# Capitolo 1

# Implementazione

In questa parte del report verranno descritte le tecnologie utilizzate per la realizzazione del sistema e le scelte fatte per la sua implementazione.

## 1.1 App Mobile

Per offrire un'app mobile che possa essere utilizzata sia su sistema operativo Android che iOS abbiamo deciso di utilizzare il framework *Flutter*, un framework open-source sviluppato di Google che si poggia su *Dart*. Il progetto è stato però testato solo su android poichè nessun componente del gruppo possedeva un dispositivo con MacOS. Per la gestione della mappa abbiamo utilizzato il pacchetto *flutter_map* che permette di integrare comodamente delle mappe Leaflet all'interno delle propria app, mentre per quanto riguarda la libreria per fare Activity Recognition abbiamo utilizzato un pacchetto chiamato *flutter_Activity_Recognition*, un wrapper alle librerie di sistema Android e iOs per fare riconoscimento dell'attività dell'utente.

## 1.2 Splash Screen

Lo splash screen è la prima schermata che viene mostrata all'utente quando avvia l'app. Questa schermata è molto semplice e mostra solo il logo dell'app, una semplice icona di un'auto con il nome dell'applicazione. Questa schermata ha lo scopo di intrattenere l'utente durante il caricamento dell'app e di chiadergli i permessi per la localizzazione e per il riconoscimento dell'attività.

## 1.3 MapWidget

MapWidget rappresenta la schermata principale dell'applicazione. In questa schermata l'utente può vedere la sua posizione sulla mappa che mostrerà i poligoni. Cliccando su un poligono verrà visualizzato un Toast che indica il numero di parcheggi disponibili per quella zona. Sono inoltre presenti (solo in questa versione Beta dell'app a scopo dimostrativo) dei pulsanti che permettono di simulare l'arrivo e la partenza dall'area di parcheggio, pulsanti che non sarebbero presenti in un'ipotetica versione commerciale dell'app.
Per la gestione della mappa abbiamo utilizzato il pacchetto *flutter_map* che permette di integrare comodamente delle mappe Leaflet all'interno delle propria app. Per la visualizzione dei poligoni ci siamo affidati al Layer Polygon di Leaflet, quindi non sono altro che un layer aggiuntivo sulla mappa.

Il riconoscimento della posizione è stato fatto tramite il pacchetto Geolocator che permette di ottenere la posizione dell'utente in tempo reale.

Lato implementativo, questo widget è il widget più importante di tutta l'applicazione. Difatti nell'initState di questo widget esso si occuperà di inizializzare anche altre classi usate all'interno del progetto, come la classe per fare il riconoscimento dell'attività dell'utente. Lo snippet di codice più importante di questa classe è probabilmente la funzione che viene chiamata nella callback del listener dell'activity Recognition: updateCurrentActivity. Questa funzione si occupa di aggiornare l'attività corrente dell'utente e di inviare un messaggio al server per indicare che l'utente è entrato o uscito dall'area di parcheggio.

```
    void updateCurrentActivity(ar.ActivityType activityType) {

    if (currentActivity == ar.ActivityType.IN_VEHICLE && activityType == ar
      sendActivity(ParkingType.ENTERING, currentLocation, this.context);
    } else if (currentActivity == ar.ActivityType.WALKING && activityType =
      sendActivity(ParkingType.EXITING, currentLocation, this.context);
    }
    print(activityType)
    markerActivity = activityType;
    setState(() {
      if((activityType == ar.ActivityType.WALKING || activityType == ar.Ac
          currentActivity = activityType;
    });
  }
```

Lo state currentActivity verrà aggiornato solo se l'attività riconosciuta è *WALKING* o *IN_VEHICLE* poichè spesso è molto facile che guidando o camminando ci si fermi, portando il sistema a riconoscere l'attività come *STILL*. Questo causerbbe un'impossibilità da parte dell'app di riconoscere un cambiamento netto di attività da *WALKING* a *DRIVING* e viceversa. Tutte le altre attività vengono comunque riconsociute e salvate nella variabile markerActivity che, tramite uno switch permette di cambiare la visualizzazione a schermo, permettendo all'utente di capire quale attività viene riconosciuta in quel momento dal sistema.

### 1.3.1 Activity Recognition

Per la parte di Activity Recognition, come precedentemente accenato abbiamo utilizzato il pacchetto *flutter_Activity_Recognition* che permette di fare riconoscimento dell'attività dell'utente. Il pacchetto riconosce varie attività come: *WALKING*, *RUNNING*, *IN_BICYCLE*, *IN_VEICHLE*, *STILL*, *UNKNOWN*. L'implementazione del sistema di activity recognition è stata fatta tramite la classe *ActivityRecognitionClass*. Questa classe viene inizializzata all'interno del MapWidget (la schermata principale dell'app). Al momento della sua inizializzazione questa classe farà partire un listener che farà partire una callback ogni qualvolta viene riconosciuta una nuova attività. Importante notare come verranno eseguite operazioni (aggiornamento dell'attività corrente ed eventuali transizioni) solo dal momento in cui l'attività riconosciuta viene riconosciuta con un grado di confidenza alto. In più, tutto funziona se e solo se l'utente ha dato i permessi all'app di far riconoscere la propria attività.
Mostriamo ora il costruttore della classe e la callback che viene eseguita ogni qualvolta viene riconosciuta una nuova attività.

```
ActivityRecognition(Function updateCurrentActivity){
    this.updateCurrentActivity = updateCurrentActivity;
    isPermissionGrants();
    _activityStreamSubscription = activityRecognition.activityStream
    .handleError(_handleError)
    .listen(_onActivityReceive);
  }

  void _onActivityReceive(Activity activity) {
    //Took only the activities that interest us
    if(activity.confidence == ActivityConfidence.HIGH)
      this.updateCurrentActivity(activity.type);
  }
```

## 1.4 Backend

Per il backend abbiamo deciso di utilizzare *Node.js*, un runtime Javascript che permette di creare applicazioni web veloci e scalabili, il Database invece, è stato creato utilizzando *PostgreSQL* con estensione *PostGIS* per la gestione dei dati spaziali.

## 1.5 Frontend

Per motivi di riusabilità del codice e dei componenti abbiamo deciso di utilizzare il framework React per il frontend, un framework *Javascript* open-source sviluppato da Facebook che permette di creare componenti riutilizzabili. Per la gestione delle mappe sia nel frontend abbiamo utilizzato la libreria *Leaflet*, un framework Javascript open-source che permette di creare mappe interattive, dal quale abbiamo preso anche due pacchetti per fare Clustering e per visualizzare la Heatmap dei dati. Infine per la parte grafica è stato utilizzato il framework *Material-UI*, un framework Javascript open-source che permette di creare componenti grafici in stile *Material Design*.