Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

HOMEGUARD: SVILUPPO DI UNA PIATTAFORMA IOT PER LA SICUREZZA DOMESTICA

Relatore:
Prof.
Federico Montori

Presentata da:
Matteo Raggi

II sessione - secondo appello
Anno Accademico 2023 / 2024

Sommario

HomeGuard è un sistema di monitoraggio e sicurezza per la propria casa. È composto da una applicazione in Kotlin, una fotocamera e un allarme fornito di sensore NFC e sensore magnetico per l'apertura della porta.
Applicazione e dispositivo fisico sono collegati utilizzando un servizio Cloud sviluppato da Amazon: Amazon Web Services (AWS).
Questo sistema di allarme ha la possibilità di attivarsi e disattivarsi da solo, basandosi sulla posizione degli utenti che hanno scaricato l'app.
Ha accesso alla posizione in ogni momento, in questo modo può attivare l'allarme quando non rileva più utenti all'interno dell'abitazione.
Grazie a questa funzionalità, l'allarme è molto semplice da utilizzare e non richiede manutenzione. Può essere interamente gestito in automatico dal sistema, senza mai dover modificare lo stato in modo manuale.
Altri metodi di gestione implementati sono: attraverso un pulsante sull'applicazione oppure usando una delle tessere NFC registrate.

Introduzione

La tesi 'HomeGuard' si concentra sul fornire un nuovo sistema di monitoraggio della casa, controllabile da Applicazione Mobile.
L'Applicazione Mobile funge da punto di controllo per la gestione completa della sicurezza domestica.
Comunicando con la struttura Backend costruita sulla piattaforma Serverless AWS (Amazon Web Services) potrà inviare e ricevere aggiornamenti e segnali dai dispositivi Hardware.

La parte Hardware è formata da dispositivi ESP32 che attraverso numerosi sensori permettono di controllare lo stato dell'abitazione.
È presente una fotocamera di sicurezza, un sensore magnetico per la porta, un sensore NFC, un buzzer sonoro e un display LCD.
Ciò che permette a questa nuova applicazione di essere innovativa sul mercato è la possibilità di gestire gli accessi alla casa in modo automatico così da garantire una gestione dell'allarme molto più semplice e intuitiva.

L'utente avrà la possibilità di consentire l'accesso alla posizione, in questo modo l'app saprà sempre dove esso si trova.
Sapendo quando sarà in casa o quando sarà fuori, l'app potrà gestire l'allarme di conseguenza, in modo automatico.
E grazie alla possibilità di creare più utenti della casa, l'allarme si attiverà solo quando gli utenti saranno tutti fuori.

Ma oltre a questo, con l'applicazione è possibile:

- Attivare/disattivare l'allarme
- Visualizzare chi è in casa
- Vedere i record dei precedenti allarmi scattati (orario, fotografie, video)
- Gestire ogni dispositivo connesso (come l'altoparlante, i sensori di movimento, il sensore di impronte,...)
- Vedere il filmato di sicurezza in tempo reale
- Registrare impronte o tessere nfc per attivare e disattivare l'allarme dell'abitazione

Indice

Elenco delle figure

Capitolo 1

Stato dell'arte

In questo capitolo viene analizzato lo stato di avanzamento attuale delle tecnologie utilizzate per questo progetto.

1.1 Introduzione all'Internet of Things

Il termine "Internet of Things", in italiano "Internet delle cose", viene usato per indicare oggetti Smart che si connettono ad Internet e che possono interagire tra di loro e con altri servizi.
Questo livello di connettività permette al sistema di essere efficiente, sostenibile e affidabile.[1]
La capacità di connettersi a Internet rende possibile, per la prima volta, una gestione e un monitoraggio remoto di oggetti di uso comune.

L'IoT si sta evolvendo continuamente, sempre più dispositivi vengono aggiunti ogni giorno e l'industria è ancora agli inizi.[2]
Secondo IoT Analytics State of IoT Summer 2024 report, alla fine del 2023 erano presenti 16.6 miliardi di dispositivo connessi, quindi una crescita del 15% rispetto all'anno precedente.
Si prevede una ulteriore crescita del 13%, fino ad un totale di 18.8 miliardi di dispositivi entro la fine del 2024. [3]

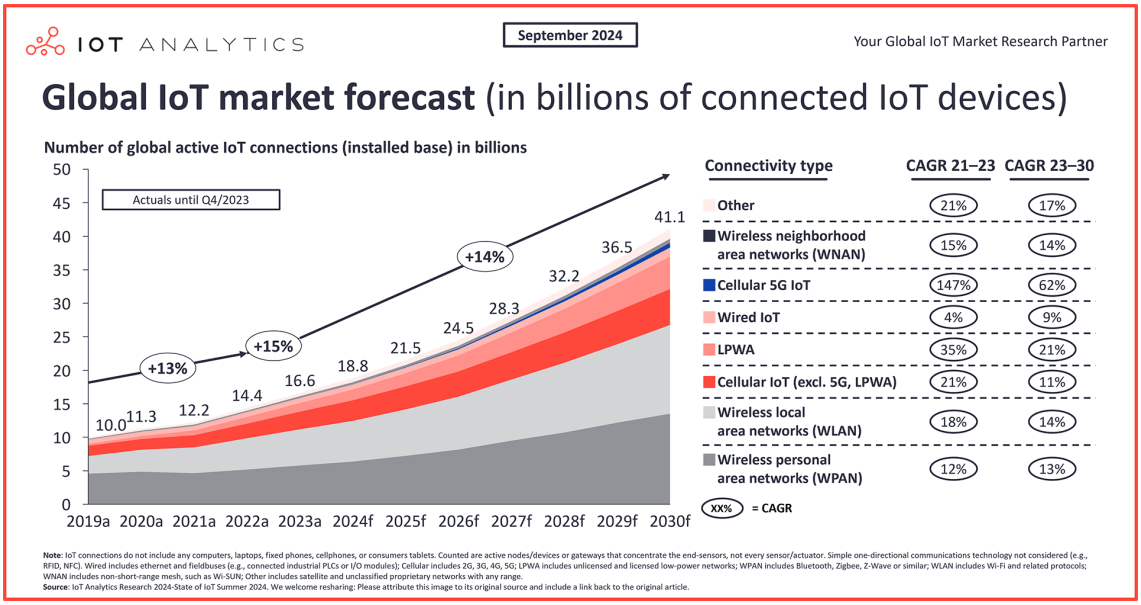

Figura 1.1: Numero di dispositivi IoT nel mondo e previsione [3]

La vera potenzialità di questa tecnologia è creare un'unica rete di dispositivi controllabili da un singolo endpoint.
Lo scopo è che ogni dispositivo Smart possa essere compatibile con qualsiasi tipo di Applicazione IoT.
Ci sono molti campi di applicazione di questa tecnologia. Nel mondo del lavoro per velocizzare alcune task, in casa per semplificare la gestione della sicurezza e delle luci oppure per la realizzazione di Smart City.
Nonostante negli ultimi anni dispositivi come Alexa, Apple Homepod e Google Home si siano espansi a macchia d'olio, non esistono ancora soluzioni che evitano all'essere umano compiti che non vorrebbero svolgere.

HomeGuard si posiziona nel settore delle Smart Home. La scelta di creare un prodotto in questa sezione di mercato è determinato dal fatto che è un settore in grande espansione, ma ancora molto sottovalutato e privo di soluzioni realmente utili.

Figura 1.2: Divisione di Use Case dell'IoT [4]

1.2 Smart Home e Home Security

L'obiettivo delle Smart Homes è quello di dare agli utenti sicurezza, convenienza, comfort, efficienza energetica, intrattenimento e in generale di migliorare la loro qualità della vita a casa.[5]
Beneficiando dell'espansione dell'IoT, le persone si aspettano che i servizi forniti dalle Smart Home siano in continua evoluzione, e che i servizi offerti migliorino le loro vite.
Solitamente il termine Smart Homes è usato per descrivere l'utilizzo di monitoraggio avanzato e funzionalità di controllo per le abitazioni.[6]
Le persone cercano comodità e sicurezza, specialmente dentro le mura di casa. E l'IoT è perfetto per questo.
Offre la possibilità di creare automazioni, utili per non dover pensare a sicurezza, pulizia

e risparmio di energia in casa.
Proprio per questi motivi il binomio Casa - Internet of Things è vincente, ed è destinato ad espandersi sempre di più.

Per quanto riguarda in particolare la sicurezza della casa, le persone sono interessate ad una gestione semplice e, ovviamente, sicura, ed è proprio quello che l'IoT può fornire. Si tratta anche di un particolare campo delle Smart Home, perché gli allarmi sono poco diffusi nelle abitazioni italiane.
Probabilmente per motivi di: costo, manutenzione, difficoltà di utilizzo o semplicemente perché non pensano di averne bisogno.
Un sistema di allarme IoT si potrebbe espandere molto, avendo costi bassi ed essendo semplice da utilizzare.

1.3 Cosa rende HomeGuard innovativo

Esistono molti sistemi di monitoraggio della casa, alcuni anche con Applicazione per la gestione da remoto.
Ma questo progetto si è concentrato sul migliorare quelli che sono i punti di forza di avere un sistema di Internet of Things nella propria abitazione.
L'obiettivo principale è stato rendere l'utilizzo dell'allarme estremamente semplice, e l'ho voluto fare implementando un sistema di gestione automatico.

Con HomeGuard, dopo l'installazione iniziale, non c'è bisogno di modificare lo stato dell'allarme manualmente. Verrà controllato da solo grazie alla posizione del tuo dispositivo mobile che l'applicazione riceverà costantemente.
L'IoT in questo modo semplifica estremamente la vita dell'uomo, sollevandolo da task stressanti o di cui spesso ci dimentichiamo.

Si concentra sul fornire un servizio che sia comodo ed efficiente: l'allarme di HomeGuard fa dimenticare al proprietario di averlo installato, ed è questo l'obiettivo di un sistema di sicurezza della casa, che non deve essere un peso aggiuntivo.

L'intero progetto è anche aperto al possibile utilizzo di altri dispositivi di sicurezza, proprietari e non.

Capitolo 2

Panoramica del progetto

In questo capitolo verranno analizzate le tecnologie presenti in questo progetto. Esse sono tre:

- Dispositivi Hardware ESP32
- Backend Serverless su Amazon Web Services
- Applicazione Android in Kotlin

Verranno spiegate in modo indipendente tra di loro, così da mostrare le funzionalità anche al di fuori dell'utilizzo che ne viene fatto in questa tesi. La loro implementazione in questo progetto verrà spiegata nel capitolo successivo in modo approfondito.

2.1 Dispositivo Esp32

Il Dispositivo ESP32 è l'Hardware del progetto è ciò che viene definito "Smart Object" in un progetto di Internet of Things.
ESP32 è un microcontrollore, come il suo predecessore Arduino, ma con sensori Wi-Fi e Bluetooth integrati.[7]

Un microcontrollore è un componente che contiene in un unico e piccolo chip diversi componenti come RAM, ROM, ingressi I/O e altri.

La presenza dei sensori Wi-Fi e Bluetooth rende l'ESP32 un dispositivo adatto ad essere programmato per progetti di IoT. Inoltre, essendo un chip poco costoso, si adatta anche a sviluppo amatoriale e sperimentale.

Dal punto di vista della programmazione, è molto simile ad Arduino. Entrambi utilizzano il linguaggio di programmazione C e possono usare lo stesso IDE. Nel mio caso ho deciso di utilizzare il compilatore ufficiale di Arduino.
La creazione di uno Smart Object si divide in due parti:

- Assemblaggio
- Sviluppo codice

Lo sviluppo del suo codice verrà analizzato nella parte di Implementazione di questa tesi. L'Assemblaggio nella sezione seguente.

2.1.1 Assemblaggio

I dispositivi che utilizzo sono due: uno è l'Allarme e l'altro la Fotocamera. L'ESP32 di Allarme è collegato a:

- Display oled integrato
- Sensore magnetico per rilevare l'apertura della porta
- Buzzer sonoro
- Sensore RFID per la lettura di tessere NFC

Sono collegati secondo questo schema:

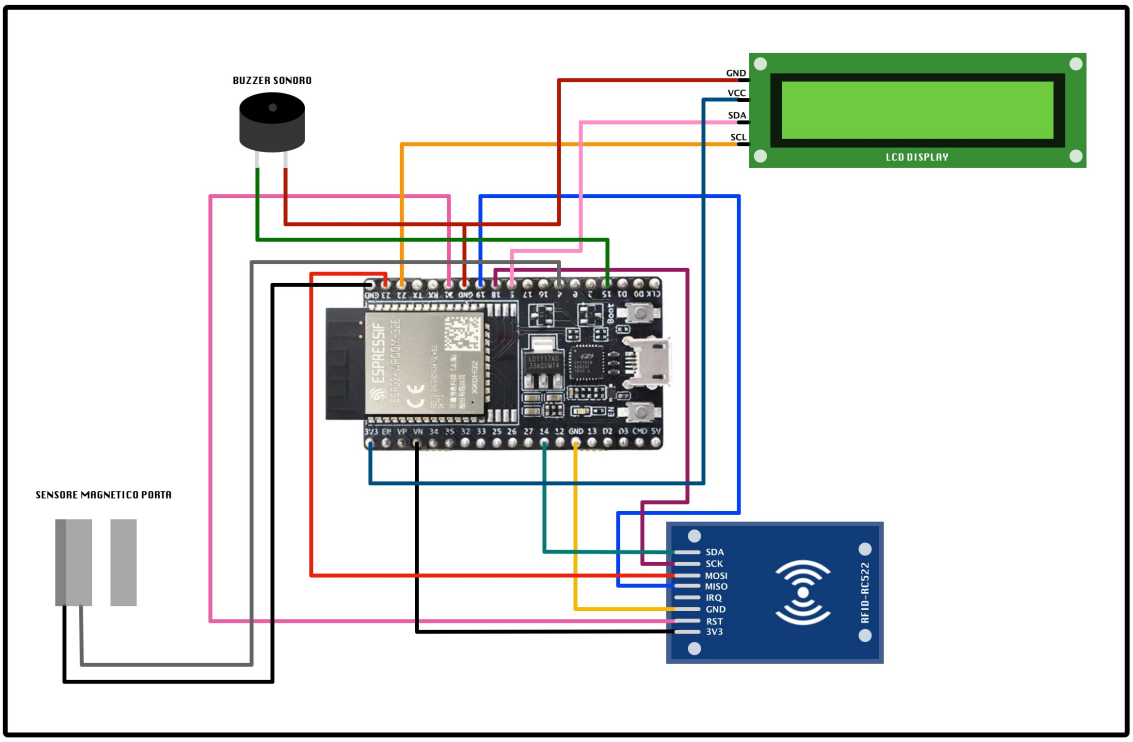

Figura 2.1: Collegamento ESP32 con Sensori

Il Display Oled serve a mostrare all'utente se l'allarme è acceso o spento e a mostrare se la tessera NFC è stata rilevata e accettata quando viene scannerizzata.
Il sensore magnetico per l'apertura della porta è composto da due pezzi, vengono posti vicini tra di loro ma uno sulla porta e uno sul muro. Quando la porta viene aperta e si allontanano, viene rilevato il cambiamento.
Il buzzer sonoro suona ad intermittenza quando l'allarme scatta, quindi quando il sensore della porta riceve un cambiamento di stato e l'allarme è attivo.
Il lettore RFID è pronto a leggere le tessere che vengono registrate tramite l'applicazione Mobile.

L'ESP32 che funge da fotocamera invece non è collegata a niente di esterno. È un dispositivo chiamato ESP32 CAM e ha una fotocamera già integrata.

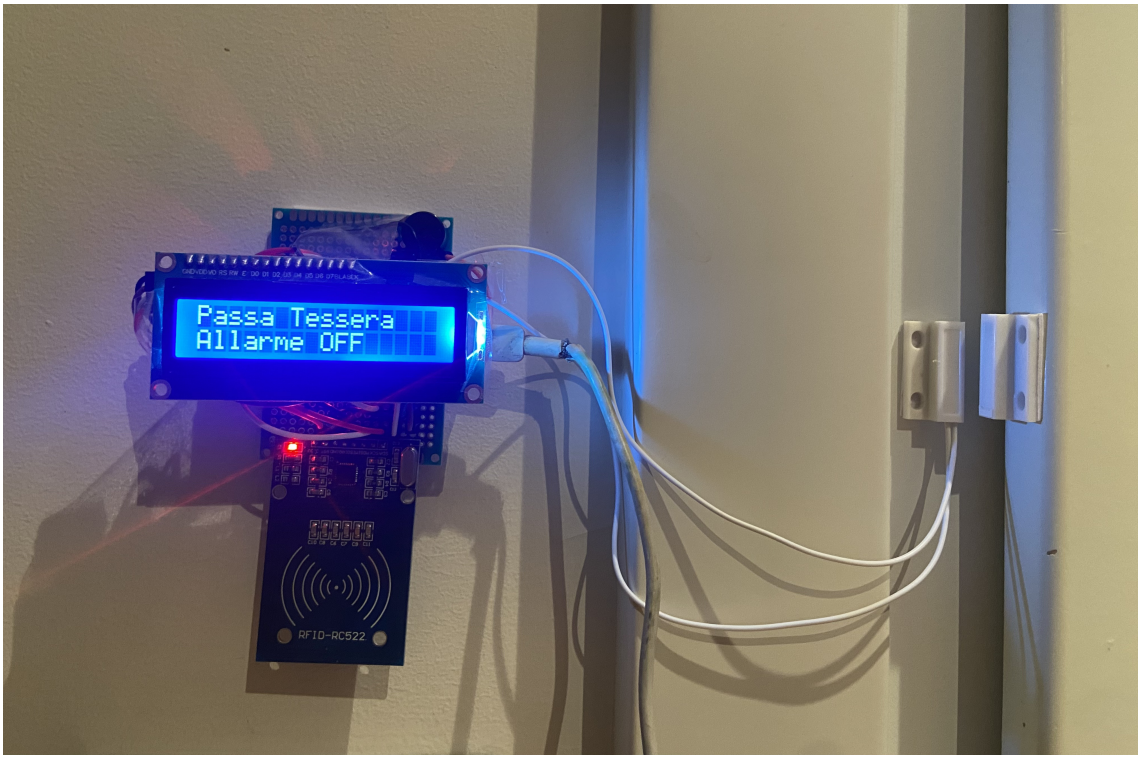

Figura 2.2: Dispositivo ESP32 saldato e applicato

2.1.2 Input e Output dell'ESP32

Per capire bene come collegare i sensori esterni al microprocessore, è necessario conoscere bene lo schema dei Pin dell'ESP32.

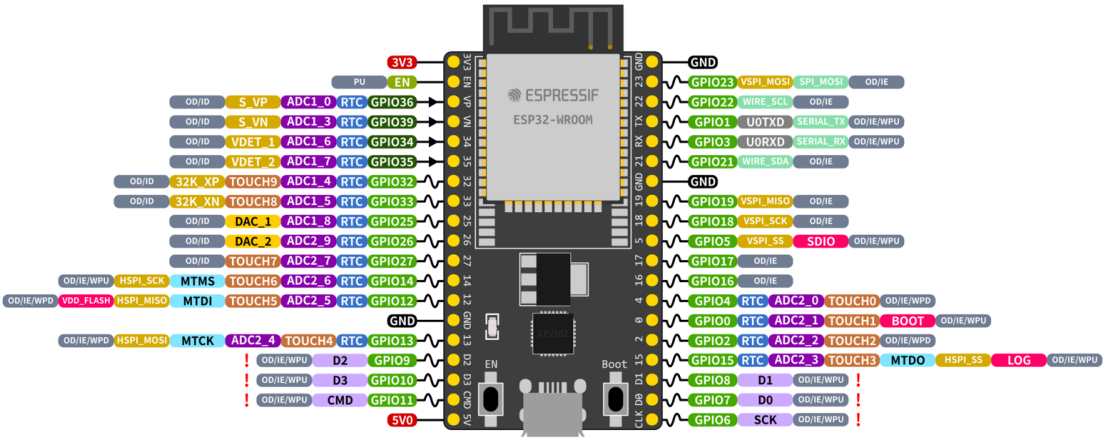

Figura 2.3: Pinout di un ESP32 [8]

I Pin non sono tutti uguali, si dividono in diversi tipi in base al colore che gli viene assegnato. La maggior parte sono Pin di Input/Output, quelli chiamati "GPIO". Mentre il Pin nero (GND) è il Ground: questi pin funzionano come collegamenti di terra, stabilendo la tensione di riferimento per l'intero sistema.
I Pin rossi invece (VIN o 3.3V) sono collegati alla fonte di energia del microprocessore.

I Pin fisici sono una delle limitazioni di questo microprocessore low-cost, infatti potrebbero non essere sufficienti a collegare tutti i sensori. L'altra limitazione è la memoria della scheda che è completamente occupata dallo Sketch (codice in C) caricato.
Questi due problemi sono anche i motivi per i quali ho dovuto utilizzare due ESP32 diversi. Oltre al fatto che l'allarme e la fotocamera è meglio che siano posizionati in luoghi differenti della abitazione.

2.2 Amazon Web Services (AWS)

Amazon Web Services (AWS) è il servizio Cloud più completo e utilizzato del mondo e offre più di 200 servizi.
Il cloud computing consiste nella distribuzione on-demand delle risorse IT tramite Internet, con una tariffazione basata sul consumo.
Piuttosto che acquistare, possedere e mantenere i data center e i server fisici, è possibile accedere a servizi tecnologici sulla base delle proprie necessità affidandosi a un fornitore cloud quale, appunto, Amazon Web Services (AWS).[9]

2.2.1 I vantaggi del Cloud Computing

Il primo vantaggio è l'Agilità di sviluppo e di implementazione. Il cloud permette di accedere a molte tecnologie diverse in modo semplice e veloce.
Quindi ti fornisce la possibilità di aumentare in modo rapido le risorse utilizzate, e di iniziare ad utilizzare altri servizi complementari in tempo quasi immediato e questo offre la libertà di sperimentare, testare nuove idee per differenziare le esperienze dei clienti e trasformare il tuo progetto.[9]

Il secondo vantaggio è l'Elasticità. Con il cloud non c'è la necessità di allocare in anticipo le risorse che pensiamo ci possano servire in futuro. Potrai ridimensionare tali risorse in qualsiasi momento, per aumentare o ridurre in modo istantaneo le capacità, adattandole alle necessità dell'azienda.

Il terzo vantaggio è il risparmio sui costi. Il cloud permette di evitare spese fisse in favore di una spesa variabile, pagando solo per le risorse realmente consumate.
Inoltre, le spese variabili sono nettamente inferiori rispetto a quanto potresti pagare autonomamente, grazie alle maggiori economie di scala.

L'ultimo vantaggio è la possibilità di distribuire il prodotto globalmente in pochi minuti. Grazie al cloud, possiamo espanderci in nuove regioni geografiche in un attimo.

Per esempio, l'infrastruttura AWS offre una copertura globale, così che tu possa distribuire la tua applicazione in più luoghi fisici in modo semplice. Fare in modo che le applicazioni siano vicino agli utenti finali riduce la latenza e migliora la loro esperienza.

2.2.2 I servizi di AWS che ho utilizzato

Amazon Web Services, come detto in precedenza, ha più di 200 servizi. Alcuni di essi ricoprono ruoli differenti, altri simili tra loro. Ma ci sono servizi che ricoprono un ruolo più importante nella piattaforma cloud, e vengono utilizzati maggiormente.
Io ne ho utilizzati principalmente sette. In questa sezione spiegherò brevemente lo scopo di ognuno di essi.

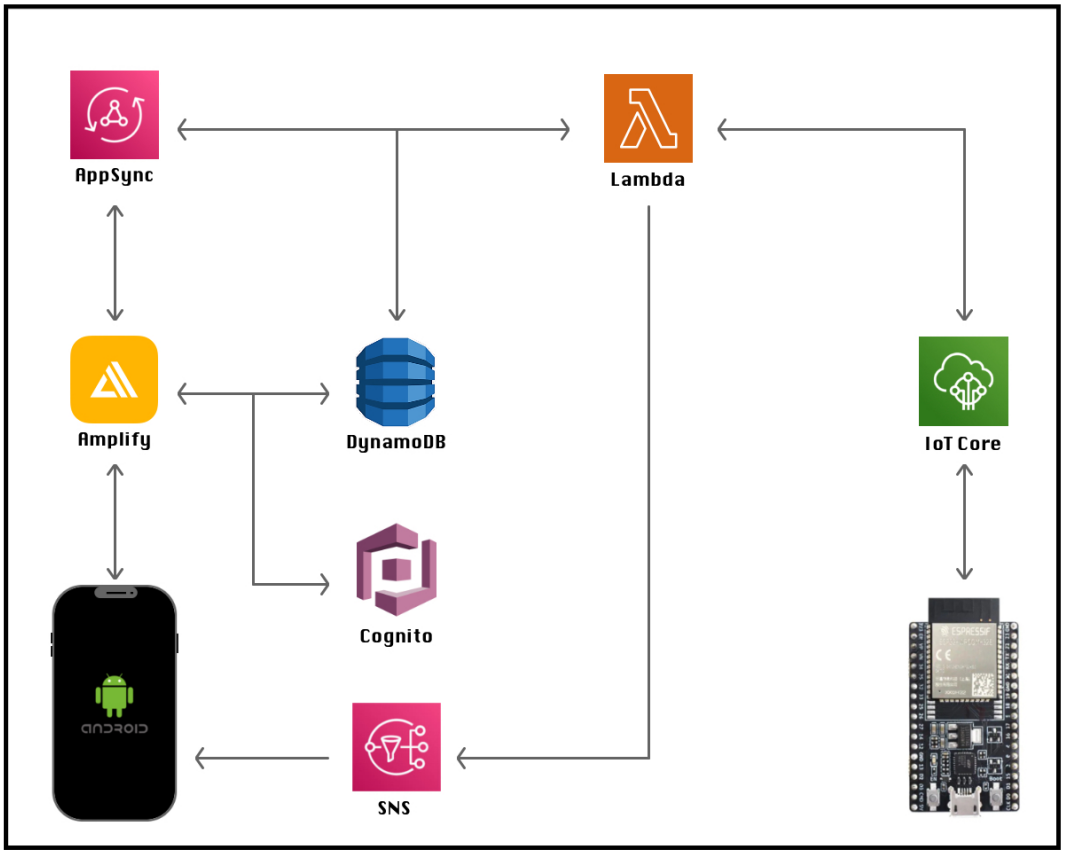

Figura 2.4: Rete di collegamenti dei servizi aws usati

DynamoDB

DynamoDB è un database NoSQL e serverless, totalmente gestibile dalla piattaforma Amazon. Essendo un servizio Serverless, con DynamoDB non è necessario effettuare il provisioning di alcun server o gestire alcun software e offre una manutenzione senza tempi di inattività. Questo database è progettato appositamente per offrire prestazioni, scalabilità, gestibilità e flessibilità migliorate rispetto a database relazionali tradizionali.[9]

Essendo un servizio di database completamente gestito, AWS si occupa della sua gestione. Conduce autonomamente l'installazione, le configurazioni, la manutenzione, la sicurezza, i backup, il monitoraggio e altro ancora.
In questo modo, quando si crea una tabella DynamoDB, questa è immediatamente pronta per i carichi di lavoro di produzione.

IoT Core

AWS IoT Core fornisce un servizio cloud che serve a connettere i dispositivi IoT fisici ad altri dispositivi o servizi cloud.
Una volta connesso un qualsiasi dispositivo ad IoT, la piattaforma potrà connetterlo a tutti gli altri servizi cloud che provvede.

AWS IoT fornisce i seguenti protocolli per comunicare con il dispositivo intelligente: MQTT (Message Queuing and Telemetry Transport), MQTT over WSS, HTTPS (Hypertext Transfer Protocol - Secure) e LoRaWAN (Long Range Wide Area Network).

Il ruolo di questo servizio è fondamentale in un progetto di Internet of Things, essendo il ponte tra il dispositivo fisico e il cloud. Il collegamento dell'oggetto smart viene fatto con estrema attenzione alla sicurezza informatica. Ogni "things" ha un identificativo unico sul cloud e solo gli oggetti registrati possono comunicare con il servizio MQTT o HTTP fornito da Amazon.

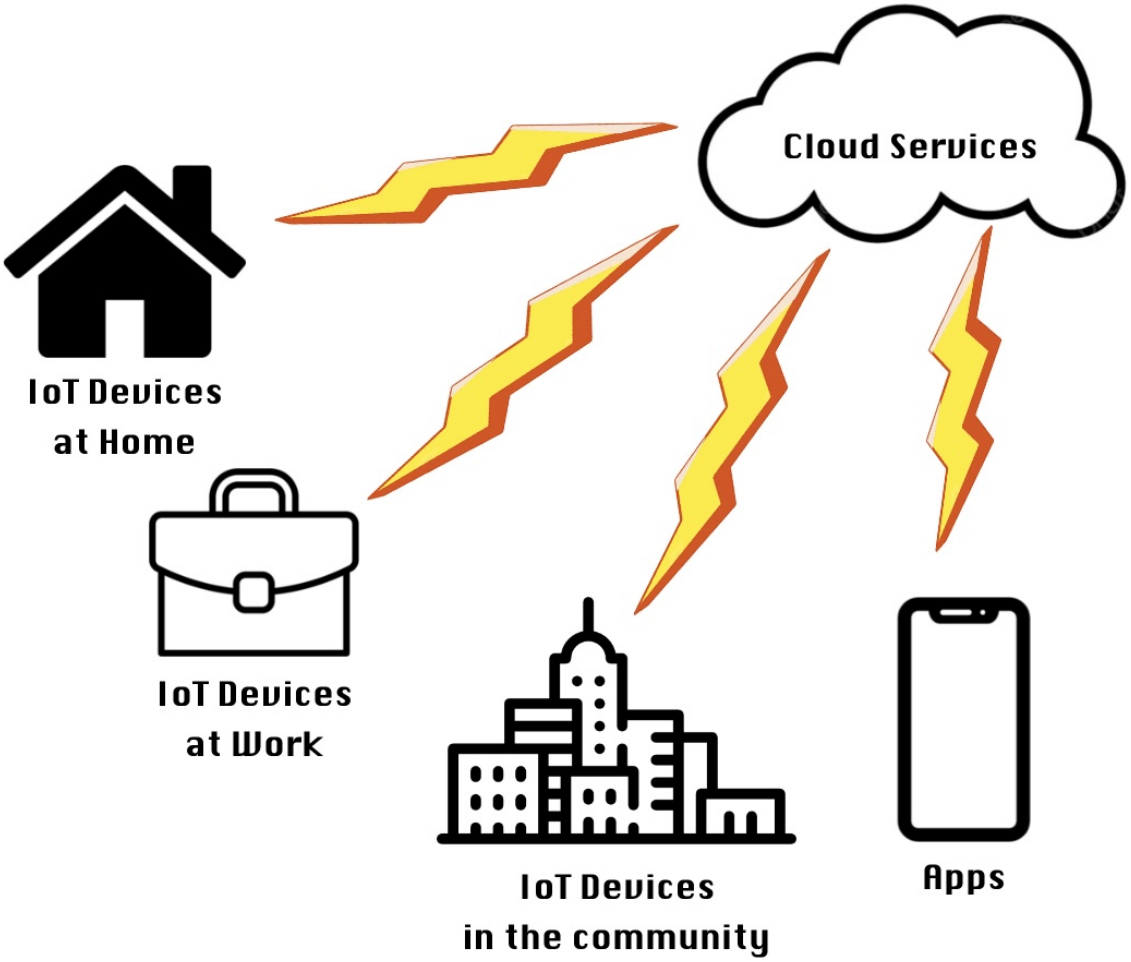

Figura 2.5: Il ruolo di aws iot core in un progetto iot

È quando il dispositivo viene connesso che diventa, da oggetto comune, un oggetto smart. Smart significa che ottiene la sua identità in rete, potendo quindi collegarsi, attraverso servizi come quelli di Amazon, ad altri servizi, applicazioni o oggetti smart.
In questo progetto, come verrà spiegato nella sezione di Implementazione, i dispositivi vengono collegati ad altri Smart Objects (Allarme - Fotocamera) e all'Applicazione Mobile Android.
Essa fungerà da punto di controllo per tutti gli oggetti intelligenti.

Creare un dispositivo Smart sulla piattaforma Amazon significa creare un Certificato

e una Policy.
I certificati autenticano le connessioni dei dispositivi e dei client. Devono essere registrati con AWS IoT e attivati prima che un dispositivo o un client possa comunicarci.
Le policy consentono di controllare l'accesso alle operazioni di AWS. Questo serve ad evitare che ogni elemento presente nella console cloud possa interagire in modo libero e compiere qualsiasi operazione.
Un esempio di operazione che deve essere permessa per poter essere eseguita è il cambiamento di valori in un database DynamoDB.

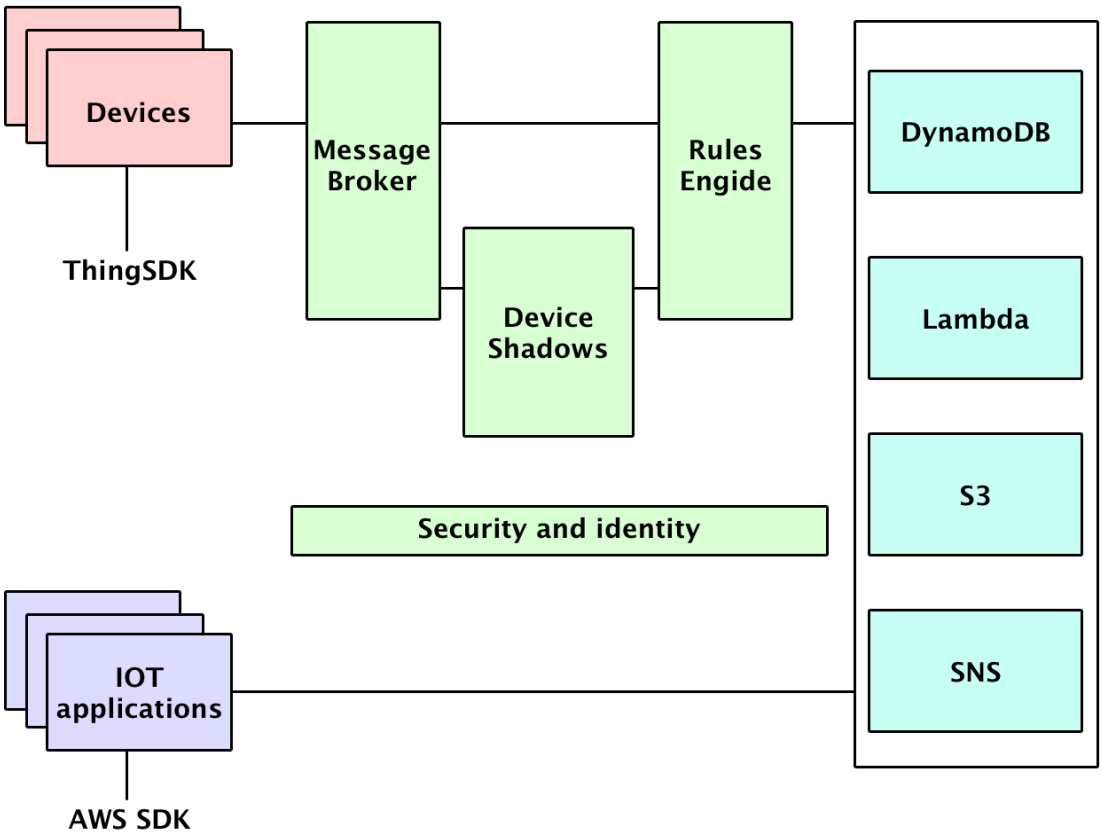

Figura 2.6: I dispositivi connessi al cloud possono interagire con altri servizi AWS

Cognito

Amazon Cognito è una piattaforma di identità per app web e per applicazioni mobili.
È un server di autenticazione e servizio di autorizzazione per token.
Con Amazon Cognito, puoi autenticare e autorizzare gli utenti dalla directory utente integrata.

Configurando un pool di identità si possono autorizzare utenti autenticati o anonimi ad accedere alle tue risorse.
I pool di identità utilizzano il controllo degli accessi basato sui ruoli e sugli attributi per gestire l'autorizzazione degli utenti ad accedere alle risorse.
L'app assegna la sessione di credenziali all'utente e fornisce l'accesso autorizzato a servizi come AWS Amazon S3 e Amazon DynamoDB.

Simple Notification Service

Amazon Simple Notification Service è un servizio che manda messaggi dagli editori agli abbonati. Gli editori comunicano in modo asincrono con gli abbonati creando e inviando messaggi a un argomento, che rappresenta un canale di comunicazione.
I clienti possono abbonarsi all'argomento e ricevere messaggi pubblicati.

Viene spesso utilizzato per mandare, in modo automatico e interamente gestito dalla piattaforma, notifiche push ad applicazioni mobili.
È utile se la notifica deve essere mandata quando cambia un valore in database, in modo tale da poter gestire tutto in cloud.

Lambda

Il servizio Lambda è sicuramente quello che ho utilizzato maggiormente nella piattaforma AWS.
Lambda esegue il codice su un'infrastruttura di elaborazione ad alta disponibilità e gestisce tutta l'amministrazione delle risorse di elaborazione, compresa la manutenzione del server e del sistema operativo, il provisioning e la scalabilità automatica della capacità

e la registrazione.
Con Lambda, tutto quello che occorre fare è fornire il proprio codice in uno dei runtime di linguaggio supportati da Lambda.

Questo servizio può essere utilizzato per scopi differenti tra loro.
Può essere usato con Applicazioni Web o Mobili per scalare automaticamente verso l'alto e verso il basso ed essere eseguite in una configurazione ad alta disponibilità su più data center. Oppure può essere usato per creare un backend serverless utiizzando appunto le funzioni lambda per gestire richieste web, di applicazione mobili o di Smart Object.

Il codice viene organizzato in funzioni Lambda e il servizio esegue la funzione solo quando necessario. Scegliamo noi quando chiamarla in base ai collegamenti che possono essere fatti con tutti gli altri servizi.
Un esempio può essere il seguente: una funzione lambda che viene triggerata quando cambia il valore in database e manda una notifica con il servizio SNS ai dispositivi interessati. Potendo scegliere sia trigger che destinazione per ogni funzione, è possibile creare automazioni interamente sul cloud.
Il linguaggio di programmazione che si vuole utilizzare può essere scelto tra varie opzioni tra cui Java, Nodejs, Pyhton e Ruby.

Amplify

Amplify è il servizio Amazon che viene utilizzato per collegare la propria applicazione alla console AWS e a tutti i suoi servizi in modo semplice e veloce.
Collegata una applicazione Mobile o Web ad Amplify è possibile implementare la UI relativa al Login e alla Registrazione senza doverla creare da zero, ma utilizzando pagine costruite da AWS stesso.
In automatico viene creato il sistema per gestire l'autenticazione di utenti. Dalla console Online di AWS Amplify è possibile collegare all'applicazione:

- un Pool di Utenti per salvare gli account creati
- un bucket S3 per salvare dati di grandi dimensioni

- un database DynamoDB
- delle funzioni Lambda
- delle API GraphQL o Rest

AppSync

AWS AppSync consente di connettere le proprie applicazioni a dati ed eventi con GraphQL e Pub/Sub sicuri, serverless e ad alte prestazioni.
GraphQL è sia un linguaggio di query che un runtime per l'esecuzione di tali query. GraphQL consente ai clienti di richiedere esattamente i dati di cui hanno bisogno, fornendo un'alternativa più flessibile ed efficiente rispetto REST.

A differenza di REST, che si basa su endpoint predefiniti, GraphQL utilizza un unico endpoint in cui i clienti possono specificare i propri requisiti di dati sotto forma di query e mutazioni.

2.3 Applicazione Android

Il terzo tassello di questo progetto è l'applicazione Android. Essa è il punto di controllo di tutto il sistema.
Considerando l'infrastruttura del progetto, si potevano scegliere anche soluzioni alternative per la gestione remota dell'Allarme della casa.
Per esempio una applicazione Web di più facile realizzazione.

Il motivo per cui è stata scelta una applicazione Android è la necessità di utilizzare delle funzionalità esclusive dei software nativi:

- monitoraggio continuo della posizione in background
- utilizzo del sensore nfc

Il monitoraggio continuo della posizione serve a rendere l'intero sistema IoT un sistema automatico in grado di attivare e disattivare l'allarme senza fare nulla.

L'utilizzo del sensore NFC del dispositivo serve invece a registrare delle tessere che saranno poi utilizzabili dal dispositivo IoT di Allarme.
L'applicazione è stata sviluppata in Kotlin, utilizzando il servizio AWS Ampify per poterla connettere facilmente con il servizio cloud e quindi con i dispositivi Smart.
Per rendere lo sviluppo più semplice è stato utilizzato Jetpack Compose.
Jetpack Compose è il toolkit raccomandato per la creazione di Applicazioni Android moderne, per la creazione di UI nativa.
Semplifica ed accelera lo sviluppo della UI di una applicazione Android. Aiuta a costruire il prodotto con meno codice, strumenti potenti e API per Kotlin intuitive.

2.4 Threat Model

Un Threat Model è una rappresentazione strutturata di tutte le informazioni che influiscono sulla sicurezza di un'applicazione.
In sostanza, si tratta di una visione dell'app e del suo ambiente attraverso la lente della sicurezza. Ma prima di capire quali possono essere le minaccie, analizziamo quali sono gli obiettivi di sicurezza del progetto:

- Assicurare che i messaggi scambiati tra il dispositivo e il cloud siano autentici e non siano stati alterati
- Assicurare che il sistema sia sempre operativo e non vulnerabile a Denial of Service (DoS) o sabotaggi
- Solo utenti autorizzati possono modificare lo stato dell'allarme

2.4.1 Potenziali minacce del sistema

Le vulnerabilità di un progetto di IoT sono date dall'interconnettività delle rete del sistema che porta a problemi di accessibilità da parte di elementi anonimi e inaffidabili.[10]
Sicurezza e Privacy sono quindi i problemi principali da affrontare.
Per capire bene quali sono i rischi di cybersecurity che un sistema informatico deve affrontare, si possono usare dei framework.

Il più utilizzato è STRIDE. Esso è stato sviluppato da Microsoft ed è uno dei framework di modellazione delle minacce più popolari per l'ingegneria del software.[11]
Usa diagrammi di data-flow come mappature per identificare le minacce.

STRIDE è un acronimo per.

- Spoofing (fingersi un'altra persona)
- Tampering (modificare qualcosa sul disco o database)
- Repudiation (dire di non aver fatto qualcosa o di non esserne responsabile)
- Information Disclosure (qualcuno che ottiene informazioni che non sarebbe autorizzato ad ottenere)
- Denial of Service (sovraccaricare le risorse necessarie per provvedere un servizio)
- Elevation of Privilege (permettere a qualcuno di fare qualcosa che non potrebbe fare)

Analizzando ognuno di questi punti, ed applicandoli al progetto, sono riuscito ad ottenere quelli che sono i possibili attacchi che questo sistema di sicurezza potrebbe subire.
Essi possono essere:

- Ricevere messaggi MQTT falsi al dispositivo IoT
- Modifica di messaggi inviati da AWS al dispositivo IoT
- Accedere al database su AWS e modificare lo stato di allarme
- Attacchi di DoS per interrompere la connessione
- Accesso fisico al dispositivo per causare malfunzionamenti

2.4.2 Azioni che possono essere intraprese per mitigare ogni minaccia

L'architettura che utilizza AWS IoT Core per la comunicazione ha alcune protezioni integrate. Ecco un'analisi di ogni tipo di attacco e quanto il mio progetto può difendersi.

Ricevere messaggi MQTT falsi al dispositivo IoT

Questo attacco è difendibile con AWS Iot Core.
AWS IoT utilizza certificati X.509 e la crittografia TLS per proteggere le comunicazioni tra il dispositivo e AWS.
Questo garantisce che solo i dispositivi e gli endpoint autorizzati possano pubblicare e sottoscrivere messaggi MQTT.
Inoltre, AWS IoT ha il controllo delle politiche che permettono di limitare chi può inviare o ricevere messaggi.

I certificati digitali garantiscono che solo i dispositivi autentici possano comunicare con il server AWS IoT.
Le policy di AWS IoT limitano chi può pubblicare e ricevere messaggi dai tuoi topic MQTT, prevenendo la pubblicazione di messaggi non autorizzati.

Modifica di messaggi inviati da AWS al dispositivo IoT

Anche questo attacco viene difeso direttamente dall'utilizzo di AWS IoT, che integra dei metodi per contrastarlo.
La crittografia TLS garantisce che i messaggi inviati da AWS non possano essere modificati da un attaccante senza che venga rilevato.
Anche se un attaccante tentasse di modificare un messaggio in transito, il dispositivo rileverebbe la mancata validità del certificato o del messaggio firmato e potrebbe scartarlo.

Accedere al database su AWS e modificare lo stato di allarme

Questo attacco dipende dal livello di sicurezza che hai implementato su AWS.
Se l'accesso ai database (ad esempio DynamoDB o RDS) è gestito correttamente tramite IAM (Identity and Access Management) con permessi strettamente limitati, allora sarebbe difficile per un attaccante accedere direttamente ai dati e modificarli.

Io ho utilizzato in modo accurato la suite di Amazon, configurando in modo attento l'accesso ai dati, con l'utilizzo appunto di IAM per l'identità digitale sulla piattaforma. Difficilmente un possibile hacker riuscirebbe ad entrare su AWS e modificare i valori sul database.

Infatti AWS utilizza le politiche IAM per controllare l'accesso ai database, quindi puoi limitare chi può leggere o modificare i dati.

Se le politiche di accesso non sono configurate correttamente, o se ci sono falle nei sistemi di accesso (come credenziali compromesse o politiche IAM troppo permissive), un attaccante potrebbe accedere al database e modificare i dati.

Attacchi di DoS per interrompere la connessione

Un attacco di Denial of Service (DoS) punta a sovraccaricare la rete o i servizi, rendendoli inaccessibili.
Questo attacco è quasi impossibile da prevenire totalmente, ma si può cercare di rendere molto difficile.
AWS IoT Core offre una certa resilienza grazie alla sua scalabilità e robustezza, ma un attacco DoS su larga scala potrebbe comunque riuscire a interrompere il servizio.

AWS è progettato per essere resiliente agli attacchi DoS e AWS Shield fornisce una protezione contro questi attacchi.
Inoltre la scalabilità automatica dei servizi AWS può ridurre l'impatto di un attacco di DoS.

Se un attacco DoS riuscisse a saturare la connessione internet del dispositivo o bloccare il server AWS, il dispositivo non sarebbe più in grado di comunicare con AWS IoT Core.
Potrebbero verificarsi interruzioni del servizio se l'attacco è su larga scala e ben coordinato.

Un possibile miglioramento sarebbe implementare logiche locali di fallback sul dispositivo hardware in modo tale che, in caso di disconnessione da AWS, il dispositivo possa continuare a funzionare offline.
Questo però non servirebbe a molto, non potendo usare l'allarme in tempo reale. Potrebbe solo permettere di visualizzare le foto di sicurezza una volta terminato l'attacco.

Accesso fisico al dispositivo per causare malfunzionamenti

Se un attaccante ha accesso fisico al dispositivo, è difficile difendersi correttamente. L'attaccante potrebbe manomettere l'hardware, riprogrammare il dispositivo o distruggere i suoi sensori.

L'unico metodo valido per evitarlo è implementare delle misure di sicurezza fisica. Come il posizionare la fotocamera di sorveglianza il luoghi poco visibili o particolarmente sicuri e protetti.
Si può monitorare lo stato del dispositivo e inviare avvisi se si rilevano anomalie, per esempio, una perdita di connessione o riavvii non previsti.

Capitolo 3

Implementazione

Dopo aver fatto un'introduzione sullo stato attuale del mercato e un'un'analisi delle tecnologie utilizzate in questo progetto, viene ora mostrato l'utilizzo che ne è stato fatto. Come prima cosa viene esposto il flusso della piattaforma. Quindi il comportamento del sistema dal punto di vista dell'utente che lo utilizza per la prima volta.
In questa parte verrà portata particolare attenzione all'interazione tra i vari componenti del sistema, mostrando il comportamento del progetto.

Verranno poi analizzati tutti i dettagli implementativi delle varie sezioni:

- Sketch ESP32
- Implementazione AWS e funzioni Lambda
- Applicazione Android

3.1 Flusso della piattaforma

Le tre parti del progetto analizzate precedentemente devono essere collegate tra di loro. Il dispositivo ESP32, i servizi di AWS e l'applicazione Mobile, devono funzionare come un unico sistema.

Il collegamento tra le parti avviene nella configurazione iniziale, appena installata l'Applicazione sul dispositivo.

3.1.1 Configurazione iniziale

I primi passi per iniziare a utilizzare HomeGuard sono:

- Accendere il dispositivo IoT
- Installare l'Applicazione sul proprio dispositivo

Inoltre è necessario essere nella propria abitazione al momento del Set Up iniziale per i seguenti motivi:

- Accesso al Wi-Fi domestico
- Alimentazione del dispositivo IoT via cavo
- Accesso alla posizione dell'abitazione

Appena aperta l'applicazione si viene reindirizzati alla procedura di Set Up iniziale, invece che alla Homepage, perché verrà rilevato che non è ancora mai stato fatto un Login sul dispositivo attuale.
Se invece è stato fatto si accede alla Homepage.
Queste schermate sono realizzate in automatico da AWS Amplify, che gestisce da solo tutta l'autenticazione dell'applicazione Mobile.

La procedura di Set Up iniziale comprende:

- creazione di un nuovo account
- creazione del proprio utente della casa
- associazione dell'allarme ESP32

Le schermate per il Log In e per la Registrazione del nuovo Account sono le seguenti:

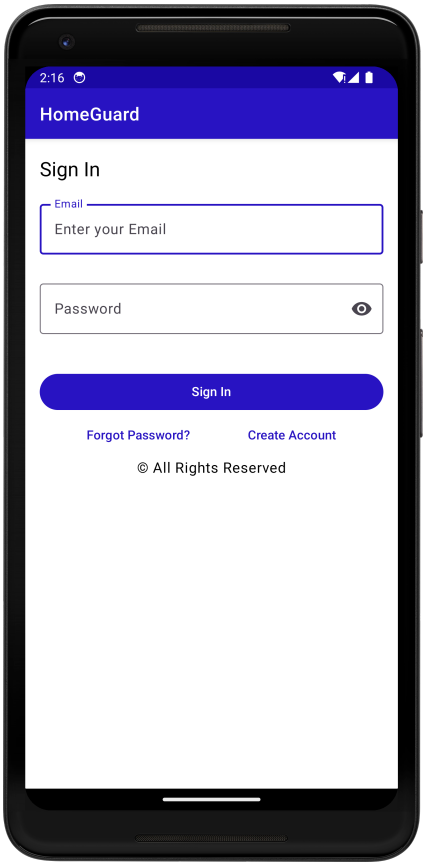

Figura 3.1: Schermata di Log In AWS

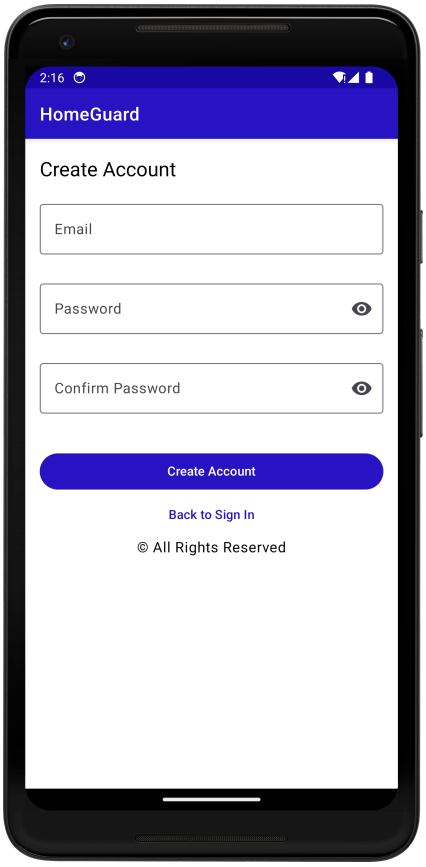

Figura 3.2: Schermata di registrazione

Quando un utente viene creato le sue informazioni vengono salvate in una pool di utenti su AWS.
Poi, una volta che un utente viene aggiunto a questa pool, viene triggerata una funzione Lambda lo crea anche su DynamoDB. Questa funzione verrà analizzata meglio nella sezione dedicata ad Amazon Web Services.
Una volta creato l'Account, Amplify riconoscerà che nel dispositivo è già presente una autenticazione e reindirizzerà verso l'Homepage.

Questo avverrà anche nelle successive aperture dell'applicazione. Non sarà necessario rifare il Log In ogni volta.
La creazione dell'Account corrisponde alla creazione della "casa". Quindi un Account corrisponderà ad una singola abitazione.

In una casa possono esserci però più persone e ognuna di esse può creare un utente personale. Questo utente servirà per la gestione automatica dell'allarme come verrà spiegato in seguito in maniera più approfondita.

Per questo motivo il secondo step è proprio creare il primo utente della casa. Nella creazione del primo utente andranno inseriti:

- nome
- immagine profilo da fotocamera o galleria (facoltativa)
- pin dell'allarme

Il pin dell'allarme è un codice stampato fisicamente sul dispositivo e serve a rendere più sicuro l'accoppiamento di esso con il profilo della casa appena creato.

In questo modo quando la registrazione sarà completata, potrai essere sicuro che il tuo allarme possa essere controllato solo dal tuo profilo.
Sarà l'unica volta che verrà chiesto il suo inserimento.

Quando si conferma la creazione del primo utente della casa vengono eseguite delle operazioni con il database DynamoDB su AWS.
Nome e immagine profilo vengono usate per creare un nuovo elemento "person" che sarà collegato con un id all'account della casa creato in precedenza.
Il codice del dispositivo IoT verrà invece aggiunto proprio all'elemento che identifica la casa nel database. In questo modo ci potrà essere solo un Account abitazione registrato con quel dispositivo fisico.

Figura 3.3: Registrazione del primo utente

Una volta inserite le informazioni dell'utente e il pin fisico si potrà procedere alla associazione dell'allarme.
In questo momento il dispositivo fisico di allarme non ha ancora subito cambiamenti. Non si può quindi connettere ad internet e di conseguenza non potrà comunicare con AWS. Noi dobbiamo dare le credenziali del nostro Wi-Fi domestico al dispositivo, in modo da consentirgli queste operazioni.

L'allarme a questo punto sarà già in modalità Access Point, non avendo nessuna credenziale salvata per il Wi-Fi. Questo significa che noi potremmo collegarci alla sua rete. Sempre all'interno dell'applicazione vengono scannerizzate le rete wifi vicine. Dovrai selezionare la rete Access Point del dispositivo di Allarme, chiamata "SecurityAlarm", e connetterti.

Dopo esserti connesso potrai accedere ad un indirizzo IP predefinito nel quale inserire le credenziali del tuo Wi-Fi domestico che verranno poi inviate all'allarme.

Figura 3.4: Scannerizzazione reti Wi-Fi

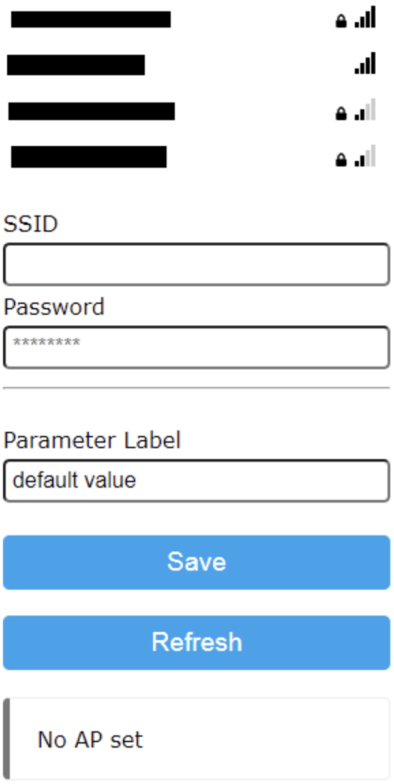

Figura 3.5: Schermata indirizzo IP

Fatto questo, la configurazione sarà completa: il dispositivo potrà connettersi ad Internet e quindi ad AWS.
Comunicherà con Amazon sul suo canale privato, quello identificato dal Pin che è stato in precedenza inserito nelle informazioni dell'Account della abitazione.
In questo modo entrambi sapranno quale è il canale di comunicazione corretto e potranno comunicare tra di loro attraverso il servizio Cloud.
Durante la procedura di registrazione iniziale, viene anche impostata in automatico una

nuova zona di Geofencing, che servirà a identificare la posizione della casa e a cambiare lo stato dell'allarme in base alla posizione degli utenti.

3.1.2 Comunicazione ESP32 - Applicazione Android

La comunicazione è gestita da Amazon Web Services. Le due parti non scambiano informazioni direttamente tra di loro, ma comunicano solo con AWS, che si occupa poi di inoltrare i messaggi.
I dispositivi IoT comunicano usando MQTT, mentre l'applicazione usa delle API GraphQL.

MQTT

MQTT è un protocollo di messaggistica utilizzato come standard nell'Internet of Things. È sviluppato per essere un protocollo di messaggistica publish/subscribe estremamente leggero.
Questo è ideale per connettere dispositivi remoti con risorse e larghezza di banda limitate. MQTT oggi è utilizzato in una larga varietà di industrie, come per esempio quella automotiva o di telecomuncazioni. [12]

I motivi che rendono MQTT un protocollo preferibile alle richieste HTTP quando si trattano dispositivi poco potenti e che devono costare poco sono i seguenti:

- È leggero ed efficiente
- È una comunicazione bi-direzionale (cloud-device)
- È molto scalabile
- Ha diversi livelli di qualità selezionabili per la consegna dei messaggi
- È ottimizzato per ambienti con connessioni instabili
- Rende facile criptare messaggi con TLS e autenticare client con OAuth

L'architettura è Publish/Subscribe. Questo significa che c'è un Topic, che sarebbe un canale, al quale ci si può iscrivere per ricevere i messaggi che vengono pubblicati su di esso. Chi si iscrive è un Subscriber.
Chi pubblica su un Topic è invece chiamato Publisher.
C'è anche un Broker che è colui che riceve i messaggi dai publisher e li distribuisce a tutti i subscriber che si sono iscritti al topic corrispondente.

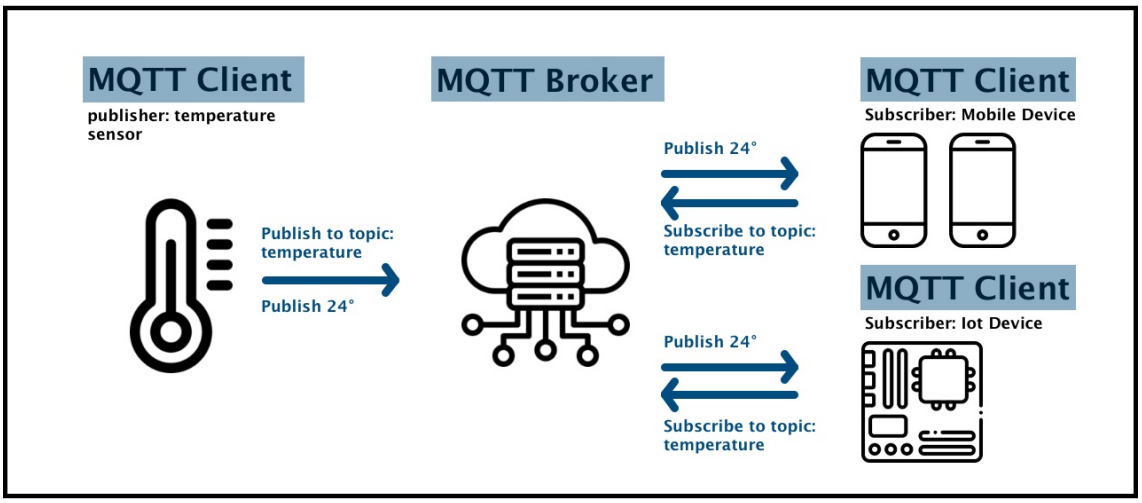

Figura 3.6: Architettura MQTT

GraphQL

GraphQL è un linguaggio di query per API e un runtime per eseguire quelle query con i dati esistenti. GraphQL fornisce una descrizione completa e comprensibile dei dati nella tua API, dà ai client il potere di richiedere esattamente ciò di cui hanno bisogno e nulla di più, rende più facile l'evoluzione delle API nel tempo e abilita strumenti potenti per gli sviluppatori. [13]

Nelle applicazioni moderne GraphQL è una delle opzioni preferite insieme alle API REST. La scelta pende sempre tra queste due opzioni.
I motivi per i quali si può preferire GraqlQL sono i seguenti:

- si vuole ridurre al minimo il numero di richieste e risposte

- ci sono più origini dati e si vogliono combinare in un unico endpoint
- le richieste variano molto e quindi ci si aspettano risposte molte diverse

Invece REST è meglio utilizzarlo per applicazioni più piccole e lineari.

3.2 Sketch Esp32

Nella sezione precedente abbiamo visto quali sono tutti i collegamenti dei due dispositivi fisici che ho creato.
In questa sezione di implementazione invece analizzeremo il loro codice.
Il programma che viene eseguito su un dispositivo IoT come Arduino o ESP32 viene chiamato "Sketch" e si compone di due parti principali: la funzione di Setup e la funzione di Loop.

La funzione di Setup viene chiamata ogni volta che il dispositivo viene acceso, per inizializzarlo. Oppure ogni volta che viene premuto il pulsante di reset, che riavvia il dispositivo senza dovergli togliere l'alimentazione, molto utile per il debugging.
La funzione di Loop invece è quella che viene chiamata dopo il Setup, e racchiude tutto il funzionamento del dispositivo. La parte di Sketch presente nella funzione di Loop è quella che viene chiamata ogni intervallo di tempo.

3.2.1 Allarme, Sensore NFC e Sensore porta

Questo è il primo dispositivo, e anche il più complesso dei due.
Prima della funzione di setup vengono:

- incluse tutte le librerie che si servono
- definiti i nomi dei topic di pubblicazione e iscrizione MQTT
- definito il nome del file json dove salveremo le credenziali Wi-Fi e l'homeId
- definiti il numero delle porte nelle quali abbiamo collegato i sensori
- inizializzato schermo, sensore nfc, wifimanager e client MQTT

Setup

In ogni funzione di setup di ogni Sketch, come prima cosa, viene inizializzato il Serial Monitor. Il Serial Monitor è il metodo di logging dei dispositivi IoT, fondamentale per testare il codice in produzione.
Questa funzione, essendo la prima parte di codice eseguita una volta avviato il dispositivo, serve a compiere quelle operazioni che vanno eseguite solo una volta all'inizio del programma.

Quindi in questo caso si chiamano i metodi Init o Begin di schermo, sensore nfc, wifimanager e del file di configurazione. Se tutti questi metodi vengono eseguiti con successo, significa che tutte le librerie stanno funzionando correttamente, e il loop può iniziare in sicurezza.
Se le funzioni ritornano esito positivo significa che i sensori sono stati rilevati e collegati correttamente.

In questo caso, sempre nella funzione di Setup, dopo aver chiamato la funzione per caricare le credenziali Wi-Fi dal file Json e la funzione WifiManager, viene chiamata "connectAWS" che serve a connettere il dispositivo al client MQTT di AWS.
Analizziamo meglio queste tre funzioni.

La funzione "loadConfigFile" serve a recuperare le informazioni salvate nel file "config.json". Esse sono: nome della rete Wi-Fi(SSID), password e id dell'Account che gli viene associato.
Con la libreria SPIFFS si può operare sui file salvati nel dispositivo IoT.
Come prima cosa, nella funzione, controllo se esiste il file che sto cercando. Se viene trovato, lo apro e accedo alle sue informazioni.

Una volta ottenute le informazioni, le salvo sulle variabili globali inizializzate prima della funzione di Setup. Queste variabili potranno essere visualizzate in ogni parte del codice.
Dopo aver chiamato questa funzione, se il dispositivo è già stato registrato ad un ac-

count, le nostre variabili conterranno le credenziali per accedere al Wi-Fi. Altrimenti, sarà necessario passare alla procedura di configurazione iniziale, ma a questo ci penserà la seconda funzione chiamata.

La funzione "WiFiManagerSetup" serve a gestire la connessione al Wi-Fi. Se abbiamo recuperato le credenziali, prova a connettersi usando quelle. Se non abbiamo le credenziali salvate, entra in modalità Access Point (AP).
Quando entra in modalità AP possiamo connetterci alla sua rete e inserire le credenziali che richiede. Questo lo facciamo direttamente dall'applicazione come descritto nella sezione precedente.

L'ultima funzione di Setup è "connectAWS". Questa parte di codice fa uso di un file esterno chiamato secrets.h che contiene tutte le chiavi di associazione del dispositivo con AWS. Queste chiavi servono per una questione di sicurezza. Infatti non tutti i dispositivi possono comunicare sul canale MQTT utilizzato dal device di allarme e AWS. Possono farlo solo i dispositivi registrati sulla piattaforma, quindi in questo caso solo l'allarme e la fotocamera.

Durante il tentativo di connessione con AWS il dispositivo invia le chiavi che ha registrato nel suo file di sicurezza e il suo nome identificativo nel Cloud. Amazon controlla se al suo ID corrispondono le chiavi, se è così allora viene autenticato e può aprire la connessione con esso.

Questo sistema rende AWS il provider adatto per questo genere di operazioni e connessioni che richiedono un livello di sicurezza elevato. Usare un semplice sistema di comunicazione MQTT senza questi meccanismi di protezione avrebbe creato un grosso problema di cybersecurity.
La procedura dettagliata di registrazione del dispositivo nel Cloud verrà descritta nella successiva sezione dedicata ad AWS.

```
1  void connectAWS() {
2    while (WiFi.status() != WL_CONNECTED) {
3      WiFi.mode(WIFI_STA);
4      delay(500);
5      WiFi.begin(ssid, password);
6    }
7
8    net.setCACert(AWS_CERT_CA);
9    net.setCertificate(AWS_CERT_CRT);
10   net.setPrivateKey(AWS_CERT_PRIVATE);
11
12   client.setServer(AWS_IOT_ENDPOINT, 8883);
13   client.setCallback(messageHandler);
14
15   if (client.connected()) {
16     publishDeviceOnlineMsg();
17   } else {
18     return;
19   }
20
21   client.subscribe(AWS_IOT_SUBSCRIBE_TOPIC);
22 }
```

Come prima cosa controlliamo che la connessione al Wi-Fi non sia stata persa, e in caso la ripristiniamo. Poi impostiamo tutti i certificati e le chiavi di cui parlavo in precedenza, e ci connettiamo all'endpoint di AWS.
Impostiamo la funzione di callback che verrà chiamata quando riceveremo un messaggio in questo canale, e ci connettiamo.

Se la connessione è avvenuta con successo, mandiamo subito un messaggio sul topic MQTT scelto per comunicare che siamo online.
Questo messaggio serve ad aggiornarci sullo stato dell'allarme, infatti appena AWS riceverà il messaggio di Device Online, risponderà con lo stato dell'allarme del nostro Account. Vedremo dopo come AWS può reagire e rispondere a questi eventi.

Questo è come viene inviato il primo messaggio:

```
1 void publishDeviceOnlineMsg() {
2   StaticJsonDocument<200> doc;
3   doc["id"] = THINGID;
4   doc["status"] = "Device Online";
5
6   char jsonBuffer[512];
7   serializeJson(doc, jsonBuffer);
8   client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
9 }
```

Creiamo un Json con il nostro id (il codice presente fisicamente sul dispositivo) per farci associare all'Account corretto, e pubblichiamo il messaggio sul Topic predefinito. Con questo viene conclusa la funzione di setup del dispositivo di Allarme. Viene poi eseguita la funzione di Loop.

Loop

La funzione di Loop gestisce quattro parti:

- sensore magnetico della porta, buzzer sonoro e invio record
- schermo lcd
- lettore nfc
- recezione messaggi MQTT

La prima parte controlla quando il sensore magnetico della porta cambia stato da HIGH a LOW o viceversa, e reagisce di conseguenza.
Quando i sensori magnetici si allontanano, significa che la porta si sta aprendo, e quindi il Buzzer inizia a suonare ad intermittenza. Questo solo se l'allarme è attivo ovviamente. Il Buzzer non ha una grande potenza, quindi non genererà un suono che si possa sentire anche all'esterno dell'abitazione. Viene usato questo prodotto low-cost solamente per scopi di ricerca, ma può essere sostituito da un altoparlante più potente.

Oltre ad attivare il Buzzer, il dispositivo pubblica immediatamente il Record di allarme scattato. Chiama quindi la funzione publishRecord.
Questa funzione è simile alla precedente utilizzata per indicare la connessione ad AWS del dispositivo.
Però a differenza della precedente viene inviato l'homeId (identificativo dell'Account in database) che abbiamo salvato prima. In questo modo AWS, una volta ricevuto il messaggio, potrà collegare il record al profilo corretto.

La seconda parte riguarda lo schermo LCD. Esso deve mostrare ai membri della casa lo stato dell'allarme in tempo reale.
Il loop controlla ogni giro se lo stato dell'allarme è cambiato. Se è così, modifica la scritta dello schermo.

La terza parte riguarda il lettore NFC.
Il lettore viene messo in fase di lettura per tutto il loop, in attesa di tessere. Appena rileva una tessera, si disattiva per 5 secondi e manda un messaggio ad AWS.
Il motivo per cui viene disattivata per 5 secondi è per non inviare più di un messaggio al Cloud.
In questo caso non ci interessa il contenuto della tessera ma solo lo UID, per verificare se corrisponde ad uno degli UID delle tessere registrate in database per il profilo in questione. Il messaggio ad AWS quindi conterrà lo UID della carta.

Ma dopo aver inviato il messaggio, il ruolo di questa sezione non è ancora compiuto. Infatti lo stato dell'allarme non viene ancora cambiato.
Però quando AWS riceve il messaggio e verifica che lo UID sia presente database, risponde al device IoT con il nuovo stato dell'allarme modificato. Così lo schermo può cambiare scritta e l'utente sa che la tessera è stata riconosciuta.

La parte finale è la più importante e riguarda la recezione dei messaggi. Di questo si occupa la funzione messageHandler.
Ogni messaggio che viene inviato da AWS al dispositivo contiene un valore "alarm".

Questo valore viene estratto dal messaggio e salvato in locale per cambiare lo stato dello schermo.
Se il messaggio contiene un valore "homeID" allora ci salviamo anche quello. Questo dovrebbe avvenire solo con la recezione del primo messaggio dopo la registrazione, per ricevere l'Id dell'account a cui ci siamo associati.

```
1  void messageHandler(char* topic, byte* payload, unsigned int length) {
2    StaticJsonDocument<200> doc;
3    DeserializationError error = deserializeJson(doc, payload, length);
4    if (error) {
5      return;
6    }
7
8    bool newAlarmState = doc["alarm"].as<bool>();
9
10   if (doc.containsKey("homeID")) {
11     const char* newHomeId = doc["homeID"];
12     strncpy(homeId, newHomeId, sizeof(homeId));
13     homeId[sizeof(homeId) - 1] = '\0';
14   }
15
16   alarmOn = newAlarmState;
17
18   //aggiornamento schermo
19
20   saveConfigFile();
21 }
```

3.2.2 Fotocamera di sicurezza

Questo dispositivo è il più semplice dei due, e serve solamente a scattare le foto di sicurezza. Non ha dispositivi esterni collegati ed è composto da un unico ESP32 Cam, un ESP32 fornito di fotocamera integrata.

La struttura del codice è molto simile a quella del dispositivo precedente, dovendo anch'esso comunicare con AWS allo stesso modo.
In questo caso però i messaggi inviati al cloud conterranno le foto di sicurezza. È interessante analizzare la funzione che pubblica il messaggio con la foto: "publishFullPhoto".

```
1  void publishFullPhoto(const uint8_t* photoData, size_t photoSize) {
2      String encodedImage = base64::encode(photoData, photoSize);
3      size_t jsonBufferSize = JSON_OBJECT_SIZE(2) + encodedImage.length()
           + 200;
4      char* jsonBuffer = (char*)malloc(jsonBufferSize);
5
6      StaticJsonDocument<2048> doc;
7      doc["id"] = homeId;
8      doc["image"] = encodedImage;
9
10     client.publish(AWS_IOT_PUBLISH_TOPIC, jsonBuffer);
11
12     free(jsonBuffer);
13 }
```

La funzione riceve due parametri in input: photoData e photoSize.
Il primo contiene i dati grezzi dell'immagine, il secondo la dimensione dell'immagine in numero di byte.
Essa viene poi codificata in Base64 utilizzando la funzione base64::encode. Questo step è necessario perché i dati binari devono essere convertiti in testo per essere inviati come parte di un messaggio JSON.

Poi viene allocato dinamicamente un buffer (jsonBuffer) con la dimensione calcolata e viene costruito un documento JSON.
Il documento conterrà l'homeId e la stringa dell'immagine in base64.
Una volta creato il documento, il messaggio viene pubblicato, subito dopo viene deallocata la memoria utilizzata per l'immagine, per evitare memory leaks.

3.3 Applicazione Android

L'applicazione Android è l'altro elemento del progetto, il centro di gestione dell'ESP32. L'applicazione comunica con i dispositivi Hardware attraverso la struttura Backend costruita sulla piattaforma Serverless di Amazon, che sarà trattata nella sezione successiva.
Ciò che permette a questa applicazione di essere innovativa sul mercato è la possibilità di gestire gli accessi della casa in modo automatico così da garantire una gestione dell'allarme molto più semplice e intuitiva.

Non sarà necessario cambiare costantemente lo stato dell'allarme, perchè esso verrà modificato in modo automatico. L'applicazione avrà sempre accesso alla posizione dell'utente, e in questo modo saprà sempre dove esso si trova.
Sapendo quando sarà in casa o quando sarà fuori, l'app potrà gestire l'allarme di conseguenza, in modo automatico, e grazie alla possibilità di creare più utenti della casa, l'allarme si attiverà solo quando tutti gli utenti saranno fuori.
Un'altra caratteristica che rende il sistema di sicurezza più comodo da gestire è la possibilità di usare delle tessere NFC per l'autenticazione.

L'utente può registrare nella propria abitazione un numero infinito di tessere NFC, che una volta inserite, potranno essere utilizzate per autenticarsi entrando in casa.
Ma oltre a questo, con l'applicazione è possibile:

- Attivare/disattivare l'allarme manualmente
- Visualizzare chi tra gli utenti è in casa
- Vedere i record dei precedenti allarmi scattati (con orario e fotografie)
- Selezionare il proprio utente della casa
- Controllare quante tessere sono state registrate
- Impostare un pin di sicurezza

3.3.1 Il collegamento con AWS

La prima funzionalità aggiunta all'applicazione è l'autenticazione: login e registrazione. Queste sono state implementate attraverso AWS Amplify.
In automatico AWS, se l'utente non è autenticato, mostra una schermata dove si può fare Log In, oppure dove si può procedere alla registrazione. Queste due sono schermate che vengono create dalla piattaforma cloud.

Oltre a svolgere questo ruolo le funzioni di AWS vengono chiamate ogni volta che dobbiamo fare query o mutation. In pratica, attraverso Amplify, viene gestita tutta la connessione dell'applicazione con il suo backend serverless sul cloud.

Tutte le schermate necessitano di una gestione del caricamento per attendere che i dati vengano presi dal cloud, prima di mostrarli.
Questo serve ad evitare che ci siano "Content Shift". Un Content Shift è quando il contenuto viene caricato dopo che l'utente vede per la prima volta la schermata, e che questo contenuto sposti verso il basso tutto il resto della pagina.

Quello che può causare è, oltre ad un problema estetico, un click errato di un pulsante, per esempio.
Inoltre può anche evitare che l'utente veda il dato non aggiornato, solitamente quello impostato di default, invece di quello ottenuto dal database.
L'unico dato che viene salvato in locale è l'utente della casa che viene selezionato.

3.3.2 Schermata Home

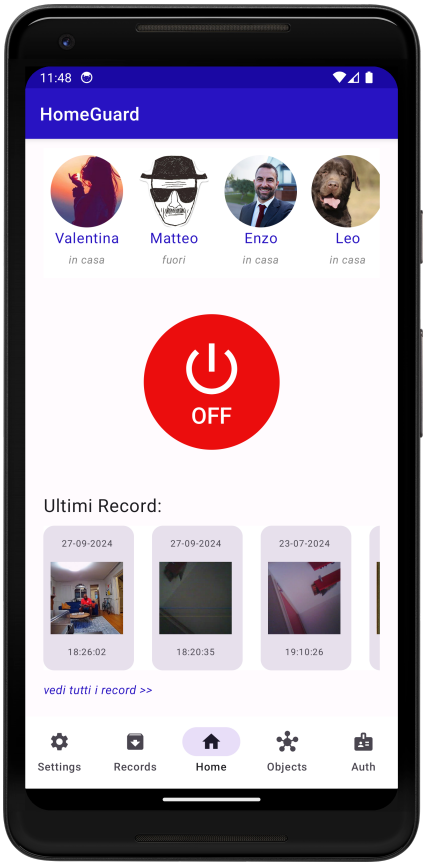

Figura 3.7: Schermata Home

Figura 3.8: Creazione nuovo utente

La schermata Home presenta 3 sezioni:

- Lista delle persone della casa
- Pulsante dell'allarme
- Lista degli ultimi record di allarme

La lista delle persone della casa è una Lazy Row che prende dal database tutte le persone associate all'abitazione e le mostra con il loro nome e la foto profilo, se presente. Viene mostrato poi lo stato della persona: In casa o Fuori. Inoltre alla fine della lista c'è un pulsante +, il quale apre un form nel quale si può creare un altro utente come è

già stato fatto nel Set Up iniziale.
È poi presente un pulsante grande, quello che indica lo stato dell'allarme. Ha due stati (ON - Verde e OFF - Rosso).
Dopo il click, mentre lo stato del pulsante cambia in Database, c'è una rotella di caricamento per indicare che il pulsante è stato cliccato con successo.

Infine c'è un'anteprima della lista degli ultimi record. Quindi tutte le ultime volte che l'allarme è scattato (può succedere con il sensore magnetico della porta o con sensori di movimento).
Quando l'allarme parte viene informato anche un altro disposivo: la fotocamera "ESP32 Cam", che scatta le foto del record e le inserisce in database.
Quindi nella LazyRow dove sono mostrati i record si possono visualizzare: data, orario e la prima fotografia in evidenza.

Per visualizzare tutte le foto di sicurezza basta cliccare sul Record in questione e aprire la sua pagina singola, dove saranno presenti tutte le foto.
Ovviamente, dato che questa pagina carica molti dati dal database, viene gestita con una rotella di caricamento finchè sia lo stato dell'allarme che la lista di persone e record viene caricata correttamente.

Le chiamate al Database che vengono fatte, sia in questa schermata, che in altre, sono rese molto semplici dall'utilizzo di Amplify.
Io ho deciso di utilizzare GraphQL e non Rest, che sarebbe comunque stata un'opzione messa a disposizione da AWS.
Vediamo l'esempio di una chiamata al Database che viene fatta molto di frequente nel codice:

```
1 suspend fun getUser(id: String): User {
2     return suspendCoroutine { continuation ->
3         Amplify.API.query(
4             ModelQuery[User::class.java, id],
5             { response ->
6                 val user = response.data
```

```
7                   if (user != null) {
8                       Log.i("User", "Retrieved user: $user")
9                       continuation.resume(user)
10                  }
11              },
12              { error ->
13                  Log.e("User", "Error querying user: $error, per id: $id
                        ")
14              }
15          )
16      }
17  }
```

È sufficiente chiamare una Query Amplify passando:

- Il modello che ci interessa
- L'id per il quale dobbiamo filtrare la ricerca

Sapendo che questa query restituirà un singolo elemento, essendo l'id univoco, prendiamo la risposta come singolo User, e la restituiamo dalla funzione.
Ci sono operazioni più complesse che possono essere eseguite con il Database, per esempio le Mutation.
Ma l'unica cosa che cambia è che nelle mutation dobbiamo creare il nuovo elemento in locale e poi passarlo alla funzione "mutate". Questo viene fatto usando un builder fornito da Amplify che è collegato agli schema Graphql presenti nel codice.

3.3.3 Schermata Records e Single record

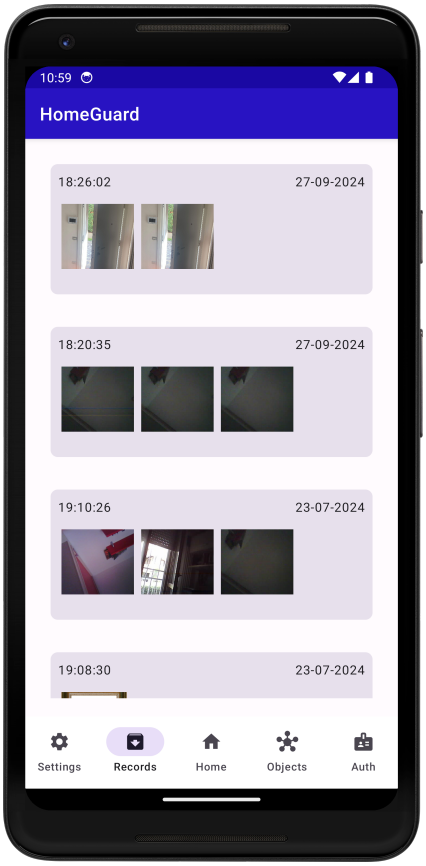

Figura 3.9: Schermata Record

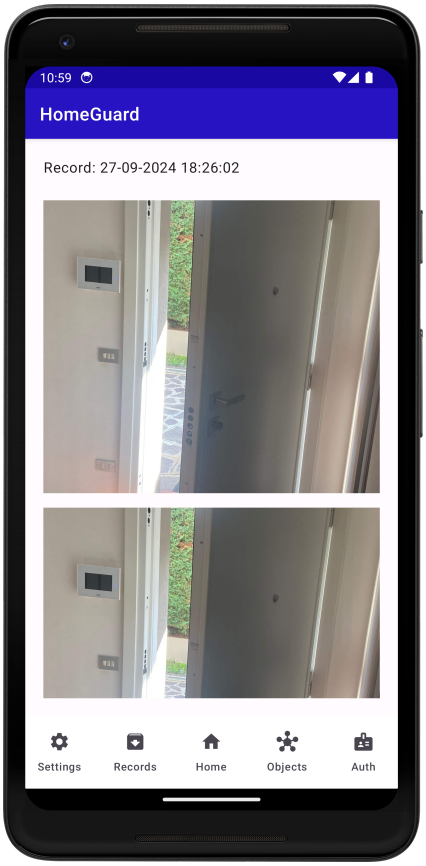

Figura 3.10: Schermata record singolo

La schermata dei Record è una LazyColumn e presenta in colonna tutti i record dell'abitazione in ordine dal più recente al meno.
Sono mostrate in piccolo tutte le foto del record, così si possono vedere dall'anteprima, scorrendo orizzontalmente.
Mentre cliccando sul record si raggiunge la schermata di record singolo nella quale si possono visualizzare le immagini di sicurezza in dimensione maggiore.

I Record vengono disposti in ordine cronologico, con i più recenti nella parte alte della schermata.

L'applicazione manda una notifica a tutti gli utenti appena l'allarme scatta e c'è un nuovo record di sicurezza da visualizzare.

3.3.4 Schermata Objects

Figura 3.11: Schermata Objects

La schermata Object è la schermata nella quale è possibile registrare altri dispositivi al nostro Account.
Si possono registrare un numero infinito di Sensori di movimento, Camere di Sicurezza o Sensori per le porte.

Aggiungere più sensori di movimento e sensori per le porte può essere utile per far scattare l'allarme non solo quando viene aperta la porta principale dell'abitazione.

I ladri potrebbero entrare dalle finestre, o da porte secondarie.
Altre fotocamere di sicurezza potrebbero invece fornire diversi punti di vista per poter identificare meglio gli eventuali ladri presenti nell'abitazione.

Cliccando su uno dei bottoni presenti nella schermata si viene reindirizzati verso la procedura guidata di registrazione, simile a quella presente nel SetUp iniziale per associare l'Allarme principale.

3.3.5 Schermata Authentication

Figura 3.12: Schermata Authentication

Figura 3.13: Registrazione nuova tessera NFC

La schermata di autenticazione serve a gestire tutti i metodi per autenticarsi con l'allarme fisico.
Come scritto in precedenza è possibile gestire l'allarme in modo automatico con solo la posizione. Ma in alternativa ci possono essere altri metodi, in particolare: Pin e Tessere NFC.

Si può impostare e visualizzare il Pin in questa schermata. Così una volta impostato può essere inserito in un eventuale tastierino del dispositivo Hardware per poter modificare lo stato dell'allarme anche manualmente.
Oltre al Pin, in questa schermata è possibile visualizzare il numero di tessere NFC connesse, ma non solo.
Infatti, cliccando sul tasto per aggiungere delle tessere NFC all'account, compare un popup che chiede di avvicinare la tessera al dispositivo.

In quel momento si sta attivando il servizio per poter leggere le tessere. Quando la tessera viene registrata si può chiudere il popup ed il servizio si fermerà.
L'identificativo della tesserà verrà salvato nel database di AWS in modo tale da poterlo rendere accessibile anche all'ESP32, che riconoscerà l'ID.
Nella prossima sezione analizziamo in modo più specifico il servizio NFC implementato.

3.3.6 Schermata Settings e Mappa

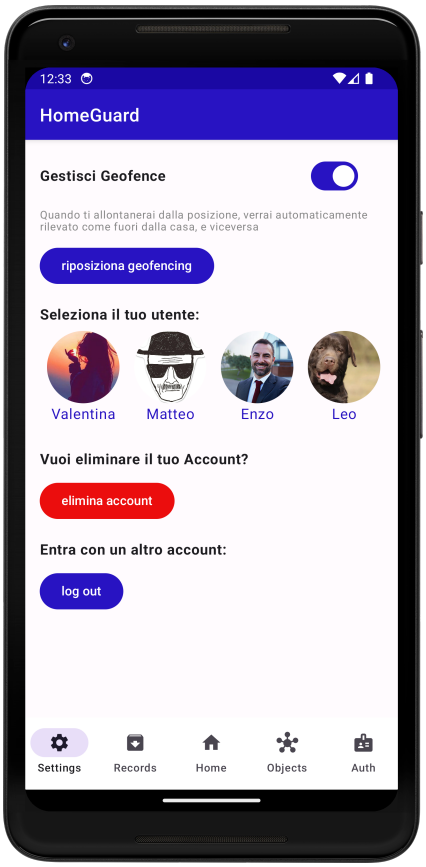

Figura 3.14: Schermata Settings

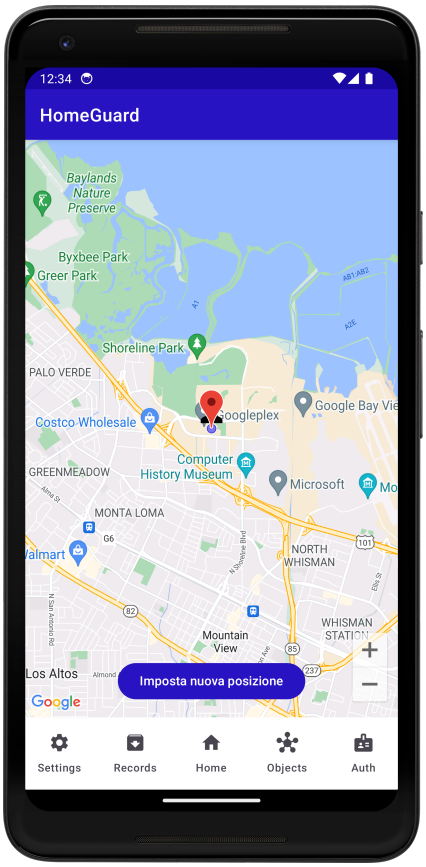

Figura 3.15: Schermata Mappa e Geofence

La sezione Settings presenta 4 parti:

- Pulsante per la schermata Mappa e Geofence
- Selezione della persona della casa
- Eliminazione Account
- Log Out

Il Log Out e l'eliminazione Account sono autoesplicativi.
Per la selezione degli utenti della casa viene mostrata la lista di persone, dalla quale poi, cliccando, si può selezionare il proprio utente.
Questo serve a capire, quando la posizione di un dispositivo entra ed esce dalla zona di geofence, chi delle persone della casa sta uscendo o entrando.

Infine la schermata della mappa.
Il pulsante nelle impostazioni che recita "riposiziona geofencing" serve proprio ad aprire una schermata dove, usando una libreria di google maps vengono mostrate due posizioni. La prima è la posizione dell'utente. La seconda è la posizione del geofencing, quindi della casa.
Essa mostra anche un'area circolare intorno ad essa, che rapprensenta l'area di raggio del geofence.
Infine, nella parte bassa della mappa, è presente il pulsante per reimpostare il geofence da dove era prima, alla posizione dell'utente in quel momento.

3.3.7 Servizio NFC

La gestione del servizio NFC avviene nella MainActivity, dalla funzione "enableForegroundDispatch".
La prima cosa che viene fatta è la creazione di un NFC Adapter, che può essere creato solamente se il dispositivo è fornito di sensore NFC o se l'NFC del dispositivo è abilitato. Dopo, se l'Adapter è stato abilitato con successo, può far partire l'ascolto di tessere NFC con "nfcAdapter.enableForegroundDispatch".

Questa funzione ha bisogno di un pendingIntent. Il PendingIntent in questo caso serve a specificare quale attività deve gestire l'evento NFC quando un tag viene rilevato mentre l'app è in esecuzione.
Viene utilizzato per garantire che l'Applicazione possa rispondere immediatamente a eventi NFC, anche quando è in esecuzione in primo piano, senza dover competere con altre app che potrebbero essere in grado di gestire eventi NFC.
Oltre a questo ha anche bisogno di IntentFilter per evitare di interessarsi a qualsiasi tipo

di evento, dato che a noi interessa solamente la lettura di un nuovo tag.

```
fun enableForegroundDispatch() {
    val nfcAdapter = NfcAdapter.getDefaultAdapter(this)
    if (nfcAdapter != null) {
        if (nfcAdapter.isEnabled) {
            val pendingIntent = PendingIntent.getActivity(
                this, 0, Intent(this, javaClass).addFlags(Intent.
                    FLAG_ACTIVITY_SINGLE_TOP),
                PendingIntent.FLAG_MUTABLE
            )
            val intentFiltersArray = arrayOf(
                IntentFilter(NfcAdapter.ACTION_NDEF_DISCOVERED),
                IntentFilter(NfcAdapter.ACTION_TAG_DISCOVERED),
                IntentFilter(NfcAdapter.ACTION_TECH_DISCOVERED)
            )
            nfcAdapter.enableForegroundDispatch(this, pendingIntent,
                intentFiltersArray, null)
        }
    }
}
```

Avviato il servizio di lettura tessere, serve il ricevitore, che gestisce gli intenti che arrivano all'Applicazione. E questo viene fatto con handleNfcIntent.
Quando viene ricevuto un intent la funzione estrae il tag identificativo dalla tessera e lo salva in database.

```
private fun handleNfcIntent(intent: Intent) {
    val authViewModel: AuthViewModel by viewModels()
    if (intent.action == NfcAdapter.ACTION_TAG_DISCOVERED ||
        intent.action == NfcAdapter.ACTION_NDEF_DISCOVERED) {
        val tag: Tag? = intent.getParcelableExtra(NfcAdapter.EXTRA_TAG)
        if (tag != null) {
            val tagId = tag.id
            val tagIdHex = tagId.joinToString(separator = "") { String.
                format("%02X", it) }
```

```
9               CoroutineScope(Dispatchers.IO).launch {
10                  registerNFC(tagIdHex)
11              }
12          }
13      }
```

3.3.8 Servizio Geofence

La gestione del geofence è divisa in due parti: la creazione e il receiver.
La funzione di creazione è nel Viewmodel della schermata della mappa, e viene chiamata quando viene cliccato il pulsante per impostarlo.

```
1  fun updateGeofence(context: Context, lat: Double, lon: Double) {
2      if (!::geofencingClient.isInitialized) {
3          return
4      }
5      val pendingIntent: PendingIntent by lazy {
6          val intent = Intent(context, GeofenceReceiver::class.java)
7          PendingIntent.getBroadcast(
8              context,
9              0,
10             intent,
11             PendingIntent.FLAG_UPDATE_CURRENT or PendingIntent.
                   FLAG_MUTABLE
12         )
13     }
14     currentGeofenceId?.let { geofenceId ->
15         geofencingClient.removeGeofences(listOf(geofenceId))
16     }
17
18     geofenceList.add(Geofence.Builder()
19         .setRequestId("GEOFENCE_ID")
20         .setCircularRegion(lat, lon, 75f)
21         .setExpirationDuration(Geofence.NEVER_EXPIRE)
22         .setTransitionTypes(Geofence.GEOFENCE_TRANSITION_ENTER or
               Geofence.GEOFENCE_TRANSITION_EXIT)
23         .build())
```

```
24
25      val geofencingRequest = GeofencingRequest.Builder().apply {
26          setInitialTrigger(GeofencingRequest.INITIAL_TRIGGER_ENTER)
27          addGeofences(geofenceList)
28      }.build()
29
30      geofencingClient.addGeofences(geofencingRequest, pendingIntent).run
            {
31          addOnSuccessListener {
32              currentGeofenceId = geofenceList[0].requestId
33              isGeofenceActive = true
34              //salvare geofence lat e lng
35          }
36      }
37  }
```

La funzione crea un pendingIntent, per ricevere i cambiamenti di posizione degli utenti. Rimuove precedenti Geofence, se esistono, e ne crea uno nuovo.
La creazione di un nuovo Geofence avviene con un geofence builder, che richiede le seguenti informazioni:

- L'ID del geofence
- Le coordinate e la larghezza della circonferenza
- La durata del geofence
- Il tipo di transizioni alla quale siamo interessati (nel mio caso entrata e uscita)

Successivamente, il Geofencing client aggiunge richiesta e pending intent. Se questo avviene con successo, il Geofence è stato creato.
Infine abbiamo il GeofenceReceiver, che gestisce l'arrivo degli Intenti per il Geofence. Ogni volta che viene ricevuto un evento, dopo i controlli su errori e specifiche, agisco di conseguenza. Se l'evento è di entrata disattivo l'allarme, se è di uscita lo attivo (se non ci sono altre persone in casa).

```
1     override fun onReceive(context: Context?, intent: Intent?) {
2         val geofencingEvent = intent?.let { GeofencingEvent.fromIntent(
              it) }
3         if (geofencingEvent.hasError()) {
4             return
5         }
6         val geofenceTransition = geofencingEvent?.geofenceTransition
7         if (geofenceTransition == Geofence.GEOFENCE_TRANSITION_EXIT) {
8             if (context != null) {
9                 handleGeofenceExit(context)
10            }
11        } else {
12            if (context != null) {
13                handleGeofenceEnter(context)
14            }
15        }
16    }
```

3.4 Implementazione AWS e funzioni Lambda

Amazon Web Services è il componente del sistema che si occupa di collegare le altre due parti, quindi riveste un ruolo fondamentale nel progetto.
Rende l'allarme un dispositivo Smart, dandogli un'identità online, e permette di gestire il Backend totalmente in Cloud.
Il lavoro svolto su AWS si suddivide in:

- Registrazione dei dispositivi sul Cloud con IoT Core
- Utilizzo di Amplify per l'applicazione mobile
- Utilizzo di AWS Simple Notification Service per l'applicazione mobile
- Utilizzo delle funzioni Lambda per compiere azioni e automazioni in Cloud

3.4.1 Registrazione dei dispositivi IoT sul Cloud con IoT Core

Per dare ad un normale dispositivo fisico un identità Online bisogna registrarlo in rete. Ovviamente il dispositivo deve avere la connessione ad Internet. Quando sarà connesso ad Internet, potrà essere registrato ed associato ad un servizio Cloud come IoT Core. Vediamo quindi tutti i passi da compiere sulla piattaforma Cloud per poterlo registrare in sicurezza.

Per prima cosa bisogna dare un nome all'oggetto e associare una policy. La policy decide ciò che il dispositivo potrà e non potrà fare sulla piattaforma Cloud.
Bisogna sempre concedere solo i permessi minimi per ciò che deve svolgere, mai di più. Questo per questioni di sicurezza.
La privacy policy nel mio caso doveva concedere al dispositivo di poter:

- Connettersi ad IoT Core con il suo specifico nome
- Pubblicare messaggi in un canale MQTT preciso
- Iscriversi all'ascolto di messaggi di un determinato canale MQTT
- Ricevere messaggi da un determinato canale MQTT

Dopo la creazione e associazione della Policy, vengono generati in automatico dei certificati. Questi certificati saranno richiesti durante la procedura di connessione dal dispositivo Smart.

Vanno quindi salvati in un file da importare nel codice Sketch dell'ESP32.
Questi certificati sono:

- Endpoint dei servizi di trust di Amazon (CA1)
- Certificato del dispositivo
- Chiave privata (univoca e non scaricabile in seguito)

Nel codice del dispositivo Smart vanno anche inseriti il nome del dispositivo scelto e l'endpoint di AWS.

3.4.2 AWS Amplify e servizi connessi

Amplify serve a connettere facilmente l'applicazione mobile a tutti i servizi AWS. Semplicemente connettendo l'applicazione a questo servizio, si possono utilizzare anche altre funzionalità della piattaforma Cloud. Tutte gestibili dalla console Amplify.
Esse sono:

- Cognito
- DynamoDB
- AppSync

Cognito viene usato in automatico quando si connette l'app ad Amplify per conservare e gestire il Login e la Registrazione di utenti.
Infatti, come detto in precedenza, la parte di autenticazione dell'applicazione viene gestita in automatico. E questo viene fatto appunto attraverso Cognito.
In automatico verranno salvati id utente, email e stato di conferma dell'account.

DynamoDB invece è gestibile successivamente, e non viene abilitato in automatico. Nonostante questo, si può comunque collegare direttamente all'applicazione creando il database dallo Studio Amplify.
Nella schermata "Data" si possono creare gli Schema GraphQl, che vengono poi riportati in automatico sia nell'applicazione che in un database DynamoDB.

Nella tesi sono presenti tre tabelle, tutte in relazione tra di loro:

- User
- Person
- RecordData

La tabella principale, quella dove vengono salvate le informazioni di ogni profilo, è User. User ha i seguenti campi: id, alarm (boolean), deviceIds (array di stringhe), thingsIds

(array di stringhe), pin (stringa), email (stringa), nfc (array di stringhe), latGeofence(stringa), lngGeofence(stringa), createdAt (stringa) e updatedAt (stringa).

Poi c'è Person, dove vengono salvate le informazioni di ogni utente della casa. Ci sono quindi qui Person per un singolo User.
Person ha i seguenti campi: id, userID (Relationship Source con User), inside (boolean), name (stringa), photo (stringa), updatedAt (stringa) e createdAt (stringa).

Infine c'è RecordData che rappresenta il singolo record di allarme scattato.
Ci sono più RecordData per un singolo User.
RecordData ha i seguenti campi: id, userID (Relationship Source con User), timestamp (stringa), photo (array di stringhe), photoBase64 (array di stringhe), createdAt (stringa) e updatedAt (stringa).

Infine AppSync, l'ultimo servizio utilizzato in modo integrato con Amplify, serve a gestire le chiamate GraphQL.
Viene importato lo schema creato sullo Studio Amplify e si può utilizzare per provare Query e Mutation.

3.4.3 Utilizzo di Simple Notification Service (SNS)

Simple Notification Service è un servizio non presente su Amplify che ho dovuto implementare separatamente.
Ho usato SNS per inviare notifiche in tempo reale e in automatico quando viene rilevato un cambiamento a campi specifici del database DynamoDB, in particolare con l'aggiunta di un nuovo record e con il cambiamento dello stato dell'Allarme da true a false e viceversa.

SNS per inviare delle notifiche push ai dispositivi ha bisogno del device token. Questo token viene estratto dal dispositivo e mandato in cloud ogni volta che l'applicazione si apre, dato che può cambiare nel tempo.

Ogni volta che devo mandare una notifica ad un account cerco tutti i token relativi ad esso e mando una notifica push a quelli presenti.

3.4.4 Le funzioni Lambda

Le funzioni Lambda sono la parte più importante dell'implementazione di AWS. Sono quelle che permettono al progetto di funzionare senza un Backend scritto a mano.
La possibilità di scrivere queste funzioni sul Cloud ed eseguirle in momenti precisi, permette di semplificare di molto l'architettura del Backend.
Sono state scritte tutte in Python.
Le funzioni utilizzate sono:

- CreateUserInDynamoDB
- SNS-Alarm
- SNS-Record
- CreateEndpointSNS
- deviceToDynamoDbFun

CreateUserInDynamoDB

Ogni funzione può avere sia Trigger che Destinazioni. In questo progetto ogni funzione Lambda ha dei Trigger, quindi viene chiamata in dei momenti precisi in modo automatico. Questa funzione viene Triggerata ogni volta che un nuovo utente viene aggiunto su AWS Cognito.
La funzione prende l'evento di aggiunta dell'utente e, estraendo email e id, crea il nuovo elemento in database.
Ecco un esempio di come avviene la creazione dell'utente in database da una funzione Lambda scritta in Python:

```
1 user_item = {
2     'id': {'S': cognito_user_id},
3     'alarm': {'BOOL': False},
```

```
4       'email': {'S': email},
5       'deviceIds': {'L': []},
6       'thingsIds': {'L': []},
7       'createdAt': {'S': current_time},
8       'pin' : {'S': ""},
9       'nfc' : {'L': []},
10  }
11
12  dynamodb.put_item(TableName=table_name_user, Item=user_item)
```

SNS-Alarm

Questa funzione viene triggerata quando il valore dell'allarme in database cambia, e serve a notificare gli utenti di questa modifica.
La funzione controlla che l'evento di dynamoDb che gli è arrivato è un evento di cambiamento di stato dell'allarme e se è così prende il suo nuovo valore.
In base al nuovo valore manda una notifica con un testo differente.
Poi prende i deviceId relativi all'utente il quale allarme è stato modificato e manda il messaggio a tutti quelli.

SNS-Record

Questa funzione è simile alla precedente ma manda una notifica quando viene aggiunto un record in database.
Viene mandata la notifica: "Allarme scattato!", dato che quando viene aggiunto un record significa che i sensori hanno rilevato un movimento.

CreateEndpointSNS

Questa funzione serve per creare endpoint ogni volta che in database viene aggiunto un device token.
Per mandare le notifiche, oltre al device token, serve la creazione di un endpoint per lo stesso. Che quindi va creato subito dopo, altrimenti non si potranno ricevere ancora

notifiche. Questo è come viene creato un nuovo endpoint per un dato token:

```
sns_client.create_platform_endpoint(
    PlatformApplicationArn=platform_application_arn,
    Token=device_id
)
```

DeviceToDynamoDbFun

Questa funzione viene triggerata ogni volta che IoT Core riceve un messaggio MQTT su un preciso canale scelto in precedenza.
Viene impostato l'ascolto sul canale con il quale tutti i device IoT comunicano con AWS e ogni volta che viene mandato qualcosa, si opera di conseguenza eseguendo questa funzione.
Ci sono diversi ruoli che deve coprire:

- Aggiornamento di stato dell'allarme
- Aggiunta di un record per uno specifico user
- Messaggio di accensione per sincronizzazione
- Aggiunta di foto nel record
- Aggiunta di tessera nfc per uno specifico user

Ogni messaggio inviato da un dispositivo IoT conterrà l'id dell'Account associato ad esso, per fornire alla funzione i dati necessari a compiere le sue operazioni.
In qualche caso potrebbe esserci la necessità di rispondere al dispositivo che ha inviato il messaggio.

In questo caso si manda così:

```
iot_response = iot_client.publish(
    topic=f'esp32door/{thingId}',
    qos=1,
    payload=json.dumps(photo_message)
)
```

Capitolo 4

Casi d'uso e Sviluppi Futuri

Vediamo quali sono i principali casi d'uso del progetto, e come può essere migliorato in sviluppi futuri.

4.1 Casi d'uso

Il sistema di sicurezza HomeGuard serve a proteggere la casa dagli intrusori. Questo viene fatto attraverso l'allarme, la fotocamera e l'applicazione Mobile.
I possibili use case del sistema sono:

- Esco di casa e l'allarme si attiva in automatico
- Entro in casa e l'allarme si disattiva in automatico
- Sono in vacanza e i ladri provano ad entrare in casa
- Disattivo o Attivo l'allarme utilizzando la tessera

4.2 Possibili sviluppi futuri

Il sistema può essere migliorato molto in futuro. Si possono fare grosse migliorie sia dal punto di vista Hardware che Software.

4.2.1 Altri sensori

Per quanto riguarda l'Hardware, si possono agiungere dei sensori da posizionare in altre zone della casa.
Sensori di movimento, per rilevare la presenza di individui estranei anche se non entrano dalla porta ma dalle finestre. Oppure altri sensori di porta per entrate secondarie.

Si può usare poi una camera di sorveglianza di qualità maggiore, dato che al momento il dispositivo utilizzato è un prodotto low-cost.
Si può alimentare il dispositivo a pile invece che con la corrente, anche se porterebbe sia benefici che lati negativi.

Sicuramente sarebbe un dispositivo di più facile implementazione. La durata della batteria sarebbe però da verificare, in quanto è importante che l'allarme non si spenga improvvisamente o abbia durata troppo breve.

4.2.2 Software

Anche il Software può essere migliorato molto.
Si può rendere l'Applicazione un punto di controllo per ogni tipo di dispositivo IoT della casa, invece che solamente per dispositivi proprietari.
Implementando questa compatibilità con sempre più brand di dispositivi Smart, si potrebbe rendere l'applicazione un ottimo prodotto per chiunque, non solo per i possessori dell'Hardware proprietario.

Possono essere implementate funzionalità come la programmazione dell'allarme, la visualizzazione di una videocamera di sorveglianza in tempo reale e la possibilità di scattare foto quando si preferisce.
Può anche essere integrato un sistema di Intelligenza Artificiale per il monitoraggio e il controllo dell'allarme automatico basato sulle abitudini degli utenti.

Questo sarebbe possibile dato che l'applicazione ha accesso in tempo reale e in ogni momento alla posizione degli utenti.
Le possibilità quando si parla di software sono praticamente infinite, soprattutto in un campo come questo che è in continua espansione.

Conclusioni

Il sistema di sicurezza HomeGuard è un progetto IoT innovativo, che mira a rendere l'utilizzo degli oggetti intelligenti più semplice e automatico.
Lo scopo è ampliare l'utilizzo degli oggetti smart, anche e sopratutto per persone che non sono avvezze all'utilizzo della tecnologia.

L'Internet of Things non deve essere una nuova tecnologia per pochi, anzi deve essere il modo per rendere la tecnologia utilizzabile da tutti.
La possibilità di attivare e disattivare l'allarme in modo automatico punta a rendere la gestione dell'allarme della casa non necessario. Si lascia la gestione all'applicazione, senza dover fare nulla.

Questa direzione per lo sviluppo di sistemi come HomeGuard può portare l'IoT in ogni casa, risolvendo quello che è il problema attuale di questo settore informatico.
Infatti uno dei principali ostacoli nel settore della sicurezza informatica domestica è la mancanza di soluzioni accessibili, capaci di offrire servizio senza compromettere la facilità d'uso o richiedere competenze tecniche all'utilizzatore.

Bibliografia

[1] Subhas Chandra Mukhopadhyay. Internet of Things: Challenges and Opportunities, 2014

[2] Chase, Jim. "The evolution of the internet of things." Texas Instruments 1.1388 (2013): 1-7.

[3] Satyajit Sinha. (3-09-2024). State of IoT 2024: Number of connected IoT devices growing 13% to 18.8 billion globally. https://iot-analytics.com/number-connected-iot-devices.

[4] A. Al-Fuqaha, M. Guizani, M. Mohammadi, M. Aledhari and M. Ayyash, "Internet of Things: A Survey on Enabling Technologies, Protocols, and Applications," in IEEE Communications Surveys & Tutorials, vol. 17, no. 4, pp. 2347-2376, Fourthquarter 2015, doi: 10.1109/COMST.2015.2444095.

[5] Charlie Wilson, Tom Hargreaves, Richard Hauxwell-Baldwin, Benefits and risks of smart home technologies, Energy Policy, Volume 103, 2017, Pages 72-83, ISSN 0301-4215, https://doi.org/10.1016/j.enpol.2016.12.047.

[6] Wenda Li, Tan Yigitcanlar, Isil Erol, Aaron Liu, Motivations, barriers and risks of smart home adoption: From systematic literature review to conceptual framework, Energy Research & Social Science, Volume 80, 2021, 102211, ISSN 2214-6296, https://doi.org/10.1016/j.erss.2021.102211.

[7] Babiuch, Marek, Petr Foltýnek, and Pavel Smutný. "Using the ESP32 microcontroller for data processing." 2019 20th International Carpathian Control Conference (ICCC). IEEE, 2019.

[8] Esp32 pinout. https://www.electronicshub.org/.

[9] Amazon Web Services Official Documentation. https://aws.amazon.com/it/.

[10] Tawalbeh, L.A.; Tawalbeh, H. Lightweight crypto and security. In Security and Privacy in Cyber-Physical Systems: Foundations, Principles, and Applications; Wiley: West Sussex, UK, 2017; pp. 243–261.

[11] Johnstone, M.N.: Threat Modelling with Stride and UML. In: Australian Information Security Management Conference (November), vol. 18 (2010).

[12] MQTT Official Documentation. https://mqtt.org/.

[13] GraphQL Official Documentation. https://graphql.org/.

Ringraziamenti

Per prima cosa voglio ringraziare il mio relatore, Federico Montori, per avermi dato sempre ottimi consigli e per essersi reso disponibile a risolvere i miei dubbi.

Ringrazio i miei genitori, per avermi sempre supportato durante questi anni e per aver creduto in me nonostante le prestazioni "non ottime" alle superiori.

Ringrazio la mia famiglia, per tutti i complimenti non voluti che mi sono stati fatti in questi anni. Vorrei ricordare anche il vostro "superpotere": conoscere i risultati dei miei esami prima di me. Ogni volta, il voto sembrava essere già arrivato a tutti grazie alla prontezza di una persona in particolare nel diffondere la notizia.

Ringrazio Vale ed Elia, che hanno realizzato una grande performance attoriale nel video di dimostrazione della tesi. Grazie Vale, per aver reso i pomeriggi di studio meno tristi e solitari, potendo parlarti per un pò dopo ore davanti al computer. Grazie Frigo, per tutti i progetti fatti insieme e per i kebab mangiati in questi anni.

Ringrazio Fra, per aver sempre condiviso le sue passioni con me. Anche con Arduino, che ha dato vita all'idea di questa tesi.

Ringrazio la Rebby, per avermi sempre aiutato in tutti questi anni. Da quando ci conosciamo mi hai sempre spronato a fare meglio. Dalle superiori, quando mi aiutavi a studiare anche se non ne avevo voglia, fino ad oggi.
Sei sempre stata più felice di me per i miei successi: dal primo esame, quando eravamo

in montagna insieme, al secondo in quarantena, fino all'ultimo, quando mi hai fatto una crostata buonissima per festeggiare.

Vederti veramente contenta per i miei risultati è sempre stato più bello della mia soddisfazione nel raggiungerli. E vederti triste per gli esami non passati, mi rendeva meno triste.

Sapere che eri fiera di me e di quello che stavo facendo mi ha sempre dato la motivazione di cui avevo bisogno.

Questo percorso non sarebbe mai stato uguale senza di te, e non sarebbe stata uguale neanche la mia vita. Mi hai insegnato cosa significa amare una persona, non avendo mai paura di mostrare i tuoi sentimenti.

Quindi una parte di questo traguardo è anche merito tuo, per avermi dato la forza di studiare, sapendo che a fine giornata ci saremmo potuti vedere.