

POLITECNICO DI TORINO

CORSO DI LAUREA MAGISTRALE in

INGEGNERIA INFORMATICA



Tesi di Laurea Magistrale

**Progettazione e implementazione di una
applicazione mobile per la rilevazione e la
segnalazione in tempo reale di violenza di
genere**

Realatori

Prof. SARAH AZIMI

Prof. LUCA STERPONE

Dott. CORRADO DE SIO

Candidata

ALESSANDRA PALMA

Aprile 2024

Sommario

Safe Smart City (S2CITIES) è un progetto dedicato a rilevare violenza e richieste di aiuto in tempo reale tramite i dispositivi di videosorveglianza che si inserisce nell'ambito delle *smart cities* e ha come obiettivo quello di offrire una soluzione innovativa per la rilevazione, la segnalazione e la gestione di eventi associati ad atti di violenza ripresi dai dispositivi di videosorveglianza, al fine di aumentare il livello di sicurezza all'interno dell'ambiente in cui viviamo e lavoriamo. Nello specifico il presente elaborato di tesi ha previsto la progettazione e lo sviluppo di tre moduli distinti e comunicanti tra di loro. Un web server, basato su Node.js e Typescript, raccoglie ed elabora i dati inviati dalla videocamera di sorveglianza posta all'interno di un edificio, non appena viene rilevato e processato un segnale di aiuto. Alle guardie di sicurezza che vigilano la struttura viene messa a disposizione una applicazione mobile per dispositivi Android che segnala loro la richiesta di soccorso sotto forma di notifica push. Il messaggio di *alert* riporta la data, l'ora e il luogo della segnalazione oltre che l'identificativo della videocamera. In questo modo la guardia può recarsi sul posto ed eventualmente chiedere aiuto ai colleghi o chiamare le forze dell'ordine direttamente dall'app. Le guardie di sicurezza, gli edifici e le videocamere vengono registrati nel sistema da utenti abilitati a farlo (admin e supervisor) attraverso l'uso di una piattaforma web che, così come l'app, comunica col server mediante API RESTful. Tali dati sono storicizzati in una base dati non relazionale, quale MongoDB, sotto forma di documenti. Inoltre, si è scelto di adoperare Typescript come principale linguaggio di programmazione e i framework React e React Native rispettivamente per la parte applicativa web e

mobile. Il risultato del lavoro consiste in un prototipo funzionante e utilizzabile che mira a fornire una base su cui eseguire futuri miglioramenti e sviluppi.

Ringraziamenti

Per la realizzazione di questo progetto ci tengo a ringraziare prima di tutto la mia relatrice Sarah Azimi per la sincera gentilezza e l'estrema disponibilità dimostratami.

Allo stesso modo grazie a Corrado De Sio per i consigli e il supporto morale.

Grazie all'azienda Getapper, senza il loro prezioso aiuto non sarebbe stato possibile raggiungere il traguardo. Un ringraziamento particolare va ad Antonio Ascone che mi ha pazientemente seguito nei momenti di difficoltà.

E infine grazie alla mia famiglia che mi è sempre stata vicina e ai miei amici che mi hanno supportato e sopportato per tutto questo tempo.

Indice

Elenco delle tabelle	VII
Elenco delle figure	VIII
Acronyms	XI
1 Introduzione	1
1.1 Motivazioni	1
1.2 Obiettivi e caso di studio	3
2 Stato dell'arte	5
2.1 Smart Safe City	5
3 Il progetto <i>s2cities</i>	9
3.1 Architettura generale del sistema	9
3.1.1 System-on-Chip	9
3.1.2 Web Server	10
3.1.3 App Mobile	11
3.1.4 Dashboard	12
3.2 Analisi dei requisiti di sistema	13

3.2.1	Scenario di riferimento	13
3.2.2	Attori e ruoli	13
3.2.3	Requisiti funzionali	15
3.2.4	Requisiti non strettamente funzionali	19
4	Sviluppo e progettazione dell'applicazione Android	21
4.1	Struttura dell'applicazione	22
4.1.1	Progettazione dell'interfaccia utente	22
4.1.2	Diagramma dei casi d'uso	23
4.2	Progettazione delle attività	25
4.2.1	Autenticazione	25
4.2.2	Recupero credenziali	26
4.2.3	Visualizzazione e modifica del profilo	27
4.2.4	Impostazione dello stato di servizio	28
4.2.5	Ricerca e visualizzazione contatti	29
4.2.6	Notifica Evento	30
4.2.7	Gestione delle segnalazioni	31
4.2.8	Archiviazione della segnalazione	32
4.3	Progettazione Dati	33
4.3.1	Il formato JSON	34
4.3.2	Le relazioni tra i dati	34
5	Implementazione della soluzione	37
5.1	Uno sguardo alle tecnologie utilizzate	38
5.2	Implementazione del Web Server	44
5.2.1	Il pattern MVC	44

5.2.2	Accesso alle risorse	46
5.2.3	Gestione della concorrenza	51
5.3	Implementazione del servizio FCM	52
5.4	Funzionalità principali realizzate	58
5.4.1	L'app s2cities	58
5.4.2	Il pannello di controllo	77
6	Conclusioni	83
6.1	Sviluppi futuri	84
	Bibliografia	87

Elenco delle tabelle

5.1	Accesso alle risorse	47
-----	--------------------------------	----

Elenco delle figure

1.1	Rilevazione del segnale e inoltro della richiesta di soccorso	2
3.1	Architettura generale	10
3.2	Diagramma delle entità	14
4.1	Mappa del sito	22
4.2	Diagramma dei casi d'uso	23
4.3	Progettazione dei dati principali	35
5.1	Architettura MVC	45
5.2	Models folder	45
5.3	Event Loop	51
5.4	Architettura del servizio FCM	53
5.5	LoginScreen	59
5.6	ForgetPswScreen	59
5.7	Email recupero credenziali	61
5.8	HomeScreen	62
5.9	Notifica push	63
5.10	AlertScreen	63

5.11 Segnalazione in corso	64
5.12 ActionsScreen	64
5.13 AcceptedActionScreen	67
5.14 ClosedActionScreen	67
5.15 DismissedActionScreen	68
5.16 MessagesScreen	70
5.17 Dialog gestione segnalazione	70
5.18 EditActionScreen	72
5.19 SettingsScreen	74
5.20 ProfileScreen	74
5.21 NotificationSettingsScreen	76
5.22 EditNotificationSettings	76
5.23 Menu	77
5.24 Dashboard relativa al Campus Borsellino	78
5.25 Form per la registrazione di una nuova struttura	79
5.26 Form per l'inserimento di un nuovo utente	80
5.27 Email credenziali d'accesso	81
5.28 Form per la registrazione di una videocamera di sorveglianza	82
5.29 Categorie	82
5.30 Posizione della videocamera	82

Acronyms

API

Application Programming Interface

CNN

Convolutional Neural Network

CRUD

Create, Read, Update e Delete

FCM

Firebase Cloud Messaging

MVC

Model View Controller

REST

REpresentational State Transfer

SPA

Single Page Application

OTA

Over-the-air

Capitolo 1

Introduzione

1.1 Motivazioni

L'Organizzazione Mondiale della Sanità afferma che nel corso della sua vita, una donna su tre subisce violenza fisica o sessuale da parte di un uomo [1]. In Italia i dati Istat mostrano che il 31,5% delle donne ha subito nel corso della propria vita una qualche forma di violenza fisica o sessuale. Le forme più gravi di violenza sono esercitate da partner o ex partner, parenti o amici. Gli stupri sono stati commessi nel 62,7% dei casi da partner [2]. Le misure di isolamento legate al Covid-19 nel 2020 hanno portato ad un incremento di casi di violenza domestica su donne, ragazze e bambini. Tra marzo e maggio 2020, sono più che raddoppiate le richieste di aiuto al 1522 (numero anti-violenza del Dipartimento Pari Opportunità) rispetto allo stesso periodo del 2019 [3]

E' per questo motivo che durante il periodo di lockdown si è diffuso sui social, specialmente su Tik Tok il così detto "Signal For Help", un modo silenzioso per chiedere aiuto attraverso un particolare gesto della mano che si realizza tenendo

il pollice piegato nel palmo e chiudendo le altre dita verso il basso. Il segnale di aiuto è stato introdotto per la prima volta in Canada dalla Canadian Women's Foundation il 14 aprile 2020 [4] e il 28 aprile 2020 negli Stati Uniti dalla Women's Funding Network (WFN).

Riconoscere tale gesto, ancora in uso dopo la pandemia per segnalare situazioni di pericolo, è stata la sfida portata avanti dal gruppo di ricerca del dipartimento di Informatica e Automatica del Politecnico: rilevare violenza e richieste di aiuto in tempo reale tramite i dispositivi di videosorveglianza [5].

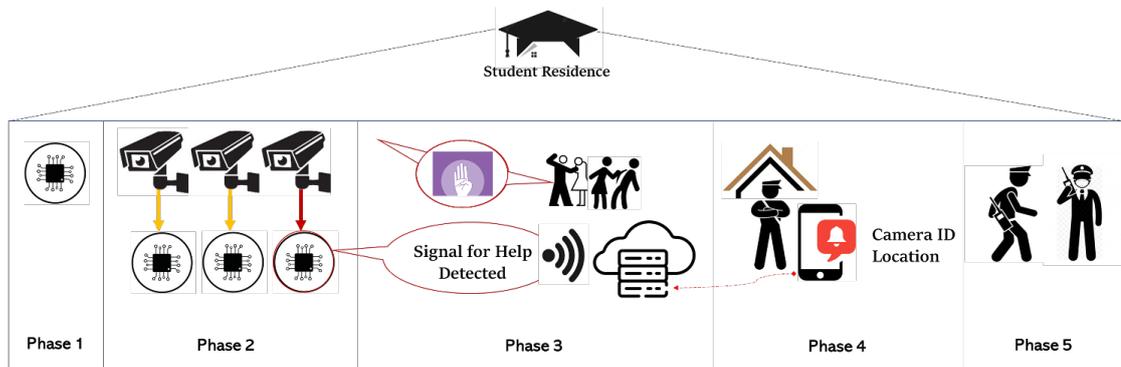


Figura 1.1: Rilevazione del segnale e inoltro della richiesta di soccorso

Come mostra la figura 1.1, la piattaforma di riconoscimento proposta si articola in cinque fasi:

1. Riconoscimento del gesto di violenza attraverso l'implementazione di algoritmi di machine learning eseguita su *low-cost edge* device
2. Integrazione della piattaforma sviluppata con i sistemi di sorveglianza già presenti in una determinata area (*smart cities*)

3. Messa a punto di un meccanismo di comunicazione tra il dispositivo di riconoscimento ed un *server in cloud* che elabora le informazioni ricevute
4. Invio di una *notifica push* alle guardie di sicurezza che sorvegliano la zona per sollecitare l' intervento
5. Inoltro della richiesta di soccorso alle forze dell'ordine

1.2 Obiettivi e caso di studio

Il progetto di tesi che mi ha visto coinvolta per vari mesi si inserisce nel suddetto contesto e propone lo sviluppo delle ultime tre fasi citate sopra. Pertanto, l'obiettivo è stato quello di ideare un'applicazione mobile al fine di segnalare, attraverso un sistema di *notifiche push*, l'evento di violenza in corso alle autorità di competenza. Contestualmente alla realizzazione di un' applicazione mobile ad uso della sorveglianza vigilante, il mio impegno in questo scenario ha previsto inoltre:

- lo sviluppo di un *server Web* in grado di sincronizzarsi con la piattaforma di riconoscimento precedentemente sviluppata, intercettare il "Signal for help" catturato dalle telecamere di sorveglianza e inoltrarlo ai vari client connessi alla rete
- lo sviluppo di una *piattaforma Web* ad uso esclusivo degli amministratori di sistema.

Sebbene sia proposta una soluzione generalizzata del problema, si è scelto di circoscrivere la questione alla residenza universitaria Borsellino che si trova nelle immediate vicinanze della Cittadella Politecnica, immaginando così come caso di

studio i segnali di aiuto generati dalle telecamere di sorveglianza presenti all'interno della struttura.

Capitolo 2

Stato dell'arte

Negli ultimi anni l'avvento dell'Internet Of Things ha permeato il nostro modo di vivere e di relazionarci con gli altri e con l'ambiente che ci circonda. Questo continuo processo di cambiamento ha come obiettivo generale la definizione di un mondo “migliore” in cui vivere, in cui la nostra qualità di vita è superiore. Ma cosa significa “migliore” e come questo processo avviene? L'uso di tale aggettivo introduce il concetto di *smart safe city*.

2.1 Smart Safe City

Una *smart city* è un'area urbanistica intelligente e sostenibile, che ha come obiettivo quello di raggiungere una sostenibilità equilibrata nelle dimensioni che compongono la città, grazie all'utilizzo lungimirante delle tecnologie digitali [6]. Negli ultimi tempi al concetto di *smart city* si affianca sempre più quello di *safe city*: usare il progresso tecnologico sul territorio per garantire la sicurezza delle persone che ci vivono.

Numerosi sono i progetti che si inseriscono in quest'ambito e spaziano da settori come la mobilità e i trasporti, a quelli che riguardano l'industria, la salute, il turismo, l'educazione, la sostenibilità. Molti di questi condividono il medesimo approccio: catturare in tempo reale delle informazioni, in situazioni più o meno emergenziali, analizzare i dati che arrivano da varie risorse e identificare il problema in modo da intervenire tempestivamente.

E' questo per esempio il processo alla base di questa ricerca [7], condotta nella città di Torino, che ha come obiettivo quello di monitorare il fenomeno delle alluvioni - problema molto sentito in Italia a causa della peculiare morfologia del territorio - favorendo la comunicazione tra i coordinatori delle emergenze, i soccorritori e i cittadini. I sensori sparsi sul territorio raccolgono i dati che possono essere di varia natura (densità di popolazione, livello del fiume, pioggia etc.) e provenire anche da fonti esterne. Un'applicazione Web realizzata in Python abbinata ad un servizio RESTful rende visibile tale collezione di dati ai coordinatori delle emergenze che possono geolocalizzare il problema e assegnare dei task agli operatori sul campo. Infine un'applicazione mobile sviluppata per sistemi Android permette la localizzazione via GPS dell'operatore.

Nella prospettiva di costruire una società 5.0 [8], il binomio tra integrazione dei dati e visualizzazione tramite tecnologie web non può che manifestarsi anche nell'ambito dell'healthcare. Di particolare interesse sono le piattaforme di healthcare che usano IoT, BigData e algoritmi di intelligenza artificiale per facilitare la vita delle persone anziane che vivono da sole e che richiedono cura e assistenza. Quando gli anziani subiscono incidenti domestici (come cadute o incidenti) o subiscono o improvvisi malori come un ictus o un infarto cerebrale, la velocità e l'efficacia della risposta hanno un impatto critico sulla determinazione della sopravvivenza e della

prognosi [8]. Gli autori di questo studio [9] dimostrano come è possibile rilevare le cadute utilizzando immagini e dati audio processati in tempo reale, inviare un segnale al server in cloud e quindi allertare i servizi di emergenza.

Lo sviluppo di applicazioni mobile ha reso semplice monitorare il nostro stato di salute anche attraverso l'uso di dispositivi indossabili. Questa descritta qui [10] in particolare utilizza oltre al sensore di accelerometro del device anche uno smartwatch per la rilevazione dei battiti cardiaci. Si serve delle librerie di Google Maps e del segnale GPS per determinare quale è la struttura ospedaliera più vicina al verificarsi di casi di emergenza come una caduta o un incendio.

Un altro settore spesso connesso alle *smart cities* è quello della mobilità urbana. Rendere una città più sicura significa anche ridurre il tasso di mortalità dovuto agli incidenti stradali. Secondo un rapporto dell'OMS infatti gli incidenti stradali rimangono la principale causa di morte tra i bambini e i giovani di età compresa tra i 5 e i 29 anni [11]. In questo campo, ad esempio, un lavoro di ricerca degno di nota è quello condotto in Arabia [12] dove il tasso di mortalità dovuto a incidenti stradali è molto alto. E' stato possibile creare un modello di rilevamento e tracciamento dei veicoli utilizzando il rilevatore di oggetti YOLOv5 con il localizzatore DeepSORT per rilevare e tracciare i movimenti delle macchine assegnando loro un numero di identificazione univoco. L'algoritmo di deep learning è in grado di valutare e classificare il grado severità di un incidente ed eventualmente inviare subito una segnalazione all'ospedale più vicino.

Tutte queste soluzioni tecnologiche, che promuovono al contempo l'intelligenza e la condivisione delle informazioni tra gli operatori della sicurezza di un'area urbanistica, mirano alla realizzazione di una città rifunzionalizzata, orientata verso la prevenzione del crimine urbano, in cui i sistemi di videosorveglianza e le

tecnologie di intelligenza artificiale ed elaborazione real-time dei dati contribuiscono a migliorare la qualità di vita del cittadino.

Capitolo 3

Il progetto *s2cities*

In questo capitolo verrà preso in considerazione il processo di sviluppo alla base della soluzione proposta. Dapprima sarà presentata una panoramica generale dei componenti strutturali della piattaforma, in seguito si effettuerà un'analisi approfondita dei casi d'uso e dei requisiti funzionali e non funzionali.

3.1 Architettura generale del sistema

3.1.1 System-on-Chip

Le videocamere di videosorveglianza usate in questo lavoro di ricerca [5] sono dispositivi dotati di un'unità di elaborazione basata su system-on-chip (SoC) che consente funzioni avanzate e potenti analisi basate sulla tecnologia deep learning in modalità edge. Il rilevamento in tempo reale dei gesti della mano che determinano il "Signal for Help" avviene mediante algoritmi di riconoscimento basati su due ben note architetture CNN: MediaPipe e MobileNet. Tali algoritmi sono implementati su una scheda di sviluppo Xilinx Ultra96-V2, dotata di un processore ARM e

logica programmabile. Il sistema operativo Linux che gira sul dispositivo integrato ospita uno strato software in Python che esegue gli algoritmi sopra menzionati e tende un filo comunicativo tra la piattaforma di riconoscimento e il server in cloud all'occorrere dell'evento.

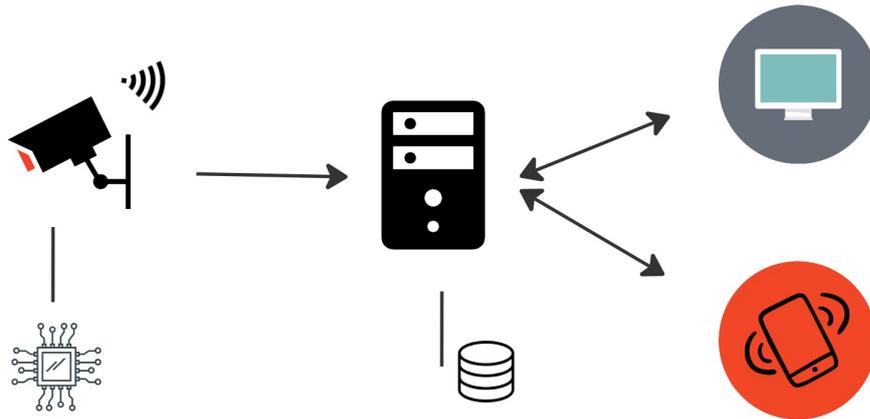


Figura 3.1: Architettura generale

3.1.2 Web Server

Come si evince dalla figura 3.1, il centro nevralgico dell'architettura così formata è rappresentato dal Web Server che consente l'interazione tra i vari dispositivi connessi alla rete. Si è scelto di implementare un Web Server che segue l'architettura REST, esponendo delle REST API attraverso cui è possibile stabilire un dialogo unidirezionale o bidirezionale tra i vari componenti. Il principale vantaggio di questo modello comunicativo, nello scenario proposto, è il totale distacco dall'implementazione hardware delle varie parti, che si realizza mediante l'uso del protocollo web HTTP. Pertanto, non appena l'unità di elaborazione installata sulla videocamera rileva una richiesta di aiuto, quest'ultima invia una HTTP POST Request all'API appositamente costruita e messa a disposizione dal server. Il server

valida, storicizza ed elabora i dati ricevuti dalla videocamera di sorveglianza, e usa apposite librerie per generare un messaggio e inoltrarlo ai client registrati al servizio (questo aspetto funzionale verrà meglio approfondito nel capitolo 5).

In accordo con quelli che sono i principi REST, i dati sono univocamente identificabili tramite un URI e sono rappresentati nel formato testuale JSON, su cui è possibile agire tramite i metodi standard previsti dal protocollo HTTP: POST, GET, PUT/PATCH, DELETE. Tali metodi seguono il paradigma CRUD (Create, Read, Update, Delete) e manipolano le risorse presenti nella base dati restituendo come risposta lo stato dell'operazione (successo o fallimento).

3.1.3 App Mobile

La caratteristica principale che rende questo progetto un progetto innovativo è la messa a punto di un'applicazione mobile in grado di favorire l'interazione tra l'utente e l'ambiente circostante. Lo scopo principale dell'applicazione è quello di avvisare l'utente ogni volta che un comportamento sospetto di violenza è in atto nelle vicinanze ed è ripreso dalle videocamere. Al manifestarsi del segnale di aiuto la videocamera di sorveglianza invia al server il proprio identificativo in modo da geolocalizzare l'aggressione. Il server inoltrerà la richiesta di aiuto agli utilizzatori dell'app che si trovano nella zona, in modo che possano accorrere ed eventualmente a loro volta segnalare l'aggressione ai colleghi o alle forze dell'ordine. La funzionalità primaria di ricevere e inviare notifiche push è affiancata da altre funzionalità secondarie, discusse ampiamente nei paragrafi seguenti. In questa prima fase di rilascio l'applicazione è pensata per essere eseguita sui dispositivi Android, dal momento che utilizza le funzionalità offerte nativamente da questo

sistema operativo, tuttavia si considera la possibilità di usare piattaforme diverse per eventuali sviluppi futuri.

3.1.4 Dashboard

Al fine di garantire una corretta gestione del flusso dei dati è stato necessario infine progettare un pannello di controllo pensato per gli utenti designati ad amministrare e monitorare l'andamento del sistema. Si è scelto di realizzare pertanto un'applicazione Web che fornisca agli utenti autorizzati tutte le funzionalità di cui si ha bisogno, in particolare:

- gestione delle strutture: è possibile registrare una nuova struttura al servizio. Per struttura si intende un edificio o in generale un complesso identificabile da un indirizzo, in cui sono installate delle videocamere di sicurezza e in cui opera un team di vigilanza.
- gestione degli utenti: si può visionare e gestire l'elenco degli utenti che fanno parte del sistema, che siano i membri della security di una determinata struttura (gli utilizzatori dell'app mobile) o gli amministratori di sistema
- gestione delle videocamere di sorveglianza: è possibile registrare le videocamere di sorveglianza associate alla struttura fornendo tutti dettagli necessari riguardanti la loro posizione e le loro caratteristiche.

Il portale Web fornirà un pannello di controllo con funzionalità diverse a seconda del ruolo svolto dall'utente loggato. L'applicativo Web mira a utilizzare le tecnologie di sviluppo Web più moderne per cercare di offrire un'esperienza utente semplice e intuitiva. Tali tecnologie saranno meglio discusse nei capitoli seguenti.

3.2 Analisi dei requisiti di sistema

3.2.1 Scenario di riferimento

I meccanismi di interazione discussi finora possono essere applicati nell'ambito di vari contesti possibili (ospedali, scuole, supermercati, luoghi di lavoro, ecc.). Per quanto sia possibile generalizzare il problema, durante lo studio dei casi d'uso si è fatto riferimento ad uno scenario ipotetico che vede coinvolta una struttura all'interno dell'area del Politecnico, il Campus Borsellino, struttura che accoglie molti studenti e in cui si immaginano installate delle videocamere di sorveglianza. Si farà spesso riferimento a tale scenario nel corso della divulgazione, per meglio comprendere le dinamiche e il funzionamento del sistema e analizzarne i requisiti.

3.2.2 Attori e ruoli

Introdotta il contesto di riferimento, si possono ora identificare gli attori del sistema e i ruoli che giocano all'interno di esso. I consumatori che interagiscono con le applicazioni mobile e desktop sono di tre tipi:

1. Le *guardie di sicurezza* che vigilano l'edificio
2. Gli *amministratori di sistema* dell'edificio
3. I *supervisor di sistema* dell'organizzazione

Per semplicità di linguaggio queste tre figure verranno indicate con i rispettivi ruoli e termini inglesi *guards*, *admins*, *supervisors*.

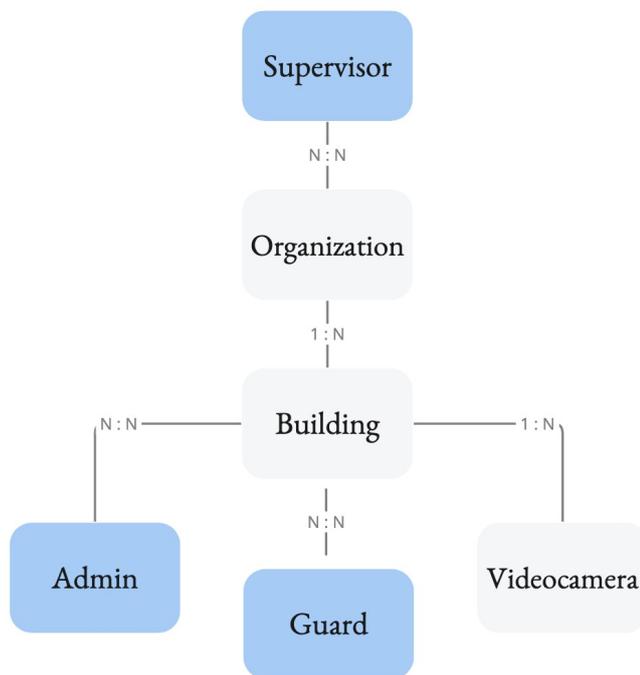


Figura 3.2: Diagramma delle entità

Il diagramma esposto in figura 3.2 rimarca la subordinazione dei ruoli svolti, oltre ad evidenziare il livello di gerarchia esistente tra le varie entità del sistema. Verrà di seguito illustrato.

Gli utenti *supervisor* sono quelli che hanno il più alto grado di privilegi. Uno o più utenti *supervisor* sono associati ad una *organization* che a sua volta può comprendere uno o più edifici (*building*) dislocati sul territorio. Ad esempio, il Politecnico di Torino rappresenta un'organizzazione articolata in strutture didattiche, amministrative e residenziali (di cui il Campus Borsellino fa parte). Tramite le credenziali fornitegli, un utente *supervisor* può accedere all'apposito pannello di controllo dell'applicazione per tenere sott'occhio e monitorare la situazione dell'ambiente in cui lavora. In particolare può:

- visionare tutte le informazioni relative ad una determinata struttura dell'ente presso cui lavora come nome, indirizzo, totale numero utenti, totale numero videocamere ecc. Può inserire un nuovo edificio o eliminare definitivamente uno già esistente.
- fornire le credenziali di accesso agli utenti *admin* e *guard*. Questo implica inserirle nel sistema e associarle ad una o più strutture esistenti.
- registrare, modificare, eliminare le videocamere di un edificio

Gli utenti *admin* a livello logico e funzionale costituiscono una sottocategoria dei supervisor di sistema. Per come è stata progettata la soluzione, i primi idealmente sono a più stretto contatto con gli ambienti organizzativi rispetto ai secondi ed hanno una visione di insieme più ristretta, limitata alle sole strutture di loro competenza. I *supervisor* hanno invece una visione d'insieme globale che include tutte le strutture che appartengono all'organizzazione. Anche gli amministratori di sistema, tramite le credenziali ricevute per email, hanno accesso al proprio pannello di controllo con ruolo *admin*

Se *admin* e *supervisor* sono gli effettivi utilizzatori della piattaforma web, agli utenti *guard* è destinato l'utilizzo dell'applicazione mobile per ricevere le segnalazioni.

I ruoli *supervisor*, *admin* e *guard* non sono mutuamente esclusivi: un generico utente nel sistema può ricoprire anche tutti e tre i ruoli. Si ribadisce ancora una volta che il ruolo di *supervisor* implica quello di *admin*, non vale il contrario.

3.2.3 Requisiti funzionali

L'applicazione mobile dovrà fornire le seguenti funzionalità all'utente *guard*:

1. *Autenticazione*: durante il primo accesso l'utente deve poter eseguire il login inserendo il suo username e il codice segreto inviatogli per email. L'utente avrà precedentemente avuto cura di fornire all'amministratore di sistema di riferimento il proprio indirizzo email per la creazione dell'utenza.
2. *Gestione delle credenziali di accesso*: in seguito al primo accesso, l'utente può decidere di cambiare la password, rispettando i vincoli di sicurezza imposti. Qualora avesse dimenticato la password può richiedere il re-invio delle credenziali specificando lo username.
3. *Impostazione dello stato e del luogo di lavoro*: per abilitare correttamente la ricezione delle notifiche push, l'utente può cambiare in qualsiasi momento il suo stato di servizio, scegliendo tra "in servizio", "occupato", "in pausa" e "fuori servizio" e impostare il suo luogo di lavoro se presta servizio in diverse strutture.
4. *Gestione dei contatti*: l'utente deve avere a disposizione una lista dei colleghi che lavorano con lui e a cui può fare riferimento nell'eventualità di una richiesta di soccorso. Per ognuno di loro sarà possibile reperire informazioni anagrafiche (avatar, nome, cognome, email, telefono) e lo stato.
5. *Ricezione delle notifiche*: le notifiche che riceve l'utente sono di due tipi: messaggi *di allarme* e messaggi *di testo*. I messaggi *di allarme* vengono inviati dal sistema quando viene rilevata una richiesta di soccorso nelle vicinanze. I messaggi *di testo* vengono inviati da una guardia per sollecitare l'intervento di un collega. L'utente riceve una notifica se il suo stato è impostato come "in servizio" e si trova nella medesima struttura in cui è installata la videocamera

che rileva il segnale di aiuto. La ricezione delle notifiche avviene secondo il seguente criterio:

- se l'applicazione è completamente chiusa la notifica push si accoderà nel centro notifiche. Il click comporta l'apertura dell'app e il re-indirizzamento alla relativa schermata di alert
- se l'applicazione è aperta ma non in uso il comportamento è analogo al precedente punto
- se l'applicazione è aperta ed in uso la notifica compare sotto forma di *banner* all'interno dell'app stessa e l'utente può essere reindirizzato alla schermata di alert cliccandoci sopra.

6. *Gestione della segnalazione*: l'apertura di una notifica push comporta la visualizzazione di una schermata di alert il cui contenuto deve evidenziare chiaramente luogo data e ora della segnalazione. L'utente può decidere se prendere in carico la richiesta di soccorso oppure ignorarla. Entrambe le azioni comportano l'apertura di un *ticket* che terrà traccia della segnalazione e delle azioni intraprese. L'utente che ha accettato la richiesta di soccorso si occuperà di accertarsi dell'urgenza, recandosi personalmente sul posto indicato. In casi più gravi in cui è richiesto l'intervento delle forze dell'ordine, l'utente deve poter chiamare la polizia direttamente dall'applicazione, attraverso un pulsante collocato nella schermata di gestione della segnalazione in corso. Sempre qui può attenzionare uno o più colleghi di lavoro, invitando loro un messaggio di aiuto nel prestare soccorso. Una volta gestita l'emergenza si potrà archiviare il ticket segnandolo come "chiuso" aggiungendo una motivazione e opzionalmente un report.

7. *Consultazione delle segnalazioni in gestione*: l'utente può consultare una lista di tutte le segnalazioni aperte, ignorate o già concluse in modo da poter visualizzarne i dettagli in qualsiasi momento o riaprire una gestione precedentemente interrotta.
8. *Consultazione dello storico notifiche*: tutte le notifiche push che arrivano sono raccolte in uno storico notifiche consultabile dall'utente in un *centro notifiche*.
9. *Gestione del profilo*: all'interno di un'area personale deve essere offerta la possibilità di vedere e modificare i propri dati e scegliere un'avatar come immagine di profilo. Deve essere altresì possibile eseguire il logout.

Il pannello di controllo deve fornire i seguenti requisiti funzionali agli utenti *supervisor*:

1. *Autenticazione e gestione credenziali di accesso*: requisiti analoghi a quelli citati sopra per l'applicazione mobile.
2. *Gestione delle strutture*: il primo accesso per l'utente mostrerà la dashboard vuota. Il primo passo da fare è registrare una struttura con nome e indirizzo. Una volta inserita la struttura è possibile associare uno o più *admin*, registrare le guardie e le videocamere.
3. *Gestione degli utenti admin*: l'utente ha la possibilità di visualizzare l'elenco degli *admin* di tutta l'organizzazione o del singolo edificio. Può filtrare la lista cercando per nome, cognome, email o telefono. L'inserimento di nuovi *admin* implica la creazione e contestualmente l'invio per email delle credenziali di accesso.

4. *Gestione delle guardie di sicurezza*: l'utente ha la possibilità di visualizzare l'elenco delle *guards* di tutta l'organizzazione o del singolo edificio. Può filtrare la lista cercando per nome, cognome, email o telefono. Può inserire singolarmente una nuova guardia compilando i campi richiesti o caricare un file .csv per l'inserimento multiplo. L'inserimento di nuove guardie implica la creazione e contestualmente l'invio per email delle credenziali di accesso. Infine l'utente può modificare o eliminare una guardia.

5. *Gestione delle videocamere*: l'utente può visualizzare l'elenco delle videocamere di sorveglianza di una struttura. Può inserire una videocamera, modificarla ed eliminarla. Può effettuare anche qui un caricamento multiplo tramite l'upload di un file .csv.

Il pannello di controllo dell'utente admin deve fornire tutti i requisiti funzionali citati sopra ad eccezione dei punti 2 e 3.

3.2.4 Requisiti non strettamente funzionali

I requisiti non funzionali elencati brevemente qui sotto esprimono dei vincoli sulla qualità del prodotto finale in termini di usabilità e sicurezza.

1. *Interfaccia utente*: l'applicazione Web deve esporre un'interfaccia utente semplice e intuitiva accessibile da qualsiasi postazione della rete. Deve essere sviluppata in TypeScript usando il framework React.js. L'applicazione mobile deve essere sviluppata con il framework React Native e deve funzionare sui dispositivi Android.

2. *Web Services*: il web Server deve esporre delle API REST attraverso l'uso di Node.js. L'interazione tra le varie parti avviene tramite scambi di messaggi in formato JSON.
3. *Persistenza dei dati*: il progetto dovrà fare uso di un database locale, utilizzato per memorizzare tutte le informazioni rilevanti e per permettere l'implementazione di tutte le funzionalità CRUD.
4. *Sicurezza del sistema*: il sistema farà uso di un meccanismo di autenticazione basato su username e password. Le password usate in fase di autenticazione e in generale tutti i codici segreti non sono memorizzati in chiaro nella base dati ma vengono prima sottoposti ad un algoritmo di hash. Occorre validare ogni richiesta http e proteggere l'accesso alle API esposte dal server attraverso un processo di autorizzazione che assicura che l'utente ha i permessi giusti per accedere alla risorsa richiesta.
5. *Implementazione del sistema di notifiche*: l'applicazione utilizzerà le funzionalità offerte nativamente dal sistema operativo Android e farà uso di un servizio esterno, Firebase Cloud Messaging, per l'invio dei messaggi.
6. *Implementazione di un servizio di mail delivery*: il sistema deve gestire in sicurezza l'invio delle mail verso gli utenti e il recupero delle credenziali di accesso.

Gli aspetti sopra elencati verranno trattati con maggior dettaglio nei capitoli 4 e 5 dell'elaborato.

Capitolo 4

Sviluppo e progettazione dell'applicazione Android

Dopo aver presentato il progetto e analizzato i requisiti funzionali si passa ora a discutere la fase di progettazione dell'applicazione mobile. Dapprima si illustreranno le scelte di presentazione all'utente delle funzionalità per poi passare alla definizione dei casi d'uso con la progettazione delle attività principali. Infine verrà presentato il modello documentale della base dati.

Il termine generico *utente* farà sempre riferimento, nel corso del capitolo, all'utente di tipo *guard*.

4.1 Struttura dell'applicazione

4.1.1 Progettazione dell'interfaccia utente

Durante la fase di sviluppo, per determinare l'ossatura del progetto e stabilire le funzionalità e le relazioni tra i vari modelli di schermata dell'applicazione, è stato utile realizzare dei *wireframe* schematici come linea guida. Il software utilizzato per la loro creazione è Miro, uno degli strumenti attualmente più diffusi per la creazione di *wireframe* e *mockup*.

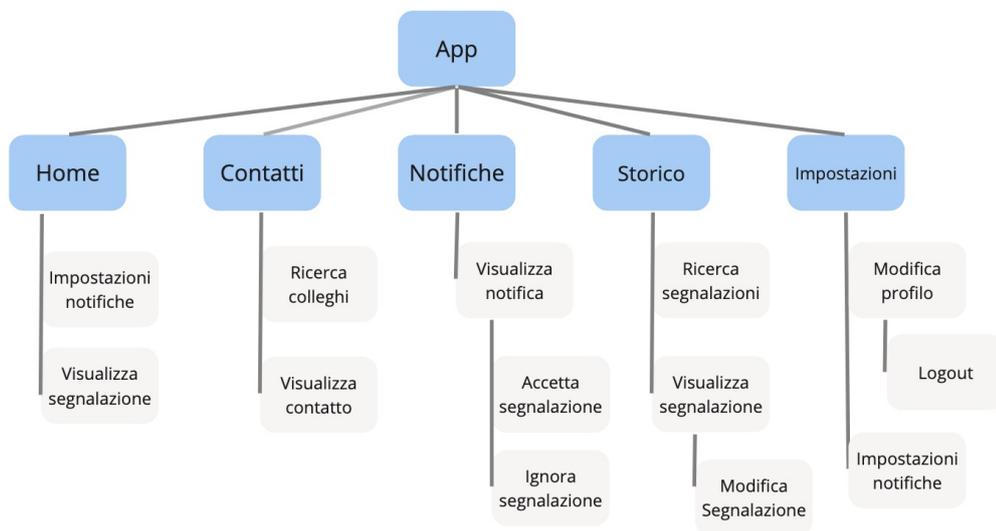


Figura 4.1: Mappa del sito

Nella figura 4.1 viene riportato uno schema creato per sintetizzare la mappa del sito dell'applicazione.

Come si può notare l'applicazione presenta cinque aree distinte:

- La pagina *Home* è il punto di accesso all'app dopo il login. Qui vengono richiamate tutte le principali funzionalità delle altre sezioni.

- La pagina *Contatti* fornisce all'utente l'elenco dei colleghi di lavoro e i dettagli per ciascun collega.
- La pagina *Messaggi* rappresenta il *centro notifiche* dell'applicazione. Una notifica porta con sè un messaggio che può essere una segnalazione di soccorso (*notifica di allarme*) o una sollecitazione (*notifica di testo*). Ogni volta che arriva una notifica questa viene segnata come non letta in questa sezione.
- La pagina *Storico* consente all'utente di gestire le segnalazioni in corso e di archivarle.
- La pagina *Impostazioni* permette di modificare il profilo dell'utente e configurare la ricezione delle notifiche attraverso l'impostazione del suo *stato di servizio*.

4.1.2 Diagramma dei casi d'uso

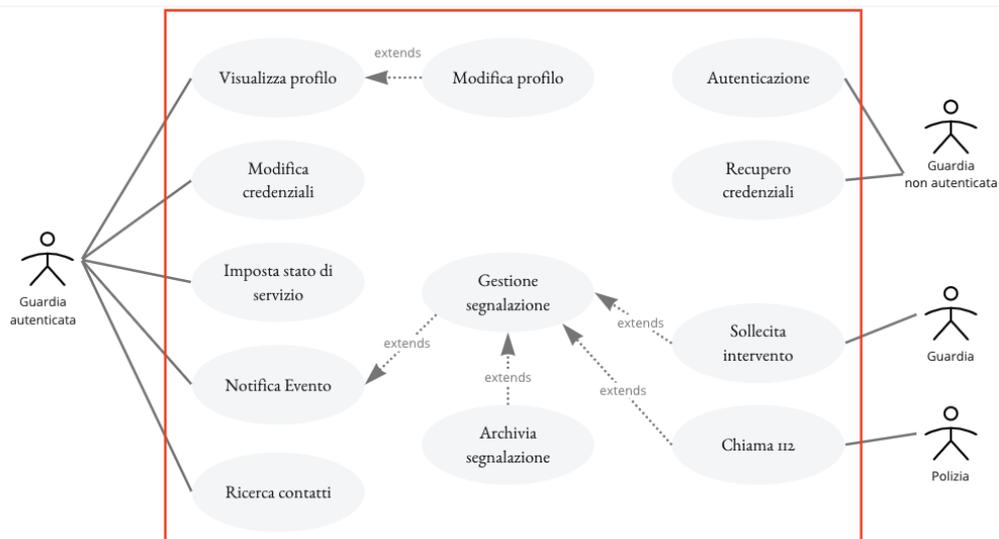


Figura 4.2: Diagramma dei casi d'uso

In contemporanea alla definizione delle schermate principali che compongono l'applicazione avviene la determinazione dei casi d'uso su cui essa si regge. L'analisi dei requisiti per la realizzazione dell'applicazione mobile ha portato ad una classificazione dei comportamenti fondamentali desiderati che segue a grandi linee lo schema riportato in figura 4.2 e che è in questo paragrafo esposta. L'*attore* che interagisce col sistema è la guardia di un team di security che vigila all'interno della struttura e rappresenta il target dell'applicazione. Egli può interagire con i suoi colleghi di lavoro e con le forze dell'ordine. I casi d'uso principali individuati sono:

- Autenticazione
- Visualizzazione profilo
- Modifica profilo
- Recupero credenziali
- Modifica credenziali
- Impostazione stato di servizio
- Notifica Evento
- Gestione segnalazione
- Richiesta intervento polizia
- Richiesta intervento colleghi
- Archiviazione segnalazione
- Ricerca contatti

Il caso d'uso *Notifica Evento* consiste nell'apertura di un messaggio di notifica di un evento di emergenza.

Il caso d'uso *Gestione Segnalazione* è un'estensione del caso d'uso *Notifica Evento* in quanto la segnalazione viene creata nel momento in cui si accetta o si rifiuta la

richiesta di soccorso notificata.

I casi d'uso *Archiviazione Segnalazione*, *Richiesta intervento polizia* e *Richiesta intervento colleghi* estendono il caso d'uso *Gestione Segnalazione* in quanto è necessario che la segnalazione esista e sia nello stato "accettata".

4.2 Progettazione delle attività

L'obiettivo di questa fase è stato definire dettagliatamente il flusso delle azioni nel sistema, facendo attenzione a descrivere con scrupolo il comportamento che esso svolge nel processo decisionale mentre risponde agli eventi che circondano i passaggi coinvolti. La progettazione delle attività, oltre a completare le funzionalità dei casi d'uso, è stata utile per evidenziare l'interazione tra utente e sistema, la sequenza delle azioni svolte dall'utente nonché le risposte e informazioni generate.

4.2.1 Autenticazione

L'utente che vuole utilizzare l'applicazione deve fornire il suo indirizzo email e i suoi dati anagrafici all'amministratore o supervisor di sistema che si occuperà di abilitarlo al servizio registrando l'utenza. Il sistema invierà all'utente una mail contenente un codice segreto da utilizzare come password. Successivamente al login l'utente potrà impostare se desidera una nuova password. Ottenute le credenziali l'utente può eseguire l'autenticazione inserendo negli appositi campi lo username (email) e il codice segreto ricevuto.

Nome	Autenticazione
Attori	Utente
Precondizioni	L'utente ha installato l'applicazione
Flusso degli eventi	<ol style="list-style-type: none">1. L'utente inserisce username e password nella schermata di login e clicca su Login2. Il sistema autentica l'utente e lo rimanda alla schermata principale dell'app
Eccezioni	<ol style="list-style-type: none">2. Se le credenziali fornite non sono valide il sistema permette all'utente di riprovare Si ricomincia dal punto 1.

4.2.2 Recupero credenziali

Se l'utente non ricorda o ha smarrito le proprie credenziali di accesso attraverso la funzionalità *Recupero credenziali* può richiedere il reset della password specificando il proprio indirizzo email. Il sistema invierà per email un link tramite il quale può reimpostare una nuova password.

Nome	Recupero credenziali
Attori	Utente
Precondizioni	L'utente ha installato l'applicazione
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente inserisce il proprio indirizzo email nella schermata di recupero password 2. Il sistema invia una email con un link per reimpostare la password 3. Il cliente clicca sul link 4. Il sistema chiede di digitare una nuova password, due volte 5. L'utente digita la nuova password e conferma 6. Il sistema memorizza le nuove informazioni e aggiorna il profilo dell'utente
Eccezioni	<ol style="list-style-type: none"> 1. Se l'indirizzo email inserito non è associato ad una utenza il sistema non autorizza l'invio della mail 2. Se l'invio della mail fallisce il sistema visualizza un messaggio di errore. Si riprende dal punto 1. 3. Se il link non è valido o è scaduto il sistema reindirizza l'utente alla schermata di login 5.1. Se le password inserite non coincidono il sistema non autorizza l'invio della mail 5.2. Se le password inserite non rispettano i vincoli di sicurezza il sistema non autorizza l'invio della mail

4.2.3 Visualizzazione e modifica del profilo

L'attuale configurazione del sistema prevede un profilo utente base composto da nome, cognome, email, telefono e avatar. Al momento non si ritengono indispensabili altre informazioni anagrafiche che tuttavia potranno essere aggiunte successivamente in base a future necessità.

Nome	Visualizzazione e modifica del profilo
Attori	Utente
Precondizioni	L'utente ha effettuato l'accesso
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente accede al profilo personale nell'area Impostazioni 2. L'utente modifica uno o più campi tra quelli previsti e conferma <ol style="list-style-type: none"> 2.1 L'utente sceglie una immagine del profilo presente nella sua galleria 3. Il sistema memorizza le nuove informazioni e aggiorna il profilo dell'utente
Eccezioni	<ol style="list-style-type: none"> 2. Se i dati non sono validi, al passo 3 il sistema non esegue il salvataggio dei dati <ol style="list-style-type: none"> 2.1 Il sistema permette all'utente di reinserire le informazioni e riprovare <p>Si ritorna allo scenario principale a partire dal passo 1</p>

4.2.4 Impostazione dello stato di servizio

Ogni utente è caratterizzato da uno *stato di servizio* che comprende uno stato (in servizio, fuori servizio, in pausa, occupato) e un luogo di lavoro. Un utente può essere associato a più luoghi di lavoro contemporaneamente. Questo perché si è considerata l'ipotesi che una guardia di sicurezza può prestare servizio in strutture diverse appartenenti alla stessa area di controllo (organizzazione). Ad esempio una guardia che lavora presso il Politecnico di Torino, in una settimana di lavoro, può prestare servizio al Cmapus Borsellino ma anche in un dipartimento dislocato all'interno della stessa area. L'impostazione dello stato di servizio consiste nella configurazione di questi due parametri (stato e struttura). E' un caso d'uso molto importante che determina la corretta ricezione delle notifiche push e il conseguente intervento: al momento della rilevazione del segnale di aiuto, il sistema invierà una notifica a tutte le guardie che rispettato i seguenti vincoli:

- stato di servizio: "in servizio"
- struttura: struttura da cui proviene il segnale di aiuto

Dal momento che è possibile che l'utente sia associato contemporaneamente a più strutture, è sua cura scegliere opportunamente quella giusta tra quelle a lui proposte.

Nome	Impostazione dello stato di servizio
Attori	Utente
Precondizioni	L'utente ha effettuato l'accesso
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente accede alla schermata di configurazione delle notifiche nelle Impostazioni 2. L'utente seleziona il luogo di lavoro tra quelli proposti e lo stato di servizio e conferma 3. Il sistema aggiorna le informazioni di stato sull'utente 4. Il sistema aggiorna la lista dei colleghi in base al luogo di lavoro selezionato

4.2.5 Ricerca e visualizzazione contatti

All'utente viene data la possibilità di avere sempre a disposizione una lista aggiornata delle guardie di sicurezza che lavorano con lui nella stessa struttura. Per ogni contatto sono disponibili i seguenti dati: immagine di profilo, nome, cognome, telefono, email, struttura in cui lavora, stato (in servizio, fuori servizio, occupato, in pausa). Questa caratteristica risulterà importante soprattutto quando, nel caso di una segnalazione in corso, si vuole sollecitare l'intervento di un collega che è in servizio nelle vicinanze. In qualsiasi momento l'utente può consultare la lista dei colleghi nella sezione *Contatti* dell'applicazione e effettuare una ricerca digitando nel campo di ricerca una parola chiave tra nome, cognome, telefono e email del collega desiderato. In questa versione dell'app non sono presenti ulteriori criteri di filtraggio dei dati.

Nome	Ricerca e visualizzazione contatti
Attori	Utente
Precondizioni	L'utente ha effettuato l'accesso
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente visualizza la lista dei colleghi nell'area Contatti 2. L'utente inserisce una parola chiave nella barra di ricerca per filtrare la lista e selezionare l'elemento desiderato 3. Il sistema fornisce le informazioni relative all'utente selezionato
Eccezioni	<ol style="list-style-type: none"> 2. Se la parola chiave non produce alcun risultato si riprende dal punto 1.

4.2.6 Notifica Evento

Il sistema invia all'utente *in servizio* una notifica push ogni volta che viene rilevato un evento di emergenza nella zona. Una notifica consiste in un banner che compare sullo schermo e mostra un breve messaggio che suggerisce all'utente di aprire l'applicazione. La sua apertura comporta la visualizzazione di una schermata di *alert* in cui sono esposti i dettagli della segnalazione in corso: da dove arriva il segnale, ora e data di rilevazione e l'identificativo della videocamera coinvolta. L'utente è posto davanti alla scelta di accettare la richiesta di soccorso o ignorarla. Se l'utente prende in carico la segnalazione questa comparirà come "Nuova segnalazione" nella sezione *Storico* e sarà gestita come nel caso d'uso successivo. Se l'utente decide di ignorare la segnalazione essa comparirà nello Storico con stato "Segnalazione ignorata". E' bene ricordare che la ricezione di una notifica sul dispositivo si manifesta in uno qualunque dei seguenti momenti: quando l'applicazione è in uso oppure quando è aperta ma non in primo piano oppure quando è completamente chiusa.

Nome	Richiesta di soccorso
Attori	Utente
Precondizioni	L'utente ha configurato correttamente la ricezione delle notifiche
Flusso degli eventi	<ol style="list-style-type: none"> 1. Il sistema invia all'utente una notifica che segnala l'emergenza 2. L'utente apre il messaggio e visualizza la richiesta di soccorso 3. L'utente accetta la richiesta di soccorso 3. Il sistema salva la pratica di soccorso come "nuova" e re-indirizza l'utente nella sezione Storico dove può gestire la segnalazione
Eccezioni	<ol style="list-style-type: none"> 2. L'utente non apre il messaggio 2.1 Il sistema indica il messaggio come non letto nella sezione Messaggi 3. L'utente decide di ignorare il messaggio 3.1 Il sistema salva la pratica di soccorso come "ignorata" e re-indirizza l'utente nella sezione Storico dove può gestire la segnalazione

4.2.7 Gestione delle segnalazioni

Una segnalazione presente nella sezione *Storico* può trovarsi in uno dei seguenti stati: *nuova*, *ignorata*, *gestita*.

Una segnalazione è *nuova* se è appena stata accettata la richiesta di soccorso. La schermata si presenta con una intestazione di colore rosso, per indicare l'urgenza e per evidenziare la distinzione con le altre segnalazioni concluse (colore verde) e ignorate (colore blu). Attraverso questa schermata, l'utente, che si è accertato del caso, può sollecitare i suoi colleghi di lavoro a collaborare attraverso l'apposita funzione messa a disposizione. Il sistema invierà loro un messaggio per segnalare la necessità di intervento. La schermata presenta anche un pulsante di SOS per consentire l'interazione diretta con le forze dell'ordine.

Nome	Richiesta intervento colleghi
Attori	Utente
Precondizioni	L'utente ha ricevuto una notifica e ha accettato la richiesta di soccorso
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente visualizza la schermata della segnalazione in corso 2. L'utente richiede l'intervento dei colleghi 3. Il sistema invia una notifica agli utenti selezionati
Eccezioni	<ol style="list-style-type: none"> 2. Se nessun collega è disponibile il pulsante di Invio è disabilitato 3. Se l'invio non è riuscito viene visualizzato un messaggio di errore e si ritorna al punto 2.

Nome	Richiesta intervento polizia
Attori	Utente
Precondizioni	L'utente ha ricevuto una notifica e ha accettato la richiesta di soccorso
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente visualizza la schermata della segnalazione in corso 2. L'utente clicca il pulsante di SOS per contattare il 112 3. Il sistema richiede la conferma di far partire la chiamata 4. L'utente clicca su "Chiama"

4.2.8 Archiviazione della segnalazione

L'utente, a fine emergenza, può modificare lo stato della segnalazione e quindi decidere di archivarla dando informazioni su come la segnalazione è stata gestita ed eventualmente producendo un breve report su quanto accaduto. Il sistema salva le informazioni e cambia lo stato della segnalazione da *nuova* a *gestita*. Mentre una segnalazione *ignorata* può tornare nello stato *gestita*, una segnalazione *gestita* non può essere più riaperta. Tutte le segnalazioni con stato *nuovo* che non vengono

gestite dall'utente entro la fine della giornata vengono marcate automaticamente dal sistema come gestite.

Nome	Archiviazione della segnalazione
Attori	Utente
Precondizioni	L'utente ha ricevuto una notifica e ha accettato la richiesta di soccorso
Flusso degli eventi	<ol style="list-style-type: none"> 1. L'utente seleziona nella sezione Storico la segnalazione aperta che intende archiviare 2. Il sistema richiede all'utente di aggiungere una descrizione su quanto accaduto e compilare un report 3. L'utente inserisce i dati e conferma 4. Il sistema salva in nuovo stato della segnalazione
Eccezioni	<ol style="list-style-type: none"> 3. Se i campi non sono correttamente compilati il sistema non consente l'archiviazione della pratica Si riprende dal punto 2.

4.3 Progettazione Dati

Prima di procedere con l'implementazione del modulo server è stato necessario capire come rappresentare i dati scambiati tra le varie interazioni viste nella sezione precedente. L'applicazione mobile non gestirà un vero e proprio database locale, ma incapsulerà i dati provenienti dal server in delle classi che modellano le strutture dati su cui è basato il database. Si è scelto di utilizzare MongoDB come base dati non relazionale. Sebbene sia un database *schema-less* e non richieda che i documenti abbiano la stessa struttura, la maggior parte delle collezioni modellate risulteranno omogenee per facilitare lo sviluppo, limitare il margine di errore e migliorare la comprensione stessa del codice.

4.3.1 Il formato JSON

Invece di memorizzare l'informazione nelle righe di una tabella, MongoDB memorizza dati strutturati utilizzando un formato simile a JSON (JavaScript Object Notation). Si tratta di un formato di dati facile da leggere e leggero per lo scambio di dati, il che significa che richiede meno larghezza di banda per essere trasmesso tra client e server rispetto ad altri formati più pesanti, come XML. Supporta la struttura gerarchica dei dati, consentendo di annidare oggetti e array all'interno di altri oggetti e array. Questo è utile per rappresentare dati complessi in modo organizzato e accessibile. Inoltre si integra perfettamente con le applicazioni come questa che utilizzano JavaScript, essendo basato sulla sintassi degli oggetti JavaScript, e questo lo rende ideale per lo sviluppo frontend perchè consente una manipolazione facile e diretta dei dati ricevuti dal server senza la necessità di conversioni complesse.

4.3.2 Le relazioni tra i dati

In MongoDB, le relazioni tra i dati sono gestite in modo diverso rispetto ai tradizionali database relazionali. E' possibile implementare relazioni tra documenti utilizzando due principali approcci:

1. Riferimenti: Questo approccio prevede l'uso di campi che contengono valori che fanno riferimento ad altri documenti. Ad esempio, si potrebbe avere un campo in un documento che contiene l'ID di un altro documento in un'altra collezione. In questo modo, si stabilisce una relazione tra i documenti. Tuttavia, è importante notare che la gestione delle relazioni deve essere effettuata manualmente dall'applicazione.

2. Documenti annidati: In questo approccio, i dati correlati sono incorporati direttamente all'interno del documento principale. Ad esempio, invece di avere un riferimento a un documento separato, si possono incorporare tutti i dati relativi all'interno del documento padre. Questo è utile quando i dati correlati sono piccoli e non cambiano spesso.

Guard	User	Building
<pre>{ "_id": ObjectId, "firstname": String, "lastname": String, "status": Number, "userId": ObjectId, "buildingsIds": ObjectId[], "organizationId": ObjectId, "workplaceId": ObjectId, "phone": String, "avatar": String, }</pre>	<pre>{ "_id": ObjectId, "username": String, "psw": String, "role": String[] }</pre>	<pre>{ "_id": ObjectId, "name": String, "location": String, "addressLine1": String, "addressLine2": String, "guardsIds": ObjectId[], "camsIds": ObjectId[], "adminsIds": ObjectId[] }</pre>
Videocamera	Alert	Action
<pre>{ "_id": ObjectId, "buildingId": ObjectId, "name": String, "model": String, "status": Number, "location": Object }</pre>	<pre>{ "_id": ObjectId, "camSerial": String, "camName": String, "buildingName": String, "location": Object, "timestamp": String, "guardsIds": Object[] }</pre>	<pre>{ "_id": ObjectId, "alertId": ObjectId, "guardId": ObjectId, "status": Number, "report": String, "timestamp": String, "markedAd": String }</pre>

Figura 4.3: Progettazione dei dati principali

In base ai requisiti del progetto precedentemente analizzati, l'approccio più frequentemente utilizzato sarà il primo, soprattutto quando sono presenti relazioni multi-a-molti, in quanto è più facile mantenere la coerenza dei dati durante le

operazioni CRUD. Per modificare i dati di un documento occorrerà una sola scrittura e si ha la garanzia che tale modifica sarà valida per tutti i documenti che lo referenziano. Dal momento che le informazioni in gioco sono tante, la scelta di usare un identificativo per collegare i documenti tra loro evita il fenomeno della *ridondanza* dei dati, mantendendo la dimensione dei documenti (e quindi del database) più contenuta. La figura 4.3 mostra ad esempio alcuni dei modelli di dati più importanti e come questi sono relazionati tra di loro.

La collezione che modella la classe Guard, oltre a contenere una serie di informazioni personali come nome, cognome, telefono, avatar e stato di servizio, ospita al suo interno dei riferimenti a documenti esterni. Per esempio, una guardia di sicurezza che opera all'interno del Politecnico è rappresentato da un oggetto *Guard*. Esso è associato ad uno *User* che modella le credenziali utente, ad una *Organization*, il Politecnico, e a più *Buildings* che modellano le strutture presenti nell'area universitaria. Il *workspaceId* fa riferimento ad una di queste e indica la struttura presso cui la guardia presta servizio. Il vantaggio dell'uso dei riferimenti è evidente: se un utente modifica le proprie informazioni, queste modifiche verranno riflesse in tutti i documenti associati a quell'utente senza la necessità di aggiornare in modo massivo ogni singolo documento che lo contiene. L'onere si sposta sul lato applicativo: per ricostruire i dati di un documento coinvolto in una relazione si devono effettuare tante letture quanti sono i documenti referenziati.

Capitolo 5

Implementazione della soluzione

Questo capitolo si concentrerà sugli aspetti tecnici e implementativi della soluzione proposta. L'implementazione del progetto *s2cities* è stata divisa in tre parti principali, che sono state sviluppate nel seguente ordine:

1. implementazione *web server*;
2. implementazione *admin dashboard* e *supervisor dashboard*;
3. implementazione *app mobile*

Se inizialmente è stato necessario sviluppare dapprima un server e avviare una base dati per gettare le fondamenta, questo ordine di sviluppo è diventato sempre più ininfluente, man mano che il progetto ha preso forma, fino a gestire componenti back-end e front-end insieme. Le prossime pagine seguono il processo di codifica a partire dalle tecnologie e strumenti utilizzati fino alla presentazione del prodotto

finale. Si menzionerà l'architettura MVC e si vedrà come ciascuno dei componenti del pattern MVC si manifesta in S2CITIES. Ci si soffermerà sul meccanismo di invio e ricezione delle notifiche push spiegando come funziona e come è stato implementato il servizio.

5.1 Uno sguardo alle tecnologie utilizzate

Durante la prima fase di sviluppo del progetto l'azienda di supporto ha messo a disposizione due importanti strumenti di lavoro:

1. un *template* di partenza (sia per la parte back-end che per la parte front-end): definisce la struttura di base del progetto, impostando un albero di directory e file template da cui partire con lo sviluppo full-stack. Il template ha previsto una configurazione iniziale basata su Node.js e l'installazione preliminare di alcuni pacchetti indispensabili al suo funzionamento come Next.js, React, Axios, Typescript, Redux, MongoDB e altri ancora.
2. un *generatore di codice*: uno strumento utile per evitare di scrivere del codice boilerplate all'interno del programma. Molte sezioni di codice infatti si ripetono con modifiche minime o nulle. Si tratta di un pacchetto Node.js che usa il generatore Yeoman come strumento di *scaffolding* per automatizzare una serie di operazioni di gestione e distribuzione del software tramite alcune semplici direttive dettate dal programmatore [13].

L'utilizzo di uno scheletro impostato e del generatore di codice ha agevolato molto il lavoro di organizzazione del codice favorendone una maggiore leggibilità e comprensione. Ha inoltre velocizzato il processo di sviluppo e ridotto la quantità di codice boilerplate generato.

Come già accennato in precedenza, il motore su cui si basa l'intero sviluppo full-stack è Node.js. Si tratta di un' ambiente runtime JavaScript che consente di eseguire codice JavaScript non solo lato front-end ma anche sul server Web. In questo modo si utilizza un unico linguaggio di programmazione per l'intera applicazione. Uno dei maggiori punti di forza di Node.js è che utilizza l'architettura *Single Threaded Event Loop* per gestire più connessioni allo stesso tempo (questo aspetto verrà approfondito in seguito). Inoltre elabora le richieste in modo non bloccante il che lo rende efficiente e adatto allo sviluppo del pannello di controllo per l'admin/supervisor, trattandosi di una *Single Page Application* (SPA) in cui l'intera applicazione viene caricata in una singola pagina mentre la richiesta di componenti specifici avviene in background. Grazie a Node.js è stato possibile usufruire di numerosi framework e librerie che hanno facilitato il lavoro di sviluppo. Node.js infatti ha un'organizzazione modulare: all'interno del codice template la cartella `node_modules` incorpora dei pacchetti (*modules*) di codice JavaScript che vengono utilizzati per estendere le funzionalità di base di Node.js. Per installare tali pacchetti esterni nel progetto, è stato usato il Node Package Manager (NPM), lo storage open-source che raccoglie oltre un milione di pacchetti ufficiali o creati dagli sviluppatori stessi. Una volta installato un pacchetto Node.js, il suo nome e la rispettiva versione vengono registrati nel file `package.json` che descrive le proprietà del progetto. Viene qui riportata una breve descrizione delle principali dipendenze utilizzate:

- Next: consente di avviare un server di sviluppo su `http://localhost:3000` e creare un applicazione React con il rendering lato server (*Server-side Rendering*)
- React: è una libreria JavaScript open-source basata su componenti per la

creazione di interfacce utente veloci e dinamiche

- Expo: è un framework che estende React Native per sviluppare app native iOS a Android
- Material UI: è una libreria di componenti React che implementa le specifiche di Google Material Design, utile per realizzare gli elementi della Dashboard
- Redux: è un contenitore di stato utilizzato per centralizzare la logica applicativa lato front-end
- Axios: è un pacchetto che permette di effettuare richieste HTTP e interagire con le API
- Mongodb: è il driver ufficiale di MongoDB, fornisce l'API per i database a oggetti MongoDB in Node.js
- Moment: è una libreria di gestione delle date in JavaScript che consente di analizzare, validare, manipolare e formattare le date

Come strumento di versionamento del codice si è scelto di usare Git, un sistema di controllo di versione distribuito e open-source, ideato per gestire progetti anche di grandi dimensioni in modo veloce ed efficiente [14]. Per una migliore gestione del codice, il progetto *s2cities* è stato suddiviso in due micro-progetti corrispondenti a due repositories distinte su github:

- *s2cities-frontend*
- *s2cities-backend*

La repository *s2cities-frontend*

La repository *s2cities-frontend* contiene il codice relativo allo sviluppo mobile del progetto e utilizza la piattaforma Expo che L'applicazione mobile, oggetto di questa

tesi, è stata scritta usando il framework React Native che, come il framework React.js, ha permesso un approccio basato su componenti, blocchi di costruzione utilizzati, personalizzati e combinati insieme per creare l'interfaccia utente. I componenti racchiudono al loro interno il codice nativo e interagiscono con le API native tramite la programmazione dichiarativa di React e il Javascript. Il codice viene convertito in codice nativo, realizzando un'app cross-platmform e evitando di gestire le specificità del sistema operativo. Sebbene con React Native sia possibile sviluppare app per iOS e Android utilizzando un'unica base di codice, l'applicazione *s2cities* è pensata per essere eseguita unicamente su dispositivi Android, dal momento che non è previsto, in questo primo rilascio dell'app, il supporto alle notifiche push per altre piattaforme.

Quando viene avviata l'app con React Native, viene avviato un JavaScriptEngine (su Android, React Native utilizza V8 come motore JavaScript predefinito) che esegue il codice JavaScript e gestisce l'interazione tra il codice JavaScript e il codice nativo Android. Ci possono essere situazioni implementative particolari per cui è necessario scrivere moduli nativi personalizzati ma non è stato questo il caso dato che al momento l'unica funzionalità nativa necessaria è quella delle notiche push.

Lo sviluppo del progetto React Native è stato reso più facile grazie all'uso della piattaforma Expo che ha semplificato il processo di installazione e configurazione di tutte le librerie necessarie. Expo ha fornito un insieme completo di tooling integrato, inclusi un ambiente di sviluppo locale, un client Expo per la visualizzazione e il debug dell'app sul dispositivo fisico e virtuale, e un servizio di distribuzione *Over-the-Air* (OTA) per la distribuzione di aggiornamenti dell' app senza la necessità di passare per gli app store.

La libreria *expo* usata in *s2cities-frontend* consente di avviare localmente un server

di sviluppo su `http://localhost:8081`. L'applicazione nelle prime fasi è stata testata su Expo Go, uno strumento che accelera il processo di codifica poiché permette di eseguire codice JavaScript direttamente sul dispositivo senza attendere il processo di build. La generazione dei file binari per l'installazione dell'app sul dispositivo è demandata in fase di compilazione del codice, durante la quale viene generata la cartella android con il codice nativo. In questa fase del progetto, per testare le funzionalità native dell'app, è risultato necessario il supporto della libreria expo-dev-client e l'uso di un development build al posto di Expo Go.

La repository `s2cities-back`

Mentre `s2cities-frontend` contiene esclusivamente codice front-end, la repository `s2cities-backend`, a dispetto del nome, ingloba due applicazioni in una: una parte server ed una parte client. La repository `s2cities-backend` è nello specifico un'applicazione scritta in Typescript che fa uso del framework Next.js per realizzare sia il web server che il portale web per gli amministratori di sistema. Next.js è infatti un framework React che, attraverso le sue proprietà di rendering ibrido, permette di creare degli API endpoints e allo stesso tempo delle pagine web che interagiscono con essi. Questo avviene grazie ad un meccanismo di routing basato su *pagine*: tutti i file e tutte le cartelle create all'interno della cartella di progetto `pages` vengono automaticamente convertiti in percorsi.

```
- pages
  - admin-dashboard
    - index.tsx
  - api
    - guards
      - [id]
        - actions
          - index.ts
        - buildings
```

```
    - index.ts
- contacts
    - index.ts
- messages
    - index.ts
index.ts
```

Lo script qui sopra presenta una panoramica parziale della cartella `pages` dell'applicazione Next.js. Si vogliono evidenziare qui i due punti di partenza dell'applicazione. La cartella `admin-dashboard` utilizza i componenti React per realizzare una SPA, ovvero il pannello di controllo dell'admin/supervisor di sistema. Ogni file TypeScript rappresenta una pagina del sito e il nome del cartella determina l'URL della pagina. Ad esempio, il file `index.tsx` rappresenterà la homepage della SPA e verrà associato all'URL `/admin-dashboard`. La cartella `api` concentra tutti gli endpoint mappandoli in modo analogo alle corrispondenti pagine, realizzando così la parte back-end dell'applicazione. Per ognuno dei file `index.ts` presenti nella cartella `api`, Next.js genera un API endpoint munito di una sua logica applicativa. Ad esempio i contatti di una guardia saranno reperibili alla route `/api/guards/[id]/contacts`. In questo e in molti altri casi la directory `[id]` è dinamica (*dynamic routing*) per gestire percorsi complessi in cui i parametri risultano essere indeterminati.

Un'altra delle caratteristiche principali di Next.js riguarda l'ottimizzazione delle immagini. Consiste nel ridurre automaticamente la dimensione delle immagini durante la fase di compilazione del codice per migliorare le prestazioni del sito. Inoltre Next.js esegue il pre-rendering delle pagine generando in anticipo il codice HTML di ogni pagina insieme al minimo codice JavaScript che deve essere eseguito attraverso un processo noto come Hydration [15]. Queste tecniche contribuiscono a migliorare le prestazioni dell'applicazione e a ridurre il tempo di caricamento

complessivo.

Si è detto che uno dei vantaggi che Node.js offre è la possibilità di usare un unico linguaggio di programmazione sia lato client che lato server. In entrambi i micro-progetti si è scelto di usare Typescript, un superset di JavaScript che aggiunge tipizzazione statica e funzioni avanzate a JavaScript. I due framework Expo e Next.js offrono un supporto integrato per l'uso di questo linguaggio, il cui utilizzo viene abilitato nel file di progetto `tsconfig.json`.

5.2 Implementazione del Web Server

5.2.1 Il pattern MVC

La struttura del progetto *s2cities-backend* rispecchia molto lo stile di un'applicazione di tipo Model View Controller (MVC), sebbene non segua in modo rigido le convenzioni alla base di questo paradigma. Il pattern MVC è un pattern architetturale che separa un'applicazione in tre componenti logici distinti: il modello, la vista e il controllore.

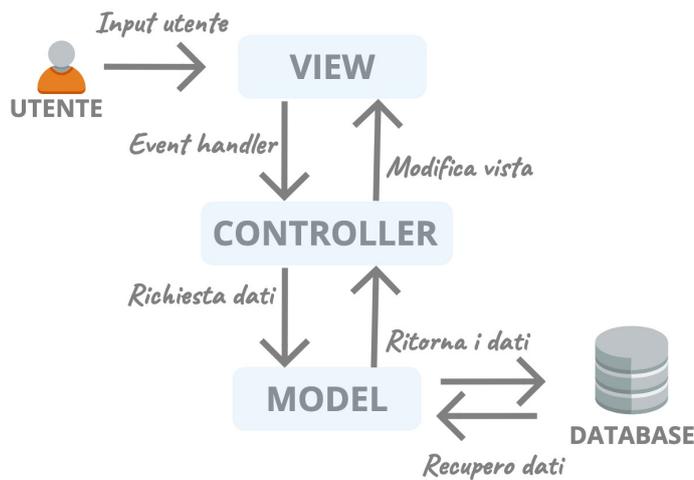


Figura 5.1: Architettura MVC

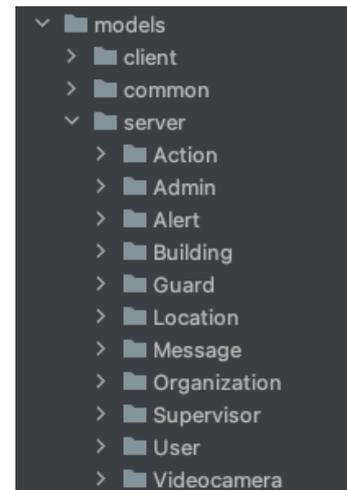


Figura 5.2: Models folder

La figura 5.1 mostra il flusso dei dati in un'architettura MVC.

La *Vista* è il componente logico responsabile della presentazione dei dati agli utenti e dell'interazione con essi. Riceve i dati dal controllore e li presenta in un formato che è comprensibile all'utente. Nella soluzione qui proposta la *Vista* è costituita dall'intera applicazione React e React Native che rappresentano le interfacce grafiche tramite cui l'utente interagisce. Il disaccoppiamento dei dati di sistema dalla loro rappresentazione ha portato ad una maggiore riusabilità del codice: è stato facilmente possibile riutilizzare gli stessi Model e Controller con viste diverse.

La *Vista* si interfaccia al *Controllore*, unico intermediario tra *Vista* e *Modello*. Il *Controllore* riceve gli input dell'utente dalla *Vista*, elabora le richieste e aggiorna il *Modello* di conseguenza. Tutta la logica di business si concentra in questo componente che, nell'applicazione Next.js, si esprime attraverso il meccanismo delle pagine esaminato nella sezione precedente. Ogni endpoint è mappato ad una sua

route ed è associato ad un file di *handler* che gestisce la richiesta, validandone il contenuto e formulando una risposta dopo aver eventualmente interagito col modello. Tutti gli endpoint sono inseriti nella cartella di progetto `endpoints`.

Il *Modello* rappresenta la struttura dei dati dell'applicazione e definisce le logiche di accesso ai dati. E' indipendente dalla *Vista* e dal *Controllore*. La cartella di progetto `models` contiene tutte le classi che modellano le strutture dati necessarie progettate nel capitolo precedente e visibili in figura 5.2. Ognuna di queste classi presenta le seguenti caratteristiche:

- implementa l'interfaccia `WithId<TSchema>` di mongodb che aggiunge un campo identificativo ad ogni documento inserito nella collezione
- è fornita degli attributi atti a definire l'oggetto che rappresenta
- è dotata di metodi, statici e non, che regolano l'interazione con la base dati. Tali metodi hanno a che fare con l'inserimento, la modifica, la ricerca e l'eliminazione di uno o più documenti.

5.2.2 Accesso alle risorse

Eseguendo il progetto `Next.js` in modalità "developer", il server web si presenta in ascolto sulla porta 3000 della macchina locale `http://localhost:3000`. La rappresentazione della risorsa è veicolata nel formato `Json` per rappresentare classi di oggetti o attributi ed è facilmente accessibile tramite un identificatore globale (URI). Tali risorse sono manipolate dalle REST API attraverso i metodi HTTP (GET, POST, PUT, PATCH, DELETE). I servizi di accesso alle risorse sono stati realizzati seguendo le direttive specificate nella tabella seguente:

Tabella 5.1: Accesso alle risorse

Metodo	URI	Risorsa
POST	http://localhost:3000/api/auth/login	Autenticazione dell'utente
GET	http://localhost:3000/api/admins/-[adminId]/buildings	Edifici associati ad un admin
GET	http://localhost:3000/api/alerts/[guardId]	Alerts di una guardia
GET	http://localhost:3000/api/buildings/-[buildingId]	Edificio
GET	http://localhost:3000/api/buildings/-[buildingId]/admins	Tutti gli admin di un edificio
GET	http://localhost:3000/api/buildings/-[buildingId]/guards	Tutte le guardie di un edificio
GET	http://localhost:3000/api/buildings/-[buildingId]/videocameras	Tutte le videocamere di un edificio
GET	http://localhost:3000/api/guards/-[guardId]/actions	Azioni di soccorso associate ad una guardia
GET	http://localhost:3000/api/guards/-[guardId]/messages	Messaggi di una guardia
GET	http://localhost:3000/api/guards/-[guardId]/buildings	Edifici associati ad una guardia
GET	http://localhost:3000/api/organizations/-[orgId]/buildings	Edifici di un'organizzazione
GET	http://localhost:3000/api/organizations/-[orgId]/guards	Tutte le guardie di un'organizzazione
GET	http://localhost:3000/api/organizations/-[orgId]/admins	Tutti gli admin di un'organizzazione
GET	http://localhost:3000/api/supervisors/[id]/-organizations	Le organizzazioni associate a un supervisor

Continua alla pagina successiva

Tabella 5.1: Accesso alle risorse (cont.).

Metodo	URI	Risorsa
POST	http://localhost:3000/api/actions	Inserimento di un'azione di soccorso
POST	http://localhost:3000/api/admins	Inserimento di un admin
POST	http://localhost:3000/api/guards	Inserimento di una guardia
POST	http://localhost:3000/api/users	Inserimento di un utente (credenziali)
POST	http://localhost:3000/api/supervisors	Inserimento di un supervisor
POST	http://localhost:3000/api/alerts	Invio di un alert signal
POST	http://localhost:3000/api/buildings	Inserimento di un edificio
POST	http://localhost:3000/api/forgetPassword	Ripristino credenziali
POST	http://localhost:3000/api/notifications	Invio di una notifica
POST	http://localhost:3000/api/videocameras	Inserimento di una videocamera
POST	http://localhost:3000/api/videocameras/enrollMany	Le organizzazione associate a un supervisor
POST	http://localhost:3000/api/guards/enrollMany	Inserimento multiplio guardie
POST	http://localhost:3000/api/admins/enrollMany	Inserimento multiplio admin
PATCH	http://localhost:3000/api/actions/[actionId]	Modifica azione

Continua alla pagina successiva

Tabella 5.1: Accesso alle risorse (cont.).

Metodo	URI	Risorsa
PATCH	http://localhost:3000/api/guards/[guardId]	Modifica guardia
PATCH	http://localhost:3000/api/buildings/-[buildingId]	Modifica edificio
PATCH	http://localhost:3000/api/videocameras/-[camId]	Modifica videocamera
PATCH	http://localhost:3000/api/messages/[msgId]	Modifica messaggio
PATCH	http://localhost:3000/api/admins/[adminId]	Modifica admin
DELETE	http://localhost:3000/api/admins/-[adminId]	Eliminazione di un admin
DELETE	http://localhost:3000/api/buildings/-[buildingId]	Eliminazione di un edificio
DELETE	http://localhost:3000/api/guards/[guardId]	Eliminazione di una guardia

Dopo aver creato i modelli e progettato il database, ogni endpoint è stato aggiunto individualmente e in modo incrementale seguendo un ordine di codifica che può essere così riassunto:

1. codifica delle *interfaces*: ogni endpoint espone una interfaccia che detta le regole sullo scambio di dati tra client e server. In particolare viene stabilito il formato dei tipi `Payload`, `QueryStringParameters` e `SuccessResponse`. Come indica il nome, il tipo `Payload` modella i dati trasmessi tra client e server in una richiesta HTTP che in Next.js è rappresentata dall'interfaccia `NextApiRequest`. Il tipo `QueryStringParameters` regola eventuali parametri della query presenti nell'URL della richiesta. Infine `SuccessResponse` definisce cosa viene inviato come risposta attraverso l'interfaccia `NextApiResponse`.

2. codifica delle *validations*: ogni richiesta che giunge al singolo endpoint viene sottoposta ad un processo di validazione dei dati. Per fare questo ci si è affidati a Yup, una libreria JavaScript che permette di definire uno schema di validazione per i diversi tipi di dati in ingresso (stringhe, numeri, array, oggetti, ecc). Attraverso il metodo `validate()`, messo a disposizione dalla libreria, si valuta se i campi degli oggetti `Payload` e `QueryStringParameters` rispettano le regole di validazione imposte.
3. codifica degli *handler*: se la validazione ha successo si procede con l'applicazione della logica di business che caratterizza in maniera diversa ciascun endpoint. Vengono gestiti i casi di eccezione che possono derivare ad esempio da un token scaduto, oppure da una incoerenza verificatasi nel database o da una azione priva di autorizzazione. In tutti questi casi viene generato un oggetto `ErrorResponse` che conterrà un messaggio e il codice di errore http pertinente al caso. Se tutto va bene invece viene creata una risposta di successo (`SuccessResponse`) con uno stato che segue lo standard numerico "2xx" e con eventualmente un messaggio e dei dati. Se la validazione non ha successo viene inviata una risposta con codice di stato 400 (Bad Request).

L'accesso a tutte le API REST (ad eccezione del percorso `/api/auth/login`) è protetto attraverso l'utilizzo di un JSON Web Token (JWT), un token che può essere cifrato e firmato tramite una chiave disponibile solo da colui che lo ha effettivamente generato [16] in modo da garantire l'autenticazione e l'autorizzazione degli utenti al servizio. Ogni volta che un utente autenticato desidera accedere a un'API protetta, deve includere il token JWT nelle intestazioni della richiesta http, in particolare viene inviato nell'intestazione `Authorization` come tipo di

autenticazione Bearer.

5.2.3 Gestione della concorrenza

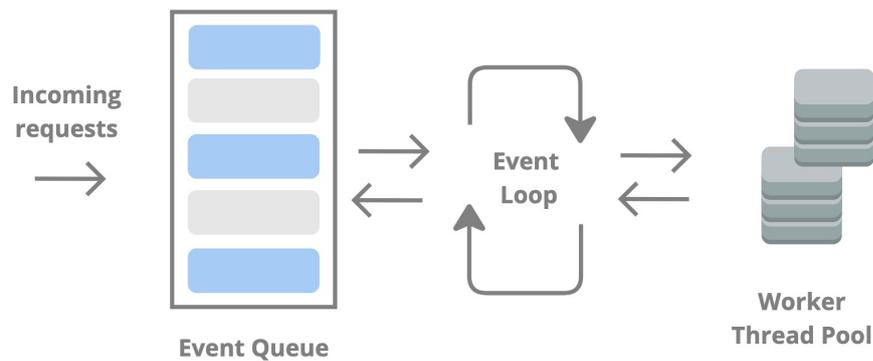


Figura 5.3: Event Loop

Il server Node.js utilizza l'architettura *Single Threaded Event Loop* per gestire più client allo stesso tempo. La figura 5.3 mostra il funzionamento dell'*event loop*. A differenza di quello che succede negli ambienti multithread, Node.js mette a disposizione un pool limitato di thread per gestire le richieste in arrivo. Appena arriva una richiesta, Node.js la inserisce in una coda di eventi (Event Queue). Il ciclo *event loop* che è costantemente in ascolto di tutte le attività, prende una richiesta dalla coda e valuta se è una richiesta bloccante (operazioni di I/O) oppure può essere svolta subito. Nel caso sia una richiesta bloccante la assegna ad un pool di thread interni (Worker Thread Pool) in questo modo non blocca altre operazioni veloci che vengono dopo. Una volta che il compito in background è completato, i risultati vengono restituiti e l'applicazione che usa Node.js può utilizzare una funzione di trigger (callback) per elaborare ulteriormente l'output. Il fatto che Node.js utilizzi pochi thread comporta un minor spreco di risorse in termini di cpu

e memoria e una maggiore velocità nell'esecuzione dei task a bassa intensità di calcolo.

5.3 Implementazione del servizio FCM

In questa sezione si vuole ora affrontare il meccanismo di invio e ricezione delle notifiche push partendo prima dal suo funzionamento e vedendo poi i dettagli implementativi del servizio.

Una notifica Push è un breve messaggio asincrono inviato da un server ad una determinata applicazione client installata nel dispositivo finale. Con la tecnologia push l'informazione, non appena disponibile, si palesa all'utente senza la necessità di un suo intervento, superando i limiti del modello di consultazione pull, centrato invece sulla continua ricerca dell'informazione da parte dell'utente stesso. Questa tecnologia apporta numerosi benefici all'app progettata, primo tra tutti la possibilità di informare l'utente, in tempo reale, di eventi occorsi nell'ambiente di lavoro - come ad esempio - eventi che possono compromettere la sicurezza dell'utenza stessa. Per il sistema progettato ci si è affidati ad un servizio gratuito di messaggistica cloud, Firebase Cloud Message, messo a disposizione da Google e basato sul modello publish-subscribe in cui gli attori principali sono raffigurati in figura 5.4.

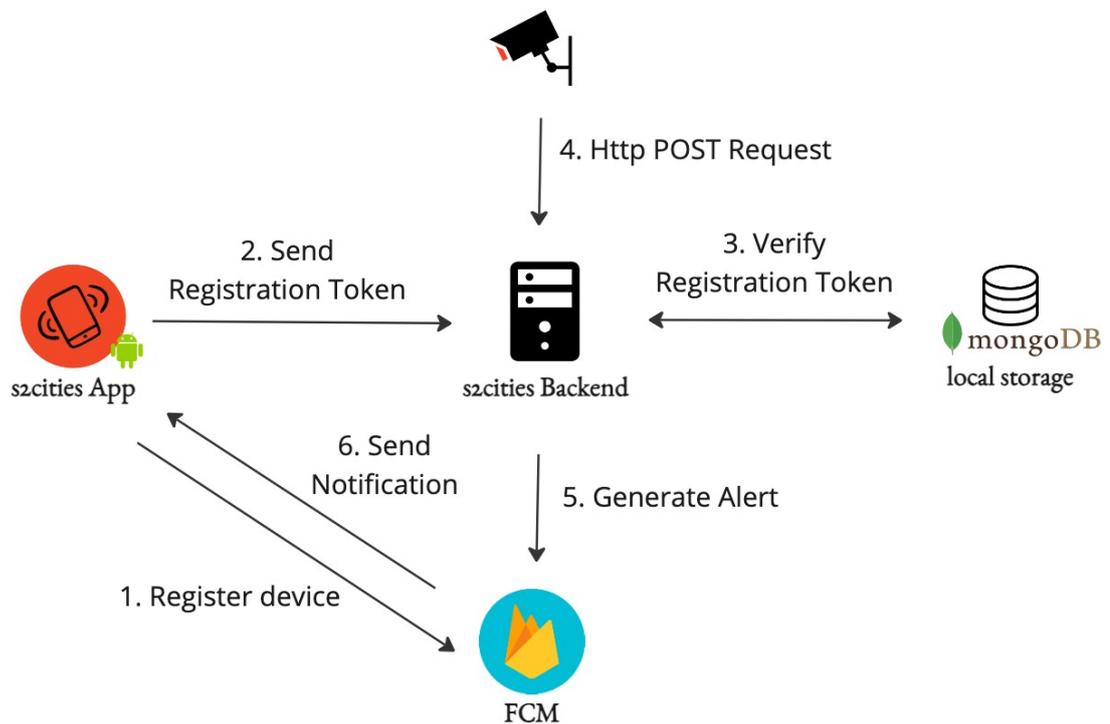


Figura 5.4: Architettura del servizio FCM

Lo schema rappresentato sopra mostra, prendendo come esempio il caso di rilevazione di un segnale di aiuto, il funzionamento dell'architettura basata su scambi di messaggi, riassumibile nei passi che seguono:

1. *Registrazione del device.* Prima che l'app possa ricevere notifiche push tramite FCM, il dispositivo su cui è installata deve essere registrato al servizio. Per fare questo ad ogni utenza con ruolo di *guard* viene associato un *registration token*, un codice che identifica in modo univoco l'istanza dell'app. Sebbene lo stesso utente possa essere associato contemporaneamente a più *registration token* (ad esempio perchè in possesso di altri dispositivi su cui può installare l'app), si predispone il progetto supponendo che una guardia utilizzi sempre lo stesso dispositivo per ricevere le notifiche.

2. *Login*. Il *registration token* è generato in fase di primo accesso all'app e inviato al web server insieme alle credenziali di accesso (username e password).
3. *Verifica del Token*. Il server memorizza localmente il codice ricevuto in modo che ogni utente *guard* abbia il suo *registration token* associato al dispositivo. In questa versione di rilascio non si effettua il controllo sull'obsolescenza del *registration token* (i token obsoleti hanno una durata di 270 giorni di inattività).
4. *Http Post Alert*. A questo punto il dispositivo è in grado di ricevere le notifiche push. Il server installato sulla videocamera di sorveglianza, al rilevamento del segnale di aiuto, invia una richiesta http di tipo POST al web server. Il payload della richiesta è un oggetto Json così fatto:

```
{
  camSerial: string,
  timestamp: string
}
```

5. *Generazione dell'alert*. In risposta alla richiesta http, il web server, in questo punto del processo, applica la logica di generazione e invio di un oggetto Alert. In particolare si scelgono il contenuto e i destinatari del messaggio di notifica dopo di che viene chiesto al FCM backend di inviare a tutte le guardie selezionate una *notification data* simile a:

```
{
  notification: {
    title: "s2Cities",
    body: "Hai una nuova segnalazione!",
  },
}
```

```
    data: {
      type: "ALERT",
      url: "/alarm",
      data: { "alertId" = $alertId }
    },
    tokens: registrationTokens
  }
```

Un oggetto Alert, la cui struttura è visibile in figura 4.3, immagazina tutte le informazioni della segnalazione ricevuta dalla videocamera. Sarà sufficiente inviare il client al suo identificativo per ricostruire l'intero oggetto.

6. *Invio della notifica push.* Il backend FCM riceve la richiesta di messaggio, genera un ID messaggio e altri metadati e invia la notifica ai dispositivi utilizzando i token di registrazione FCM. L'applicazione Android riceve quindi la notifica push che, attraverso il messaggio di alert e i dati ricevuti, invita la guardia ad intaprendere un'azione specifica.

Per utilizzare il servizio FCM, dopo un'iniziale configurazione dell'account Firebase creato ad hoc per il progetto, è stato necessario integrare:

- la libreria *React Native Firebase* lato client
- l'SDK *firebase-admin* lato server

Il pacchetto React Native Firebase mette a disposizione il metodo `messaging()` che permette di gestire le notifiche push e le interazioni dell'utente con esse. La riga di codice

```
let fcmToken = await messaging().getToken();
```

permette ad esempio di salvare nella variabile locale `fcmToken` il token di registrazione FCM utilizzato per inviare notifiche push al dispositivo. Prima di

invocare `getToken()`, è necessario tuttavia richiedere il permesso all'utente per ricevere notifiche push utilizzando il metodo `requestPermission()`.

Il comportamento dell'applicazione quando si ricevono i messaggi di alert dipende dal fatto che sia in background o in primo piano. In background, l'app riceve il payload delle notifiche nella barra delle notifiche e gestisce il payload dei dati solo quando l'utente tocca la notifica. Si utilizza il metodo `setBackgroundMessageHandler()` per gestire i messaggi push ricevuti in background quando l'app non è attiva. Quando è in primo piano invece, l'app riceve un oggetto messaggio con entrambi i payload disponibili. In questo caso la gestione avviene col metodo `onMessage()` che si occupa di ascoltare gli eventi di ricezione.

Lato server, si è preferito raggruppare e funzioni principali per l'invio delle notifiche push all'interno del file `/src/lib/firebase/service.ts`, in modo da favorire il loro riutilizzo in qualsiasi punto all'interno del codice. Tali funzioni sono qui elencate:

```
async function storeMessage(message) {...}
async function getFcmTokenByUserId(userId) {...}
async function sendNotification(notification) {...}
```

La funzione `storeMessage(message)` crea un oggetto `Messaggio`. Si distinguono due tipi di messaggio all'interno dell'applicazione. Un messaggio *di allarme* (`type = ALARM`) porta con sé una reference ad un `Alert` e viene inviato per segnalare l'emergenza in corso. Un messaggio *di testo* (`type = TEST_MESSAGE`) è un generico messaggio testuale inviato da un utente ad un altro utente. È usato per avvertire uno o più utenti della partecipazione di una guardia ad un evento di soccorso. Un messaggio è inoltre caratterizzato dal flag `read` che indica se è stato letto o meno. La funzione `getFcmTokenByUserId(userId)` recupera il registration token dell'utente destinatario del messaggio. Infine il metodo `sendNotification(notification)`

usa la funzione `sendEachForMulticast(notification)` di firebase per inviare il messaggio in multicast a tutti gli FCM registration token presenti in `notification`.

La selezione delle guardie a cui inviare la notifica push avviene secondo il seguente criterio:

1. si cerca nella base dati la videocamera che ha prodotto il segnale attraverso il `camSerial` ricevuto dal server nella richiesta http del passaggio 4:

```
const cam = await Videocamera.getBySerial(cam_serial);
```

Essa contiene tutte le informazioni necessarie che servono a geolocalizzare l'evento.

2. si recupera la struttura in cui la videocamera è installata attraverso il `buildingId` associato:

```
const building = await Building.getById(cam.buildingId);
```

L'oggetto `Building` contiene l'informazione relativa alle guardie di sicurezza che possono gestire l'evento.

3. ottenuta la struttura si procede con un'operazione di filtraggio delle guardie associate a tale struttura: si selezionano solo le guardie che sono operative ovvero il cui stato è "In servizio". Tutte le guardie che hanno uno stato diverso da "In servizio" non sono abilitate alla ricezione del messaggio:

```
const guards: Guard[] = await building.getGuards();  
  
const selectedGuardsIds: ObjectId[] = [];  
const userIds: ObjectId[] = [];  
  
guards.forEach((guard) => {
```

```
    if (guard.status === guardStatus.IN_SERVICE) {
      if (guard.workplaceId.equals(building._id)) {
        selectedGuardsIds.push(guard._id);
        userIds.push(guard.userId);
      }
    }
  });
```

Nell'esempio di codice riportato sopra si memorizza l'informazione sullo `userId` per recuperare in seguito i relativi `registration token`.

5.4 Funzionalità principali realizzate

Dopo aver analizzato la struttura del progetto prendendo in esame i vari strumenti utilizzati e la logica di sviluppo dei web services, questo capitolo vuole completare ora la discussione proponendo una presentazione grafica dell'applicazione mobile e del pannello di controllo. Si porrà il focus sugli aspetti grafici che caratterizzano l'app, talvolta motivando determinate scelte di stile, e si fornirà una breve descrizione al lettore delle funzionalità implementate.

5.4.1 L'app *s2cities*

L'applicazione mobile *s2cities* è stata testata sul dispositivo fisico Galaxy A04s con Android versione 13. Secondo quelle che sono le direttive di Expo, il file `_layout` dell'app root stabilisce come avviene la navigazione all'interno del sito. Qui viene definito uno *stack di screen* su cui è basata la navigazione con `expo-router`. Tutte le schermate principali che costituiscono l'app si trovano all'interno della cartella *screens* e fanno da wrapper a form e componenti React usati per la visualizzazione dei dati. Per questioni di manutibilità del codice, si è scelto di dividere ogni

componente creato in tre file distinti: un file dove si dichiarano i componenti utilizzati (liste, tabelle, form, ecc.), un file dove si gestisce la logica interna al componente (come ad esempio le query effettuate), un file con le regole di stile. A questi tre file si farà riferimento nel corso della discussione rispettivamente come parte *funzionale*, parte *logica* e parte *grafica* del componente.

In questa sezione i termini utente e guardia sono semanticamente intercambiabili.

Autenticazione e recupero password

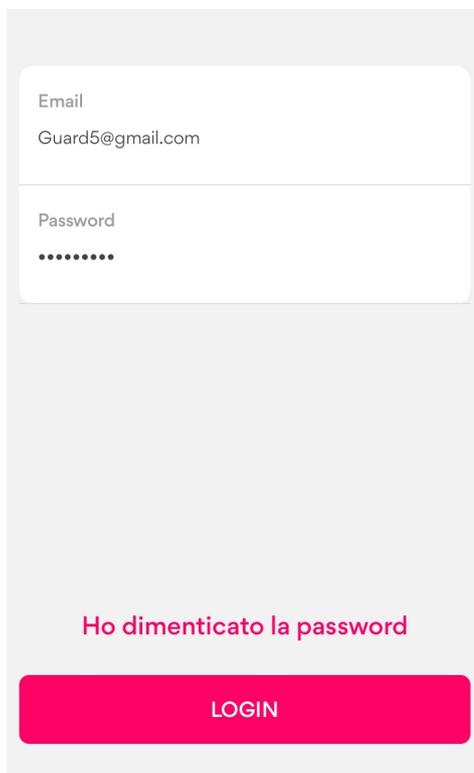


Figura 5.5: LoginScreen

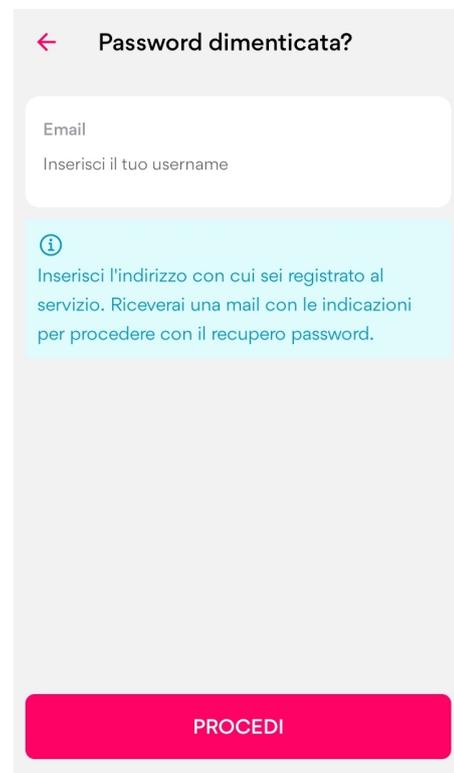


Figura 5.6: ForgetPswScreen

LoginScreen è il componente custom che realizza la schermata di Login, mostrata in figura 5.5. La sua parte funzionale è costituita dal LoginForm con i due campi testuali per l'inserimento di username e password e un AppButton. E' nella parte

logica del componente LoginScreen che ci si registra al servizio FCM, nello specifico nel metodo `useEffect()` di React dove si invocano le due funzioni asincrone di `messaging()` menzionate nel capitolo precedente. Il click sul pulsante Login provoca l'invocazione del metodo `dispatch()` di un'azione secondo quelle che sono le politiche Redux:

```
dispatch(actions.postAuthLogin.request({
  username: formData.username,
  psw: formData.psw,
  fcmToken: fcmToken
}))
```

Se il login ha successo la guardia accede alla schermata principale e l'applicazione salva le informazioni ricevute dal server nello stato `auth`, come si può intuire dalle seguenti righe di codice:

```
extraReducers: (builder) => {
  builder.addCase(extraActions.postAuthLogin.success,
    (state, action) => {
      state.user = action.payload.data.user;
      state.accessToken = action.payload.data.accessToken;
    });
}
```

In generale i messaggi di interazione tra client e server, che siano di errore o successo, vengono presentati all'utente mediante l'uso del componente Snackbar. Ogni volta che si eseguirà il logout o che l'`accessToken` (jwt token) risulterà scaduto, lo stato `auth` verrà resettato e si ritornerà alla schermata iniziale di login. Se l'utente ha dimenticato o smarrito la propria password, attraverso il link "Ho dimenticato la password" viene ridiretto alla schermata `ForgotPswScreen` (figura 5.6). Qui può inserire il proprio indirizzo email per ricevere le istruzioni per il reset password.

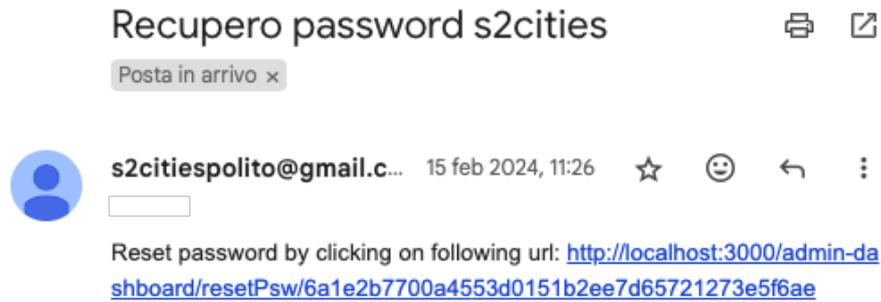


Figura 5.7: Email recupero credenziali

Il server invia all'utente una mail simile a quella raffigurata nell'immagine 5.7 con un link tramite il quale può reimpostare una nuova password. L'invio della mail (che viene realizzato lato server attraverso l'utilizzo della libreria Node.js *nodemailer*) è stato testato su un account gmail di prova. Il link è una concatenazione dell'URL con un codice segreto, il `pswResetToken`, generato attraverso un algoritmo di hashing della libreria *crypto* nel seguente modo:

```
const resetToken = crypto.randomBytes(20).toString("hex");
const pswResetToken = crypto
  .createHash("sha256")
  .update(resetToken)
  .digest("hex");
const pswResetExpires = Date.now() + 3600000;
```

Viene mantenuta nella base dati una corrispondenza tra l'utente che fa richiesta di recupero password e il `pswResetToken`, che ha la validità di un'ora. In questo modo solo l'utente in possesso del codice segreto può visualizzare correttamente la pagina richiesta e cambiare la sua password.

Schermata principale



Figura 5.8: HomeScreen

La schermata principale HomeScreen è mostrata in figura 5.8. L'intento è stato quello di raggruppare visivamente tutte le operazioni principali che una guardia può effettuare all'interno dell'applicazione, in modo da renderle disponibili a colpo d'occhio in un'unica schermata. Ad esempio, una volta autenticato la guardia può immediatamente prendere atto del suo stato di servizio ed eventualmente cambiarlo. Oppure aprendo nuovamente l'app l'utente può verificare se ci sono segnalazioni che ancora non ha gestito, segnate In Evidenza. Così come la parte funzionale, che

mette insieme *view* personalizzate, la parte logica di questo macro componente è corposa. Qui si utilizza lo *useEffect()* *hook* per fetchare i dati che servono a ricostruire lo stato iniziale *me* del container: *workPlace*, *contacts*, *buildings*, *alerts*, *actions*, *messages*.

Ricezione di un alert

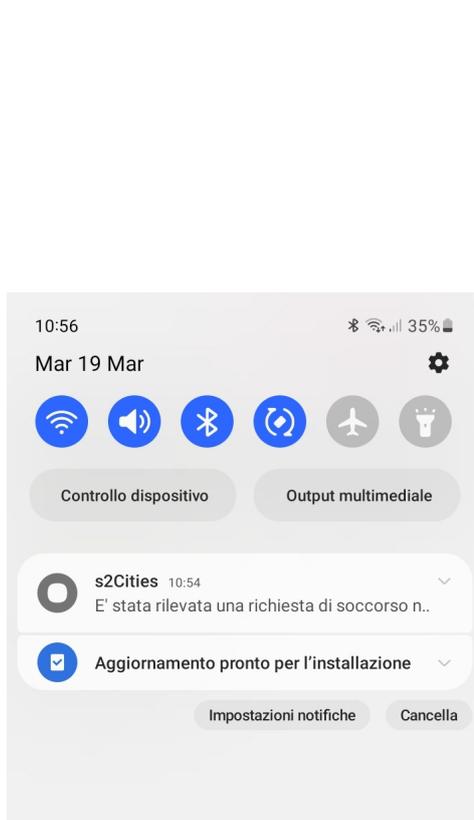


Figura 5.9: Notifica push

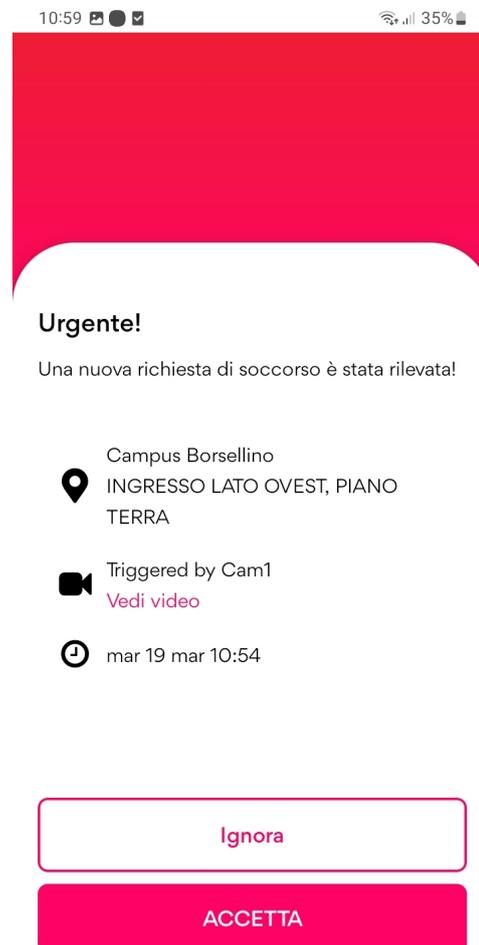


Figura 5.10: AlertScreen



Figura 5.11: Segnalazione in corso

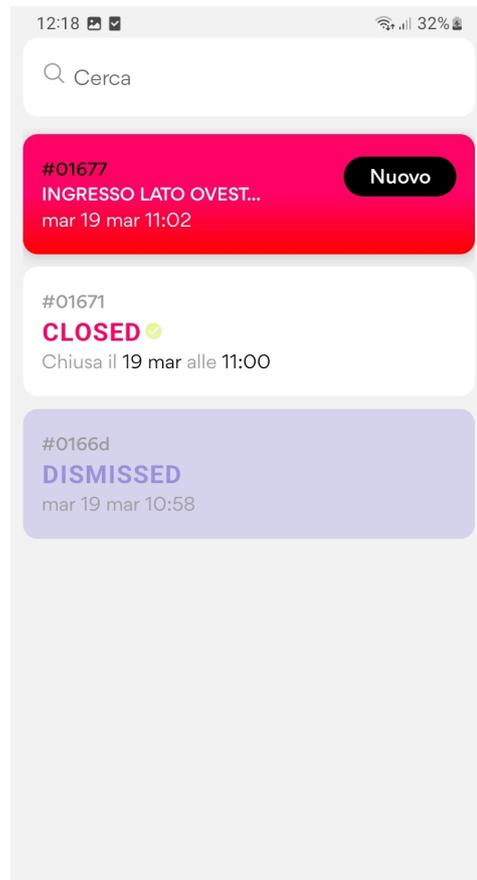


Figura 5.12: ActionsScreen

Non appena il segnale di aiuto viene rilevato, un oggetto *Alert* è stato creato dal web server e memorizzato localmente. Un *messaggio di allarme* viene inviato a tutte le guardie in servizio presso la struttura coinvolta. Lato front-end, alla ricezione di un messaggio con `type=ALERT` viene eseguita una query per aggiornare l'array di stato *alerts*.

Se l'applicazione è chiusa oppure in background il dispositivo vibra e segnala la ricezione di una notifica push nel centro notifiche ("E' stata rilevata una richiesta di soccorso nelle tue vicinanze"), come mostrato in figura 5.9. Il "tap" sulla notifica provoca la messa in primo piano dell'applicazione e la visualizzazione

di una schermata di allarme chiamata **AlertScreen**. La pagina viene aperta in modalità *modal* e il percorso del sito diventa `/modal/alarm/[alertId]` dove `alertId` è l'identificativo dell>alert ricevuto nel campo "data" della notifica. Lo *useLocalSearchParams()* hook di expo-router recupera il *search parameter* utilizzato poi per pescare nell'array `alerts` l'oggetto desiderato.

L'immagine 5.10 è un esempio di **AlertScreen**. La scelta del colore primario come sfondo contribuisce ad esprimere un messaggio di allarme ed emergenza. La parte funzionale del componente incorpora un **AlertDetailsCard**, uno specchietto che racchiude al suo interno il messaggio che arriva dal server, e due **AppButton** "Accetta" e "Ignora" che determinano l'azione. Nello specchietto riassuntivo si è scelto di dare importanza alle informazioni che possono essere utili alla guardia per individuare l'aggressione e recarsi sul posto. Ad esempio il messaggio in figura x indica che l'emergenza si sta verificando all'ingresso del Campus Borsellino e la telecamera coinvolta è la numero 1. Il link "vedi video" è al momento non attivo perchè coinvolge una funzionalità ancora non supportata in questa versione dell'applicativo. Si è pensato di lasciare comunque un *placeholder* per marcarne la potenzialità.

L'opzione "Accetta" contempla una presa in carico della segnalazione da parte della guardia che si suppone gestisca con relativa urgenza il caso recandosi sul posto dell'accaduto. Ognuna di queste due funzioni provoca la generazione di un oggetto **Azione** associato alla guardia e all>alert e il relativo cambio di stato. Tre sono gli stati di un **Azione**: **ACCEPTED**, **CLOSED**, **IGNORED**.

Si passa nello stato **ACCEPTED** non appena la guardia clicca su "Accetta". Il routing redirige l'utente verso il componente **ActionsScreen** in cui viene mostrata l'**Azione**, ovvero la segnalazione, appena creata (deducibile dal colore rosso e dall'etichetta

"Nuovo" visibile in figura 5.12). Questo componente container, selezionabile nella voce menu "Storico", raccoglie tutte le segnalazioni, correnti (ACCETPED) e passate (CLOSED/IGNORED), dalla più recente alla più vecchia. La sua parte funzionale è costituita da una barra di ricerca e una scroll view di `ActionItem`, elementi custom selezionabili atti a riassumere l'Azione intrapresa e il suo stato. Nella figura 5.12 sono riportate a titolo di esempio tutte e tre le tipologie di Azioni. Gli elementi grafici in comune sono: l'identificativo della segnalazione (il carattere # seguito dalle ultime cinque cifre alfanumeriche dell'actionId), lo stato dell'Azione, la data in cui la segnalazione è stata accettata, chiusa o ignorata. Quando un'Azione è nello stato ACCEPTED, nella sezione "In Evidenza" della schermata HomeScreen viene segnalata la presenza di una segnalazione da gestire, come mostra la figura 5.11. Cliccando sul link, l'utente è ridiretto nello Storico.

Storico delle segnalazioni



Figura 5.13: AcceptedActionScreen



Figura 5.14: ClosedActionScreen



Figura 5.15: DismissedActionScreen

In `ActionsScreen` si diramano tre percorsi differenti: in base allo stato di una segnalazione, un `ActionItem` può condurre alle schermate modali `AcceptedActionScreen`, `DismissedActionScreen` o `ClosedActionScreen`, visibili nelle rispettive figure 5.13, 5.15, 5.14.

Il percorso `/history/dismissedActionDetails/[id]` apre ad esempio la schermata in figura 5.15 che mostra una richiesta di soccorso ignorata associata all'azione con `actionId=id`. Si è data la possibilità all'utente di riaprire la segnalazione precedentemente ignorata attraverso una funzione che invia una richiesta http

PATCH al server per modificare lo stato dell'azione. Tornando automaticamente in `ActionsScreen`, per aggiornare lo stato `actions` viene eseguito il dispatch di un'azione Redux che provoca il re-rendering del componente. In generale il concetto di aggiornare gli array dei dati coinvolti nelle query di tipo POST e PATCH, in modo da avere sempre dati freschi, è una linea guida per l'applicazione. E' bene inoltre notare che la scelta di un'adeguata palette di colori è un fattore importante che condiziona l'esperienza d'uso e la percezione del messaggio veicolato. Come si può evincere dalle immagini riportate sopra, ad ogni stato di un' Azione è associato un colore.

Gestione di una richiesta di soccorso

La guardia che ha accettato la richiesta di soccorso e si è recata presso il posto indicato, cliccando sull'`ActionItem` generato nello storico, può accedere alla pagina di gestione della segnalazione caratterizzata dal percorso

`/history/acceptedActionDetails/[id]`.

Qui ancora una volta il componente `AlertDetailsCard` funge da specchio dati e, in base alla gravità del caso, l'utente può usufruire di due funzioni: chiamare la polizia e/o chiedere aiuto ai colleghi con cui lavora. La prima si realizza tramite l'`AppButton` verde mostrato in figura 5.13. Essendo questa una demo, la pressione del pulsante "112" logicamente non sortirà alcun effetto se non quello di stampare su console l'evento. La seconda funzione si limita, in questa versione dell'applicazione, ad inviare un messaggio di remainder a tutte le guardie che sono in servizio nelle vicinanze.

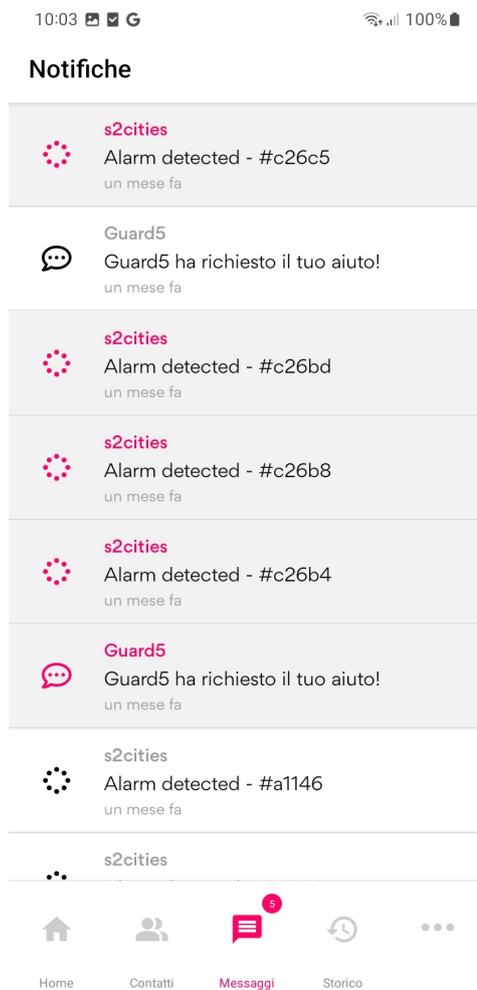


Figura 5.16: MessagesScreen



Figura 5.17: Dialog gestione segnalazione

Durante la discussione del servizio di messaggistica FCM, si è detto che i messaggi inviati dal web server possono essere di due tipi: messaggi *di allarme* e messaggi *di testo*. Entrambe le tipologie sono raffigurate nell'istantanea 5.16 che può aiutare a chiarire meglio questo concetto. La schermata rappresentata è la pagina `MessagesScreen` che usa una `FlatList` per il rendering degli elementi contenuti nello stato `messages`. La funzione del menu Messaggi è quella infatti di tenere traccia di tutti i messaggi che arrivano dal server e che hanno come

destinatario l'utente autenticato, fungendo da centro notifiche dell'applicazione. La parte logica di questo componente si occupa di recuperare al primo accesso la lista dei messaggi dal server e di mantenerla aggiornata quando questi vengono "toccati". Per una migliore esperienza utente si evidenziano i messaggi che ancora non sono stati letti, come nella figura 5.16, per distinguerli da quelli che sono già stati aperti. E' possibile distinguerli e contarli grazie al flag `read` contenuto nell'oggetto `Message`. Ogni volta che un messaggio viene letto viene invocata una callback che ne modifica lo stato:

```
const readMessage = useCallback((messageId) => {
  dispatch(actions.patchMessage.request({
    messageId: messageId
  }))
}, [])
```

La diversa natura dei messaggi viene riflessa anche nella parte funzionale di `MessagesScreen` in cui si distinguono i componenti `AlertMessage` e `TextMessage`. Come si nota in figura 5.16, il primo è caratterizzato da un'icona a forma di cerchio e dall'intestazione "s2cities" che suggerisce alla guardia che la notifica è una notifica di sistema. Mentre questo componente veicola il messaggio di "Alarm detected" per segnalare un caso di emergenza in corso (messaggio di allarme), il componente `TextMessage` veicola il messaggio proveniente da un collega che sta partecipando ad una richiesta di soccorso e che ha richiesto il suo aiuto. L'intestazione indica il mittente del messaggio, in questo caso l'operatore Guard5 ha richiesto il soccorso in loco di Guard2.

E' qui che tutte le notifiche push fanno sedimento e possono essere consultate in qualsiasi momento. L'utente che seleziona un `AlertMessage` visualizza la schermata di alert in figura 5.10, con i dettagli e i pulsanti che creano una nuova Azione. Qualora sia già presente nello storico un' Azione associata all'alert segnalato, viene

proposto un link per accedervi, tramite la schermata di Dialog mostrata in figura 5.17.

Archiviazione di una segnalazione

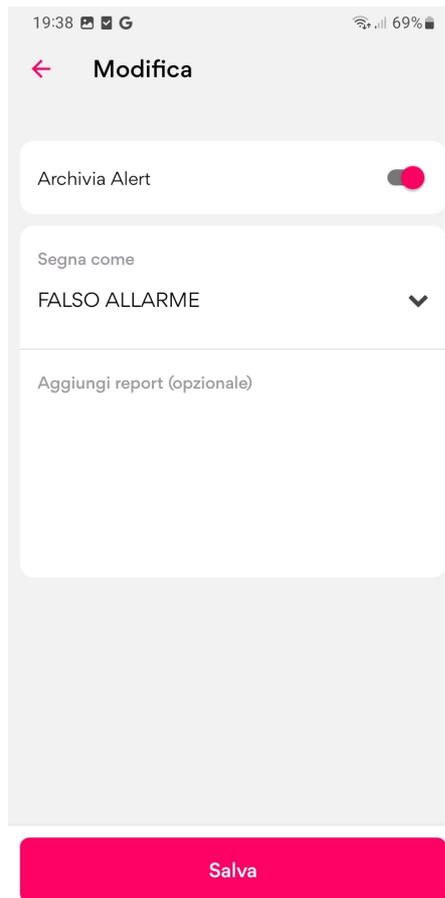


Figura 5.18: EditActionScreen

Cliccando sul link "Modifica" della schermata `AcceptedActionScreen`, si passa a `/history/editAction` che mostra la schermata di modifica dell'Azione selezionata, `EditActionScreen`. Questo componente fa da wrapper ad un altro, l'`EditActionForm`, la cui logica utilizza l'hook `useForm()` di `react-hook-form` per

gestire i dati dell’Azione da modificare. Prima che i dati siano inviati al server vengono sottoposti ad un processo di validazione. Anche qui, come nel back-end, ci si è serviti della libreria *yup* e dei metodi di validazione da essa offerti, come mostrano le linee di codice seguenti:

```
const schema = yup.object().shape<YupShape<EditActionFormData>>({
  status: yup.number().required("Seleziona un'opzione"),
  markedAs: yup.number().required("Seleziona un'opzione"),
  report: yup.string().max(1000).when(
    "markedAs", {
      is: 3,
      then: yup
        .string()
        .max(1000)
        .required("Se hai selezionato la voce ALTRO questo campo
          è obbligatorio")
    }
  ),
});
```

Il campo *markedAs* è selezionabile da un menu a tendina che espone al momento quattro opzioni predefinite (112, Falso Allarme, Gestione Interna, Altro). Lo scopo è quello di fornire una categoria che possa riassumere l’esito della segnalazione che si intende archiviare. Il campo *report* può essere opzionalmente riempito dalla guardia per inserire una breve descrizione dell’accaduto. Diventa obbligatorio se si seleziona la voce "Altro" in *markedAs*. I campi *markedAs* e *report* sono visibili in figura 5.18. Quando il form viene inviato (onSubmit) viene eseguita una http PATCH request al server web per modificare lo stato dell’Azione da ACCEPTED in CLOSED, aggiungendo i campi descrittivi. Se ha successo, il reducer modifica lo stato *actions* aggiornando l’array con i nuovi dati ricevuti e attraverso la funzione `router.back()` di expo-router la navigazione dell’utente viene riportata indietro in `/history`.

Un'Azione nello stato CLOSED si mostra come in figura 5.14, dove si è nel percorso `/(history)/closedActionDetails/[id]`. La parte logica del componente `ClosedActionScreen` non fa nulla se non recuperare l'Azione con `actionId=id` nello store Redux tramite lo *hook* `useSelector()` e passarla alla parte funzionale:

```
const myActions = useSelector(selectors.getActions)
const action = myActions.find(a => a.id === id)
```

Impostazione dello stato di servizio



Figura 5.19: SettingsScreen

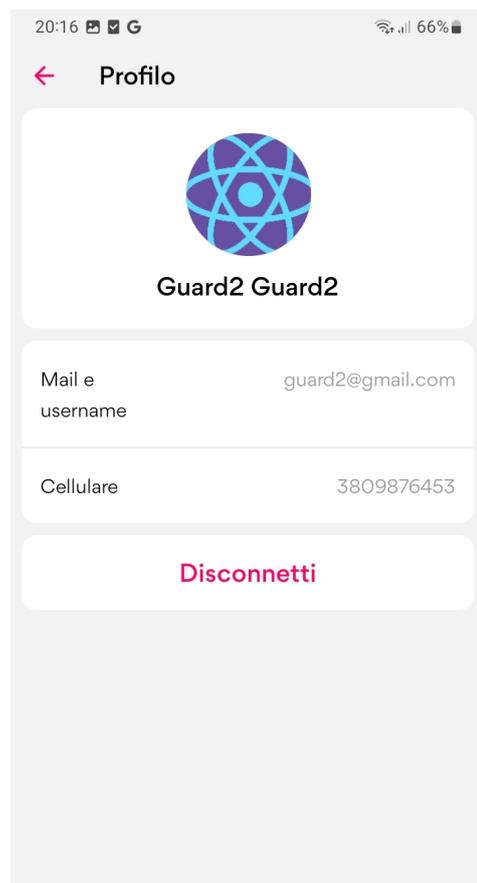


Figura 5.20: ProfileScreen

La schermata "Impostazioni" (figura 5.19) rappresenta l'ultima voce del menù ed è designata a raccogliere tutte le funzionalità extra tra cui le operazioni di configurazione delle notifiche e di modifica profilo/credenziali. Quando l'utente clicca sul MenuItem "Visualizza il tuo profilo" accede alla schermata `React ProfileScreen` in cui può controllare i suoi dati (mail di registrazione al sistema e contatto telefonico) e eventualmente impostare una immagine di profilo. La figura 5.20 mostra le informazioni minime e indispensabili dell'utente di prova Guard2. L'AppButton "Disconnetti" esegue il logout con il ripristino degli stati iniziali del contenitore Redux e il redirect a `/login`.

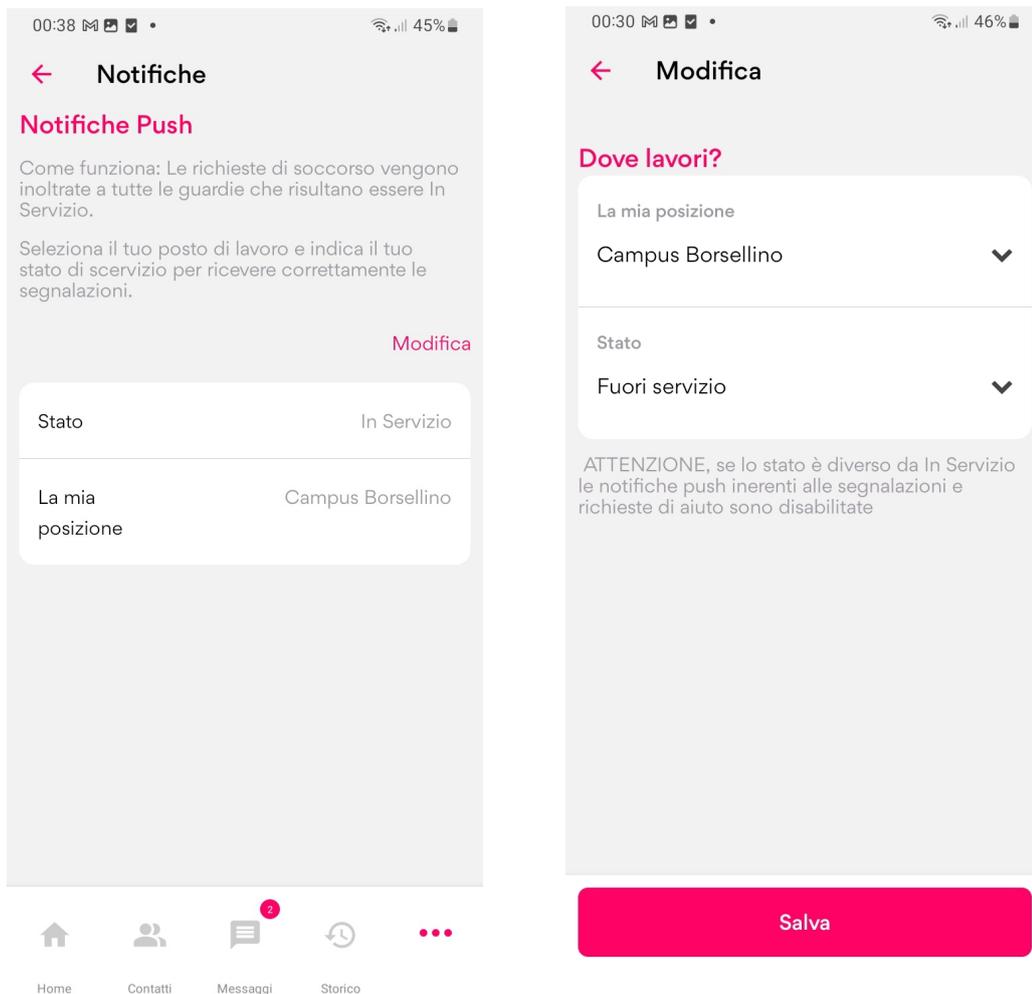


Figura 5.21: NotificationSettingsScreen **Figura 5.22:** EditNotificationSettings

Alla schermata `NotificationSettingScreen` ci si accede selezionando il MenuItem "Impostazioni Notifiche" (figura 5.21). Cliccando sul link `Modifica` la guardia seleziona tra le opzioni disponibili il luogo di lavoro e lo stato. Può scegliere tra `In servizio`, `Occupato`, `In pausa` e `Non in servizio`. Cliccando su `Salva` può apportare le modifiche. Si è considerata l'ipotesi che una guardia di sicurezza può prestare servizio in strutture diverse appartenenti alla stessa area di controllo (organizzazione). Ad esempio il menu a tendina "La mia posizione" mostra le strutture del

Politecnico in cui la guardia lavora.

5.4.2 Il pannello di controllo

Per completezza si darà ora un rapido sguardo alle funzionalità messe a disposizione nel pannello di controllo dell'utente *supervisor*, soffermandosi non tanto sui dettagli tecnici quanto sulla descrizione delle attività proposte, ponendo maggior enfasi sulle operazioni di creazione di una nuova struttura e registrazione di una guardia di sicurezza. Si terrà conto inoltre del fatto che l'utente *admin* potrà accedere ad un subset di tali funzionalità, come è stato visto nella discussione dei requisiti di sistema nel capitolo 3 di questo elaborato.

Dashboard

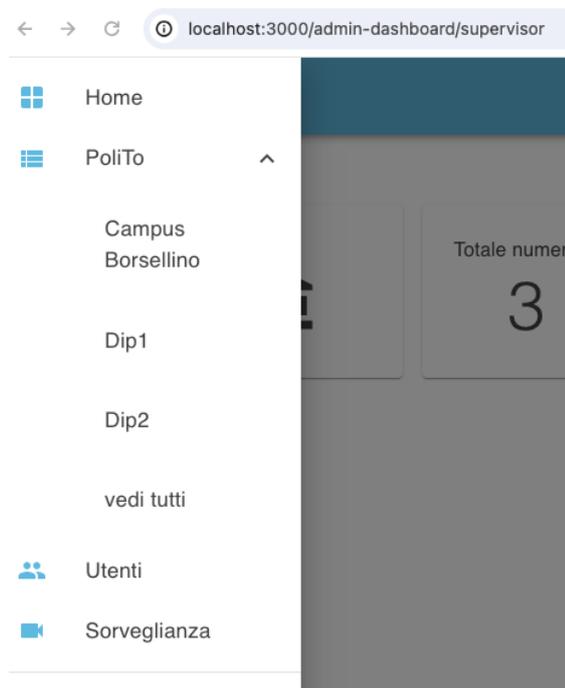


Figura 5.23: Menu

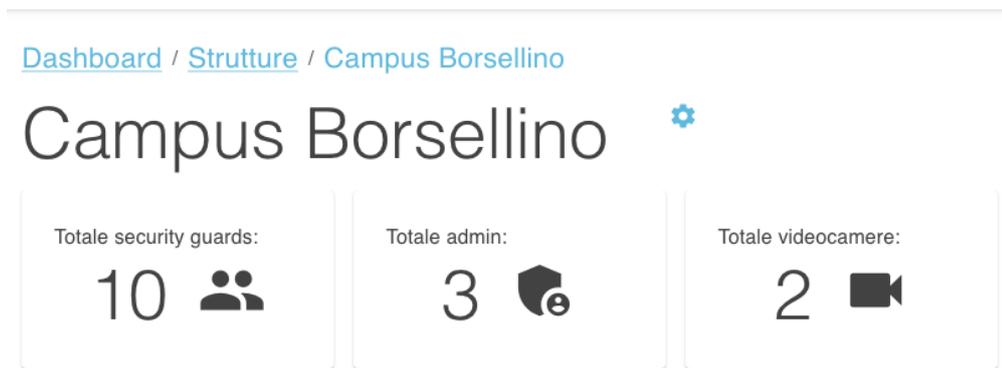


Figura 5.24: Dashboard relativa al Campus Borsellino

Una volta autenticato con il ruolo di *supervisor*, l'utente ha accesso alla sua *Home*, identificata dal percorso

`http://localhost:3000/admin-dashboard/supervisor`

Attraverso un menu laterale può navigare nelle varie sezioni del sito. Quattro sono quelle principali, evidenziate in figura 5.23: Home, PoliTO, Utenti e Sorveglianza. Come si vede nell'esempio fornito 5.24 l'utente, amministratore dell'*organizzazione* PoliTO, gestisce tre strutture. Per ogni struttura viene visualizzato uno specchietto dati riassuntivo come quello mostrato in figura 5.24: l'applicazione interroga la base dati ottenendo il numero totale delle guardie che lavorano per la struttura, il numero totale di utenti *admin* e il numero di videocamere registrate al servizio. La scelta stilistica di usare delle *cards* selezionabili per mostrare i dati rende l'esperienza più *user-friendly* per l'utente, creando un collegamento con una determinata pagina all'interno del sito.

Creazione di una struttura

[Dashboard](#) / [Strutture](#)

Buildings

Inserisci una nuova struttura riempiendo i campi sottostanti

Seleziona un'organizzazione *

Nome dell'edificio* Città* Indirizzo 1* Indirizzo 2

[Aggiungi](#) [Annulla](#)

PoliTo

Campus Borsellino

Dip1

Dip2

Figura 5.25: Form per la registrazione di una nuova struttura

In <http://localhost:3000/admin-dashboard/supervisor/buildings> l'utente visualizza, insieme alla lista delle strutture di pertinenza, la funzione "Aggiungi Struttura". Cliccando sul pulsante, appare un form di registrazione come in figura 5.25. Come in tutti i form costruiti all'interno dell'applicazione, anche qui è prevista la validazione dei dati inseriti prima dell'invocazione della funzione `onSubmit()`.

Generazione delle credenziali

[Dashboard](#) / [Strutture](#) / [Campus Borsellino](#) / [Security](#)

8 

Inserisci i dati del nuovo utente
Verrà inviata loro una mail con le credenziali di accesso

Campus Borsellino

Nome

Cognome

Email

Cellulare

Oppure carica un file .csv

Nessun file selezionato
[Scegli file](#)

AGGIUNGI Annulla

Figura 5.26: Form per l’inserimento di un nuovo utente

Nella progettazione dell’applicazione, è un requisito fondamentale che la guardia fornisca all’amministratore della struttura presso cui lavora il suo indirizzo email per la creazione dell’utenza, oltre ai dati anagrafici e il contatto telefonico. Questo permetterà all’utente *supervisor/admin* di riempire il form per la registrazione dell’utente al sistema, visibile in figura 5.26. Una scelta ragionevole è stata quella di dare la possibilità al *supervisor* di effettuare un inserimento multiplo delle utenze, caricando un file in formato Csv. Per un corretto inserimento tale file deve avere un’intestazione ben precisa, composta da: Username, First Name, Last Name, Phone, BuildingId. All’atto dell’invio, la logica espressa nell’endpoint `http://localhost:3000/api/guards` si occuperà di inviare una mail all’indirizzo specificato, simile a quella in figura 5.27. E’ bene sottolineare che, al momento, il codice segreto è generato in modo casuale dal server, attraverso la riga di codice:

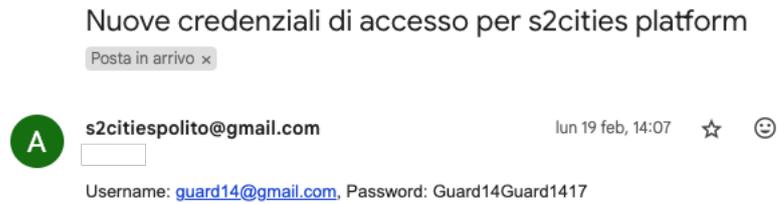


Figura 5.27: Email credenziali d'accesso

```
const psw = firstname + lastname + username.length;
```

e che tale password non è salvata in chiaro nella base dati, ma viene prima sottoposta alla funzione di hashing della libreria Node.js bcrypt. L'utilizzatore dell'applicazione mobile potrà cambiare il codice segreto dopo il primo accesso.

Gestione delle videocamere di sorveglianza

L'utente *supervisor* è la figura designata alla registrazione dei dispositivi di video-sorveglianza all'interno del sistema. Il menu Sorveglianza fornisce una vista globale sul numero di videocamere installate per organizzazione e per singola struttura. La registrazione avviene attraverso un form analogo a quello in figura 5.28 dove tutte le informazioni richieste sono obbligatorie. E' inoltre previsto anche qui l'inserimento multiplo tramite caricamento del file Csv.

Questa procedura è fondamentale poiché l'identificativo della videocamera permette di localizzare la richiesta di aiuto e di indicare la posizione alla guardia tramite degli *attributi* associati al device in fase di inserimento nella piattaforma. Gli attributi in questione sono *categoria*, *sotto-categoria* e *piano* e sono visibili in figura 5.30. La scelta di usare delle opzioni pre-configurate, come quelle visibili in figura 5.29, limita il proliferare di *dati sparsi*, conferendo uniformità e omogeneità alla base dati.

[Dashboard](#) / [Sorveglianza](#)

Videocamere

Organizzazione e Edificio

Seleziona l'organizzazione *
PoliTo

Seleziona una struttura *
Campus Borsellino

Nome e modello della videocamera

Serial Number *

Nome *

Modello *

Posizione della videocamera all'interno della struttura

Seleziona una categoria *

- ATRIO
- AULA
- AULA STUDIO
- BIBLIOTECA
- CORTILE
- DISIMPEGNO
- INGRESSO
- LABORATORIO
- LIBRERIA
- MENSA
- PORTICATO
- SALA RIUNIONI
- TERRAZZO
- UFFICIO DOCENTI
- UFFICIO TECNICO

Figura 5.29: Categorie

Figura 5.28: Form per la registrazione di una videocamera di sorveglianza

Posizione della videocamera all'interno della struttura

Seleziona una categoria *
INGRESSO

Nome *
INGRESSO A

Es. Porticato ingresso principale

Seleziona il piano *
PIANO TERRA
PRIMO PIANO
SECONDO PIANO

AGGIUNGI Annulla

Figura 5.30: Posizione della videocamera

Capitolo 6

Conclusioni

In conclusione in questo lavoro tesi è stata progettata ed implementata un'applicazione eseguibile sui dispositivi Android che ha l'obiettivo di ricevere il segnale di aiuto (*Signal For Help*) proveniente da una videocamera di sorveglianza e avvertire le guardie di sicurezza che vigilano l'area, attraverso un sistema basato su *notifiche push*.

Tale applicazione si serve di due componenti principali: un modulo server e una piattaforma web per l'inserimento e la gestione dei dati.

Si è partiti dapprima con l'introduzione del progetto *s2cities*, inquadrandolo nel contesto delle *safe smart cities* di cui si è brevemente discusso.

Dopo aver descritto l'architettura del progetto, si è quindi cominciato a trattare il processo di sviluppo alla base della soluzione proposta, effettuando un'analisi approfondita dei casi d'uso e dei requisiti funzionali e non funzionali. Durante questa fase, in particolare, sono stati ottenuti i requisiti da soddisfare nella progettazione dell'app e le funzionalità che essa deve presentare, studiandone l'interazione con l'utente.

Si è passati poi a descrivere la struttura delle repositories *s2cites-backend* e *s2cites-frontend*, indicando le tecnologie e gli strumenti utilizzati durante la fase di codifica. L'utilizzo del framework React ha permesso la realizzazione di un'app mobile moderna basata su componenti altamente personalizzabili. Node.js è un potente motore che ha consentito l'integrazione di numerose librerie che hanno semplificato in molti casi il lavoro di sviluppo. Tra queste si è fatta particolare menzione a quelle di Google Firebase che sono alla base del servizio di messagistica implementato. Nella tesi sono state inoltre introdotte tabelle e figure che descrivono le principali funzionalità realizzate.

Molto del lavoro realizzato è frutto di uno studio personale sugli argomenti trattati. Tuttavia sono risultate fondamentali le nozioni apprese durante il mio percorso universitario, specialmente nei corsi di Applicazioni Web e Programmazione Distribuita. La soluzione proposta risulta parzialmente completa rispetto alle funzionalità inizialmente progettate: come si è visto nel corso della discussione, esistono alcune funzionalità per le quali attualmente non viene fornito il supporto dal server. Nel complesso, grazie alla tecnologie usate e agli strumenti di sviluppo messi a disposizione dall'azienda di supporto, il prodotto finale, oggetto di questo lavoro di tesi, rappresenta un buon punto di partenza per futuri sviluppi.

6.1 Sviluppi futuri

Numerosi spunti e idee sono venuti fuori durante la progettazione e lo sviluppo di *s2cites*: l'applicazione realizzata, sebbene soddisfi pienamente gli obiettivi prefissati, può essere ulteriormente affinata al fine di renderla più efficiente e di migliorarne l'esperienza utente. Sono di seguito esposti alcuni punti da prendere in

considerazione per eventuali sviluppi futuri:

- *visualizzazione video*: una guardia che riceve una notifica di allarme ha la possibilità di accettare o ignorare la richiesta di soccorso in base alla visione di un breve filmato registrato dalla videocamera all'atto della rilevazione del segnale di aiuto e successivamente inviato al web server. L'idea di base sarebbe quella di inviare nel file json, insieme all'identificativo seriale della videocamera, gli ultimi dieci secondi del frammento ripreso. E' uno strumento utile che agevolerebbe il processo decisionale e accorcerebbe i tempi di soccorso.
- *mappa delle videocamere*: una guardia può visualizzare nel menu Impostazioni una mappa che indica la posizione delle videocamere all'interno della struttura presso cui lavora.
- *gestione registration token*: occorre mantenere un elenco aggiornato dei token attivi, dal momento che i registration token scadono dopo un periodo di inattività. Per fare questo si deve implementare un timestamp del token nel codice e aggiornarlo a intervalli regolari. Inoltre bisogna far sì che ad un utente sia possibile associare un *array* di token.
- *perfezionamento delle funzionalità del pannello di controllo*: non solo la guardia ma anche un utente con ruolo di admin o supervisor può ricevere una notifica di allarme. Sul portale web è necessario inserire una sezione apposita del Pannello di controllo e far in modo che il server inoltri l'Alert sulla piattaforma. Inoltre, è ragionevole fornire all'utente un meccanismo di controllo real-time delle videocamere per capire quali sono attive e quali no.

- *affinamento dell'opzione "Invita i tuoi colleghi"*: la guardia può scegliere se inviare il messaggio di sollecitazione in modalità broadcast a tutti i colleghi disponibili o selezionarli singolarmente da una apposita lista. In quest'ultimo caso il web server si occuperà di effettuare un secondo filtraggio delle guardie in servizio come destinatari del messaggio.
- *chat interna*: infine una chat interna all'app mobile potrebbe aiutare la comunicazione in tempo reale tra colleghi, portando un maggior coinvolgimento e migliorando complessivamente l'esperienza utente.

Bibliografia

- [1] URL: https://www.ansa.it/sito/notizie/speciali/editoriali/2020/11/23/violenza-sulle-donne-doppia-emergenza-in-era-covid_2fb563c7-a4c6-4937-bd21-5d30989fec8f.html (cit. a p. 1).
- [2] URL: <https://www.unwomen.org/en/what-we-do/ending-violence-against-women/facts-and-figures> (cit. a p. 1).
- [3] URL: <https://www.istat.it/it/files/2021/05/Case-rifugio-CAV-e-1522.pdf> (cit. a p. 1).
- [4] URL: <https://canadianwomen.org/signal-for-help/> (cit. a p. 2).
- [5] Sterpone Luca Azimi Sarah De Sio Corrado. «Enhanced Video Surveillance Systems for “Signal For Help” Detection on Edge Devices». In: (2023), pp. 1–4. DOI: <http://dx.doi.org/10.1109/ISTAS57930.2023.10305989> (cit. alle pp. 2, 9).
- [6] Youssef El ganadi. «Potenzialità e criticità del modello Smart City. Un’applicazione al caso di Casablanca». In: (2021), p. 9. DOI: <https://hdl.handle.net/1889/4282> (cit. a p. 5).

- [7] «ACED-IoT : Safe Cities through Cloud and the Internet of Things». In: *project 9th* (2014), pp. 50–55. URL: <https://www.asp-poli.it/wp-content/uploads/2015/12/IXCycle.pdf> (cit. a p. 6).
- [8] Osamu Kamimura Atsushi Deguchi. *Society 5.0. A People-centric Super-smart Society*. Springer Open, 2018 (cit. alle pp. 6, 7).
- [9] Sangwon LEE Choong Hyong LEE Kigon PARK. «Development of IoT Healthcare Platform Model for the Elderly using Bigdata and Artificial Intelligence». In: *International Journal of Membrane Science and Technology* (2023). DOI: <https://doi.org/10.15379/ijmst.v10i1.1435> (cit. a p. 7).
- [10] K. Pavani Prasanth Kumar M. Ravi Kumar e M. Naga Durga Bhavani. «Smart Mobile Hybrid App for Emergency Health Care». In: (2023). DOI: <http://dx.doi.org/10.2139/ssrn.4411676> (cit. a p. 7).
- [11] URL: <https://www.who.int/news/item/13-12-2023-despite-notable-progress-road-safety-remains-urgent-global-issue> (cit. a p. 7).
- [12] Rim Zaghdoud Basheer Ahmed Mohammed Imran, Mohammed Salih Ahmed, Razan Sendi, Sarah Alsharif, Jomana Alabdulkarim, Bashayr Adnan Albin Saad, Reema Alsabt, Atta Rahman e Gomathi Krishnasamy. «A Real-Time Computer Vision Based Approach to Detection and Classification of Traffic Incidents». In: *Big Data and Cognitive Computing* (2023). DOI: <https://doi.org/10.3390/bdcc7010022> (cit. a p. 7).
- [13] URL: <https://yeoman.io/> (cit. a p. 38).
- [14] URL: <https://git-scm.com/> (cit. a p. 40).
- [15] URL: <https://nextjs.org/docs> (cit. a p. 43).
- [16] URL: <https://jwt.io/> (cit. a p. 50).