

RELAZIONE DI PROGETTO DI PERVASIVE COMPUTING

Smart Evacuation

Christian Serra, Maicol Forti

Indice

1	Introduzione	7
1.1	Visione Complessiva	7
1.2	Tecnologie	8
1.2.1	Tecnologie Usate	8
1.2.2	Contributo Tecnologico Scientifico	10
2	Stato dell'arte	13
2.1	Dijkstra Algorithm Based Building Evacuation Recognition Computing and IoT System Design and Implementation	13
2.1.1	Tecnologia	13
2.2	Smart Apparatus for Fire Evacuation - An IoT based fire emergency monitoring and evacuation system	14
2.2.1	Tecnologia	14
2.3	IoT based Emergency Evacuation System	14
2.3.1	Tecnologia	15
2.4	Evacuation Supporting System Based on IoT Components	15
2.4.1	Tecnologia	15
3	Analisi dei requisiti	17
3.1	Requisiti di Business	17
3.2	Requisiti Utente	17
3.3	Requisiti Non Funzionali	18
3.4	Requisiti di Implementazione	19
3.5	Condizioni di Soddisfazione	19

4	Progettazione	21
4.1	Design Architetture	21
4.2	Scelte Rilevanti	22
4.2.1	Sito Web	22
4.3	Dati dei device	24
4.3.1	Dati Statici	24
4.3.2	Dati Dinamici	27
4.3.3	Gestione dei dati sul sito	29
4.4	Dijkstra	29
4.4.1	Definizione dei percorsi	31
4.5	Gestione dell'allarme	33
5	Implementazione	37
5.1	Sito	37
5.1.1	Login	37
5.1.2	DynamoDB	38
5.1.3	Dati dei Device	40
5.1.4	Caricamento dei dati sul sito	40
5.1.5	Mappa	42
5.1.6	Dijkstra	43
5.2	Raspberry Pi Controller	46
5.2.1	Authentication	46
5.2.2	Sensori Zigbee e Zigbee2Mqtt	47
5.2.3	Register new devices to DynamoDB	49
5.2.4	SensorController	49
5.2.5	Sensori Simulati	51
5.2.6	Allarme	51
6	Testing e performance	53
6.1	Funzionamento	53
6.2	Testing	54
6.2.1	Caso di Studio	54
6.3	Performance	55
6.3.1	Sito Web - Node.js	55

<i>INDICE</i>	5
7 Analisi di deployment su larga scala	57
7.1 Il sito web	57
7.2 Locale	57
8 Piano di lavoro	61
8.1 Divisione dei compiti	61
8.2 Piano di lavoro adottato	61
8.2.1 Versioning	62
8.2.2 Trello	62
8.2.3 Toggl	62
9 Conclusioni	63
Appendice	64
Riferimenti bibliografici	65

Capitolo 1

Introduzione

1.1 Visione Complessiva

In situazioni di emergenza indoor, come ad esempio un incendio all'interno di un edificio, avere informazioni in tempo reale riguardo la posizione dell'incendio e le possibili vie di fuga è fondamentale per permettere alle persone coinvolte di salvarsi.

Il progetto **Smart Evacuation** ha come obiettivo quello di controllare e gestire le evacuazioni indoor in caso di emergenza. Il sistema è suddiviso in:

- **Sistema di Controllo nell'Edificio:** sarà presente un controllore che raccoglie i dati dalla sensoristica e li rende disponibili al cloud. Questo controllore si occupa dell'attivazione della procedura di allarme, la quale può essere avviata automaticamente, dalla lettura di dati anomali, oppure manualmente, tramite pulsanti fisici nell'edificio. Il controllore gestirà l'evacuazione calcolando il percorso di fuga ottimale con l'algoritmo di Dijkstra [2, 5], applicato in real time in base alle condizioni delle vie di fuga. Per quanto riguarda la sensoristica, il sistema verrà sviluppato tenendo aperta la possibilità di aggiungere una varietà di sensori di diverso tipo (Fumo, Temperatura, Umidità...), con supporto a vari protocolli di comunicazione (principalmente Zigbee, ma sarà possibile anche con Wi-fi e Bluetooth).
- **Cloud:** la parte cloud raccoglierà le informazioni dall'edificio che verranno poi visualizzate su un sito web e permetterà di interagire con esse e di

conseguenza con i device IoT. In particolare:

- le informazioni relative allo stato della sensoristica verranno raccolte e rese disponibili.
- sarà possibile visualizzare e interagire con una mappa dell’edificio, che mostrerà in tempo reale i device e le vie di fuga.
- sarà inoltre possibile disattivare l’allarme con un codice dal sito web.

Si valutano eventuali estensioni:

- gestire il controllore come un sistema decentralizzato, per renderlo più resiliente.
- utilizzare device mobili per implementare una forma di crowd control e suggerire percorsi migliori.

1.2 Tecnologie

1.2.1 Tecnologie Usate

Le tecnologie utilizzate all’interno del progetto sono:

- **Cloud:**
 - **AWS IoT Core:** un servizio cloud che permette la connessione dei propri device IoT ad altri device ed ai servizi cloud messi a disposizione da AWS.
 - **EC2:** Amazon Elastic Compute Cloud è un servizio web che fornisce capacità di elaborazione sicura e scalabile nel cloud.
 - **Dynamo DB:** è un database che supporta i modelli di dati di tipo documento e di tipo chiave-valore che offre prestazioni di pochi millisecondi a qualsiasi scala, è veloce e flessibile ed è erogato come servizio
 - **AWS Lambda (Node.js):** un servizio che permette di eseguire codice per qualsiasi tipo di applicazione o servizio di back-end, senza alcuna amministrazione.

- **Website:**

- **Stack MEVN / DEVN**

- * **MongoDB:** MongoDB è un database distribuito, document-based, per uso generico.
 - * **DynamoDB:** è un database che supporta i modelli di dati di tipo documento e di tipo chiave-valore che offre prestazioni di pochi millisecondi a qualsiasi scala, è veloce e flessibile ed è erogato come servizio.
 - * **Express:** un framework per applicazioni web per Node.js, open source sotto Licenza MIT.
 - * **Vue.js:** un framework progressivo per costruire interfacce utente.
 - * **Node.js:** un runtime JavaScript costruito sul motore JavaScript V8 di Chrome.
 - **Client:** per lo sviluppo del client abbiamo fatto uso di HTML, SCS-S/CSS, JS, Vue+Axios, Bootstrap.
 - **MazeMap:** un servizio per mappe indoor.
 - **Passport.js:** un middleware di autenticazione per Node.js.
 - **aw4:** una utility che permette di fare richieste HTTP(S) utilizzando il processo di firma Signature Version 4

- **Locale**

- **Raspberry Pi**

- * **Lato Software**

- **Scala backend** Applicazione scritta in Scala che si mette in ascolto degli aggiornamenti dei vari sensori (simulati e Zigbee) e che gestisce le Shadows di AWS IoT.
 - **AWS IoT Device SDK - Java** Una libreria che agevola la comunicazione con le entità di AWS IoT aggiungendo un livello di astrazione al semplice protocollo MQTT.
 - **Maven** Strumento utilizzato per la gestione del progetto di backend. Utile per gestione delle dependencies, testing e deployment

- **Paho** Client MQTT locale utilizzato nel backend per leggere i messaggi inviati dall'applicazione Zigbee2Mqtt
- **Zigbee2Mqtt** Applicazione standalone che permette, attraverso una router fisico connesso al Raspberry Pi, di creare una rete mesh Zigbee e comunicare con i device connessi alla rete. Lo scambio di messaggi avviene tramite un canale MQTT esterno
- **Deployment con Docker e docker-compose** Utilizzati per favorire lo sviluppo su una piattaforma differente senza avere ulteriori problemi di deployment su Raspberry Pi.
- **Mosquitto** Message broker che implementa il protocollo MQTT. I suoi canali vengono utilizzati da backend e MQTT

* **Lato Hardware**

- **Raspberry Pi 3b+** Famoso SoC utilizzato per le sue dimensioni ridotte, per i consumi e per la facilità di sviluppo.
- **CC2531** Altro SoC utilizzato come router Zigbee. Connesso via USB, comunica con il Raspberry Pi per gestire i devices Zigbee connessi ad esso. Prima di poter essere utilizzato necessita di essere flashato con il giusto firmware per comunicare con l'applicazione Zigbee2Mqtt
- **Honeywell Smoke Detector** Sensore di fumo dotato di allarme sonoro che può inviare messaggi di stato attraverso il protocollo Zigbee

1.2.2 Contributo Tecnologico Scientifico

Il sistema risultante è un primo prototipo funzionante di un possibile sistema di gestione delle evacuazioni, che sfrutta le tecnologie IoT e Cloud. Non sono ancora in vendita sul mercato soluzioni di questo tipo, ma sono discusse in letteratura, in quanto rappresentano una direzione concreta per il futuro delle città intelligenti. Il sistema fornisce:

- **Cloud:**

- **AWS IoT Core**: un sistema di gestione e monitoraggio dei dati dei device fisici necessari al controllo dell'edificio, raccolti in shadow, un servizio che permette di rendere disponibile lo stato dei device ad altre applicazioni e servizi.
- **Website**: un sito web dotato di: homepage di presentazione, login, homepage utente, pagina di visualizzazione di una mappa interattiva, pagina di gestione dei device. Abbiamo adottato un design responsivo e mobile-first, per permetterne l'utilizzo su dispositivi mobili. Questo sito è hostato su EC2, comunica con le device shadow di IoT Core e fa utilizzo di DynamoDB.
 - **Stack "DEVN"**: per permettere un'integrazione automatica dei nostri componenti, abbiamo deciso di usare DynamoDB al posto di MongoDB. Sfruttando **dynamoose**, un tool di modellazione simile a mongoose, abbiamo potuto cambiare il DB di riferimento del progetto cambiando poche righe di codice. Questo DB viene utilizzato per la gestione dei login e per mantenere uno storico dei device, che possono comunicare in modo semplice con Dynamo grazie ad AWS.
- **Locale**: all'interno dell'edificio sono presenti dei device geolocalizzati necessari al monitoraggio e il controllo dell'edificio.
 - **Controllore**: un RPi si occupa di raccogliere i dati dei device a lui collegati attraverso diversi protocolli per poi caricarli su AWS IoT Core. Inoltre gestisce le situazioni di emergenza, chiamando l'algoritmo di Dijkstra per il calcolo delle vie di fuga e rendendole disponibili su AWS Iot Core.

Capitolo 2

Stato dell'arte

2.1 Dijkstra Algorithm Based Building Evacuation Recognition Computing and IoT System Design and Implementation

In [5] viene presentata una soluzione per l'evacuazione dai complessi edifici della Cina in caso di incendio.

L'approccio è affine al nostro: le tecnologie dell'Internet Of Things (IoT) vengono utilizzate per migliorare la percezione del fuoco e del fumo negli edifici, in aggiunta un set di algoritmi basati sul metodo di calcolo del percorso più breve di Dijkstra, viene progettato per funzionare sui terminali di computazione FPGA locali e calcolare il percorso di evacuazione ottimale. Il percorso viene indicato da un sistema di luci apposite.

2.1.1 Tecnologia

Rispetto al nostro progetto ci sono alcune differenze tecnologiche:

- Il processore che viene utilizzato è un FPGA, che si occupa di effettuare image recognition delle fiamme, di analizzare gli allarmi del fumo e calcolare il percorso ottimale.
- Il sistema di indicazione è implementato usando LoRa (low-power wireless communication).

2.2 Smart Apparatus for Fire Evacuation - An IoT based fire emergency monitoring and evacuation system

L'obiettivo descritto in [3] è quello di procurare agli occupanti e ai servizi di emergenza informazioni riguardo la locazione dell'incendio e fornire in tempo reale un percorso di evacuazione sicuro.

2.2.1 Tecnologia

Diversamente dal nostro progetto, viene proposto l'utilizzo di una rete mesh di allarmi antincendio intelligenti e algoritmi di pianificazione dei percorsi.

- Meshed Sensor Network: ogni modulo di allarme ha 4 componenti: un sensore per la rilevazione del fuoco, un microprocessore, un power supply e un ricetrasmittitore wireless. Questo stile permette ai ricetrasmittitori RF a medio raggio di comunicare efficientemente anche a lunga distanza. Alcune opzioni per la trasmissione RF sono (Bluetooth, BLE, Wi-Fi). In particolare sono stati scelti i "nRF24L01 transceivers (2.4GHz)".
- Central Hub e Path Planning: una server workstation viene utilizzata per raccogliere tutti i dati e calcolare la via di fuga. Per il prototipo è stato scelto un nRF24L01 transceiver connesso ad un Arduino UNO collegato ad un pc portatile.
- User Interface: viene mostrato in real-time su un sito web una immagine raffigurante il percorso d'uscita in base alla posizione.

2.3 IoT based Emergency Evacuation System

In [4] viene presentato un Emergency Evacuation System (EES) basato su Internet Of Things.

L'EES si occupa di rilevare le emergenze e allertare i civili nell'edificio, fornendo la propria guida attraverso una applicazione.

2.3.1 Tecnologia

L'architettura del sistema è fatta da 6 componenti: Arduino Mega, sensori, LED, server per processare dati, applicazione per fare da guida e definire i percorsi e il modulo GSM per gli alert.

Non è descritto un vero e proprio algoritmo per il calcolo del percorso, si sceglie tra i percorsi uno in cui non viene superato il livello di allarme.

2.4 Evacuation Supporting System Based on IoT Components

In [1] viene presentato un sistema di supporto all'evacuazione (EES) outdoor, che utilizza dispositivi IoT distribuiti nella città per guidare le persone verso le aree sicure.

2.4.1 Tecnologia

Viene proposta una organizzazione gerarchica a due livelli, dove l'Emergency Operation Center (EOC) è al comando. Al di sotto un sistema distribuito sfrutta il paradigma peer-to-peer (P2P) ed è costituito da nodi speciali, le Witness Units (WU). Queste unità computano i percorsi di evacuazione, comunicano con l'EOC e funzionano come access point per gli utenti. Altri nodi sono i sensori, anche essi distribuiti nell'area per monitorare le condizioni generali dei percorsi.

La rete è basata su LoRa.

Viene fornita inoltre una applicazione mobile per accedere alle WU e ai loro servizi. Per poter distribuire in modo corretto le WU viene utilizzato un algoritmo genetico.

Capitolo 3

Analisi dei requisiti

3.1 Requisiti di Business

- I dati dei device presenti all'interno dell'edificio dovranno essere monitorabili, e dovranno essere resi disponibili ai servizi e le applicazioni che ne possono avere necessità.
- Le situazioni di allarme dovranno essere in primo luogo identificate, in modo da fornire le informazioni utili agli utenti per poter evacuare, poi terminate manualmente quando ritenuto necessario.
- Durante le situazioni di emergenza, sarà necessario definire una via di fuga sicura per gli utenti del sistema. Sarà quindi richiesta una soluzione per il calcolo di una via di fuga ottimale in tempi brevi, in modo da permettere la corretta evacuazione dell'edificio.
- Dovrà essere resa disponibile una soluzione per il monitoraggio dei device IoT che sia intuitiva e funzionale, come ad esempio una visualizzazione su mappa.

3.2 Requisiti Utente

Per avere maggiore chiarezza si è deciso di raccogliere i requisiti funzionali e utenti sotto forma di user story. Ognuna di queste avrà un nome che la definisce e sarà

correlata di tre informazioni: l'utente a cui ci si rivolge, il bisogno da soddisfare e il motivo per cui soddisfarlo. Individuiamo Jon come utente, un normale cittadino che si trova all'interno dell'edificio per lavoro.

- Nome: *Registrazione di un account*
Storia: Come Jon, voglio poter registrare un account, così da poter effettuare il login.
- Nome: *Login e Logout*
Storia: Come Jon, voglio poter effettuare il login nell'applicazione, così da poterla utilizzare. Come Jon, voglio poter effettuare il logout dall'applicazione.
- Nome: *Visualizzazione su mappa*
Storia: Come Jon, voglio poter visualizzare una mappa interattiva, così da avere una visione globale dell'edificio a colpo d'occhio.
- Nome: *Via di fuga*
Storia: Come Jon, voglio poter visualizzare una via di fuga sulla mappa quando si presenta una situazione di emergenza, così da poter uscire dall'edificio in modo sicuro.
- Nome: *Lista dei device*
Storia: Come Jon, voglio poter visualizzare una lista dei device IoT, così da poterli monitorare.
- Nome: *Device su mappa*
Storia: Come Jon, voglio poter visualizzare i device geolocalizzati sulla mappa, in modo da monitorarli rendendomi conto della loro posizione.
- Nome: *Gestione dell'allarme*
Storia: Come Jon, voglio poter inserire un codice di allarme, così da disattivare la situazione di allarme quando ritenuto opportuno.

3.3 Requisiti Non Funzionali

- La piattaforma dovrà essere disponibile 24/7, garantendo che i servizi offerti siano sempre raggiungibili.

- Il software dovrà essere facilmente scalabile, ad esempio adottando soluzioni cloud.
- Il software dovrà essere mantenibile, in particolare le operazioni dovranno essere ben divise, così da permettere modifiche mirate in caso di problemi.
- I costi dovranno essere limitati il più possibile: ad esempio, se si usano soluzioni che si pagano in base all'uso, questo dovrà essere limitato al necessario.

3.4 Requisiti di Implementazione

I requisiti di implementazione servono a garantire la produzione di software di qualità:

- Il sito dovrà essere:
 - Responsivo e Mobile-First, in modo da potersi adattare ed essere utilizzato senza problemi su ogni dispositivo mobile, a prescindere dallo schermo.
 - Dovranno essere adottati il principio KISS e Less is More: l'interfaccia deve essere semplice con solo il necessario per il funzionamento.
 - Dovrà utilizzare elementi grafici di interazione riconoscibili, ad esempio l'icona di un bottone deve facilitare la comprensione del suo funzionamento.
- Il codice dovrà essere testato prima di essere accettato.
- Tutti i membri del team dovranno controllare il codice prodotto dagli altri e revisionarlo.

3.5 Condizioni di Soddisfazione

Alcuni ulteriori requisiti sono stati definiti come condizioni di soddisfazione. Per ognuno di essi viene definita la condizione e il target.

- Rispetto della schedula: il progetto dovrà essere consegnato, entro e non oltre il 15 febbraio 2021.

- Consegna: la consegna dovrà comprendere: il codice completo del progetto, una relazione ben strutturata, delle slide di presentazione ed un video demo.
- Comunicazione: i membri del team dovranno comunicare almeno una volta alla settimana per aggiornarsi sullo stato del progetto.
- Usabilità: il software dovrà essere di semplice utilizzo anche per i nuovi utenti, adottando un design semplice e intuitivo.
- Desirable: il design dovrà essere piacevole.
- Findable: il contenuto dell'applicativo dovrà essere semplice da navigare, a prescindere dal dispositivo.

Perché il progetto sia considerato di successo, i requisiti di business dovranno essere concretizzati e le condizioni di soddisfazione dovranno essere rispettate.

Sarà responsabilità del team di sviluppo accertarsi che lo siano.

Capitolo 4

Progettazione

4.1 Design Architetturale

L'architettura complessiva del sistema è visibile in Figura 4.1.

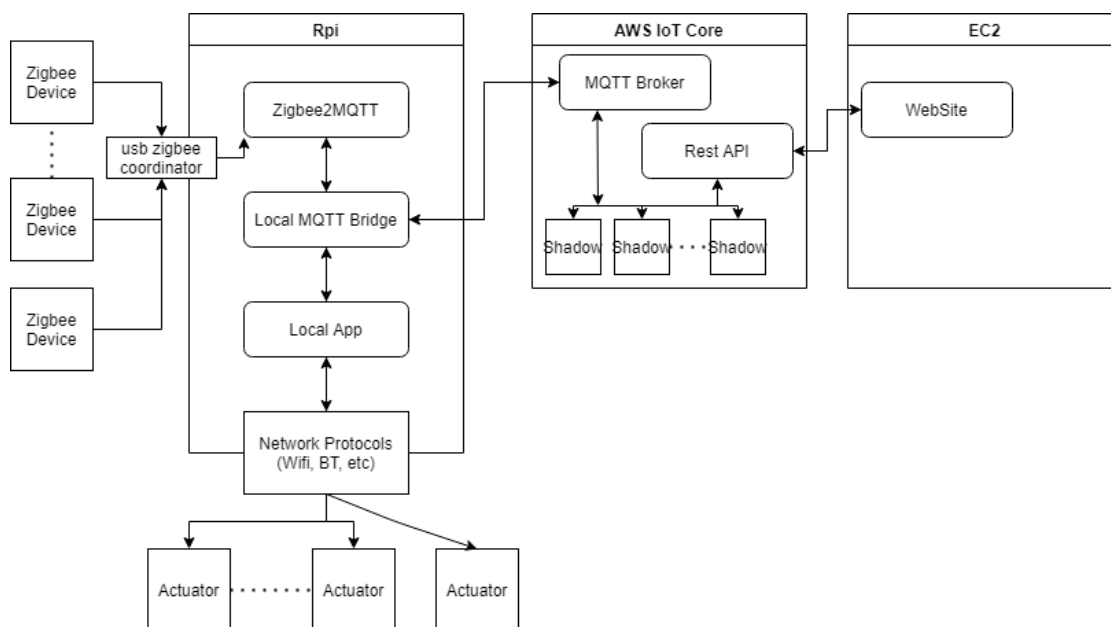


Figura 4.1: Architettura del sistema

4.2 Scelte Rilevanti

4.2.1 Sito Web

Per quanto riguarda il sito web si è deciso di adottare inizialmente lo **stack ME-VN**, su cui possediamo già delle competenze. In seguito, per integrare il sito ad AWS, MongoDB è stato sostituito con **DynamoDB**, in modo da rendere l'integrazione tra i vari componenti del sistema più veloce e automatica. Questo cambiamento è stato fatto in modo semplice utilizzando al posto di mongoose, **dynamoose**: uno strumento di modellazione per DynamoDB che permette di utilizzare degli schemi di dati simili a quelli di mongoose e di effettuare le transazioni e le query sul db.

Organizzazione del codice

Il sito è stato diviso in 3 cartelle, vedi anche figura 4.2:

- **public**: contenente i file css, js e le risorse utili come ad esempio le immagini.
- **src**: contenente tre sotto cartelle:
 - **controllers**: in cui si trovano il file di configurazione, il validatore per gli input dell'utente e il controller, che si occupa di gestire le richieste degli utenti.
 - **models**: contenente gli schemi dei dati da recuperare dal DB.
 - **routes**: contenente le route del progetto e le relative richieste da fare al controller.
- **www**: contenente le pagine HTML.

Istanza EC2

Abbiamo deciso di hostare il sito web su una istanza EC2, e di fornire a quell'istanza un identificatore di default per un edificio. In questo modo, una istanza di EC2, permetterà l'accesso agli utenti ai dati di un determinato edificio e dei suoi things.

In questo modo per gestire edifici diversi potranno essere create altre istanze di EC2, garantendo la scalabilità necessaria.

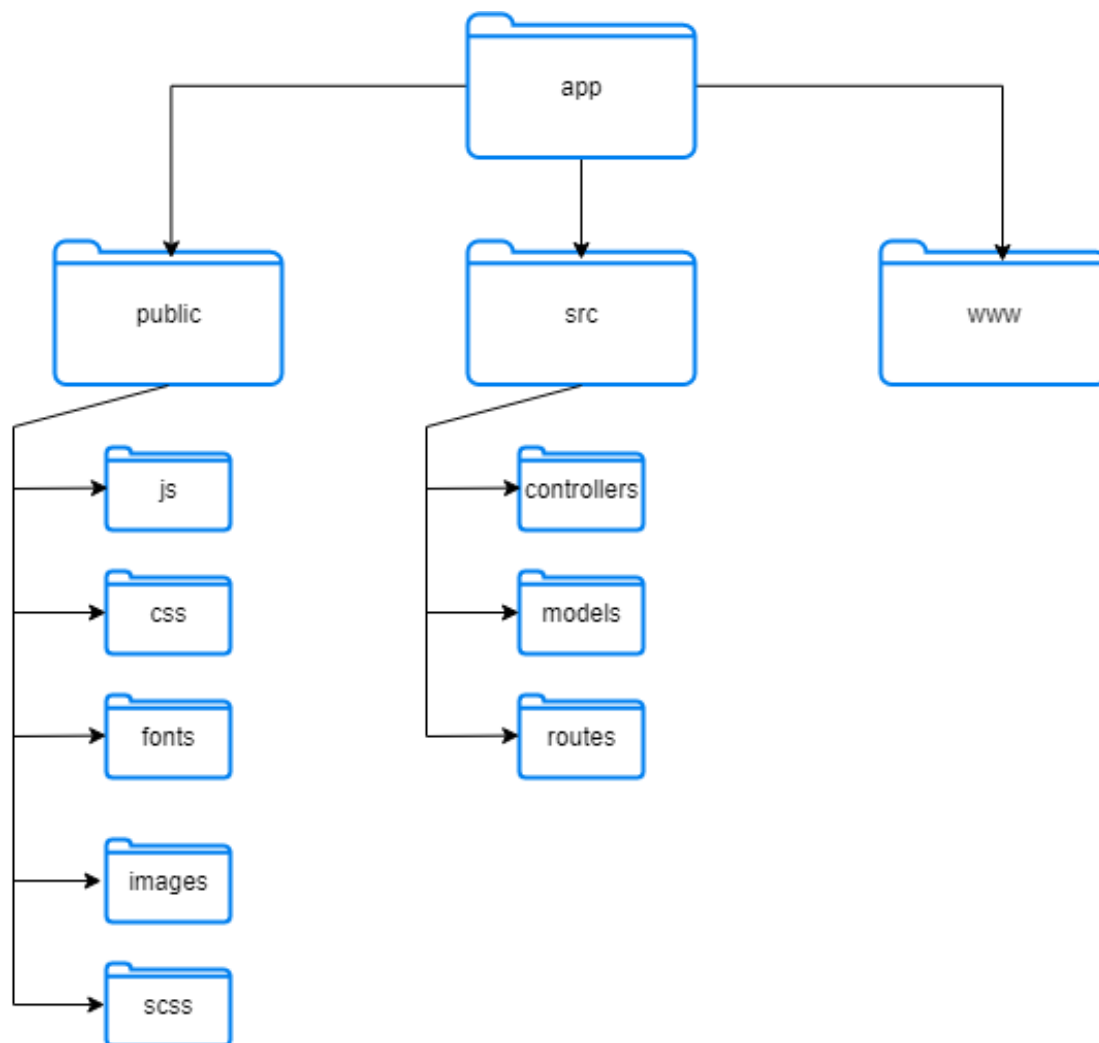


Figura 4.2: Organizzazione del codice website

4.3 Dati dei device

Un problema da affrontare è stato la gestione dei dati dei device presenti nell'edificio. Ogni device possiede due tipologie di dati:

- **Statici:** dati che non cambiano, o se cambiano succede raramente, come ad esempio la posizione del device e l'edificio a cui appartiene.
- **Dinamici:** dati che vengono aggiornati in modo continuo, come ad esempio il valore di fumo rilevato di uno smoke detector.

4.3.1 Dati Statici

Per quanto riguarda i dati statici si è deciso di caricarli su DynamoDB, in quanto sono facilmente reperibili al bisogno grazie alla flessibilità di accesso a DynamoDB attraverso chiave-valore. In particolare su di esso vengono salvati tutte le definizioni dei sensori appartenenti ai diversi edifici, come mostrato in figura 4.3. Nelle definizioni sono presenti dati relativi al modello, tipo di sensore, protocollo di comunicazione, alimentazione. In più, ogni sensore ha una mappa "exposes" che indica quali sono i dati che è in grado di rilevare, vedi figura 4.4.


```

▼ things Map {1}
  ▼ zigbeeCoordinator Map {1}
    ▼ devices Map {8}
      ► 0x00158d0003467624 Map {20}
      ► 0x10158d0003467624 Map {20}
      ▼ 0x20158d0003467624 Map {20}
        connection String : zigbee
        date_code String : 20170314
        description String : MiJia Honeywell smoke Detector
        ► exposes Map {8}
          friendly_name String : 0x20158d0003467624
          ieee_address String : 0x20158d0003467624
          img_path String : https://api.iconify.design/mdi-smoke-detector.svg
          interview_completed Boolean : true
          interviewing Boolean : false
          lat Number : 38.03100233465469
          long Number : -78.4985754560354
          model String : JTYJ-GD-01LM/BW
          model_id String : lumi.sensor_smoke
          network_address Number : 37965
          power_source String : Battery
          software_build_id String : 3000-0001
          supported Boolean : true
          type String : smoke detector
          vendor String : Xiaomi
          zFloor Number : 1
        ► A Map {18}
        ► B Map {18}
        ► C Map {18}
        ► Exit Map {18}
        ► Start Map {18}

```

Figura 4.3: Shadows Registrare su DynamoDB

```
▼ exposes Map {4}
  ▼ battery Map {8}
    access Number: 1
    description String: Remaining battery in %
    name String: battery
    property String: battery
    type String: numeric
    unit String: %
    value_max Number: 100
    value_min Number: 0
  ▼ battery_low Map {7}
    access Number: 1
    description String: Indicates if the battery of this device is almost empty
    name String: battery_low
    property String: battery_low
    type String: binary
    value_off Boolean: false
    value_on Boolean: true
  ▼ smoke Map {7}
    access Number: 1
    description String: Indicates whether the device detected smoke
    name String: smoke
    property String: smoke
    type String: binary
    value_off Boolean: false
    value_on Boolean: true
  ▼ smoke_density Map {4}
    access Number: 1
    name String: smoke_density
    property String: smoke_density
    type String: numeric
```

Figura 4.4: Dati pubblicati da un sensore simulato

4.3.2 Dati Dinamici

I dati dinamici, tra cui quelli rilevati dai sensori e il percorso ottimale di fuga, sono rappresentati su AWS attraverso le Shadows di AWS IoT. Questa decisione è dovuta alla possibilità di accederci sia tramite API REST che con protocollo MQTT, permettendo in questo modo di sfruttare il pattern di comunicazione Publish/Subscribe.

Per quanto riguarda i dati dei sensori, provenienti dal Raspberry Pi, si è deciso di utilizzare più Named Shadows associate ad un'unica Thing. Una Named Shadow è semplicemente una Shadow associata ad una Thing e riferita per l'identificativo. Mentre una Thing può avere solo una classica (unnamed) Shadow, ci possono essere multiple Named Shadow per Thing. Questo permette di ridurre il numero di Things e raggruppare logicamente i sensori, vedi figura 4.5.

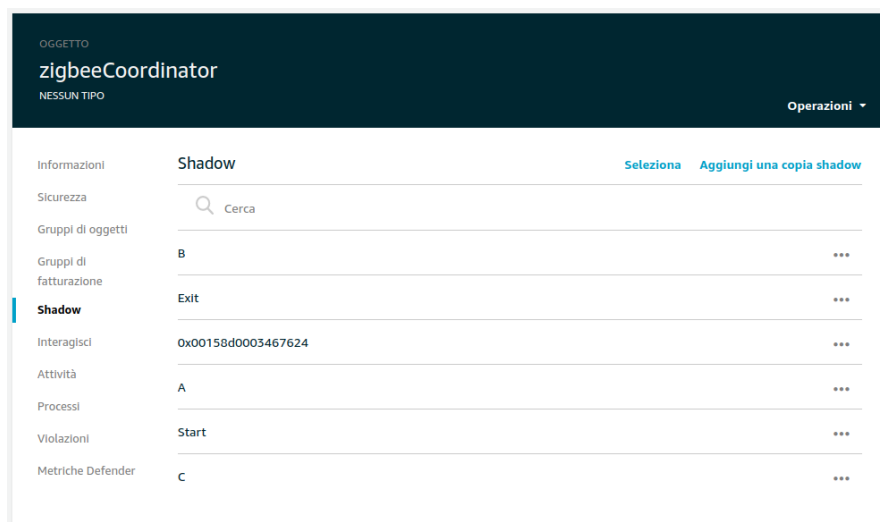
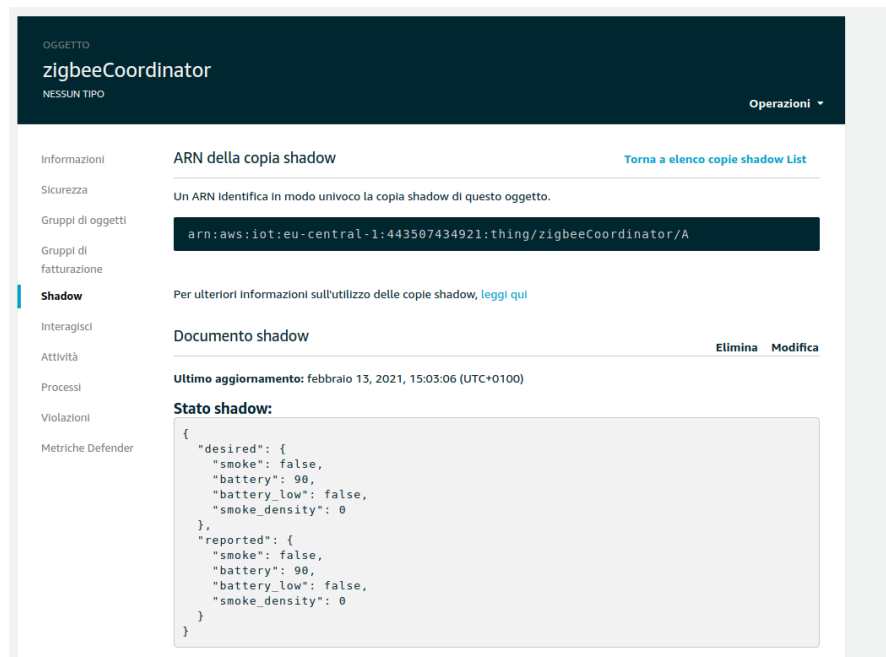


Figura 4.5: Named Shadows presenti nella Thing del Raspberry Pi

Ogni Shadow contiene uno stato "Reported" che raggruppa i dati provenienti dal sensore fisico e uno stato "Desired" che può essere modificato da altri applicativi. Ogni cambiamento allo stato "Desired" genera, tra gli altri, un evento "Shadow-Delta" che viene utilizzato per far richiedere al sensore delle modifiche sui propri dati. Vedi un esempio in figura 4.6



The screenshot displays the AWS IoT console interface for a specific object named **zigbeeCoordinator**. The left sidebar contains a navigation menu with options like Informazioni, Sicurezza, Gruppi di oggetti, and Shadow. The main content area is titled **ARN della copia shadow** and includes a link to [Torna a elenco copie shadow List](#). It explains that an ARN uniquely identifies a shadow copy and shows the ARN: `arn:aws:iot:eu-central-1:443507434921:thing/zigbeeCoordinator/A`. Below this, there's a section for **Documento shadow** with links to [Elimina](#) and [Modifica](#). It also shows the **Ultimo aggiornamento** as February 13, 2021, at 15:03:06 (UTC+0100). The **Stato shadow:** section displays a JSON object representing the shadow's state.

```
{
  "desired": {
    "smoke": false,
    "battery": 90,
    "battery_low": false,
    "smoke_density": 0
  },
  "reported": {
    "smoke": false,
    "battery": 90,
    "battery_low": false,
    "smoke_density": 0
  }
}
```

Figura 4.6: Esempio di stato di una Named Shadow

Anche l'allarme viene gestito come una Thing con una sua (classic) Shadow, dentro semplicemente viene rappresentato lo stato dell'allarme (true/false) e il percorso di fuga ottimale, dato un punto di partenza.

4.3.3 Gestione dei dati sul sito

Per quanto riguarda il sito web, i dati vengono ottenuti in due fasi (vedi figura 4.7:

- Vengono caricati da DynamoDB i dati statici relativi ai device.
- In base a questi dati, quando necessario, vengono richiesti i dati in tempo reale alle shadow di AWS IoT Core, di particolare rilevanza è lo stato reported.

Le richieste vengono fatte tramite API REST e sono autenticate con Signature Version 4.

Si è deciso di utilizzare le API REST per poter disaccoppiare il tempo di aggiornamento di una shadow da quello di aggiornamento del sito. In questo modo si può gestire il numero delle richieste separatamente, a differenza di MQTT che riceve dati ogni volta che le shadow sono aggiornate.

4.4 Dijkstra

Per quanto riguarda il calcolo delle via di fuga abbiamo deciso di utilizzare l'algoritmo di Dijkstra. (Vedi 5 per il dettaglio sull'algoritmo)

L'algoritmo richiede in input una sorgente, una destinazione, ed infine una lista dei nodi del grafo, che abbiamo deciso di rappresentare come segue:

```
1 final case class Path(vertex: String, from: String, weight: Int,  
    label: Boolean)  
2  
3 val nodeList: ListBuffer[Path] = mutable.ListBuffer[Path](...)
```

Lista di Path, dove un Path contiene: il vertice, un nodo a cui è collegato, il peso/distanza da quel nodo, un valore booleano che serve a identificare se il vertice è già stato etichettato o meno.

Queste informazioni vengono preparate dal controllore che poi si occupa di fornire il risultato di Dijkstra ad AWS IoT Core, da cui il sito web può recuperarli per mostrarli.

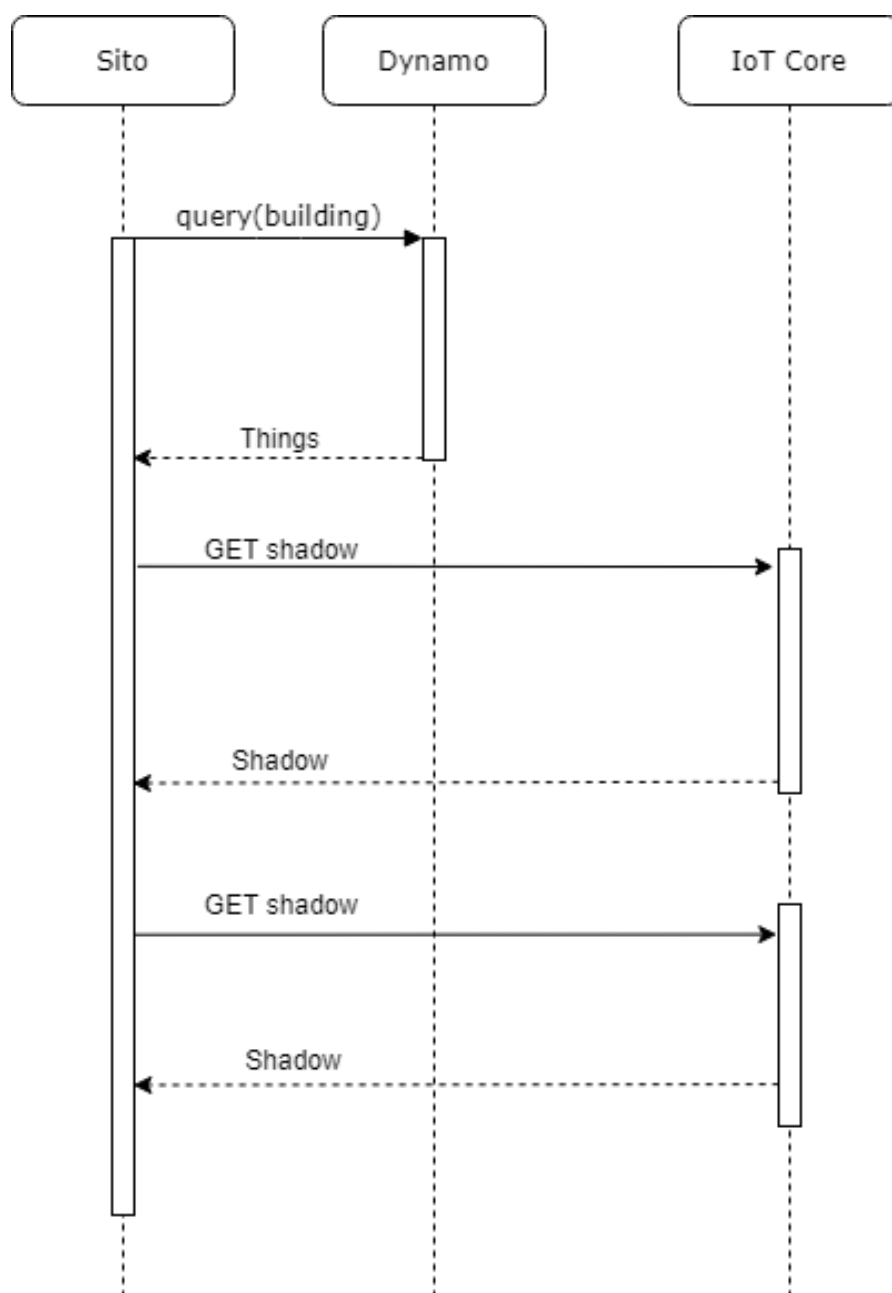


Figura 4.7: Sequenza di ottenimento dati

4.4.1 Definizione dei percorsi

La nostra scelta è stata, per semplicità del prototipo, quella di far coincidere un percorso come il collegamento tra la posizione di due device. I vertici del grafo di Dijkstra saranno di conseguenza le posizioni dei device, di cui possediamo la posizione in coordinate geografiche all'interno delle relative shadow, e il percorso finale la somma dei collegamenti tra le coppie di shadow che portano da un punto di partenza a uno di arrivo. Dijkstra lavorerà su distanze definite come numeri interi positivi, proporzionati alla distanza reale tra i device, vedi Figura 4.8. A Dijkstra saranno fornite solamente le informazioni riguardo ai device che non rilevano stati anomali, in modo tale da effettuare l'algoritmo solo sui percorsi sicuri. Dijkstra verrà eseguito in modo continuo per ricalcolare in tempo reale il percorso sui dati aggiornati.

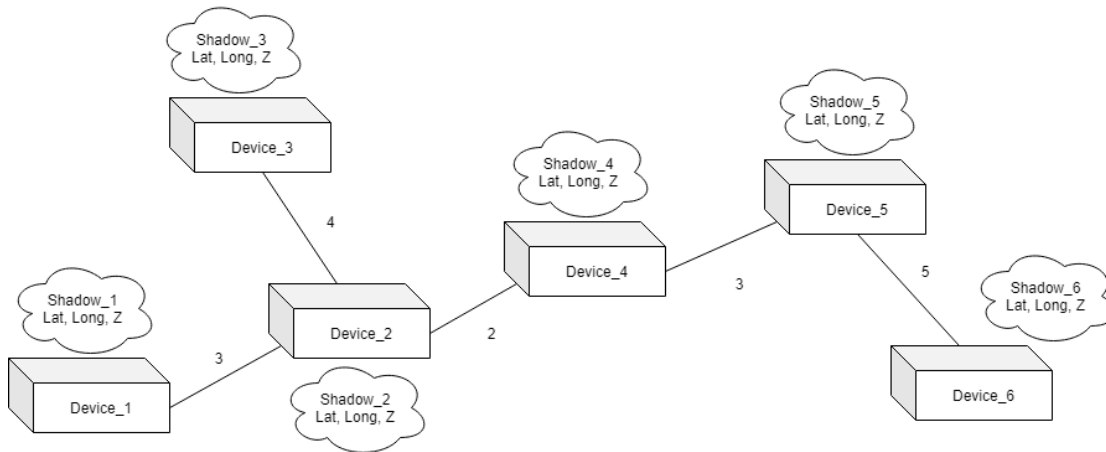


Figura 4.8: Grafo dei device shadow

Come mostrato in figura 4.9, nella soluzione attuale, ogni cerchio rappresenta un device, e i percorsi sono determinati come collegamento tra i device.

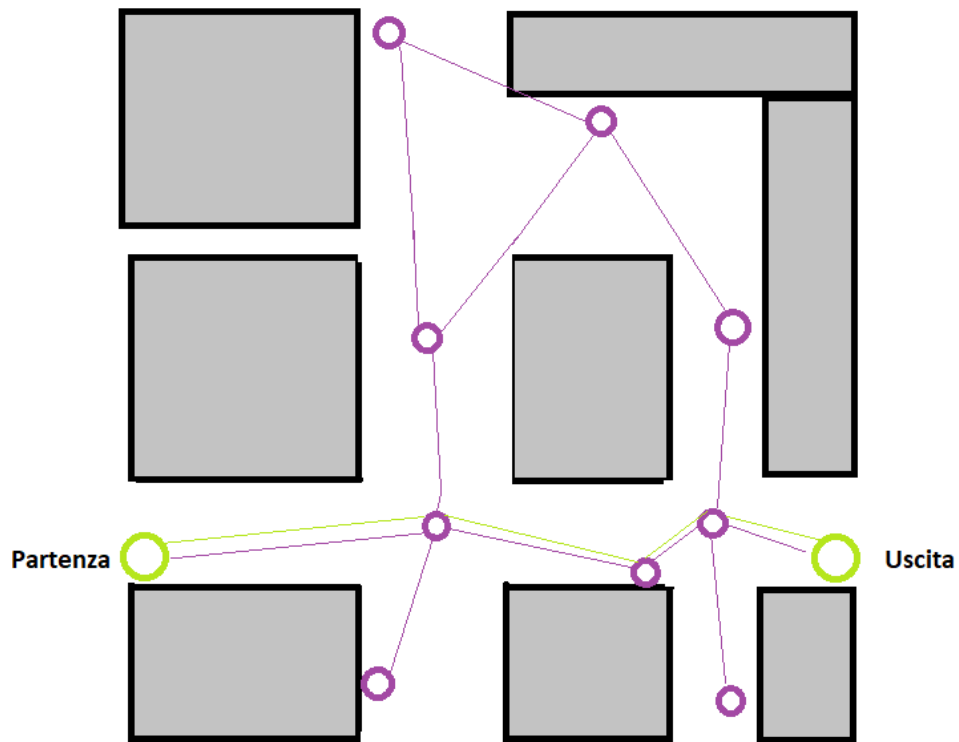


Figura 4.9: Rappresentazione con device

Questa soluzione è stata adottata in quanto l'obiettivo era fornire una proof of concept del sistema, ma in un sistema reale, potrebbe essere adottato un modello più astratto e potente.

La nostra idea sarebbe quella di definire il concetto di "Path", che racchiude al suo interno ad esempio: tutti i device (sensori, attuatori, etc) che si occupano di monitorare e gestire una determinata zona di un edificio, una lista delle persone presenti nella zona, altro. L'insieme delle rilevazioni fornisce la possibilità di determinare se quella zona identificata dal Path sia sicura. In questo modo un vertice del grafo di Dijkstra sarebbe un Path, e sarebbe possibile ad esempio utilizzare le informazioni sulla quantità di persone nelle zone per poter pesare il grafo e distribuire i percorsi.

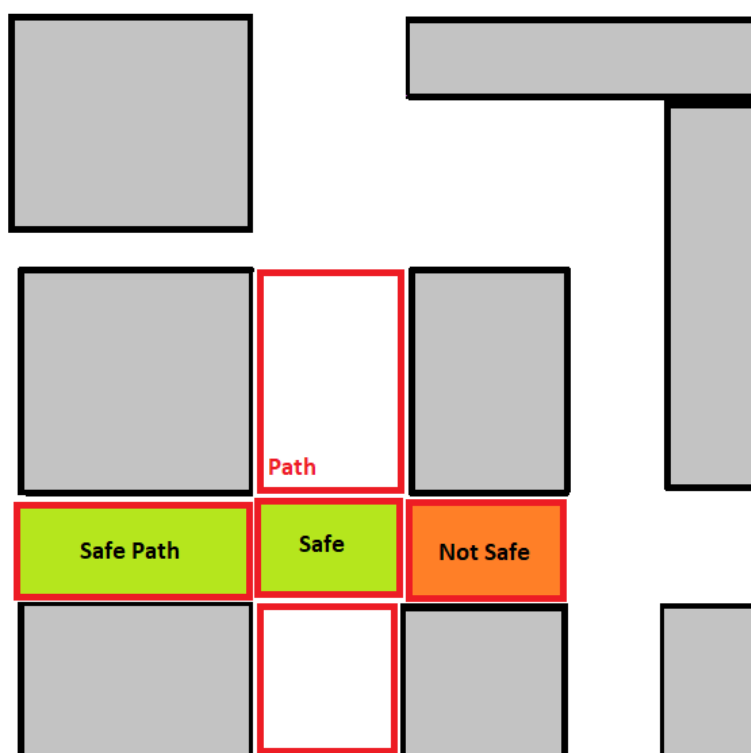


Figura 4.10: Rappresentazione con Path

4.5 Gestione dell'allarme

Per permettere una gestione rapida dell'allarme, abbiamo deciso di definire una shadow su AWS IoT Core relativa all'edificio. Questa shadow mantiene i valori utili alla gestione dell'allarme, come ad esempio un valore che definisce se l'allarme è attivo o spento, e i percorsi di fuga.

Quando il controllore rileva uno stato di allarme, aggiorna la shadow con lo stato di allarme positivo e calcola il percorso di fuga. Quando nel sito web ci si trova nella pagina della mappa, sarà prelevato periodicamente lo stato della shadow relativa all'allarme, in attesa di una situazione di emergenza, vedi Figura 4.11. Quando si presenta, viene utilizzato il valore del percorso definito sulla shadow per visualizzare la via di fuga. Come già spiegato, il percorso sarà definito come la somma dei collegamenti tra coppie di shadow.

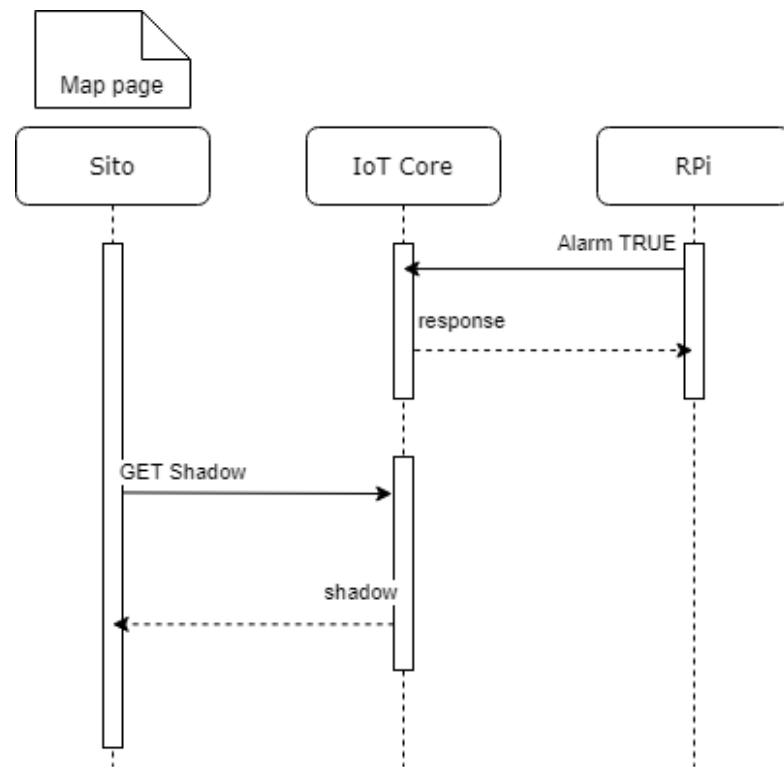


Figura 4.11: Sequenza ottenimento dati allarme

Infine il sito, una volta effettuato il login, fornisce la possibilità di inserire un codice di disattivazione dell'allarme all'interno della homepage. Questo codice potrà essere inserito manualmente, oppure con lo scan di un codice QR apposito. Questo codice sarà confrontato con quello dell'edificio di interesse, presente su DynamoDB e criptato con SHA512. Se i due codici coincidono, allora viene effettuato un update, in particolare una POST sull'URI esposto dalla shadow dell'allarme, che aggiorna lo stato desired di allarme a false. La shadow si aggiornerà sulla base della richiesta effettuata sullo stato desired, e l'allarme sarà impostato a false anche sullo stato reported. Il processo è visionabile in Figura 4.12.

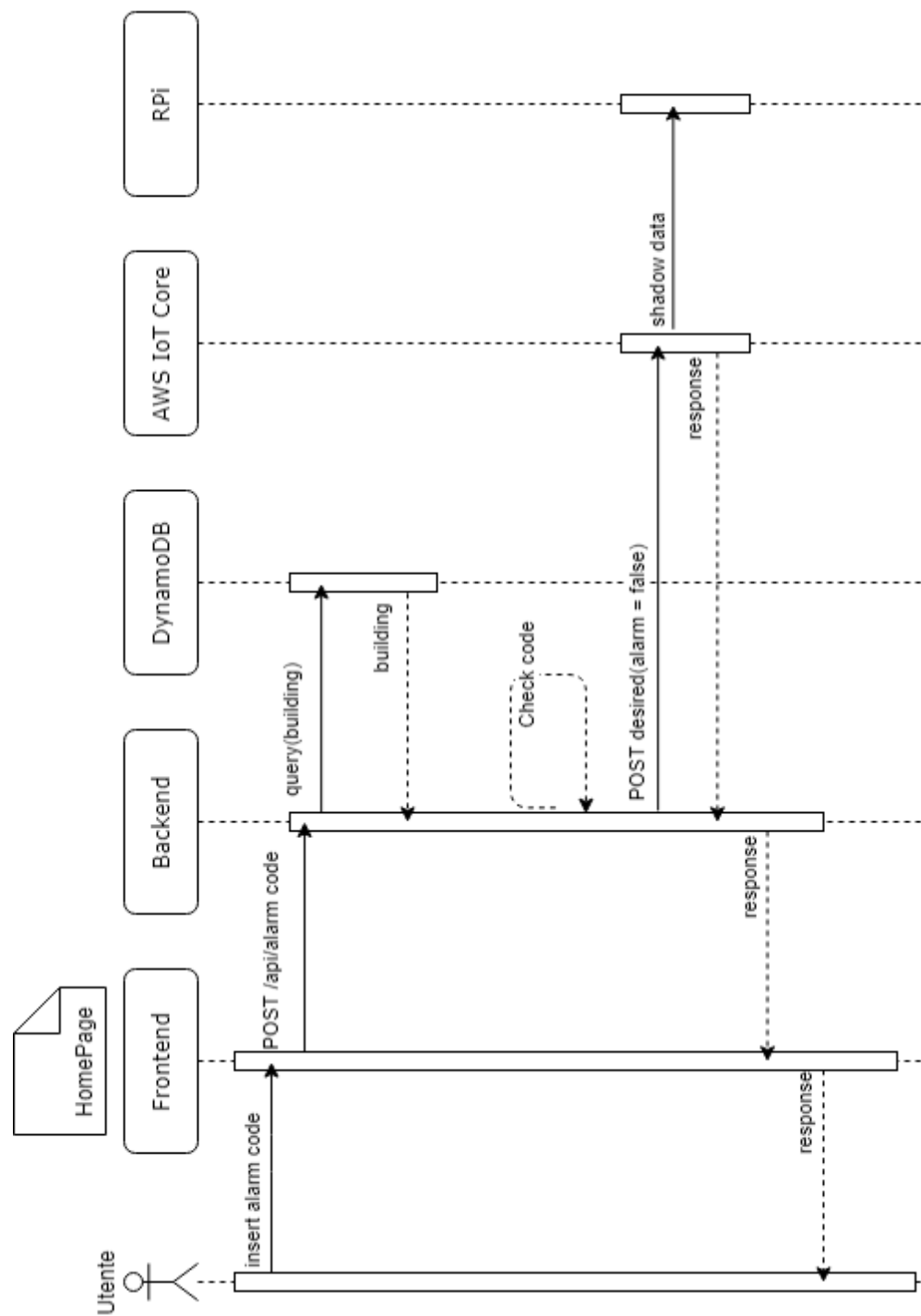


Figura 4.12: Sequenza spegnimento allarme

Capitolo 5

Implementazione

5.1 Sito

5.1.1 Login

Per poter gestire il login degli utenti al sito e le sessioni, si è fatto uso di Passport.js, un middleware di autenticazione per Node.js. La scelta è ricaduta su di esso in quanto avevamo già esperienza ed era risultato funzionale e intuitivo.

Validazione degli input

Per poter controllare che i dati forniti durante la registrazione di un account rispettino alcuni criteri, come ad esempio la lunghezza della password o una mail del giusto formato che non sia già utilizzata, è stato implementato un validatore. Ad esempio: perché l'username di un utente sia valido, deve essere almeno lungo 4 caratteri, alfanumerico e non deve già essere in uso.

```
1 body("._id")
2   .trim()
3   .isLength({min: usernameMinLength}).withMessage("Username must be
   at least " + usernameMinLength + " characters long")
4   .isAlphanumeric().withMessage("Username can only contain letters
   and numbers")
5   .custom(async value => {
6     const count = await Utenti.query("Account").eq(value).exec();
7     if (count.count != 0) {
8       return Promise.reject();
```

```
9     }  
10  }).withMessage("Username already in use"),
```

Sicurezza delle password

Per salvare le password degli utenti in modo sicuro abbiamo fatto uso di Crypto, un modulo di NodeJS, che fornisce tra le sue funzionalità, quella di poter utilizzare un algoritmo SHA512, per la criptazione delle password con sale.

```
1  var sha512 = function(password, salt){  
2    if (!salt) {  
3      const length = 32;  
4      salt = Crypto.randomBytes(Math.ceil(length/2))  
5        .toString('hex') /** convert to hexadecimal format */  
6        .slice(0,length);  
7    }  
8    var hash = Crypto.createHmac('sha512', salt); /** Hashing  
    algorithm sha512 */  
9    hash.update(password);  
10   var value = hash.digest('hex');  
11   return {  
12     salt:salt,  
13     passwordHash:value  
14   };  
15 };
```

5.1.2 DynamoDB

Node.js si connette a DynamoDB utilizzando l'aws-sdk e fornendo le credenziali di AWS.

```
1 var AWS = require("aws-sdk");  
2  
3 AWS.config.update({  
4   region: "eu-central-1",  
5   endpoint: "https://dynamodb.eu-central-1.amazonaws.com",  
6   accessKeyId: config.accessKeyId,  
7   secretAccessKey: config.secretAccessKey  
8  
9 });
```

```
10
11 var dynamodb = new AWS.DynamoDB();
```

Dynamoose

Per poter interagire con il DB, viene utilizzato Dynamoose, un tool di modellazione ispirato a Mongoose.

In primo luogo permette di definire degli schemi che rappresentano le tabelle del DB

```
1  const dynamoose = require('dynamoose');
2  var Schema = dynamoose.Schema;
3
4  var UtentiSchema = new Schema({
5    Account: {
6      type: String,
7      hashKey: true
8    },
9    Email: {
10     type: String,
11     required: 'An email is required'
12   },
13   Password: {
14     type: String,
15     required: 'A password is required'
16   },
17   Sale: {
18     type: String,
19     required: 'A salt is required'
20   },
21   Attivo: {
22     type: Boolean,
23     default: false
24   }
25 }, {
26   throughput:{read: 10, write:5}
27 });
28
29 module.exports = dynamoose.model('Utenti', UtentiSchema);
```

e poi di effettuare query e transazioni sul DB a partire da questi schemi. Ad esempio la seguente espressione ritorna l'utente con l'account che corrisponde al valore di value.

```
1 Utenti.query("Account").eq(value).exec();
```

5.1.3 Dati dei Device

5.1.4 Caricamento dei dati sul sito

Un altro aspetto importante è quello della visualizzazione dei dati dei device all'interno del sito. In particolare sono state predisposte due pagine, una con la lista dei device ed una con una mappa dei device geolocalizzati.

AWS IoT Core utilizza un modello in cui i prezzi vengono calcolati in base all'effettivo utilizzo, di conseguenza si è pensato al modo in cui si possa limitare le richieste dei dati.

Gestione device

Per quanto riguarda la pagina di gestione device, vengono presi da DynamoDB i valori statici e viene mostrata una lista dei device a partire da questi. A questo punto ogni device nella lista ha un bottone per mostrare i suoi dati in tempo reale all'interno di un modale. L'apertura di questo modale triggera una post da parte del client alla route `"/api/getshadow"` fornendo il nome del thing e la rispettiva shadow.

```
1 axios.post("/api/getshadow", {thingName: shadow.shadowThing,  
    shadowName: shadow.shadowName})
```

Questa richiesta viene effettuata ogni X secondi, per mantenere i dati aggiornati. A fronte di questa richiesta il controller si occupa di fare una richiesta ad AWS IoT Core, ottenendo il documento della shadow.

Rispettivamente, la chiusura del modale interrompe l'invio della richiesta, in modo tale da richiedere i dati solo quando effettivamente vogliamo visualizzarli.

Credenziali

Per poter utilizzare i servizi AWS è necessario possedere delle credenziali, queste sono state definite all'interno di un file di configurazione e caricate dove necessarie. TODO: Una volta hostato il sito su EC2, le credenziali sono acquisite attraverso l'IAM Role dell'istanza.

Autenticazione

Per poter effettuare una richiesta alle shadow di AWS è necessario fare una richiesta con autenticazione. Nel nostro caso è stato utilizzato il processo di firma "Signature Version 4".

In particolare, è stata utilizzata una utility chiamata "aws4" che permette di fare richieste HTTP(S) utilizzando la Signature Version 4.

In seguito la gestione del processo di firma da parte di aws4 e la richiesta verso AWS IoT Core.

```
1 exports.get_shadow = async function(req, res) {
2   try {
3     var result = aws4.sign({
4       service: 'iotdata',
5       host: config.host,
6       region: config.region,
7       method: 'GET',
8       path: '/things/'+req.body.thingName+'/shadow?name='+req.body.
          shadowName,
9       headers: {
10         'Content-Type': 'application/x-amz-json-1.0',
11       },
12       body: '{}',
13     }, {
14       secretAccessKey: config.secretAccessKey,
15       accessKeyId: config.accessKeyId
16     });
17     const ret = await request(result);
18     console.log(ret.body);
19     res.status(201).json({body: ret.body, shadowName: req.body.
          shadowName, thingName: req.body.thingName});
20   } catch (error) {
21     console.log(error);
22   }
23 }
```

```
22     res.status(501).json({errors: [error]});  
23   }  
24 }
```

Visualizzazione su mappa

La gestione dei dati su mappa è analoga a quella descritta per la gestione dei device, l'unica differenza sta nel fatto che per ogni device presente nel DB viene caricato un marker su mappa, che se premuto triggera la richiesta per ottenere i dati della shadow.

5.1.5 Mappa

Per poter permettere la visualizzazione dei device in modo geolocalizzato e delle vie di fuga si è scelto di utilizzare una mappa. In particolare si è deciso di adottare una tecnologia già presente sul mercato: MazeMap, una piattaforma che fornisce delle API in javascript di semplice utilizzo.

Una volta definita una mappa, ad esempio:

```
1   var myMap = new Mazemap.Map({  
2       container: 'map',  
3       campuses: 50,  
4       center: {lng: 78.4985754560354, lat: 38.03100233465469},  
5       zoom: 30,  
6       scrollZoom: true,  
7       doubleClickZoom: false,  
8       touchZoomRotate: false,  
9       zLevelControl: true  
10  });
```

è possibile posizionare sulla mappa dei marker

```
1   var mazeMarker = new Mazemap.MazeMarker( {  
2       color: "MazeBlue",  
3       size: 36,  
4       zLevel: shadow.shadowZFloor,  
5       zLevel: 1,  
6       innerCircle: true,  
7       imgUrl: shadow.shadowImage,  
8       imgScale: 1.1,
```

```
9      preventClickBubble: false // Allow click to go through  
    to global map layer  
10    } ).setLngLat( lngLat ).addTo(myMap);
```

e visualizzare dei percorsi.

```
1  Mazemap.Data.getRouteJSON(start, dest)  
2    .then(function(geojson){  
3      routeController.setPath(geojson);  
4  
5      // Fit the map bounds to the path bounding box  
6      var bounds = Mazemap.Util.Turf.bbox(geojson);  
7      myMap.fitBounds( bounds, {padding: 100} );  
8    });
```

La scelta di MazeMap rispetto ad altre soluzioni, come ad esempio WRLD3D o Mapbox, deriva dal fatto che fornisce delle API semplici e intuitive ed open source, a differenza di WRLD3D che risulta non solo più complesso ma anche a pagamento o Mapbox che richiede comunque un piano tariffario.

5.1.6 Dijkstra

Come definito nell'introduzione, il problema di identificazione delle vie di fuga viene demandato all'algoritmo di Dijkstra, che permette di calcolare i cammini minimi in un grafo.

In particolare, abbiamo deciso di implementare lo pseudo-codice definito all'interno dell'articolo [5].

L'algoritmo si sviluppa sui seguenti passi:

1. Scegli il vertice di partenza.
2. Definisci un set S di vertici e inizializzalo al set vuoto. L'algoritmo immagazzinerà i vertici per cui è stato trovato il percorso più breve all'interno di questo set.
3. Etichetta il vertice di partenza con 0 e inseriscilo in S .
4. Considera ogni vertice non in S , connesso direttamente al vertice appena aggiunto ad S . Etichetta il vertice non in S con l'etichetta del vertice appena aggiunto + la lunghezza/peso dell'arco. Se il vertice non in S aveva già una

etichetta, allora la nuova etichetta sarà il minimo tra quella del nuovo vertice + la lunghezza/peso dell'arco e la vecchia etichetta.

5. Scegli un vertice non in S con l'etichetta più bassa e aggiungilo ad S.
6. Ripeti il processo dallo step 4, fino a che il vertice di destinazione non fa parte di S, oppure fino a che non ci sono più vertici non in S da etichettare.

Al termine dell'algoritmo, se la destinazione è etichettata, allora esiste un percorso tra partenza e destinazione, altrimenti non è presente.

L'implementazione dell'algoritmo è stata definita come segue:

```

1  /**
2   * Dijkstra Algorithm to get the min path from source to
   * destination.
3   * @param from source
4   * @param to destination
5   * @param nodeMap graph
6   * @return a List of node if there is a path, an empty list
   * otherwise
7   */
8  def runDijkstra(from: String, to: String, nodeMap: ListBuffer[
   Path]): Option[ListBuffer[String]] = {

```

per il codice completo fare riferimento alla repository di progetto.

Ciò che questo algoritmo ritorna è un percorso, che sarà compito del controllore formattare e caricare sulla shadow per la gestione dell'allarme su AWS IoT Core, ad esempio:

```

1  "escape_path": [
2    {
3      "_1": "Start",
4      "_2": "B"
5    },
6    {
7      "_1": "B",
8      "_2": "A"
9    },
10   {
11     "_1": "A",
12     "_2": "Exit"
13   }

```

14

]

rappresenta un percorso che parte da Start, va in B, poi in A e infine ad Exit. Il sito, in presenza di un'emergenza, può recuperarlo e associare ad ogni posizione le relative coordinate geografiche definite in DynamoDB. Ottenendo delle coppie di coordinate latitudine e longitudine che individuano dei percorsi. Ognuno di questi percorsi viene mostrato su mappa, e la loro unione costituisce la via di fuga.

5.2 Raspberry Pi Controller



Figura 5.1: Setup locale con Raspberry Pi, CC2531 e Sensore di Fumo Zigbee

Il sistema serve a gestire i sensori, controllare lo stato e le azioni di allarme. Tutti i dati rilevanti (letture dei sensori, percorso ottimale di fuga) vengono inviati su AWS IoT.

5.2.1 Authentication

Per poter accedere alla propria istanza di Aws IoT ci sono diversi metodi di autenticazione. Quello utilizzato nel backend segue lo standard X.509 basato su certificati e chiavi pubbliche/private.

In particolare è necessario fornire al backend i seguenti documenti:

- **AmazonRootCA1.pem** Il certificato pubblico che identifica la Root Certification Authority di Amazon.

- **cert.pem.crt** Il certificato generato su AWS IoT ed associato ad una certa policy. Quest'ultima detta le azioni ammissibili da qualsiasi client che si connette con tale certificato.
- **private.pem.key** Chiave privata associata al certificato precedente. Sempre parte del protocollo X.509

Questi certificati e chiavi devono essere trasferiti sul Raspberry Pi manualmente e, per ovvi motivi di sicurezza, non devono essere tracciati git.

Dato l'endpoint per connettersi alla propria istanza di AWS IoT, esistono diversi protocolli di comunicazione come in figura 5.2. Il backend si connette attraverso il protocollo MQTT alla porta 8883.

Protocols, authentication, and port mappings				
Protocol	Operations supported	Authentication	Port	ALPN protocol name
MQTT over WebSocket	Publish, Subscribe	Signature Version 4	443	N/A
MQTT over WebSocket	Publish, Subscribe	Custom authentication	443	N/A
MQTT	Publish, Subscribe	X.509 client certificate	443 [†]	x-amzn-mqtt-ca
MQTT	Publish, Subscribe	X.509 client certificate	8883	N/A
MQTT	Publish, Subscribe	Custom authentication	443 [†]	mqtt
HTTPS	Publish only	Signature Version 4	443	N/A
HTTPS	Publish only	X.509 client certificate	443 [†]	x-amzn-http-ca
HTTPS	Publish only	X.509 client certificate	8443	N/A
HTTPS	Publish only	Custom authentication	443	N/A

Figura 5.2: Protocolli di comunicazione con AWS IoT

5.2.2 Sensori Zigbee e Zigbee2Mqtt

Il Raspberry Pi legge i dati dei sensori Zigbee tramite l'applicazione Zigbee2Mqtt. Quest'ultima apre una connessione seriale con il SoC CC2531 (connesso via USB al Raspberry Pi) ed estrae le informazioni sulla rete Zigbee e su tutti i device connessi ad essa. I dati rilevanti vengono pubblicati su topic specifici del canale MQTT, in particolare:

- **/zigbee2mqtt/bridge/devices** All'avvio di Zigbee2Mqtt e quando un dispositivo si aggiunge alla rete, viene pubblicato un messaggio contenente tutti

i device connessi (e la loro definizione) sul topic `/zigbee2mqtt/bridge/devices`. Il contenuto ci servirà per stabilire quali device sono in funzione e per aggiornare DynamoDB con eventuali nuovi device all'avvio del backend. Importante notare che ogni dispositivo Zigbee viene identificato dal suo `"friendly_name"`.

- **`/zigbee2mqtt/"friendly_name"`** Ogni volta che un device/sensore aggiorna il suo stato, inviando i dati al router, Zigbee2Mqtt li pubblica nel topic relativo al device stesso: `/zigbee2mqtt/"friendly_name"`. Sarà quindi necessario effettuare una subscribe anche ad ogni topic di device. Per farlo, si devono estrarre tutti i `friendly_name` dal messaggio inviato in `"/zigbee2mqtt/bridge/devices"`.

Il backend usa Paho, l'implementazione in Java di un client MQTT, e all'avvio si mette in ascolto dei messaggi in arrivo su `/zigbee2mqtt/bridge/devices`. Una volta arrivata la lista di tutti i device connessi, per ognuno di essi, inizia ad ascoltare sui topic `/zigbee2mqtt/"friendly_name"` relativi. Ogni volta che riceve un messaggio con i dati di un sensore, il backend chiama l'SDK di AWS IoT Device per sincronizzare lo stato della relativa Named Shadow su AWS IoT.

```

1  case class ZigbeeController(mqttClient: MqttClient) {
2    val topicPrefix: String = "/zigbee2mqtt"
3
4    mqttClient.subscribe(s"$topicPrefix/bridge/devices", (_, String,
5      mqttMessage: MqttMessage) => {
6      val devices = MqttMessageParser.fromMqttMessage(mqttMessage).
7        parseDevices
8      devices.keySet.foreach(subscribeToDeviceUpdates)
9      SensorsController.registerDevices(devices)
10   })
11
12   def subscribeToDeviceUpdates(name: String): Unit = {
13     mqttClient.subscribe(s"$topicPrefix/$name", (_, String,
14       mqttMessage: MqttMessage) => {
15       println(s"received message from Zigbee Sensor $name")
16       println(mqttMessage.toString)
17       val updateMap = MqttMessageParser.fromMqttMessage(
18         mqttMessage).fromUpdateToMap

```



```
15     SensorsController.onZigbeeDataUpdate(name, updateMap)
16   })
17 }
18 }
```

5.2.3 Register new devices to DynamoDB

Inoltre, quando vengono scoperti nuovi devices connessi alla rete Zigbee, è necessario registrarli su DynamoDB. Per fare ciò, è stata introdotta una Rule su AWS IoT che viene triggerata ogni volta che viene mandato un messaggio opportuno sul topic MQTT `iot/devices`. Questa Rule avvia una Lambda che effettua il parsing del messaggio e inserisce i dati dei nuovi sensori su DynamoDB. Se nel messaggio ci sono device già registrati sul database, la lambda fa un controllo per ogni campo e aggiorna solo quelli modificati. La funzione Lambda viene eseguita nel runtime di Node.js.

Il data flow completo per l'aggiornamento di DynamoDB è dunque:

Zigbee2Mqtt publish on topic `/zigbee2mqtt/bridge/devices` → Paho receives the message and forwards to AWS Rule on topic `iot/devices` → Rule triggers AWS Lambda → AWS Lambda writes on DynamoDB.

Questo è stato fatto per escludere la dipendenza di DynamoDB per il Raspberry Pi.

5.2.4 SensorController

SensorController si occupa di gestire la lista di sensori e di tenerli sincronizzati con le Shadow di AWS Iot. Per fare ciò, alla momento della creazione dei sensori, si registra agli eventi di Tipo `ShadowDeltaUpdateEvent`, del SDK di AWS IoT Device. Ogni volta che lo stato "desired" della Shadow su AWS IoT viene aggiornato, il backend riceve questo evento con dentro le modifiche richieste allo stato del sensore/device.

```

2  def listenForDeltaUpdates(newSensors: Set[String]): Unit =
    shadowClient match {
3      case None =>
4      case Some(client) =>
5          (newSensors -- sensors.keySet).foreach { shadowName =>
6              val requestShadowDeltaUpdated:
                ShadowDeltaUpdatedSubscriptionRequest = new
                ShadowDeltaUpdatedSubscriptionRequest
7              requestShadowDeltaUpdated.thingName = thingId
8              client.SubscribeToShadowDeltaUpdatedEvents(
9                  requestShadowDeltaUpdated,
10                 QualityOfService.AT_LEAST_ONCE,
11                 event => onShadowUpdate(shadowName, event))
12                 .get
13         }
14     }

```

Inoltre, affinché sia il backend stesso ad aggiornare su AWS IoT lo stato reported della Shadow, ogni volta che vengono pubblicati i dati del sensore sul topic locale `/zigbee2mqtt/"friendly_name"`, viene eseguita la funzione `"onZigbeeDataUpdate"`. Per aggiornare lo stato della Shadow, attraverso lo `ShadowClient` dell'SDK di AWS Iot Device, viene inviata una richiesta di tipo `UpdateNamedShadowRequest`.

```

1  def updateNamedShadow(shadowName: String, shadow: NamedShadow)
    : Unit =
2      sensors.get(shadowName) match {
3          case None =>
4          case Some(localValue) if localValue != shadow =>
5              sensors = sensors.updated(shadowName, shadow)
6              val request = new UpdateNamedShadowRequest()
7              request.thingName = thingId
8              request.shadowName = shadowName
9              request.state = new ShadowState()
10             request.state.reported = new util.HashMap[String, Object](
                shadow.reported.asJava)
11             request.state.desired = new util.HashMap[String, Object](
                shadow.desired.asJava)
12
13             shadowClient.head.PublishUpdateNamedShadow(request,
                QualityOfService.AT_LEAST_ONCE)

```

```
14         .thenRun(() => println(s"Updated namedShadow $shadowName  
    "))  
15         .get()  
16     }
```

5.2.5 Sensori Simulati

La gestione dei sensori simulati avviene in modo analogo ai sensori Zigbee, escludendo appunto la subscribe ai topic di Zigbee2Mqtt. Semplicemente c'è un `TimerTask` che viene schedolato ogni secondo per simulare dei dati pseudo-random che potrebbe assumere un sensore. Quindi ogni secondo, tramite lo `ShadowClient` dell'SDK di AWS Iot Device, viene inviata una `UpdateNamedShadowRequest` per ogni sensore simulato.

5.2.6 Allarme

L'allarme è rappresentato su AWS IoT come una `Thing` singola per un edificio, con una sola classic (unnamed) `Shadow`. La `Shadow` viene utilizzata come canale di comunicazione fra sito e backend del Raspberry Pi. Quest'ultimo aggiorna lo stato "reported", nel senso che è il responsabile per l'effettiva attivazione dell'allarme. Il sito invece suggerisce modifiche allo stato "desired" della `Shadow`, inviando delle `DeltaRequest` al Raspberry Pi.

L'allarme può essere attivato manualmente, se il sito fa una richiesta di Delta con `state.desired.alarm = true`, oppure automaticamente secondo una qualche euristica (per ora si attiva se un sensore riconosce del fumo). Invece, l'allarme può essere disattivato solo manualmente, con un'altra richiesta di Delta, settando lo stato della shadow `state.desired.alarm = false`.

Una volta attivato l'allarme, il backend avvia un task periodico `DijkstraAlarm` che effettua la ricerca del percorso di fuga ottimo, filtrando le zone (per ora rappresentate dai sensori) inagibili. Quando l'allarme viene disattivato, il task periodico viene fermato.

Capitolo 6

Testing e performance

6.1 Funzionamento

Un utente avrà la possibilità di accedere ad un sito attraverso un login, all'interno del quale potrà:

- Visualizzare la lista dei device dell'edificio e controllare in tempo reale i loro dati.
- Visualizzare su di una mappa interattiva i device e la via di fuga in caso di emergenza.
- Terminare una situazione di allarme attraverso l'inserimento di un codice, scritto a mano o scannerizzato da un codice QR.

Nel frattempo il controllore gestirà autonomamente i device dell'edificio, raccogliendone i dati e depositandoli sul cloud. Dai dati potrà rendersi conto della presenza di una situazione di allarme, calcolando il percorso migliore con Dijkstra e aggiornandolo sulla shadow predisposta. Questo processo di aggiornamento viene fatto in modo continuo in modo tale che i dati siano sempre aggiornati e si trovino vie di fuga in tempo reale. Il sito potrà visualizzare la via di fuga trovata da Dijkstra e la mostrerà su mappa ogni qualvolta ne viene calcolata una diversa da quella precedente.

6.2 Testing

6.2.1 Caso di Studio

Il funzionamento del sistema è stato testato principalmente attraverso un caso di studio. Un utente si registra al sito, effettua il login con le proprie credenziali e accede. A questo punto visualizza la lista di device dell'edificio e controlla l'aggiornamento dei dati. In seguito si trasferisce nella mappa dove controlla la posizione dei device e i loro dati, poi si rende conto che è presente una via di fuga, in questo modo capisce che è in atto una situazione di emergenza. A questo punto l'utente, termina la situazione di emergenza tornando in homepage e inserendo il codice di allarme manualmente nell'apposito campo.

In questo caso di studio l'utente si trova nella posizione nominata "Start" all'interno della figura 6.1, e la via di fuga sarà segnalata in verde, in quanto quella rossa, anche se più breve, non è agibile per un incendio. La via di fuga è calcolata in tempo reale da Dijkstra, non considerando i percorsi insicuri.

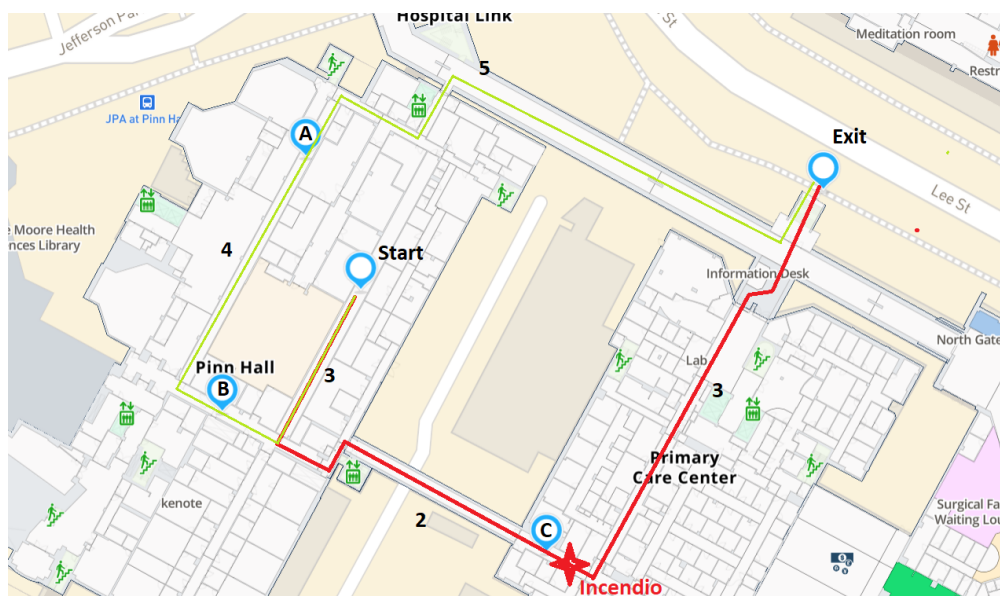


Figura 6.1: Caso di Studio: via di fuga

Il caso di studio è stato documentato attraverso un video di presentazione presente in appendice (9).

6.3 Performance

6.3.1 Sito Web - Node.js

Hardware

- GPU: RX480 4gb
- CPU: AMD Ryzen 3600 6-Core
- Ram: 16gb 3200mhz

I test delle performance del sito sono stati effettuati usando Artillery.io, che permette di fare richieste HTTP e ricevere in output i tempi. I risultati sono mostrati nella tabella 6.1.

Accesso Pagina	Scenari Lanciati	Mean response/sec	Response time (msec)
GET Home	10	1.06	min: 1, max: 4.5, median:1.2
POST Login	10	1.06	min: 1.1, max: 5.4, median: 1.3
GET Mappa	10	2.11	min: 0.6, max: 9.5, median: 1.3
GET Device	10	2.11	min: 0.7, max: 9.4, median: 1.6

Tabella 6.1: Prestazioni delle richieste HTTP

Capitolo 7

Analisi di deployment su larga scala

7.1 Il sito web

Per quanto riguarda il sito web, questo è stato progettato per essere hostato su di una istanza di EC2. Una singola istanza permetterà agli utenti in possesso di un account di effettuare il monitoraggio dei device di un particolare edificio.

In una futura release, in cui il sistema verrà adottato in edifici diversi, sarà possibile istanziare nuove istanze di EC2, una per ogni edificio, in modo da distribuire gli accessi.

Questi cambiamenti non causano particolari problematiche avendo progettato il sito utilizzando i servizi di AWS. Potrebbero essere utilizzate anche le funzionalità di auto-scaling che monitorano e regolano le capacità per mantenere prestazioni stabili e prevedibili riducendo i costi.

7.2 Locale

Il sistema, una volta venduto ad un certo cliente, fornirà un pacchetto contenente il controllore e l'insieme di device fisici utili al monitoraggio dell'edificio. Questo pacchetto può essere ampliato in base alle esigenze del cliente e in base alla situazione, con un quantitativo maggiore o minore di device, di tipologia differente e con protocolli di comunicazione diversi. Nel caso in cui venga adottato su larga

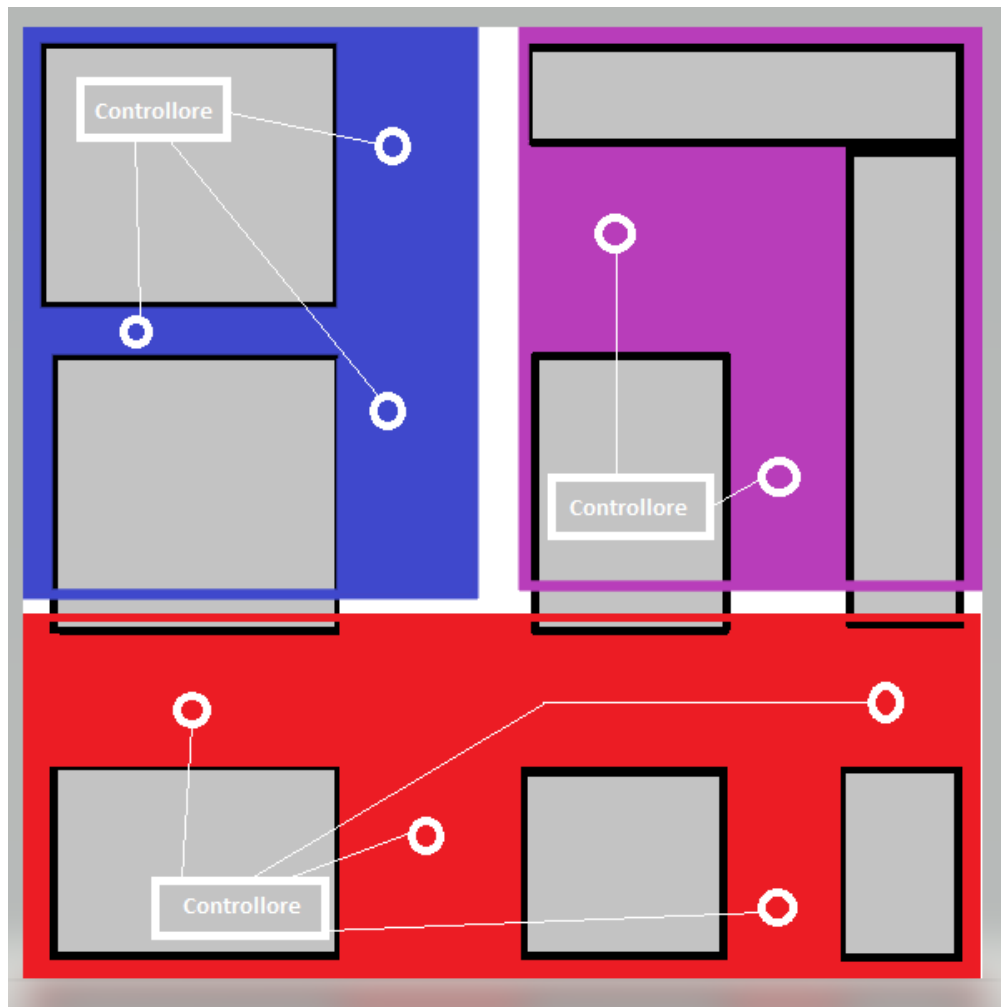


Figura 7.1: Suddivisione dell'edificio

scala, la vendita sarà aperta ad un numero maggiore di clienti. Sarà sufficiente vendere pacchetti diversi a clienti diversi, dove ogni pacchetto fa riferimento ad un edificio e contiene il proprio controllore e device correlati. Nel caso in cui un edificio sia particolarmente ampio, o con una struttura complessa che ad esempio potrebbe creare problemi a livello di connessioni, si può pensare di suddividere l'edificio in zone: in ogni zona sarà presente un sistema di controllo che gestirà soltanto l'insieme di device di quella determinata zona, vedi Figura 7.1.

In questo modo, sarà possibile gestire separatamente ogni zona, ma allo stesso tempo l'unione dei dati provenienti dalle zone differenti, fornirà una visione com-

plessiva dell'edificio. Grazie al fatto che i device saranno raccolti dai controllori e poi depositati su cloud, la visione dell'edificio da parte di servizi esterni, come ad esempio il sito web, sarà trasparente: il sito vedrà infatti tutti i dati a prescindere dalla zona, e di conseguenza sarà nascosta l'architettura locale.

Capitolo 8

Piano di lavoro

8.1 Divisione dei compiti

Seguendo la suddivisione del sistema definita in 1.1, il lavoro è stato assegnato inizialmente come segue:

- Sistema di controllo nell'edificio: la parte legata alla gestione dei device IoT e AWS IoT Core è stata assegnata a Christian Serra.
- Sito web: lo sviluppo del sito web è stato assegnato a Maicol Forti.
- Dijkstra: l'algoritmo di Dijkstra è stato implementato da Maicol Forti.

Questo tipo di suddivisione è stata presa in quanto permette di sviluppare le due parti in parallelo e in modo indipendente, la scelta specifica dell'assegnazione è stata presa in quanto Christian possiede l'hardware utile alla parte di controllo.

8.2 Piano di lavoro adottato

Grazie all'indipendenza delle parti, ogni membro del gruppo si è auto-gestito per quanto riguarda il piano di lavoro, mantenendo però dei punti in comune per quanto riguarda il sistema di reportistica.

8.2.1 Versioning

Lo strumento di versioning utilizzato è stato Git, sfruttando una repository online sulla piattaforma Github. Ogni developer ha lavorato su una propria copia della repository principale, in modo da effettuare in locale gli sviluppi richiesti. Nelle repository locali ognuno ha lavorato come meglio crede.

8.2.2 Trello

Per avere le feature da sviluppare sempre a disposizione con la possibilità di aggiornare il livello di completamento, si è deciso di utilizzare una bacheca condivisa. Ogni feature è stata decorata con un'etichetta per definire a chi appartiene e una checklist con le sotto-feature, in modo da determinare lo stato di completamento. Inoltre su trello abbiamo reso disponibili varie utilities per il progetto a cui accedere velocemente.

<https://trello.com/b/ItvtAwJC/smartcity>

8.2.3 Toggl

Per poter tracciare le ore di lavoro si è deciso di utilizzare Toggl, una comoda applicazione che permette di definire un progetto e di aggiungerci le ore di lavoro. Queste ore sono state personalizzate con varie etichette per definire il tipo di lavoro svolto. Abbiamo cercato di essere molto precisi nel registrare le ore spese in questo progetto, aggiungendo ad ogni entry una breve descrizione e alcune tag comuni per mostrare su cosa stavamo lavorando. La lista completa di ogni time-entry è disponibile in formato pdf all'interno della repo di progetto, con il nome di Timetable.pdf

Capitolo 9

Conclusioni

L'avvento dell'IoT e dei servizi Cloud, riteniamo fornisca una base solida per fornire servizi utili al miglioramento delle città e alla salvaguardia della popolazione. I casi di emergenza all'interno degli edifici, come ad esempio gli incendi, potrebbero essere controllati in modo più efficiente sfruttando questi mezzi, ma nonostante i presupposti siano promettenti, non ci sono ancora soluzioni acquistabili sul mercato.

Il progetto Smart Evacuation è stato pensato come un prototipo di partenza per un possibile sistema di monitoraggio e controllo delle situazioni di emergenza indoor, vendibile sotto forma di pacchetto pensato per la gestione di un edificio: all'acquisto di un sistema di questo tipo vengono forniti uno o più controllori, un gruppo di sensori ed attuatori, un sito web per poter monitorare la situazione in tempo reale. Il sistema sarà consegnato già configurato secondo le esigenze dell'acquirente ed installato all'interno dell'edificio.

In futuro è possibile estendere questo progetto in varie direzioni:

- Gestire il controllore dell'edificio come un sistema decentralizzato per aumentare la resilienza.
- Utilizzare i device mobili per implementare una forma di crowd control per distribuire le persone su più percorsi in modo da non sovraffollarli.
- Utilizzare la posizione GPS del cellulare per determinare la posizione attuale e di conseguenza la via di fuga.
- Utilizzare l'astrazione Path per rappresentare il percorso.

Appendice

- Slide di presentazione: sono presenti all'interno della repo di progetto sotto al nome di slide_presentazione.pdf
- Video di presentazione: il video di presentazione è presente al seguente link https://www.youtube.com/watch?v=UUcu4hpWXiE&ab_channel=MaicolForti.
- CC2531 Datasheet https://www.ti.com/lit/ds/symlink/cc2531.pdf?ts=1613296821090&ref_url=https%253A%252F%252Fwww.ti.com%252Fproduct%252FCC2531
- Honeywell Smoke Sensor Zigbee2Mqtt Info <https://www.zigbee2mqtt.io/devices/JTYJ-GD-01LM-BW.html>
- Honeywell Smoke Sensor Datasheet <https://5.imimg.com/data5/ES/AT/MY-1235321/battery-operated-smoke-detector-honeywell.pdf>

Bibliografia

- [1] Gabriel M. Eggly, Mariano Finochietto, Matías Micheletto, R. P. Centelles, Rodrigo Santos, S. Ochoa, R. Meseguer, and J. Orozco. Evacuation supporting system based on iot components †. In *UCAmI*, 2019.
- [2] V. Ferraro and J. Settino. *Evacuation and Smart Exit Sign System*, pages 363–383. Springer International Publishing, Cham, 2019.
- [3] S. Majumder, S. O’Neil, and R. Kennedy. Smart apparatus for fire evacuation — an iot based fire emergency monitoring and evacuation system. In *2017 IEEE MIT Undergraduate Research Technology Conference (URTC)*, pages 1–4, 2017.
- [4] Keerthi Mohan, A. Mahesh, Atul Vasudevan, and Krupa Panchagnula. Iot based emergency evacuation system. *INTERNATIONAL JOURNAL OF ENGINEERING RESEARCH & TECHNOLOGY (IJERT) ICIOT*, 4, 2016.
- [5] W. Wang, G. Liu, X. Chen, Z. Liu, A. Zhang, Z. Xing, L. Liu, and T. Fei. Dijkstra algorithm based building evacuation recognition computing and iot system design and implementation. In *2019 IEEE 13th International Conference on Anti-counterfeiting, Security, and Identification (ASID)*, pages 229–233, 2019.