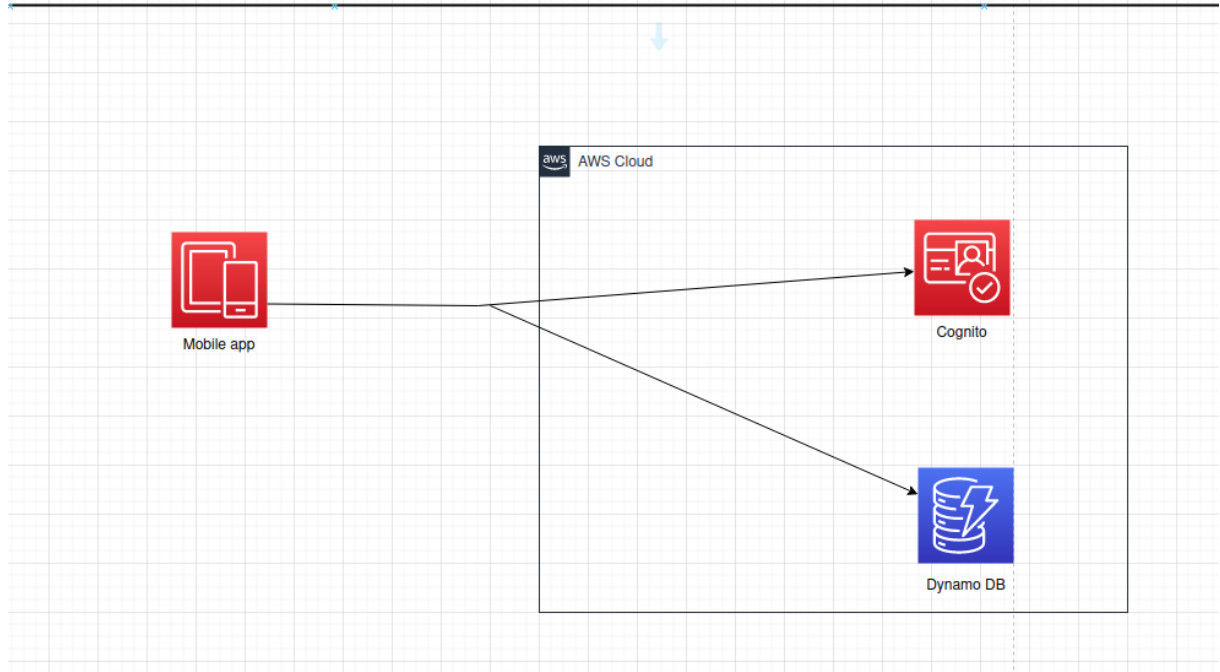
Active version: 10 Info			Builder	JSON
Policy effect	Policy action	Policy resource		
Allow	iot:Connect	arn:aws:iot:eu-central-1:248963601064:client/MyNewESP32		
Allow	iot:Subscribe	arn:aws:iot:eu-central-1:248963601064:topicfilter/actuator/esp32/data		
Allow	iot:Receive	arn:aws:iot:eu-central-1:248963601064:topic/actuator/esp32/data		
Allow	iot:Publish	arn:aws:iot:eu-central-1:248963601064:topic/sensor/esp32		
Allow	iot:Subscribe	arn:aws:iot:eu-central-1:248963601064:topicfilter/actuator/esp32/mode		
Allow	iot:Receive	arn:aws:iot:eu-central-1:248963601064:topic/actuator/esp32/mode		
Allow	iot:Connect	arn:aws:iot:eu-central-1:248963601064:client/MobApp		
Allow	iot:Publish	arn:aws:iot:eu-central-1:248963601064:topic/actuator/esp32/data		
Allow	iot:Publish	arn:aws:iot:eu-central-1:248963601064:topic/actuator/esp32/mode		
Allow	iot:Subscribe	arn:aws:iot:eu-central-1:248963601064:topicfilter/sensor/esp32		
Allow	iot:Receive	arn:aws:iot:eu-central-1:248963601064:topic/sensor/esp32		

Slika 4.5: Definiranje sigurnosnih pravila za AWS IoT sustav

Ipak, kreiranje polica nije bilo dovoljno kako bi mobilna aplikacija imala pristup objavi/pretplati sustavu na AWS IoT sustavu i Dynamo DB bazi podataka te smo zato dodali identity pool na Amazon Cognito usluzi. Mobilna aplikacija pri pokretanju kreira certifikate koji su kasnije preneseni na AWS sustav. Za to nam je poslužila već definirana polica koja sada na temelju prenesenog certifikata autorizira mobilnu aplikaciju za objavljivanje, pretplaćivanja i primanja poruka. U stvarnom sustavu trebalo bi dodati proces autentifikacije mobilne aplikacije identity pool-u. Amazon Cognito podržava autentifikaciju preko Cognita Application ID-a, Amazon-a, Google-a, Apple-a, Facebooka-a, OpenID-a, SAML-a itd. no zbog ograničenog vremena u okviru ovog projekta to nije implementirano.

Mobile app security flow

Connecting to Dynamo DB



Slika 4.6: Prikaz spajanja mobilne aplikacije na bazu podataka

Dodavanje potencijalnih novih korisnika bilo bi vrlo jednostavno. Potrebno je dodati novi ESP32 uređaj (ili uređaj sličan tome), definirati nove teme (eng. topic) za koje će imati pristup objavljivanju te omogućiti preuzimanje mobilne aplikacije na mobitel koja će sada nuditi korisniku pretplaćivanje na temu određenu njegovim uređajem.

3. IoT platforma

U projektu smo koristili AWS IoT platformu koja nam omogućava povezivanje IoT uređaja i drugih cloud usluga na jednostavan i efikasan način. Ona omogućava komuniciranje preko MQTT, HTTPS-a, MQTT over WSS i LoRaWAN protokola, a nama je posebice interesantan bio MQTT protokol. AWS IoT platforma već sadrži implementiran message broker pa je uspostavljanje komunikacije među komponentama bila značajno lakša. Postoje 3 vrste AWS IoT usluga: analitičke, kontrolne i usluga koje implementiraju određeni softver na uređaju. U analitičke usluge ubrajamo AWS IoT SiteWise, AWS IoT events, AWS TWINMAKER i AWS IoT analytics. AWS IoT SiteWise omogućava sakupljanje, organizaciju i analiziranje podataka dobivenih s uređaja dok AWS IoT events omogućuje detekciju i odgovaranje na događaje generirane od strane senzora i aplikacija. Twinmaker usluga se često koristi za stvaranje identične kopije sustava iz stvarnog svijeta, a IoT analitika služi za upravljanje i analiziranje velikih skupova podataka. Najvažnija kontrolna usluga je AWS IoT Core koji može pružiti uslugu spajanja i do milijarde IoT uređaja te usmjeravanja pripadnih poruka s AWS platforme. AWS IoT Defender služi za fizičko osiguravanje IoT uređaja, a AWS Device Management omogućava organizaciju, praćenje i upravljanje IoT uređajima u sustavima kada je skalabilnost iznimno bitna. Osim već spomenutog message brokera, platforma daje pristup još nekoliko iznimno

korisnih softvera. DeviceShadowService usluga je uz pomoć koje aplikacije mogu nesmetano komunicirati s uređajem, neovisno o tome je li uređaj spojen na Internet. U slučaju kada uređaj nije spojen na Internet, ova usluga jednostavno sprema željeno stanje (npr. aplikacija želi promijeniti stanje, ova će se akcija zapamtiti i izvesti sinkronizacija prilikom sljedećeg spajanja uređaja na Internet). DeviceRegistry usluga je koja se najčešće koristi u situacijama u kojima imamo puno uređaja te želimo zapamtiti određene informacije o svakom. Uz pomoć pravila (eng. rules) podaci se mogu lako prosljeđivati drugim uslugama i resursima, a mi smo u ovom projektu to iskoristili za slanje podataka s AWS platforme na AWS Dynamo bazu podataka. Baza je osmišljena tako da prihvaća poruke koje objavljuju uređaji te ih zajedno s vremenom dospijeca pohranjuje. MQTT poruke koje se prihvaćaju i spremaju mogu se identificirati preko tema (topics) te AWS također može objavljevati poruke u svrhu informiranja uređaja i aplikacija o stanjima i mogućim promjenama događaja. U izgradnji jednog IoT sustava iznimno je bitno i na pravilan način osigurati pristup uređaju, komunikaciju između uređaja i AWS platforme te aplikacije i AWS platforme, a za implementiranje sigurnosnih pravila poslužili smo se AWS IAM sustavom te Security and Identity sustavom sigurnosti u AWS IoT sustavu.

```
// Configure wifi_client with the correct certificates and keys
wifi_client.setCACert(AWS_CERT_CA);
wifi_client.setCertificate(AWS_CERT_CRT);
wifi_client.setPrivateKey(AWS_CERT_PRIVATE);

//Connect to AWS IOT Broker. 8883 is the port used for MQTT
mqtt_client.begin(AWS_IOT_ENDPOINT, 8883, wifi_client);

//Set action to be taken on incoming messages
mqtt_client.onMessage(incomingMessageHandler);

Serial.print("Connecting to AWS IOT");

//Wait for connection to AWS IoT
while (!mqtt_client.connect(THINGNAME)) {
    Serial.print(".");
    delay(100);
}
Serial.println();

if (!mqtt_client.connected()){
    Serial.println("AWS IoT Timeout!");
    return;
}

//Subscribe to a topic
mqtt_client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
mqtt_client.subscribe(AWS_IOT_DISABLE_TOPIC);
```

Slika 4.7: Prikaz konfiguracije ESP32 klijenta na AWS platformu

AWS IoT platforma omogućuje povezivanje milijarde IoT uređaja te slanje 1000 puta više poruka prema AWS uslugama i drugim uređajima. Naplaćuju se samo one komponente koje se koriste te ne postoji minimalna ili obavezna naknada. Naplaćivanje se vrši odvojeno za povezivanje, slanje poruka, pohranjivanje stanja uređaja, pohranjivanje metapodataka sa uređaja te transformaciju i slanje poruka.

AWS Free Tier je dostupan korisnicima u svim regijama osim GovCloud (US) regiji kroz 12 mjeseci nakon otvaranja AWS računa te su korisnici nakon njegovog isteka ili prekoračenjem limita dužni platiti za korištenje AWS IoT platforme. Kriteriji koji se uzimaju u obzir pri prekoračenju limita su 2 250 000 minuta povezanosti, 500 000 poruka, 225 000 pohranjivanja stanja uređaja ili metapodataka te 250 000 pravila i isto toliko izvršenih akcija.

Naplata ovisi o regiji, a s obzirom da smo radili u regiji Europe (Frankfurt), cijene će biti prikazane samo za tu regiju. Za povezivanje je za milijun minuta potrebno izdvojiti 0.096\$, a inkrementira se minutno. Cijena za slanje poruka je 1.2\$/ milijun poruka za prvih milijardu poruka, 0.96\$/milijun poruka za iduće 4 milijarde te 0.84\$/milijun poruka za preko 5 milijardi poruka. Za MQTT i HTTP se ne uračunavaju primljene ili poslane poruke preko rezerviranih tema. Maksimalna duljina poruke je 128kB, a mjere se u inkrementima od 5kB. Pohranjivanje stanja uređaja i metapodataka je mjereno brojem operacija koje dohvaćaju ili modificiraju te podatke. Cijena za milijun takvih operacija je 1.5\$ te se mjere u inkrementima od 1kB. Transformacija poruka omogućava transformaciju poruka dobivenih s uređaja koristeći aritmetičke operacije ili vanjske funkcije kao što su AWS Lambda te njihovo preusmjeravanje do AWS platformi kao što su Amazon simple storage service, Amazon DynamoDB ili Amazon Kinesis. Korištenje transformacije je mjereno svaki puta kada se ta transformacija izvodi te brojem akcija izvedenih u transformaciji, s minimalno jednom akcijom po transformaciji. Naplaćuje se 0.18\$ po milijunu transformacija te isto toliko za milijun akcija. Transformacije i akcije su mjerene u inkrementima od 5kB.

4. Korisničke aplikacije

Razvijene su dvije korisničke aplikacije: mobilna aplikacija i web aplikacija. Mobilna aplikacija namijenjena je korisnicima operacijskog sustava Android. Ona je razvijena u programskom jeziku Java. Integrirano programsko okruženje (IDE) koje se koristilo za razvoj android aplikacije je Android Studio. Android studio se bazira na JetBrains softveru te je namijenjen isključivo za razvoj android aplikacija. Ima zaseban “build system” koji predstavlja skup alata koji se koriste za izgradnju, testiranje i pokretanje aplikacije. AVD Manager unutar Android studia predstavlja alat za upravljanje virtualnim uređajima koji služe za pokretanje aplikacije. On dozvoljava korisnicima da sami izgrade svoj virtualni uređaj koji mogu prilagoditi svojim potrebama (npr. mogu odabrati veličinu uređaja). U ovom projektu android aplikaciju pokretana je na uređaju: Pixel 5, te je korišten API 31.

Svaka android aplikacija, pa tako i ova, sadrži datoteku naziva Android manifest koja sadrži ključne informacije o aplikaciji koje operacijski sustav mora imati prije pokretanja aplikacije (slika 4.1). Ako aplikacija treba pristupiti određenim funkcionalnostima, to se mora posebice naglasiti u Android manifestu korištenjem “<uses-permission>” elementa. Jednom kada je aplikacija instalirana na uređaj, korisnik sam odlučuje hoće li dozvoliti pristup za spajanje na Internet (kao što je ovdje slučaj).


```

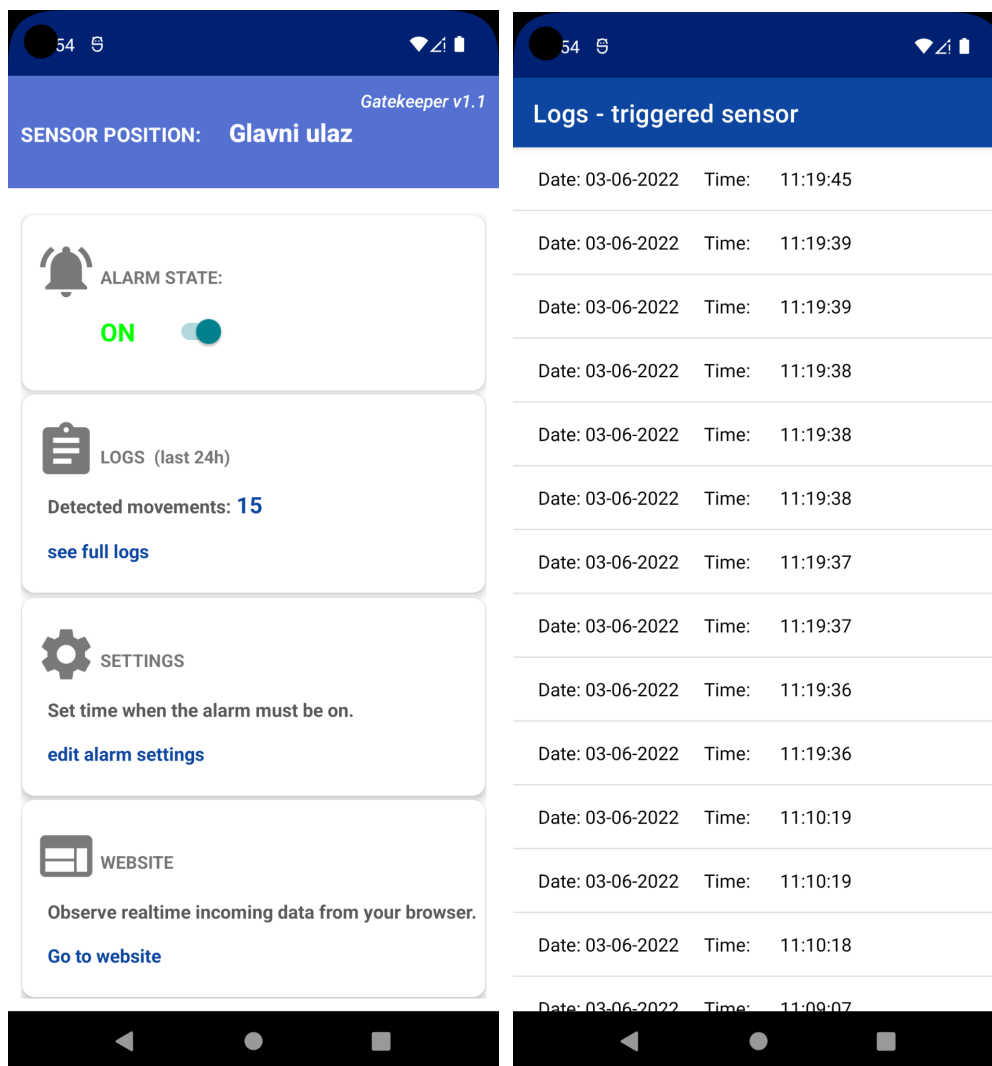
1  <?xml version="1.0" encoding="utf-8"?>
2  <manifest xmlns:android="http://schemas.android.com/apk/res/android"
3      xmlns:tools="http://schemas.android.com/tools"
4      package="com.example.gatekeeperapp">
5      <uses-permission android:name="android.permission.INTERNET" />
6      <application
7          android:allowBackup="true"
8          android:dataExtractionRules="@xml/data_extraction_rules"
9          android:fullBackupContent="@xml/backup_rules"
10         android:icon="@mipmap/ic_launcher"
11         android:label="@string/app_name"
12         android:roundIcon="@mipmap/ic_launcher_round"
13         android:supportRtl="true"
14         android:theme="@style/Theme.GatekeeperApp"
15         tools:targetApi="31">
16         <activity
17             android:name=".PubSubActivity"
18             android:exported="true"
19             android:label="@string/app_name"
20             android:theme="@style/Theme.GatekeeperApp.NoActionBar">
21             <intent-filter>
22                 <action android:name="android.intent.action.MAIN" />
23                 <category android:name="android.intent.category.LAUNCHER" />
24             </intent-filter>
25         </activity>
26     </application>
27
28 </manifest>

```

Slika 4.8 Android manifest (screenshot iz Android studia)

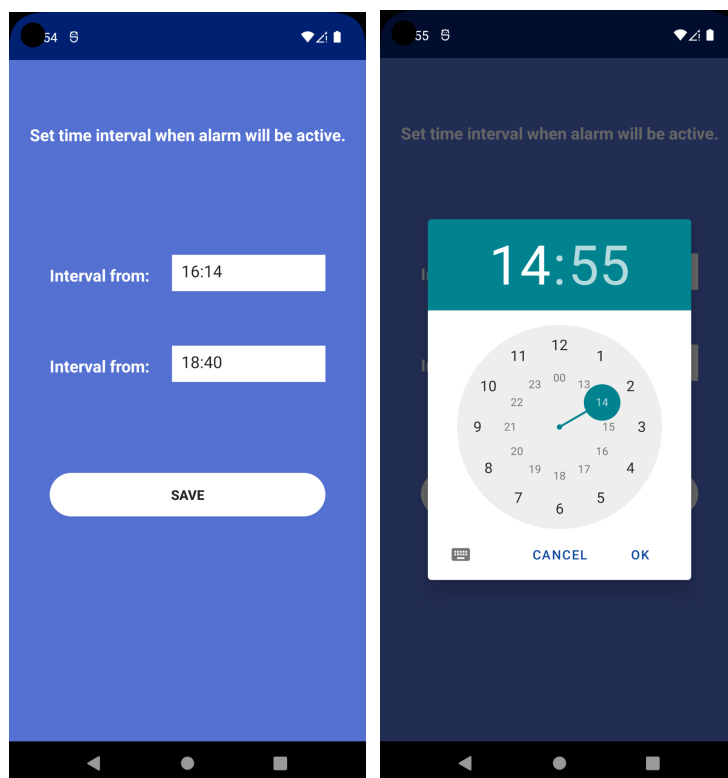
Android aplikacija se sastoji od jedne ili više aktivnosti. Svaka aktivnost predstavlja vizualno korisničko sučelje i mora biti definirana u Android manifest datoteci. Ako je akcija, unutar manifest datoteke, definirana kao “Launcher”, to znači da je to početna aktivnost koja će biti pozvana odmah nakon ulaza u aplikaciju. Svaka android aktivnost mora imati svoju odgovarajuću “layout” datoteku napisanu u programskom jeziku XML. Ta layout datoteka predstavlja korisničko sučelje aktivnosti te se u njoj definiraju widgeti i ostale komponente bitne za razvoj aplikacije. Aktivnost se razlikuje od obične Java klase po tome što proširuje (engl. extends) klasu “Activity” zbog čega mora implementirati metodu “onCreate()”. Ta metoda predstavlja stvaranje aktivnosti i unutar te metode definira se povezanost varijabli iz Java koda sa XML datotekom.

Na slici 4.9 prikazano je početno korisničko sučelje koje korisnik vidi nakon otvaranja aplikacije. Uočava se da se na zaslonu nalaze 4 različite “komponente”. Na samom vrhu nalaze se naslov aplikacije i naziv mjesta gdje se nalazi senzor. Nakon toga vidljivo je stanje alarma (je li alarm trenutno upaljen ili ugašen). Sljedeća komponenta sadrži informaciju o broju očitavanja posljednjih 24 sata te text link naziva “see full logs”. Klikom na njega, korisnik vidi detalje o tim očitanjima (datum i vrijeme očitavanja (slika 4.10).

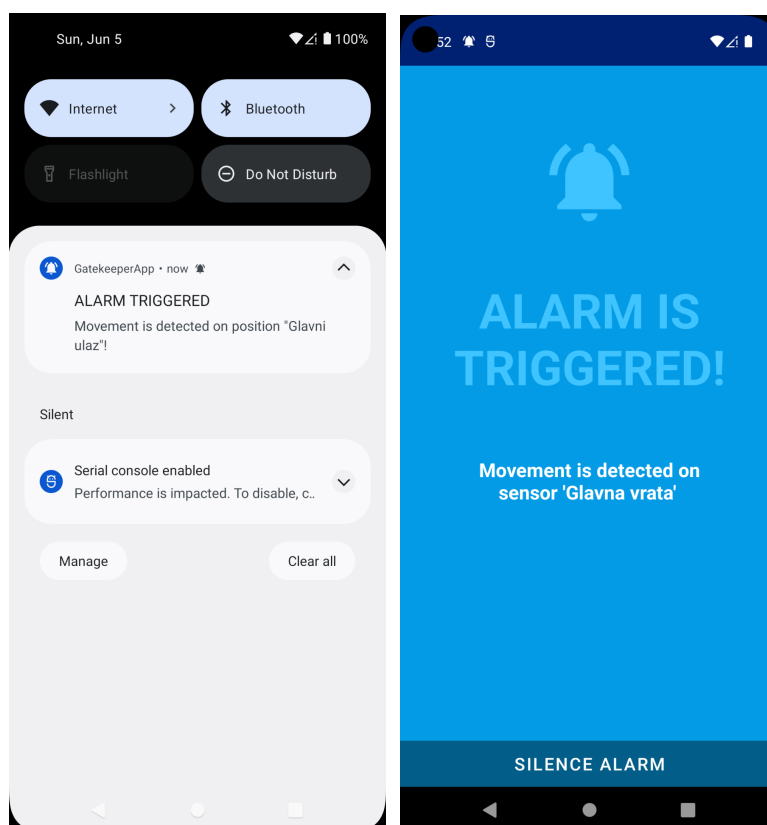


Slike 4.9 i 4.10 Početni ekran i ispis detalja o detektiranim pokretima

Komponenta “Settings” omogućava korisniku da sam postavi vremenski period u kojem želi da se alarm pali na detektirane pokrete koje senzor uoči (slike 4.11 i 4.12). Npr. ako korisnik unese vremenski period od 21 do 6 sati, to znači da će alarm samo u tom razdoblju reagirati ako senzor detektira neki pokret. Točnije, samo u tom razdoblju će se upaliti nakon čega će korisniku doći obavijest da je alarm upaljen (slika 4.13) te će ga korisnik klikom na gumb moći utišati (slika 4.14). U nekom drugom vremenskom periodu, npr. u 14 sati, alarm se neće upaliti bez obzira je li senzor detektirao neke pokrete ili ne. Posljednja komponenta naziva se “Website” sadrži text link koji bi trebao voditi na web stranicu aplikacije. Trenutno taj link nije postavljen jer web aplikacija nije deployana, već se vrti lokalno na računalu. U budućnosti je plan da se web aplikacija deploya.

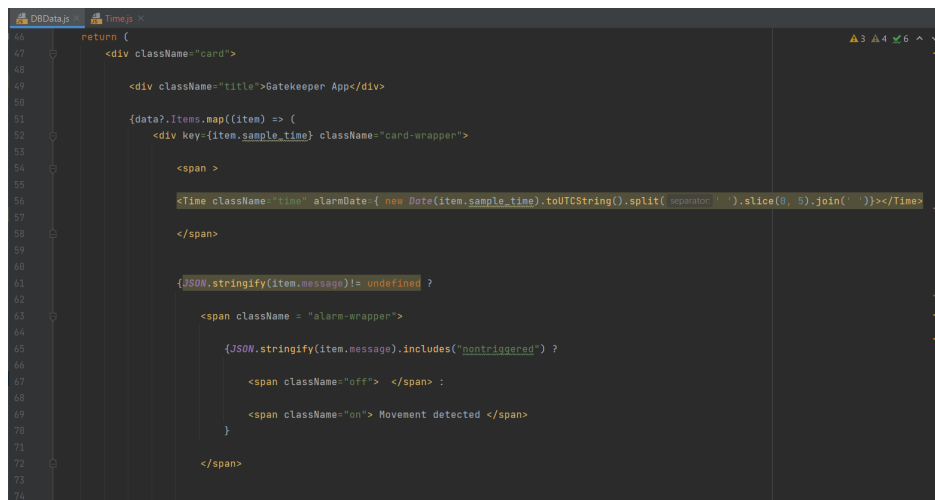


Slike 4.11 i 4.12 Odabir vremenskog intervala



Slike 4.13 i 4.14 Obavijest o alarmu i utišavanje alarma

Web aplikacija napravljena je koristeći programski okvir React. React je Javascript programski okvir koji je specijaliziran za pomoć programerima pri izradi korisničkih sučelja. React prikazuje korisničko sučelje na način da programer stvara komponente koje onda zajedno uključi u “glavnu” komponentu i na taj način nastaje stranica koja sadrži više komponenti koje zajedno djeluju kao cjelina. Primjer komponente DBData.js koja unutar sebe sadrži komponentu Time prikazan je na slici 4.15. Komponente su zapravo slične Javascript funkcijama. Primaju parametre te vraćaju React elemente koji opisuju što će se pojaviti na ekranu.



```

16  return (
17    <div className="card">
18      <div className="title">Gatekeeper App</div>
19      <div>
20        {data?.Items.map((item) => (
21          <div key={item.sample_time} className="card-wrapper">
22            <span>
23              <Time className="time" alarmDate={new Date(item.sample_time).toLocaleString().split(' ').slice(0, 5).join(' ')}>
24            </span>
25            <span className="alarm-wrapper">
26              {JSON.stringify(item.message) != undefined ?
27                <span className="off"> </span> :
28                <span className="on"> Movement detected </span>
29              }
30            </span>
31          </div>
32        )
33      }
34    </div>
35  )

```

Slika 4.15 Komponenta DBData.js (screenshot iz IntelliJ)

Izrađena web aplikacija prikazuje naziv aplikacije te popis očitavanja kada je senzor detektirao pokret (slika 4.16). Očitavanja sadrže informacije o datumu i vremenu detektiranih pokreta. Povučena su iz Dynamo baze podataka sa AWS-a te služe samo kao informacija korisniku koji na ovaj način lako može saznati kada je senzor detektirao pokret.

Gatekeeper App		
20 January 1970 3h 32min 30s		Movement detected
03 June 2022 9h 9min 7s		Movement detected
03 June 2022 9h 10min 18s		Movement detected
03 June 2022 9h 10min 19s		Movement detected
03 June 2022 9h 10min 19s		Movement detected

Slika 4.16 Web aplikacija