

Safe-Device Locator UI

Lorenzo Tanganelli Luca Patarca Nico Trionfetti

14 maggio 2021

Indice

| | |
|-------------------------------------|---|
| Introduzione | 2 |
| Processo di sviluppo | 3 |
| Prima iterazione | 3 |
| Seconda iterazione | 5 |
| Terza iterazione | 6 |
| Riferimenti bibliografici | 7 |

Introduzione

Il progetto di ricerca industriale *S.A.F.E Project*[1] ha come obiettivo la realizzazione di sistemi di arredo innovativi capaci di trasformarsi in sistemi intelligenti di protezione passiva delle persone in caso di crollo dell'edificio causato da un terremoto.

Questi sistemi di arredo smart saranno dotati di sensoristica "salva-vita" capace di pre-allertare in caso di terremoto, di rilevare e localizzare la presenza di vita dopo un crollo, di monitorare le condizioni ambientali sotto le macerie e di elaborare e trasmettere informazioni utili a chi deve portare soccorso.

Il ciclo di vita dei sensori si divide in tre scenari operativi:

- i. **Tempo di pace:** monitoraggio per il pre-allertamento (es. misure accelerometriche)
- ii. **Durante l'evento:** invio dei dati per il rilevamento dei danni (es. misure accellerometriche, inclinometriche e di spostamento) e attivazione di logiche di intervento in seguito al riconoscimento dell'evento.
- iii. **Dopo l'evento:** invio dei dati per la localizzazione delle vittime e monitoraggio ambientale al fine di guidare gli operatori nel triage di soccorso.

L'invio di dati tra i sensori ed il mondo esterno avviene utilizzando la tecnologia LoRa.

LoRa consente trasmissioni a lungo raggio e a basso consumo energetico arrivando oltre 10 km nelle zone rurali e 3–5 km in zone fortemente urbanizzate.[2]

Facendo riferimento al modello ISO/OSI la tecnologia è presente in due strati:

- **LoRa:** Il livello fisico LoRa è proprietario della Semtech e non se ne conoscono i dettagli implementativi. LoRa utilizza una modulazione a spettro espanso proprietaria, derivata della modulazione Chirp Spread Spectrum (CSS). Inoltre utilizza la codifica Forward Error Correction (FEC) come meccanismo di rilevazione e successiva correzione degli errori contro le interferenze.
- **LoRaWAN:** LoRaWAN è un protocollo del livello Media Access Control (MAC) che lavora a livello di rete per la gestione delle comunicazioni tra gateway Low Power Wide Area Network (LPWAN) e dispositivi end-node come protocollo di routing.

Lo scenario operativo post evento si divide in tre attività:

- i. **Campionatura:** mediante l'utilizzo di un drone dotato di tecnologia che supporta il protocollo LoRaWAN viene campionata l'area coperta dalle macerie. Durante la fase di volo vengono memorizzati i dati ricevuti dai sensori e la potenza del segnale.
- ii. **Analisi dati:** sfruttando opportuni algoritmi di localizzazione vengono analizzati i dati memorizzati dal drone così da determinare dei centroidi in cui si suppone si trovi il disperso.
- iii. **Guidare soccorritori:** i soccorritori, dotati di opportuni tablet, visualizzeranno una mappa con la heatmap e i centroidi risultanti dall'attività di analisi dati, così da potersi orientare per individuare i dispersi.

Il nostro progetto, all'interno di S.A.F.E., ha l'obiettivo di creare un applicativo per tablet linux e android utile nell'ultima attività post evento. Trovandosi in uno stato d'emergenza, l'applicazione punta ad avere un'interfaccia grafica semplice e funzionale così da agevolare il lavoro degli operatori. Inoltre, dovrà essere in grado di funzionare senza connessione internet in quanto il terremoto potrebbe causare l'interruzione delle comunicazioni.

Processo di sviluppo

Prima iterazione

Requisiti

Durante l'attività di analisi sono emersi i seguenti requisiti:

- Visualizzazione di mappe.
- Rendering di dati vettoriali come heatmap.
- Rendering di marker sulla base delle posizioni dei sensori.
- Rendering di dati vettoriali come centroidi.
- Geolocalizzazione del dispositivo.
- Visualizzazione delle informazioni inviate dai sensori.
- Mantenimento dello stato nel caso in cui venga terminata l'applicazione.
- Funzionamento senza comunicazione internet.

In questa iterazione abbiamo scelto di sviluppare questi requisiti:

- Visualizzazione di mappe.
- Rendering di dati vettoriali come heatmap.
- Rendering di marker sulla base delle posizioni dei sensori.
- Visualizzazione delle informazioni inviate dai sensori.
- Funzionamento senza comunicazione internet.

Progettazione e Implementazione

Terminata la prima attività di progettazione si è scelto di utilizzare il framework *Flutter*[3] basato su *Dart*[4] per lo sviluppo del frontend, così da creare un'applicazione nativa compilata sia per mobile che per desktop.

In particolare, abbiamo deciso di sfruttare l'implementazione Dart di Leaflet per Flutter denominata *Flutter_Map*[5]; Leaflet è la principale libreria JavaScript open source per mappe interattive ottimizzate per dispositivi mobili. Il pacchetto *Flutter_Map* rende disponibili una serie di implementazioni della classe *LayerOptions*, come per esempio *CircleLayerOptions*, *MarkerLayerOptions*, *PolygonLayerOptions* e *TileLayerOptions*, i quali permettono di fare il rendering della mappa e di aggiungerci informazioni.

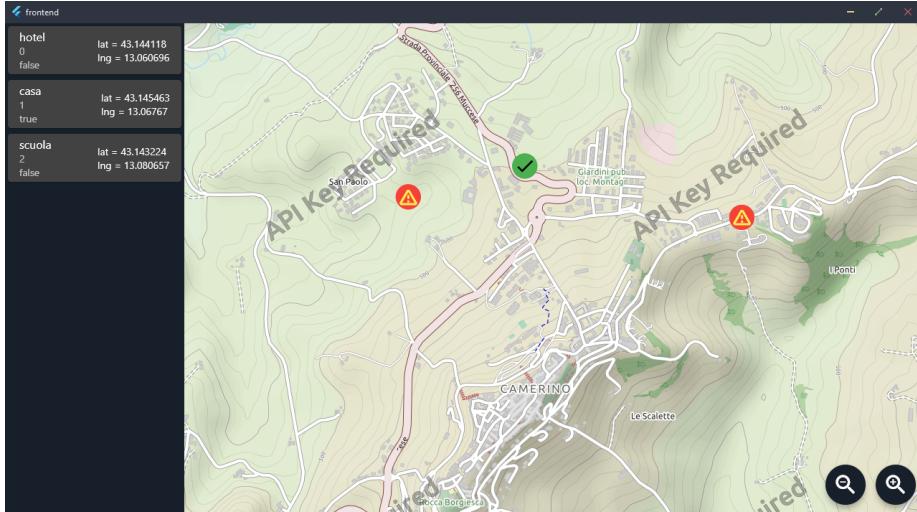


Figura 1: Marker Flutter

Purtroppo Flutter_Map non dispone di un implementazione di LayerOptions per fare il rendering di una heatmap da aggiungere come layer alla mappa. Per questo motivo è stato deciso di adattare la classe CircleLayerOptions così da creare manualmente l'heatmap.

Invece, per lo state management, si è scelto di utilizzare il pacchetto *provider*[6] così da condividere i dati più facilmente nel widget tree.

Per la generazione di mappe offline si è scelto di creare un eseguibile in *Rust*[7] che, prese delle coordinate geografiche, scarica delle immagini organizzate in cartelle:

```
./tiles/{z}/{x}/{y}/*.png
```

dove {z} indica lo zoom dell'immagine, mentre {x} e {y} indicano rispettivamente latitudine e longitudine. Per generare le mappe da rendere disponibili offline ci appoggiamo a *Thunderforest*[8]: un fornitore di tiles renderizzati dai dati di *OpenStreetMap*[9]. Una volta terminato il processo di download la directory generata viene compressa nel file *tiles.zip*. Si è deciso di scaricare le immagini da zoom 15 (scala 1:15 mila) a zoom 22 (scala 1:125) in un raggio di 5km dal punto centrale. Questa scelta è motivata dal fatto che vogliamo dare una visione generale di un quartiere o di un paese fino ad arrivare ad avere una visione dettagliata di singoli edifici.

Validazione e rilascio

Terminata l'attività di sviluppo del prototipo abbiamo soddisfatto quasi tutti i requisiti che ci eravamo prefissati per questa iterazione. Come si può notare dalla Figura 1 il prototipo visualizza una mappa con dei marker che cambiano il proprio aspetto in base allo stato del sensore: verde se è stato verificato, rosso altrimenti. Sulla sinistra si può notare la lista dei sensori con i relativi dati, in caso di tap sulla card del sensore la mappa sposta la visuale sopra le coordinate del sensore selezionato.

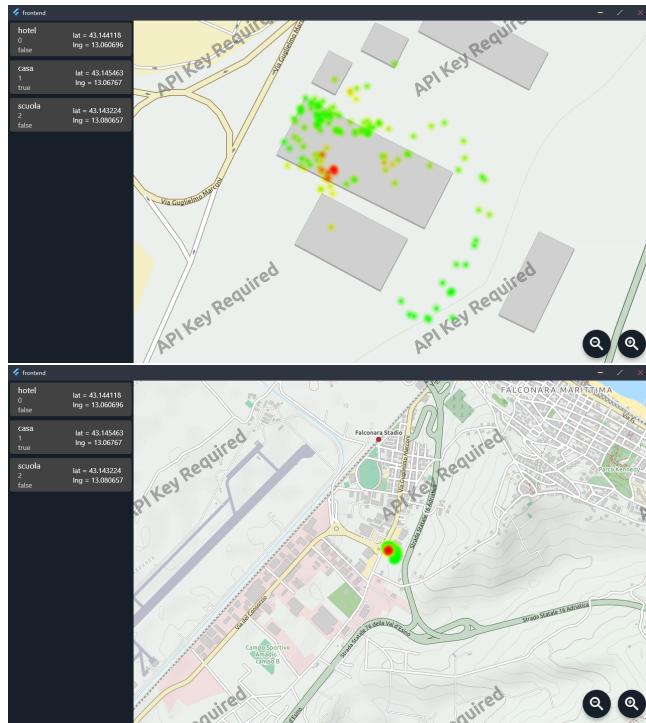


Figura 2: Heatmap Flutter

L'unico requisito che non è stato soddisfatto a pieno è stato quello di disegnare la heatmap, in quanto necessitavamo di visualizzare una mappa di calore che, in base alla scala della mappa, si ridisegnasse mettendo in relazione la distanza dei punti e l'RSSI rilevato (*Received Signal Strength Indication*).

Come si può notare nella Figura 2, abbiamo disegnato dei punti che tenevano conto dell'RSSI ma non siamo stati in grado di tenere in considerazione la densità dei punti. Questo ci portava a visualizzare una heatmap che, se avevamo tanti punti vicino ma con intensità molto bassa venivano disegnati di verde, invece noi necessitavamo che in questa circostanza la zona interessata fosse colorata di rosso.

Seconda iterazione

Visti i feedback della precedente iterazione abbiamo deciso di mantenere i medesimi requisiti riguardanti il rendering della mappa ma di rivedere progettazione e implementazione cambiando tecnologie utilizzate. Questo cambiamento è motivato dal fatto che implementare un'heatmap che soddisfa il relativo requisito avrebbe richiesto più tempo che sviluppare un nuovo prototipo con una tecnologia differente, e per questo avremmo rischiato di non rispettare le scadenze. Della precedente iterazione abbiamo mantenuto l'eseguibile per rendere disponibili i tiles offline.

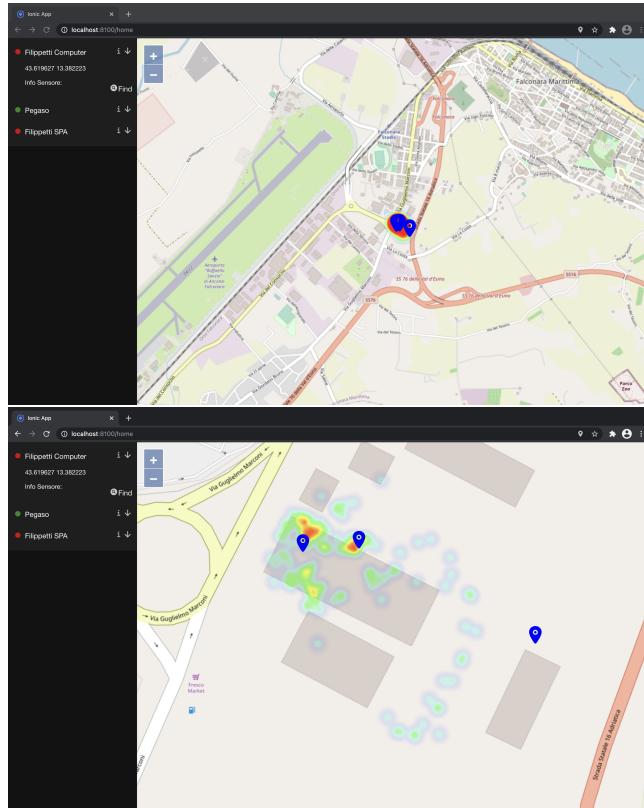


Figura 3: IonicReact

Progettazione e Implementazione

In questa iterazione si è scelto di utilizzare il framework *Ionic React*[10]: versione *React*[11] di *Ionic*[12], per sviluppare il frontend. Abbiamo scelto Ionic React così da creare app per dispositivi mobili e desktop utilizzando *Capacitor*[13] e *Electron*[14].

Ionic React utilizza standard web ed è compatibile con librerie web come *OpenLayers*[15]: libreria che permette di visualizzare tiles, marker e dati vettoriali come heatmap e centroidi.

Come mostrato in Figura 3 si è scelto di mantenere la medesima struttura del primo prototipo per quanto riguarda la UI (*User Interface*).

Validazione e rilascio

Terminata l'attività di implementazione abbiamo soddisfatto tutti i requisiti che ci eravamo prefissati di sviluppare in questa iterazione.

Si è notato che le informazioni aggiunte alla mappa, come marker e heatmap, possono essere elemento di disturbo nella situazione in cui l'operatore voglia visualizzare la mappa utilizzando una scala molto grande, per questo nelle prossime iterazioni si è scelto sviluppare un menu per selezionare quali layer mostrare.

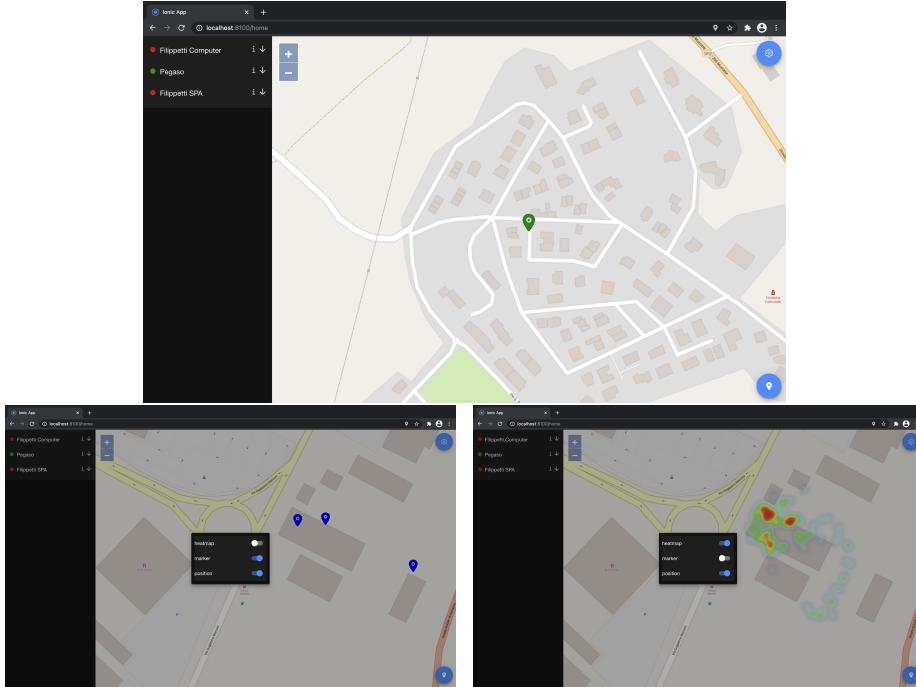


Figura 4: Geolocalizzazione e menu layers

Terza iterazione

Requisiti

Tra i requisiti individuati inizialmente abbiamo deciso di aggiungere, al prototipo validato nella precedente iterazione, le funzionalità riguardanti il requisito di geolocalizzazione del dispositivo.

Inoltre, si è deciso di implementare anche la funzionalità risultante dai feedback della seconda iterazione: un menù per selezionare quali layer mostrare sulla cartina.

Progettazione e Implementazione

Come mostrato nella Figura 4 sono stati aggiunti due pulsanti sul lato destro dell’interfaccia. Il pulsante in alto a destra permette di visualizzare un menu all’interno di un popover contenente la lista dei layer disponibili e i relativi switch per abilitarne o meno la sovrappressione della mappa. Mentre, il pulsante nell’angolo inferiore destro, esegue le funzioni di geolocalizzazione e centra la visuale della mappa sulle coordinate del dispositivo. Per individuare la posizione geografica del dispositivo utilizziamo *Cordova Plugin Geolocation*[16] il quale sfrutta il sistema GPS o network signal: WiFi e GSM, per dedurre latitudine e longitudine.

Bibliografia

- [1] *S.A.F.E Project.* URL: <http://www.safeproject.it>.
- [2] Ramon Sanchez-Iborra et al. «Performance Evaluation of LoRa Considering Scenario Conditions». In: *Sensors* 18.3 (2018). URL: <https://www.mdpi.com/1424-8220/18/3/772>.
- [3] Google. *Flutter*. URL: <https://flutter.dev/>.
- [4] Google. *Dart*. URL: <https://dart.dev/>.
- [5] John Ryan. *Flutter_Map*. URL: https://pub.dev/packages/flutter_map.
- [6] Dash-Overflow. *provider*. URL: <https://pub.dev/packages/provider>.
- [7] Rust Project Developers. *Rust*. URL: <https://www.rust-lang.org/>.
- [8] Andy Allan. *Thunderforest*. URL: <https://www.thunderforest.com/>.
- [9] Steve Coast. *OpenStreetMap*. URL: <https://www.openstreetmap.org/>.
- [10] Drifty. *Ionic React*. URL: <https://ionicframework.com/docs/react>.
- [11] Facebook e community. *React*. URL: <https://it.reactjs.org>.
- [12] Drifty. *Ionic*. URL: <https://ionicframework.com/>.
- [13] Ionic. *Capacitor*. URL: <https://capacitorjs.com>.
- [14] GitHub Inc. *Electron*. URL: <https://www.electronjs.org>.
- [15] OSGeo. *OpenLayers*. URL: <https://openlayers.org>.
- [16] apache. *Cordova Plugin Geolocation*. URL: <https://github.com/apache/cordova-plugin-geolocation#readme>.