UNIVERSITÀ DEGLI STUDI DI CAGLIARI

Dipartimento di Matematica e Informatica

Corso di laurea Magistrale

RELAZIONE TECNICA

Home security system

DOCENTE

Prof. Roberto SAIA

STUDENTI

65370 Francesco SIMBOLA

65371 Lorenzo SUSINO

Embedded Systems for the Internet of Things

A.A. 2024-2025

INDICE

Requisiti ad alto livello del progetto

Gestione di un sistema di sorveglianza dell'abitazione per effrazioni ed incendio basato su sensori IoT.

Il sistema si basa su sensori di tipo "Gas sensor" e "Volumetric sensor", le cui letture vengono archiviate in un database AWS e gestite attraverso il paradigma delle shadow things. In caso di rilevazione di un allarme, una notifica deve essere immediatamente inviata al cellulare del proprietario.

Ogni dispositivo è dotato di un display che consente di visualizzare sia il numero di allarmi verificatisi in un determinato intervallo di tempo, sia eventuali mancate notifiche al proprietario. Inoltre, il sistema offre la possibilità di monitorare lo stato dell'abitazione da remoto tramite un'interfaccia web, permettendo una gestione completa e centralizzata di ciascun sensore.

Specifiche del progetto

In questo capitolo viene descritta la specifica dei requisiti funzionali per un sistema di sorveglianza domestica basato su sensori IoT, progettato per rilevare tentativi di effrazione e situazioni di incendio.

Durante lo sviluppo abbiamo adottato diverse decisioni progettuali che, a nostro avviso, garantiscono uno sviluppo scalabile ed efficace dei requisiti del sistema. Di seguito sono presentate le decisioni progettuali adottate, insieme alle motivazioni che ne hanno guidato l'implementazione:

- **Scelta dei sensori**: per lo sviluppo del progetto abbiamo deciso di acquistare dei sensori di rilevamento della distanza e di rilevamento di gas nell'aria per cercare di sviluppare un progetto il più accurato possibile e che potesse anche essere riprodotto per nostri usi personali successivamente.
- **Node.js per il backend**: Per il backend abbiamo inizialmente valutato l'utilizzo di Node-Red o Node.js. In conclusione, abbiamo optato per Node.js per diversi motivi: la sua maggiore flessibilità e controllo sul codice, l'ampia disponibilità di librerie e strumenti per lo sviluppo backend, una più facile integrazione con il resto dell'architettura, e la nostra maggiore familiarità con questo ambiente, che ci ha permesso di essere più efficienti nello sviluppo pur affrontando comunque sfide tecniche interessanti..

- **Angular per il frontend**: Per il frontend abbiamo scelto Angular, un framework open-source ampiamente adottato nell'industria, che offre un eccellente compromesso tra professionalità ed efficacia. Grazie all'integrazione con Angular Material, è possibile sviluppare rapidamente una dashboard funzionale e performante per il sistema.
- **DynamoDB**: Per la gestione dei dati abbiamo optato per DynamoDB, un database NoSQL completamente gestito da AWS. Questa scelta è stata dettata da diversi fattori: la necessità di una soluzione scalabile, ad alte prestazioni e l'ottima integrazione con il resto dell'ecosistema AWS.

2.1 Requisiti funzionali dettagliati del sistema

Il sistema fornisce una piattaforma di monitoraggio dell'ambiente di casa. Il primo utente che si registrerà nel sistema avrà ruolo di "Amministratore", mentre i successivi avranno come ruolo "Familiare". Il pannello di accesso fornisce gli strumenti per effettuare l'accesso, la registrazione e il ripristino password tramite e-mail. Una volta effettuato l'accesso un utente si troverà dinnanzi a un'interfaccia che presenta le seguenti sezioni di controllo:

- **Casa**: Questa sezione presenta tre pannelli di monitoraggio della sezione casa. Il pannello "Dashboard" presenta lo stato della casa avvisando in caso di notifiche di emergenze non ancora risolte e permettendo la risoluzione delle stesse. I pannelli "Antincendio" e "Antieffrazione" presentano due tabelle, la prima mostra l'elenco di sensori e le loro relative ultime rilevazioni, con possibilità di recuperare lo storico delle rilevazioni presenti nel database; la seconda mostra le notifiche in attesa e risolte.
- **Profilo**: In questa sezione l'utente potrà avere accesso alle proprie informazioni ma soprattutto potrà inserire il proprio telegramID per permettere al bot di inviargli le notifiche di allarme.

Un'altra funzionalità fondamentale del sistema è la gestione delle notifiche tramite telegram bot. Per implementare ciò un utente potrà cercare su telegram il bot chiamato: "HomeSecurityEsit_bot". All'avvio della conversazione il bot fornirà il codice chat dell'utente che dovrà inserire nella sezione profile del gestionale. Da quel momento in poi il bot telegram invierà messaggi di allarme in caso di necessità.

Attori del sistema

Gli attori sono individui o entità che assumono un ruolo, attivo o passivo, nell'ambito del sistema, interagendo con esso attraverso una o più delle sue funzionalità. In questa definizione rientrano anche quelle entità che, pur non intervenendo direttamente, risultano comunque coinvolte, anche solo in modo indiretto, nel funzionamento del sistema.

1. **Amministratore**: è colui in grado di accendere e spegnere il sistema. Ha accesso a tutte le funzionalità del gestionale;
2. **Familiare**: è un utente che utilizza l'applicazione al fine di monitorare lo stato della casa, potrà avere accesso allo stato del sistema da remoto e potrà ricevere le notifiche di allarme tramite le metodologie a cui ha aderito;

Componenti del sistema

In questa sezione verranno definite le componenti del sistema e come vengono utilizzate al fine di garantire tutte le funzionalità del sistema.

1. **Sensori**: sono il cuore del sistema, per controllare eventuali incendi sono stati utilizzati dei sensori di rilevamento di gas capaci di misurare la concentrazione di GPL, i-butano, propano, metano, alcool, idrogeno e fumo nell'aria. Per controllare eventuali effrazioni sono stati utilizzati dei sensori di misurazione della distanza a ultrasuoni.

2. **Display**: i dispositivi del sistema sono dotati di display che visualizza il numero di allarmi che si sono verificati in un intervallo di tempo.

3. **Dashboard Utente**: la piattaforma utilizzata per verificare lo stato dell'applicazione, ogni utente potrà accedere con le proprie credenziali e utilizzare le funzionalità di cui ha permesso.

4. **Sistema di Notifiche**: per avvisare gli utenti in caso di allarmi è stato implementato un sistema di notifiche che permette all'utente di scegliere quali metodi di notifica vuole utilizzare tra i seguenti: telegram bot, e-mail.

5. **Base di dati**: il sistema memorizza i dati tramite una base di dati caricata nel cloud. Ogni informazione viene raggruppata nella propria categoria apposita in modo da mantenere una struttura logica coerente.

Casi d'uso

Il gestionale si occupa di fornire agli utenti del sistema protezione in caso di eventuali emergenze di incendi o effrazioni, per fare ciò consente di monitorare la propria casa da remoto. Gli utenti potranno accedere a una dashboard che mostra lo stato della casa in funzione delle letture rilevate dai sensori installati. In caso di emergenze gli utenti verranno notificati in modo da poter agire prontamente. In seguito elenchiamo i requisti funzionali che il sistema dovrà garantire:

1. **Sistema di accesso alla dashboard**:

Attore	Utente registrato
Descrizione	L'utente accede alla dashboard inserendo le proprie credenziali.
Scenario	• L'utente apre l'interfaccia di login. • Inserisce email e password. • Il sistema verifica le credenziali. • Se corrette, viene effettuato il login e viene generato un token di sessione. • Se errate, viene mostrato un messaggio di errore.

2. **Creazione di un utente**:

Attore	Utente non registrato
Descrizione	L'utente effettua la registrazione inserendo i propri dati.
Scenario	• L'utente apre l'interfaccia di registrazione. • Inserisce username, email e password. • Il sistema valida i dati. • Se corretti, viene crittata la password, viene effettuata la registrazione e viene generato un token di sessione. • Se non validi, viene mostrato un messaggio di errore.

3. **Sistema di cambio credenziali della dashboard**:

Attore	Utente registrato
Descrizione	L'utente può modificare le proprie credenziali (password).
Scenario	• L'utente apre l'interfaccia di login. • Seleziona password dimenticata. • Inserisce la mail per il recupero password. • Se corrette, viene inviata una mail contentente il link per effettuare il recupero password. • Il link della mail contiene un form di reset password.

4. **Attivazione del sistema di allarme**:

Attore	Utente amministratore
Descrizione	L'utente attiva il sistema di allarme per abilitare il monitoraggio dei sensori.
Scenario	• L'utente accede alla dashboard. • Attiva il sistema tramite il toggle. • Il sistema abilita i sensori e inizia il monitoraggio. • Lo stato del sistema cambia a "Acceso".

5. **Disattivazione del sistema di allarme**:

Attore	Utente amministratore
Descrizione	L'utente disattiva il sistema di allarme per disabilitare il monitoraggio dei sensori.
Scenario	• L'utente accede alla dashboard. • Disattiva il sistema tramite il toggle. • Il sistema abilita i sensori e inizia il monitoraggio. • Lo stato del sistema cambia a "Spento".

6. **Configurazione del sistema di notifiche**:

Attore	Utente registrato
Descrizione	L'utente configura il proprio profilo per ricevere le notifiche telegram
Scenario	• L'utente cerca HomeSecurityEsit_bot su telegram. • Avvia la chat tramite il comando "/start". • Il bot fornisce il codice chatID dell'utente. • L'utente accede alla sezione Profilo. • Tramite l'interfaccia modifica il proprio telegramID

7. **Visualizzazione delle notifiche su display**:

Attore	Sistema
Descrizione	Il sistema carica il quantitativo di notifiche non risolte del gestionale su un display
Scenario	• Un sensore rileva una condizione critica. • Il sistema genera un messaggio di allarme. • Il sistema mostra il quantitativo di allarmi generati nelle ultime 6 ore.

8. **Monitoraggio dei livelli di gas nell'aria**:

Attore	Sistema
Descrizione	Il sistema monitora eventuali incendi tramite sensori a CO2 e informa l'utente
Scenario	• Il sensore effettua letture periodiche. • I dati vengono memorizzati tramite il paradigma delle shadow e dynamoDB. • Il backend recupera i dati e li fornisce al frontend • L'utente può consultarli in tempo reale dalla dashboard.

9. **Monitoraggio effrazioni**:

Attore	Sistema
Descrizione	Il sistema monitora eventuali effrazioni tramite sensori volumetrici e informa l'utente
Scenario	• Il sensore effettua letture periodiche. • I dati vengono memorizzati tramite il paradigma delle shadow e dynamoDB. • Il backend recupera i dati e li fornisce al frontend • L'utente può consultarli in tempo reale dalla dashboard.

10. **Visualizza storico sensore**:

Attore	Utente registrato
Descrizione	L'utente visualizza lo storico degli allarmi
Scenario	• L'utente accede alla dashboard. • Seleziona la tab Antincendio/Antieffrazione • Visualizza la tabella Notifiche sistema

11. **Visualizza storico notifiche**:

Attore	Utente registrato
Descrizione	L'utente visualizza lo storico degli allarmi
Scenario	• L'utente accede alla dashboard. • Seleziona la tab Antincendio/Antieffrazione • Visualizza la tabella Notifiche sistema

12. **Spegni tutti gli allarmi**:

Attore	Utente registrato
Descrizione	L'utente spegne tutti gli allarmi in stato pending
Scenario	• L'utente accede alla dashboard. • Se sono presenti allarmi attivi preme il pulsante "Risolvi tutti" • Gli allarmi vanno in stato "resolved"

13. **Spegni un allarme specifico**:

Attore	Utente registrato
Descrizione	L'utente spegne un allarme specifico
Scenario	• L'utente accede alla dashboard. • Nella tabella notifiche preme il pulsante "Risolvi" presente nella riga dell'allarme da disattivare • L' allarme va in stato "resolved"

2.2 Implementazione del sistema

Questa sezione illustra in maniera dettagliata come il sistema ESIT HOME SECURITY implementa le funzionalità descritte nella sezione 2.1, ovvero la gestione di un sistema di sorveglianza integrato per la protezione domestica contro effrazioni e incendi. Verranno esposte in particolare le scelte progettuali riguardanti l'architettura hardware/software (HW-SW) e le tecnologie adottate. Tale descrizione è fondamentale per comprendere il flusso dei dati, il funzionamento dei vari componenti e la logica operativa alla base della soluzione, nonché per giustificare le scelte tecniche che rendono il sistema scalabile ed efficiente.

2.2.1 Architettura HW-SW

Descrizione Generale

L'architettura del sistema si fonda su un approccio IoT che prevede la distribuzione di nodi di rilevamento (dotati di sensori specifici) in ogni ambiente domestico da monitorare. Ogni nodo, basato su una scheda ESP8266, è dedicato a un particolare tipo di rilevamento:

- **Nodo Rilevamento Effrazioni**: utilizza il sensore HC-SR04 per misurare la distanza, con lo scopo di identificare eventuali anomalie legate a intrusioni o movimenti sospetti.
- **Nodo per Rilevamento Gas e Fumo**: utilizza il sensore MQ-2 per misurare la concentrazione di gas (GPL, i-butano, propano, metano, alcool, idrogeno) e rilevare la presenza di fumo, utili per individuare situazioni di incendio.

Oltre a questi sensori, ogni board è dotata di un display integrato che mostra in tempo reale il numero di allarmi verificatisi e le eventuali notifiche mancate, garantendo così un feedback immediato all'utente in loco.

Schema a Blocchi dell'Architettura

Per una migliore comprensione, di seguito si riporta uno schema a blocchi concettuale che riassume il funzionamento e la connessione tra i vari componenti:

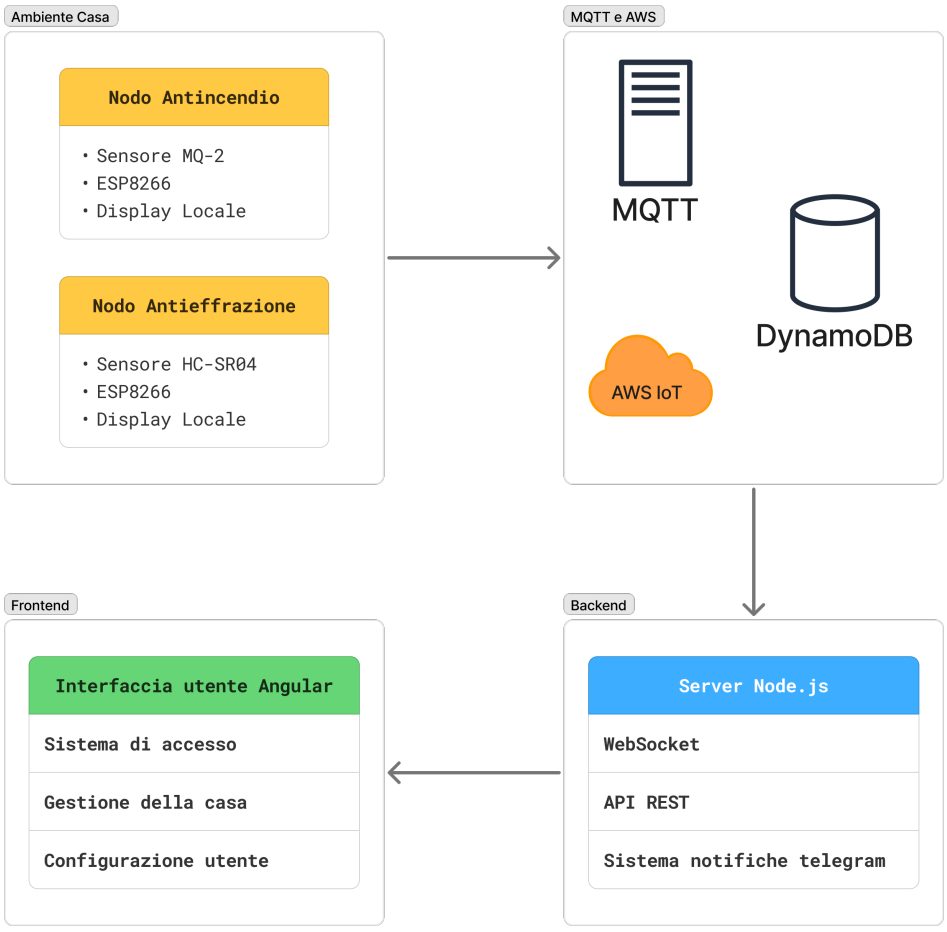

Figure 2.1: Schema Architettura

Database

Per collezionare i dati del progetto abbiamo usato dynamoDB, abbiamo creato le seguenti tabelle:

Alarms

La tabella Alarms viene utilizzata per memorizzare gli allarmi che vengono generati dal sistema, viene salvato il timestamp della rilevazione, l'id del dispositivo, lo stato dell'allarme (resolved, pending), il tipo di allarme (smoke, burglary)

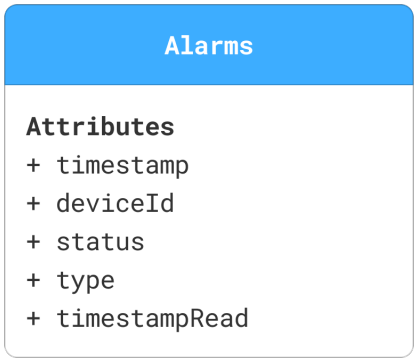

Figure 2.2: Tabella Alarms

SensorData

La tabella SensorData rappresenta il cuore del sistema, in essa vengono memorizzati i dati delle rilevazioni dei sensori che saranno accessibili dagli utenti registrati

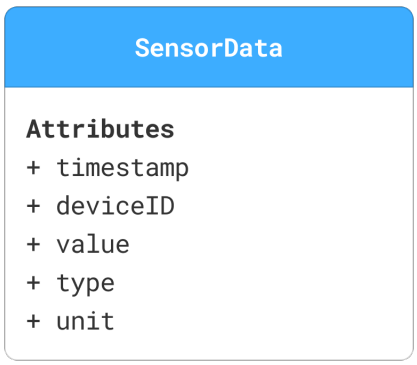

Figure 2.3: Tabella SensorData

SystemStatus

La tabella SystemStatus permette di attivare e disattivare i sistemi di sicurezza da remoto, memorizzando uno stato attivo/inattivo

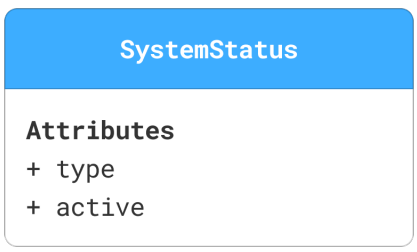

Figure 2.4: Tabella SystemStatus

Users

La tabella Users permette una corretta gestione degli utenti che utilizzeranno la piattaforma.

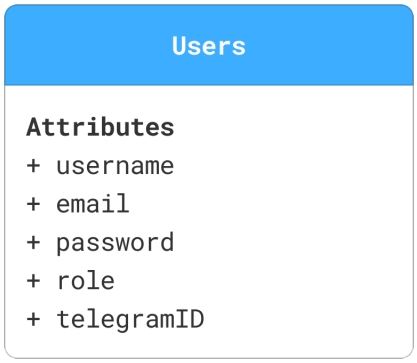

Figure 2.5: Tabella Users

Funzionamento dei Nodi di Rilevamento

- **Nodo Volumetrico (HC-SR04):**

  Il codice implementato per questo nodo prevede la configurazione dei pin dedicati al TRIG ed ECHO, la connessione alla rete WiFi e la gestione della pubblicazione dei dati tramite il protocollo MQTT. Il sensore HC-SR04 emette un impulso di 10 μs e calcola la distanza in base al tempo impiegato dall'eco a ritornare. Il valore della distanza viene convertito in formato stringa e pubblicato sul topic MQTT `sensor/distance`.

  Il ciclo di lettura avviene ogni secondo, garantendo un monitoraggio continuo e aggiornato. Il dispositivo è dotato di un display locale (LCD I2C), che visualizza due informazioni fondamentali:

  – il numero totale di allarmi relativi alle effrazioni verificatisi in un determinato intervallo di tempo (6 ore);

  – il numero di eventuali mancate notifiche al proprietario, causate da problemi di rete o di messaggistica.

  Oltre alla pubblicazione MQTT, i dati vengono inviati anche tramite API Gateway a AWS DynamoDB. Questo processo è orchestrato da AWS Lambda functions, che ricevono i dati, li parsano correttamente e li inseriscono nelle tabelle DynamoDB predefinite. Le Lambda functions sono configurate con policy IAM specifiche per garantire operazioni sicure ed efficienti sul database.

- **Nodo per Rilevamento Gas e Fumo (MQ-2):**

  Questo nodo utilizza il sensore MQ-2, che fornisce due tipologie di dati:

  - un segnale analogico (scala 0–1023), indicante la concentrazione di gas rilevata;
  - un segnale digitale (0 o 1), determinato dalla soglia impostata tramite potenziometro.

  Il codice legge simultaneamente entrambi i segnali, li converte in stringhe e li pubblica su due topic MQTT distinti: `sensor/mq2/analog` e `sensor/mq2/digital`. Anche in questo caso, il ciclo di lettura avviene ogni secondo.

  Il display locale (LCD I2C) visualizza in tempo reale:

  - il numero totale di allarmi rilevati relativi agli incendi in un dato intervallo di tempo (6 ore);
  - il numero di mancate notifiche al proprietario.

  Analogamente al nodo volumetrico, anche i dati del MQ-2 vengono trasmessi tramite API Gateway a DynamoDB. Le Lambda functions associate si occupano del parsing e della memorizzazione nelle rispettive tabelle. Le policy IAM assegnate garantiscono che le operazioni avvengano in sicurezza e nel rispetto delle best practice AWS.

Integrazione e Flusso dei Dati

Una volta che i dati vengono pubblicati dai nodi di rilevamento, essi transitano tramite la rete WiFi verso il broker MQTT e verso DynamoDB. Il broker, in questo caso ospitato su "broker.mqtt-dashboard.com", agisce come hub centrale per la gestione e il routing dei messaggi. AWS IOT ci permette di implementare il paradigma delle shadow inserendo in esse i valori inviati ai topic MQTT. Node.js si connette successivamente sia alle shadow che al database di DynamoDB per recuperare i dati che gli servono.

Flusso di Elaborazione: Node.js è configurato per eseguire una serie di operazioni quali:

- Recupero dati in tempo reale dalle shadow.
- Recupero dei dati dal database.
- Modifica dei dati del database.
- Inoltro notifiche tramite telegram bot.
- Gestione di WebSocket per il frontend.

Archiviazione e Visualizzazione:

I dati vengono inviati a una base dati cloud (DynamoDB), che consente una conservazione strutturata e sicura delle informazioni. Parallelamente, una dashboard sviluppata in Angular fornisce all'utente un'interfaccia intuitiva per il monitoraggio in tempo reale e la gestione delle notifiche. Questa integrazione garantisce un sistema flessibile e scalabile, capace di rispondere prontamente a situazioni di emergenza e di fornire una panoramica completa dello stato della casa.

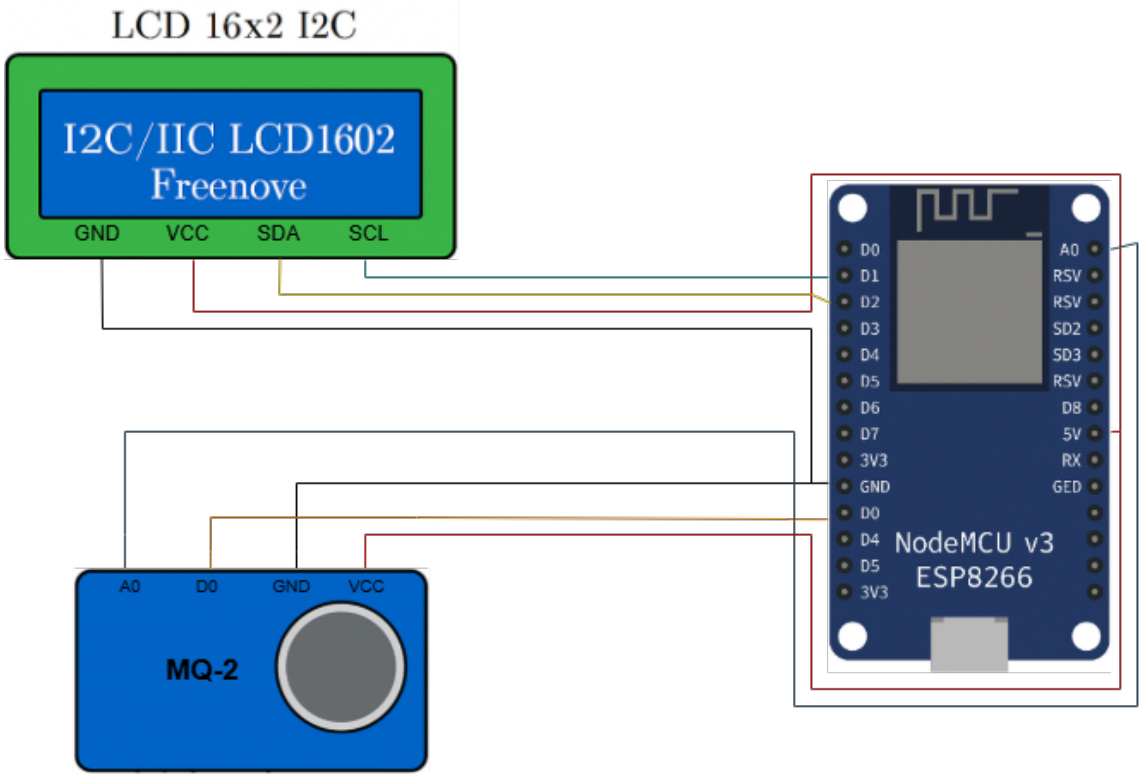

Figure 2.6: Schema 1: Circuito per il Sensore Volumetrico (HC-SR04) su NodeMCU

Descrizione Schema 1: In questo schema il sensore HC-SR04 è collegato al NodeMCU in maniera tale da fornire i dati relativi alla distanza rilevata.

- Il pin **Vcc** del sensore è alimentato tramite la sorgente di 5V (oppure il VIN, a seconda della configurazione) e il pin **GND** è collegato a terra.
- Il segnale di trigger viene inviato al pin **D5** del NodeMCU, mentre il segnale di echo viene ricevuto dal pin **D6**.
- Inoltre, il NodeMCU è dotato di un display (collegato via I2C tramite SDA/SCL) che permette la visualizzazione locale dei dati e del numero di allarmi.

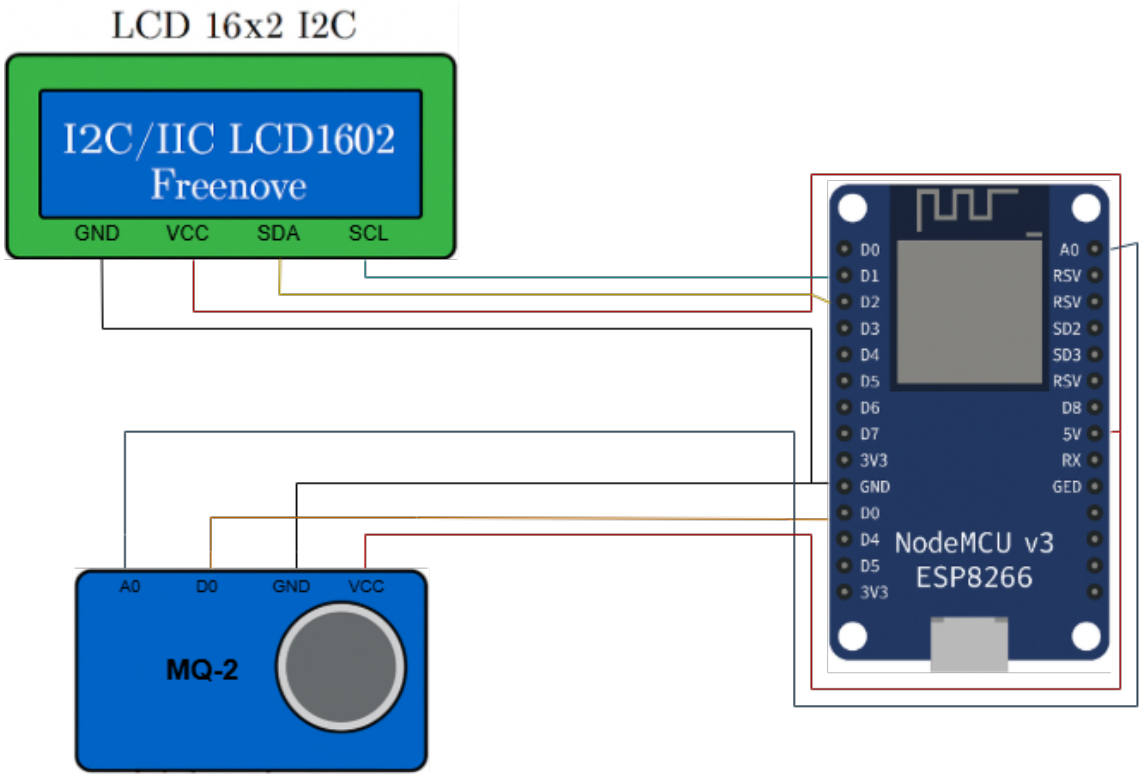

Figure 2.7: Schema 2: Circuito per il Sensore Gas/Fumo (MQ-2) su NodeMCU

Descrizione Schema 2: In questo schema il sensore MQ-2, utilizzato per il rilevamento di gas e fumo, è integrato con il NodeMCU.

- Il sensore riceve l'alimentazione (Vcc e GND) dalla stessa fonte utilizzata per il NodeMCU.
- L'uscita analogica del sensore è collegata al pin **A0** del NodeMCU, mentre l'uscita digitale (la soglia regolata tramite potenziometro) è collegata al pin **D0**.
- Anche in questo caso, il NodeMCU comunica con un display (connesso via I2C) che visualizza in tempo reale le letture e lo stato degli allarmi.

Questi schemi riassumono graficamente il setup hardware previsto per il progetto ESIT HOME SECURITY, evidenziando come ciascuna board dedicata al sensore (sia HC-SR04 che MQ-2) sia interfacciata con il NodeMCU e un display per il feedback locale, nonché la connessione all'alimentazione necessaria per il corretto

funzionamento dei componenti.

2.2.2 Tecnologie Utilizzate

Hardware

Il sistema ESIT HOME SECURITY è basato su una serie di componenti hardware selezionati per garantire affidabilità, rapidità di risposta e scalabilità:

Sensori:

- **HC-SR04:** Sensore ad ultrasuoni utilizzato per il rilevamento volumetrico. La sua capacità di misurare la distanza in tempo reale lo rende ideale per identificare intrusioni o movimenti sospetti attorno all'abitazione.
- **MQ-2:** Sensore in grado di rilevare la presenza di gas (tra cui GPL, i-butano, propano, metano, alcool, idrogeno) e di fumo. Questa caratteristica lo rende fondamentale per il monitoraggio preventivo di situazioni di incendio.

Schede di Controllo: Ogni sensore è associato a una board basata su **ESP8266**. Questa scelta è motivata dalla necessità di avere una connessione WiFi integrata che permetta la comunicazione immediata dei dati al sistema centrale tramite il protocollo MQTT. Le board sono inoltre equipaggiate con display LCD o OLED che visualizzano, in tempo reale, i dati rilevati e il numero di allarmi generati.

Display Locale: Ogni board dispone di un display dedicato, utilizzato per la visualizzazione delle informazioni critiche:

- Numero di allarmi verificatisi in un determinato intervallo di tempo.
- Notifiche in tempo reale, utile sia per il monitoraggio in loco che per verificare la corretta trasmissione dei dati.

Software e Protocolli di Comunicazione

La componente software del sistema è altrettanto fondamentale e si basa su scelte tecnologiche che garantiscono facilità di integrazione e robustezza:

- **Node.js:** Utilizzato per il backend, Node.js è un ambiente di esecuzione JavaScript lato server che consente la gestione efficiente e scalabile dei dati provenienti dai sensori. Grazie alla sua architettura event-driven e all'ecosistema ricco di librerie (come Express, WebSocket, AWS SDK), Node.js permette di implementare flussi di elaborazione modulari e sicuri per la validazione, il filtraggio e l'invio automatico di notifiche.
- **Protocollo MQTT:** Il protocollo MQTT, leggero e adatto a contesti IoT, viene utilizzato per la trasmissione dei dati dai nodi di rilevamento al broker centrale. Grazie alla sua architettura publisher/subscriber, MQTT garantisce una comunicazione affidabile e a bassa latenza, essenziale per la gestione di emergenze e per il monitoraggio in tempo reale.
- **Angular per il Frontend:** L'interfaccia utente è stata sviluppata con Angular, un framework open-source noto per la sua robustezza e scalabilità. Angular, integrato con Angular Material, permette di realizzare una dashboard moderna e responsive, in grado di presentare i dati in maniera chiara e immediata, facilitando l'interpretazione dei dati raccolti dai sensori.
- **AWS DynamoDB:** I dati vengono salvati in una base dati ospitata nel cloud AWS. Questa scelta non solo assicura la sicurezza e la resilienza del sistema, ma permette anche di gestire grandi volumi di dati, utili per analisi storiche e reportistica avanzata.

Scelte Progettuali e Vantaggi

La combinazione delle tecnologie sopra descritte risponde a diversi criteri chiave:

1. **Scalabilità:** Il sistema è progettato in modo modulare, permettendo l'aggiunta di ulteriori nodi o sensori senza compromettere le performance complessive. L'uso di ESP8266 e MQTT facilita l'espansione in ambienti di dimensioni maggiori o in presenza di nuove funzionalità.

2. **Affidabilità e Risposta Immediata:** L'utilizzo di Node.js per la gestione dei flussi di dati consente il monitoraggio continuo dello stato dei sensori e l'attivazione istantanea di notifiche in caso di anomalie. Grazie alla comunicazione asincrona tramite WebSocket e all'integrazione con servizi come Telegram ed email, il sistema garantisce una risposta tempestiva. Inoltre, il display locale presente su ogni board offre un ulteriore livello di supervisione, rendendo l'intero sistema autonomo e resiliente anche in caso di perdita della connettività esterna.

3. **Interoperabilità:** L'integrazione tra sensori, board ESP8266, broker MQTT e dashboard Angular crea un ecosistema in cui le informazioni fluiscono in maniera fluida e trasparente. Ciò consente non solo una gestione centralizzata ma anche la possibilità di interagire con il sistema tramite dispositivi mobili e interfacce web.

4. **Semplicità di Implementazione e Manutenzione:** Grazie alla natura modulare del sistema, eventuali aggiornamenti o manutenzioni possono essere eseguiti in maniera mirata su singoli componenti, riducendo i tempi di inattività e semplificando il troubleshooting.

Sintesi Concettuale dei Codici Arduino

Per chiarire ulteriormente il funzionamento del sistema, si riassume di seguito il comportamento dei due codici sviluppati per i nodi di rilevamento.

Codice per il Sensore Volumetrico (HC-SR04)

- **Inizializzazione:** La board ESP8266 viene inizializzata con una connessione al WiFi tramite credenziali preimpostate. Dopo la connessione, viene configurato il broker MQTT e impostati i pin del sensore HC-SR04 (TRIG e ECHO).
- **Ciclo di Lettura:** Il sensore emette un impulso di 10 µs sul pin TRIG e misura il tempo impiegato dal segnale di ritorno sull'pin ECHO. Questo tempo, moltiplicato per la velocità del suono e diviso per due, consente di calcolare la distanza.
- **Trasmissione Dati:** Il valore della distanza viene convertito in formato stringa e pubblicato sul topic MQTT "sensor/distance". Contestualmente, il display locale viene aggiornato per visualizzare il valore misurato e il numero di allarmi verificatisi.

Codice per il Sensore per Rilevamento Gas e Fumo (MQ-2)

- **Inizializzazione:** Analogamente al nodo volumetrico, la board stabilisce una connessione WiFi e configura il broker MQTT. Viene inoltre impostato il pin digitale per la lettura del segnale di soglia, mentre il pin analogico viene utilizzato per misurare la concentrazione di gas.
- **Ciclo di Lettura:** Il codice esegue la lettura simultanea del segnale analogico e di quello digitale del sensore MQ-2. I valori ottenuti vengono convertiti in stringhe e pubblicati su due differenti topic: "sensor/mq2/analog" per il dato analogico e "sensor/mq2/digital" per il dato digitale.
- **Feedback Locale:** Il display associato alla board mostra in tempo reale il valore delle letture e l'eventuale attivazione degli allarmi, garantendo un riscontro immediato all'utente.

Conclusioni

Il sistema ESIT HOME SECURITY rappresenta una soluzione integrata e innovativa per il monitoraggio e la protezione delle abitazioni. Grazie all'architettura HW-SW modulare e all'adozione di tecnologie all'avanguardia (ESP8266, MQTT, Node.js, Angular e AWS), il sistema non solo risponde alle esigenze funzionali descritte nella sezione 2.1, ma garantisce anche una notevole scalabilità, affidabilità e facilità d'integrazione.

In sintesi, ogni nodo del sistema (che si tratti del sensore volumetrico o del sensore per rilevamento gas e fumo) è dotato di una board dedicata, capace di inviare dati in tempo reale e di offrire un feedback locale tramite display. I dati raccolti vengono centralizzati tramite il broker MQTT e gestiti da Node.js, il quale assicura la corretta gestione e visualizzazione dei dati tramite una dashboard sviluppata in Angular. La base dati cloud su AWS DynamoDB rappresenta l'ultimo anello di questa catena, garantendo la memorizzazione storica e la possibilità di effettuare analisi approfondite.

Questa struttura consente non solo di reagire prontamente a situazioni di emergenza, ma anche di monitorare in maniera continuativa lo stato dell'abitazione, offrendo un sistema di sorveglianza efficace.

In conclusione, l'implementazione delle funzionalità, attraverso l'adozione di una architettura HW-SW ben definita e l'utilizzo di tecnologie avanzate, costituisce la spina dorsale del progetto ESIT HOME SECURITY, assicurando una protezione integrata e una gestione efficiente degli allarmi, sia in loco che da remoto.

Contesto, problematiche da affrontare, motivazione delle scelte progettuali svolte

Il progetto nasce dall'esigenza di realizzare un sistema integrato e scalabile, in grado di gestire e monitorare dati provenienti da sensori e di fornire un'interfaccia utente intuitiva per il controllo in tempo reale. Le principali problematiche individuate sono le seguenti:

- **Gestione e integrazione dei dati**: È fondamentale assicurare una raccolta e trasmissione dei dati affidabile, in modo che le informazioni provenienti dai sensori possano essere elaborate e visualizzate senza ritardi o perdite.
- **Scalabilità del sistema**: Il sistema deve poter espandersi facilmente per integrare ulteriori funzionalità o sensori, senza compromettere le performance complessive.
- **Usabilità e monitoraggio**: La necessità di una dashboard chiara e funzionale ha guidato la scelta di strumenti che facilitino la visualizzazione dei flussi di lavoro e l'interazione da parte dell'utente.

Le scelte progettuali adottate sono state attentemente ponderate al fine di garantire un sistema robusto, scalabile e facilmente manutenibile. Le principali tecnologie utilizzate sono le seguenti:

- **Node.js**: È stato adottato per lo sviluppo del backend per la sua natura event-driven e il supporto nativo ad operazioni asincrone, ideali per la gestione di flussi continui di dati provenienti da sensori IoT. La sua efficienza nella gestione delle connessioni WebSocket ha permesso l'implementazione di un sistema di aggiornamento in tempo reale, fondamentale per la tempestiva rilevazione e gestione degli allarmi.

- **Angular**: La scelta di Angular per il frontend è motivata dalla sua architettura modulare e dalla capacità di integrare strumenti come Angular Material, permettendo lo sviluppo di un'interfaccia utente responsive, moderna e facilmente scalabile. La dashboard consente agli utenti di monitorare in tempo reale lo stato dei sensori e di interagire con il sistema attraverso controlli intuitivi.

- **DynamoDB**: Per la persistenza dei dati è stato utilizzato Amazon DynamoDB, un database NoSQL completamente gestito, scelto per la sua scalabilità automatica e la bassa latenza. Questa soluzione ha permesso di archiviare efficientemente letture dei sensori, allarmi e informazioni sugli utenti, garantendo performance elevate anche in presenza di grandi volumi di dati.

- **TelegramBot**: Per migliorare la reattività del sistema e garantire un canale di comunicazione immediato con l'utente, è stato integrato un bot Telegram. Questo componente consente l'invio automatico di notifiche push in caso di rilevamento di situazioni di pericolo, assicurando un elevato livello di prontezza operativa anche in contesti remoti.

Validazione e sperimentazione

Per ogni funzionalità implementata, sono stati progettati specifici test funzionali con l'obiettivo di verificare l'aderenza del sistema ai requisiti definiti. In particolare:

- **Verifica della trasmissione dei dati dai sensori**: Si testerà che i dati rilevati dai sensori vengano trasmessi correttamente al backend sviluppato in Node.js. Il test dimostrerà la precisione e l'affidabilità della raccolta dati, assicurando che nessuna informazione venga persa o alterata durante il trasferimento e che i dati vengano salvati correttamente in DynamoDB.
- **Validazione dei flussi di lavoro**: Attraverso simulazioni di situazioni reali, si verificherà che i processi automatizzati implementati in Node.js (come il controllo dello stato dei sensori, il salvataggio delle letture, e l'attivazione delle notifiche) si comportino correttamente. In particolare, si testerà che l'invio delle notifiche tramite il bot Telegram avvenga puntualmente in caso di allarme.
- **Test dell'interfaccia utente**: Si eseguiranno test sull'interfaccia sviluppata in Angular per verificare la reattività e l'accuratezza della dashboard. Questi test simulano le interazioni degli utenti e controllano che la visualizzazione dei dati in tempo reale sia corretta e intuitiva.

Per valutare le performance del sistema, sono state definite le seguenti metriche chiave:

- **Latenza di Sistema**: Misurazione del tempo impiegato dal sistema per acquisire, elaborare e visualizzare i dati provenienti dai sensori.
- **Tempo di Risposta dell'Interfaccia**: Valutazione della rapidità con cui la dashboard risponde alle interazioni degli utenti e aggiorna i dati in tempo reale.
- **Throughput**: Capacità del sistema di gestire simultaneamente un elevato volume di dati e richieste, senza degradare le prestazioni.

La modalità di valutazione di queste metriche prevede l'esecuzione di test in ambienti controllati.

Manuale di riuso

5.1 Introduzione

Questo manuale fornisce istruzioni dettagliate per consentire il riuso autonomo del progetto ESIT HOME SECURITY, illustrando come riprodurre completamente l'ambiente hardware e software sviluppato. Di seguito si riportano tutte le fasi necessarie, inclusi riferimenti alle cartelle contenenti i sorgenti, gli eseguibili e le configurazioni necessarie. Include riferimenti a sorgenti, configurazioni, Lambda functions, API Gateway e setup AWS.

5.2 Requisiti Preliminari

- Account AWS con credenziali amministrative
- Schede ESP8266 (NodeMCU)
- Sensori HC-SR04 (sensore volumetrico) e MQ-2 (sensore gas e fumo)
- Ambiente di sviluppo Arduino IDE
- Node-JS installato e configurato
- Angular per il frontend

5.3 Configurazione Ambiente AWS

Creazione Resource su AWS

- DynamoDB: creare tabelle come descritto con chiavi primarie coerenti (es. timestamp).
- API Gateway: creare REST API, definire risorse /alarms, /sensor-data, /system-status; collegare metodi GET/POST alle rispettive Lambda.
- Lambda: caricare codice, assegnare variabili ambientali per DynamoDB, testare con eventi simulati.
- IAM: creare ruoli specifici per ogni Lambda con permessi PutItem, GetItem, Scan.
- IoT Core: creare Things (burglary, smoke), generare certificati, allegare policy IoT che permettano Connect, Publish, Subscribe, Receive.
- Shadow Configuration: associare Thing Shadow, definire documentazione JSON per stato desiderato, riportato e differenze.

5.3.1 DynamoDB

- Creare 4 tabelle: Alarms, SensorData, SystemStatus, Users.
- Alarms → timestamp, status, type
- SensorData → deviceId, type, value, unit, timestamp
- SystemStatus → stato generale dei dispositivi
- Users → username, email, password, role, telegramId

5.3.2 API Gateway

- Creare API REST con risorse/metodi:
- /alarms → GET, POST → Lambda getAlarms
- /sensor-data → POST → Lambda saveSensorData
- /system-status → GET → Lambda getSensorStatus

5.3.3 Lambda Functions

- getAlarms: legge ultimi 10 allarmi (ultime 6 ore), salva nuovi allarmi, tabella Alarms
- saveSensorData: salva dati sensore (SensorData) e utenti (Users)
- getSensorStatus: legge stato sistema, tabella SystemStatus

5.3.4 IAM Roles & Policies

- Creare IAM Role per ogni Lambda con policy:

```
{
"Effect": "Allow",
"Action": ["dynamodb:PutItem", "dynamodb:GetItem", "dynamodb:Scan"],
"Resource": "*"
}
```

5.4 Shadow Things AWS IoT

- Creare Things: burglary (HC-SR04) e smoke (MQ-2)
- Scaricare certificati (pem.crt, private.key, AmazonRootCA1.pem)
- Organizzarli in cartelle dedicate per dispositivo
- Creare policy IoT per publish/subscribe su topic

5.5 Configurazione Arduino

- Installare librerie necessarie: PubSubClient, ESP8266WiFi, ArduinoJson, ESP8266HTTPClient WiFiClientSecure, MQTT, time.h, Wire.h, LiquidCrystal_I2C.
- Caricare codici: hc_sr04_node.ino, mq2_node.ino, SmokeSensorAWSLast.ino, BurglarySensorAWSLast.ino.
- Integrare certificati usando WiFiClientSecure, configurando: cacert, client_cert, privkey tramite BearSSL::X509List e BearSSL::PrivateKey.
- Struttura cartelle Arduino:
  - configuration.h → contiene WiFi SSID, password, endpoint AWS, MQTT topic.
  - errors.h → definisce codici e messaggi di errore per debug locale.
  - libreriaProgetto.h → gestisce funzioni comuni.
  - Certificati → cacert, client_cert, privkey.
- Logica di funzionamento Arduino:
  - **Setup**: connessione WiFi, configurazione certificati, inizializzazione LCD.
  - **Loop**: lettura dati sensore, pubblicazione MQTT su shadow AWS, invio HTTP POST a API Gateway, gestione riconnessioni MQTT, aggiornamento display.

5.6 Backend

La directory del backend è organizzata nel seguente modo

- **/controller:** Contiene la logica principale delle operazioni lato server, come la gestione degli utenti, la lettura dei dati dai sensori, il recupero degli allarmi. Ogni file definisce le funzionalità associate a uno specifico ambito del sistema.
- **/routes:** In questa cartella sono definiti i percorsi (endpoint) REST esposti dal server. Ogni route importa il controller corrispondente e ne esegue le funzioni in risposta alle richieste HTTP provenienti dal frontend.
- **/services:** Contiene servizi riutilizzabili, come l'invio dei messaggi tramite Telegram Bot, la gestione dell'autenticazione con JWT. Sono moduli autonomi che possono essere facilmente estesi o sostituiti.
- **/. (root):** La directory principale contiene il file di avvio del server ('server.js' o 'index.js'), la configurazione dei WebSocket per l'invio in tempo reale degli aggiornamenti, il file '.env' per le variabili ambientali e i moduli principali per la connessione a DynamoDB, AWS IoT e altri servizi.

In seguito presentiamo una guida al riuso del backend.

5.6.1 Prerequisiti

Prima di procedere con la configurazione del backend, è necessario disporre dei seguenti prerequisiti:

- Node.js installato (versione consigliata >= 18)
- Account AWS con le seguenti risorse:
  - Tabelle DynamoDB: `Users`, `Alarms`, `SensorData`, `SystemStatus`
  - AWS IoT Core configurato con i dispositivi (thing) presenti nella variabile `DEVICE_LIST`
- Bot Telegram attivo con il token registrato

5.6.2 Installazione e Struttura

1. Creare la directory del progetto e inizializzarlo:

```
mkdir backend && cd backend
npm init -y
```

2. Installare le dipendenze:

```
npm install express dotenv cors axios jsonwebtoken bcrypt nodemailer
npm install @aws-sdk/client-dynamodb @aws-sdk/lib-dynamodb
npm install  @aws-sdk/client-iot-data-plane ws
```

3. Creare un file `.env` nella root del progetto:

```
PORT=1880
ACCESS_TOKEN=your_jwt_secret
TELEGRAM_API_TOKEN=your_bot_token
AWS_REGION=eu-north-1
DEVICE_LIST=esp8266-01,esp8266-02
IOT_ENDPOINT=your-iot-endpoint.iot.eu-north-1.amazonaws.com
```

4. La struttura del progetto sarà la seguente:

```
backend/
 controllers/
    sensorController.js
    userController.js
 routes/
    sensor.js
    user.js
 services/
    authentication.js
    checkRole.js
    telegram.js
 bot.js
 db.js
 index.js
 main.js
 server.js
 .env
 package.json
```

5.6.3 Configurazione Express

Il backend utilizza `Express.js` come framework principale per la gestione delle API REST. L'applicazione viene configurata per accettare richieste HTTP da qualsiasi origine grazie all'integrazione di `CORS`, e viene abilitata la lettura dei payload JSON nelle richieste in ingresso.

Le rotte sono organizzate in modo modulare e separate in due principali gruppi:

- **/user:** tutte le operazioni relative agli utenti (registrazione, login, gestione profilo, ecc.)
- **/sensor:** operazioni relative alla gestione degli allarmi e dei sensori (letture, stato sistema, storico, ecc.)

Il file principale `server.js` si occupa di avviare il server HTTP, che rimane in ascolto sulla porta specificata nel file `.env`. In questo modo l'applicazione è pronta a gestire sia le richieste REST provenienti dal frontend sia le connessioni WebSocket connesse alla trasmissione in tempo reale dei dati dei sensori.

L'architettura è pensata per essere scalabile e facilmente estendibile: è possibile aggiungere nuove funzionalità semplicemente creando nuove rotte o controller.

5.6.4 Servizi aggiuntivi

- **main.js**: Avvia il WebSocket server, recupera dati da AWS IoT Shadow e li invia ai client ogni 2 secondi.
- **bot.js**: Avvia il bot Telegram e restituisce il chatId a chi invia `/start`.
- **telegram.js**: Funzione per invio di messaggi agli utenti registrati.

5.6.5 Avvio del progetto

```
node server.js
node bot.js
```

5.6.6 Test del sistema

Per garantire il corretto funzionamento del sistema, sono stati eseguiti diversi test funzionali e di integrazione, sia manuali che automatizzati. In particolare:

- **Test di autenticazione (login/signup):** Sono stati condotti test sia tramite frontend (form Angular) che tramite strumenti come `Postman`, per verificare che la registrazione degli utenti, la protezione tramite token JWT e l'accesso ai dati personali funzionino correttamente.
- **Verifica della ricezione e visualizzazione degli allarmi:** Il sistema è stato testato simulando l'invio di dati dai sensori. È stata verificata la corretta visualizzazione degli allarmi all'interno della dashboard, oltre alla reattività in tempo reale tramite WebSocket. Sono stati testati anche i meccanismi di aggiornamento dello stato dei sensori tramite AWS IoT (Shadow).

- **Notifiche Telegram:** L'integrazione con il bot Telegram è stata validata verificando che, in caso di allarme con stato `pending`, venga inviata una notifica all'utente registrato con il relativo `telegramId`. Sono stati inoltre simulati più eventi per testare l'invio multiplo e la gestione della coda.

Questa fase di test ha permesso di individuare e correggere tempestivamente eventuali criticità, garantendo un sistema stabile, affidabile e pronto per l'utilizzo in ambienti reali.

5.7 Frontend

La directory del frontend, sviluppata con **Angular**, è strutturata in maniera modulare per facilitare la scalabilità e la manutenzione del progetto. Di seguito viene descritta la struttura principale all'interno della cartella `src/app`:

- **/house:** Contiene i moduli funzionali legati alla gestione dei sensori e degli allarmi.
  - **/antincendio:** Modulo dedicato alla gestione dei sensori di fumo/gas e agli allarmi associati.
    - `notifiche-incendio`: visualizza e gestisce gli allarmi attivi di tipo incendio.
    - `sensor-history-dialog`: finestra modale per la cronologia letture di un singolo sensore.
    - `sensori-incendio`: mostra lo stato attuale di tutti i sensori di fumo.
  - **/antieffrazione:** Modulo dedicato ai sensori di movimento e intrusione.
    - `notifiche-effrazione`: simile alla controparte incendio, mostra gli allarmi attivi di tipo volumetrico.
    - `sensori-effrazione`: visualizza i sensori effrazione attivi e i relativi valori.
  - `antincendio.component.ts / antieffrazione.component.ts`: fungono da entry point per ciascun modulo, aggregando i componenti interni.
  - `dashboard`: mostra in tempo reale lo stato di sistema (colore verde/rosso/-grigio) e consente, per utenti admin, di attivare/disattivare i sistemi.
  - `house.service.ts`: raccoglie tutte le chiamate HTTP ai sensori e agli allarmi, fungendo da layer tra il backend e i componenti.
- **/profile:** Contiene il componente di visualizzazione e modifica del profilo utente (incluso Telegram ID).
- **/shared:** Racchiude servizi e componenti riutilizzabili:
  - `alarms-websocket`: servizio che gestisce la ricezione WebSocket per gli allarmi in tempo reale.

  - `sensor-websocket`: servizio per gli aggiornamenti dei sensori in tempo reale (dashboard).
  - `global`: definizioni di costanti globali.
  - `snackbar`: servizio per notifiche rapide.
  - `token-interceptor`: intercetta le richieste HTTP per aggiungere il token JWT.
- **Componenti di autenticazione:**
  - `/auth, /login, /register, /forgot-password, /reset-password`: flusso completo di autenticazione, recupero e cambio password.
  - `route-guard`: impedisce l'accesso a pagine riservate se non autenticati.
- **Layout e navigazione:**
  - `header, sidenav, menu`: componenti responsabili dell'interfaccia grafica di navigazione (header, sidebar).
  - `spinner`: visualizzazione caricamenti tramite `ngx-ui-loader`.

Ogni modulo è stato pensato per essere indipendente e facilmente manutenibile. L'uso di servizi condivisi consente la separazione tra logica di business e presentazione, facilitando anche l'esecuzione di test unitari.

5.8 Risorse AWS (ARN)

- DynamoDB:
  - Alarms → arn:aws:dynamodb:...:table/Alarms
  - SensorData → arn:aws:dynamodb:...:table/SensorData
  - SystemStatus → arn:aws:dynamodb:...:table/SystemStatus
  - Users → arn:aws:dynamodb:...:table/Users
- Lambda:
  - getAlarms → arn:aws:lambda:...:getAlarms
  - saveSensorData → arn:aws:lambda:...:saveSensorData
  - getSensorStatus → arn:aws:lambda:...:getSensorStatus

5.8.1 Conclusione

Seguendo queste istruzioni, sarà possibile replicare integralmente il sistema ESIT HOME SECURITY, garantendo una configurazione sicura, scalabile e integrata tra hardware, backend e frontend.