UNIVERSITÀ DEGLI STUDI DI GENOVA

DIPARTIMENTO DI INGEGNERIA NAVALE, ELETTRICA, ELETTRONICA E DELLE COMUNICAZIONI

CORSO DI STUDIO IN INGEGNERIA ELETTRONICA E TECNOLOGIE DELL'INFORMAZIONE

Tesi di Laurea Triennale
Ottobre 2021

“PROGETTO E IMPLEMENTAZIONE DI UN SISTEMA EMBEDDED PER LA REALIZZAZIONE DI HEATMAP”

Candidato: Mattia Musumeci
Relatore: Prof. Riccardo Berta
Correlatore: Dott. Luca Lazzaroni

Sommario

L’obbiettivo di questa tesi è quello di progettare e implementare un sistema embedded allo scopo di rilevare e memorizzare su cloud informazioni sulla posizione dell’utente che trasporta il dispositivo e, in un secondo momento, utilizzare tali dati di geolocalizzazione per ottenere una heatmap degli spostamenti, ovvero, una rappresentazione grafica dei dati in cui i singoli valori sono rappresentati tramite colori.
Nello specifico, viene utilizzata una scheda T-Beam ESP32 dotata di moduli GPS e Wi-Fi integrati. I dati raccolti da questo dispositivo vengono memorizzati sul cloud attraverso le funzionalità fornite delle API di Measurify.
A questo punto, i campioni sono visualizzabili attraverso un’applicazione per dispositivi mobile realizzata tramite il framework Flutter. Prima di questo, è necessario autenticarsi su Measurify in modo che sia effettivamente possibile comunicare con il cloud e prelevare i campioni memorizzati. L'applicazione comprende diverse schermate con le quali è possibile eseguire molteplici azioni, tra cui, selezionare la data che verrà utilizzata per filtrare i campioni, visionare la heatmap degli spostamenti e analizzare alcune statistiche, ricavate tramite computazione locale, sui dati di geolocalizzazione.

INDICE

1. Introduzione

1.1. Internet of Things

Con **Internet of Things (IoT)** ci si riferisce a un ecosistema di oggetti fisici in comunicazione tra loro, tramite connessione internet, allo scopo di raccogliere ed elaborare informazioni.

L'idea è quella di creare rappresentazioni virtuali di dispositivi "smart" in modo che essi possiedano un'identità e una personalità virtuale. Si possono distinguere due categorie di dispositivi:

- **sensori**: raccolgono e forniscono informazioni al cloud dove vengono memorizzate.
- **attuatori**: utilizzano le informazioni memorizzate sul cloud per compiere determinate azioni.

Anche dispositivi non "smart" possono essere utilizzati in un sistema IoT: prendendo come esempio un sensore, si può pensare che esso affidi le informazioni raccolte a un **sistema di controllo** e che sia quest'ultimo a occuparsi dell'interazione con il cloud.

Figura 1: Concetto di un sistema IoT

L'esempio più semplice che viene in mente quando si tratta di sistemi IoT è la "Smart Home": in questo contesto i dispositivi di interesse sono oggetti di utilizzo quotidiano come lampadine, frigoriferi, termostati etc. Generalmente, come sistema di controllo viene utilizzato un qualsiasi assistente vocale che permette di interfacciarsi ai dispositivi prima nominati e controllarli con smartphone, computer o anche comandi vocali.

Nel mondo odierno, sono già state sviluppate diverse architetture di funzionamento per sistemi IoT e una delle prime, molto semplice, si compone di soli tre strati che possono essere definiti come:

1. Lo **strato della percezione**, o anche strato fisico, che comprende sensori, per captare e raccogliere informazioni dall'ambiente esterno, e attuatori per compiere azioni in corrispondenza di determinati stimoli o eventi.
2. Lo **strato del network** si occupa della connessione tra dispositivi smart, dispositivi di rete e server. Il suo scopo è quello di trasmettere e processare i dati provenienti dai sensori dello strato della percezione e per inviare stimoli agli attuatori.
3. Lo **strato delle applicazioni** mette a disposizione specifici servizi all'utente. Esso definisce le varie applicazioni in cui il sistema dell'Internet of Things può essere utilizzato.

Un sistema IoT può essere realizzato anche in una rete privata in cui soltanto i dispositivi all'interno di tale rete hanno la capacità comunicare tra di loro: questo meccanismo viene chiamato **Intranet of Things**. Tuttavia, la vera potenzialità di un sistema IoT si manifesta quando la comunicazione tra dispositivi avviene tramite rete internet, ma questo comporta alcune problematiche legate alla sicurezza.
Dato il meccanismo di funzionamento di internet, ogni utente ha la capacità di prelevare pacchetti dalla rete, leggerli e ottenere qualsiasi informazione. In questo modo è possibile capire il funzionamento di un determinato sistema e ottenere accesso a dispositivi privilegiati con i quali normalmente non si potrebbe comunicare. Questa problematica è nota come **"man in the middle"**.

Le soluzioni che vengono adottate per risolvere questa problematica sono due:

- **Crittografia**: si occupa di garantire che i dati siano leggibili unicamente dal destinatario.
- **Autenticazione**: si occupa di garantire l'autenticità del server.

1.2. Progetto

Questo progetto di tesi si basa fortemente sul concetto di sistema IoT e, in particolare, si occupa dello sviluppo del primo e il terzo strato prima descritti. Si possono distinguere due parti principali:

1. Lo sviluppo e l'implementazione di un sistema embedded per la raccolta di campioni GPS.

2. Lo sviluppo di un'applicazione mobile che mostra i campioni GPS attraverso una heatmap.

Per quanto riguarda la prima parte, il microcontrollore utilizzato in questa tesi è un ESP32. E' possibile programmare questo dispositivo attraverso l'IDE di Arduino che, tramite apposita configurazione, permette il caricamento del codice anche su dispositivi diversi da quelli prodotti da Arduino. In questo contesto, si parla di realizzazione di **sketch Arduino**, ovvero, un insieme di istruzioni C che saranno poi eseguite dal dispositivo. Questo non esclude la possibilità di utilizzare librerie preesistenti per facilitare e velocizzare la scrittura del programma e, in particolare, sono state utilizzate librerie per abilitare la connessione Wi-Fi, semplificare l'utilizzo del GPS integrato e inviare richieste HTTP , contenenti i campioni di geolocalizzazione, alle API di Measurify.

Figura 2: Logo Arduino

Per la seconda parte, è stata sviluppata un'applicazione mobile facendo uso di **Android Studio**. Grazie a questo ambiente di sviluppo è stato possibile eseguire testing e debugging del software direttamente tramite smartphone. A questo fine, è necessario collegare lo smartphone, attraverso cavo USB, al computer sul quale Android Studio è in esecuzione e confermare sul dispositivo che si vuole consentire il debugging.
L'applicazione è stata realizzata grazie a un framework cross-platform chiamato **Flutter**, basato sul linguaggio di programmazione **Dart**, che fornisce alcune funzionalità di debugging molto interessanti come l'hot reload del codice. Questa funzionalità evita la necessità di seguire il build del software ogni volta che si introduce un cambiamento all'interno del programma.

Figura 3: Logo Flutter

L'applicazione smartphone presenta diverse schermate:

- Nella prima vengono richieste delle credenziali per l'accesso a Measurify in modo che, in seguito, si possano prelevare i campioni memorizzati sul cloud. Se le credenziali inserite non fossero corrette, verrebbe visualizzato un messaggio di errore, altrimenti verrebbe aperta una nuova schermata composta, a sua volta, da tre sotto-schermate.

- Nella prima sotto-schermata l'utente deve scegliere una data che sarà utilizzata per prelevare i campioni da Measurify. L'idea è appunto quella di visualizzare i campioni in base alla data selezionata.

- La seconda sotto-schermata è il cuore dell'applicazione in quanto è in questa schermata che i campioni vengono visualizzati attraverso l'utilizzo di una heatmap. Questo è stato reso possibile soltanto grazie all'utilizzo di un plugin per Flutter, realizzato da Google e basato sul sistema di Google Maps, per la realizzazione di questa tipologia di mappe.

- Nella terza sotto-schermata sono mostrate alcune statistiche riguardo ai campioni prelevati per il giorno selezionato dall'utente. Questi valori sono calcolati sfruttando unicamente il processore dello smartphone e non tramite computazione su cloud.

2. Metodi e strumenti utilizzati

Questa sezione è divisa in sotto-sezioni dedicate alla discussione degli strumenti utilizzati per lo svolgimento del progetto e alla metodologia di applicazione di tali strumenti.

2.1. LILYGO TTGO T-Beam V0.7 ESP32

Per ESP32 si intendono una serie di microcontrollori aventi determinate prestazioni, un basso costo e consumo e con moduli Wi-Fi e Bluetooth integrati.

In particolare, in questa tesi è stata utilizzata la scheda LILYGO TTGO T-Beam V0.7 ESP32 visibile nelle figure 4 e 5. In particolare, nella figura 5 è mostrato lo schema delle connessioni del dispositivo.

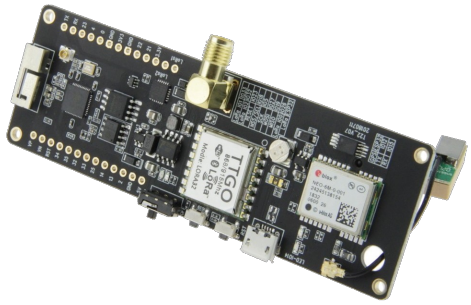

Figura 4: LILYGO TTGO T-Beam

Le principali caratteristiche sono le seguenti:

- Processore Xtensa 32-bit LX6 dual-core a 240MHz
- 520KB di memoria SRAM interna e 4MB di memoria flash esterna.
- Connettività Wi-Fi 802.11 b/g/n/e/i (supporto per WPA,WPA2,WPA2-Enterprise/SPS).
- Connettività Bluetooth (classica 4.2 e BLE)
- Modulo GPS NEO-6M con alimentazione 3V-5V e antenna in ceramica.
- Alimentazione USB 5V/1A, batteria 3.7-4.2V/500A

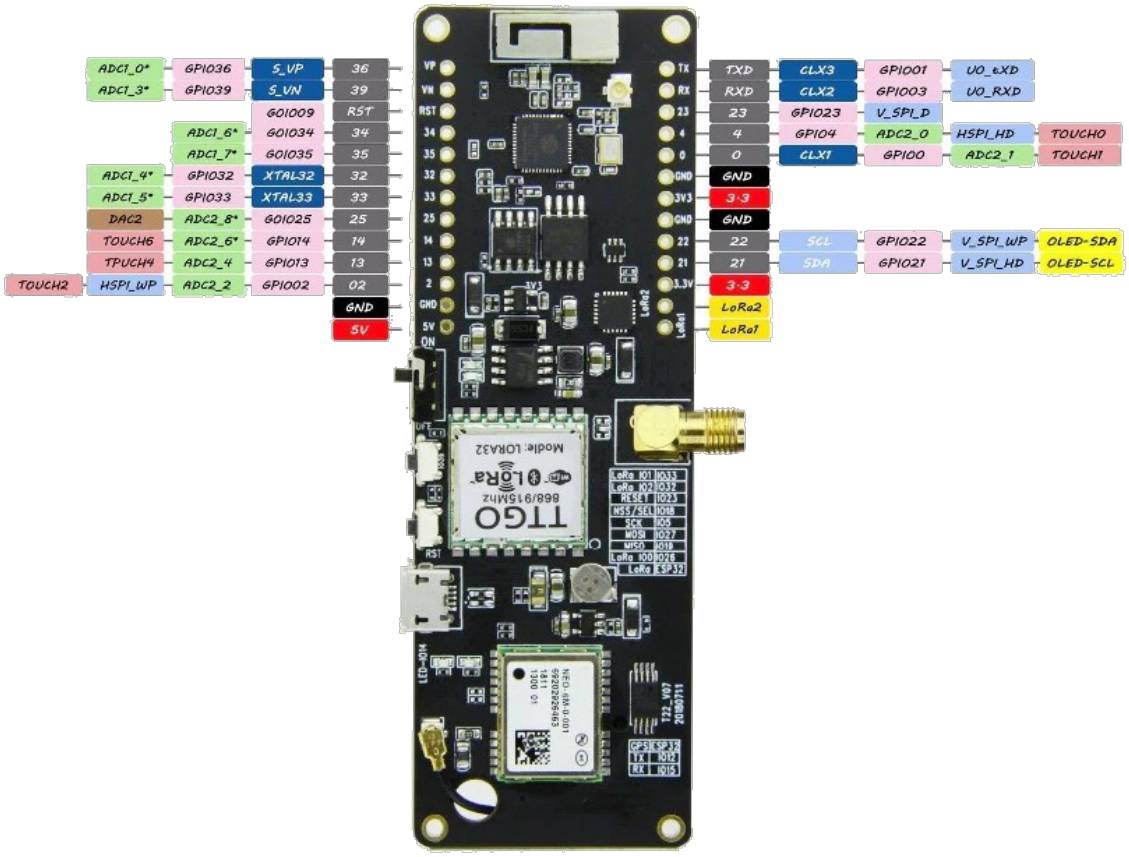

Figura 5: Pinout LILYGO TTGO T-Beam

2.2. Measurify

Measurify è un framework basato sul cloud con lo scopo di gestire dispositivi smart in un ecosistema IoT. Esso si basa sul concetto di **misura**, molto comune in un ambiente IoT, in modo da rendere più semplice e veloce il processo di astrazione necessario a coinvolgere differenti contesti. Questo significa che per utilizzare le funzionalità messe a disposizione da Measurify è sufficiente modellare gli oggetti fisici attraverso degli oggetti software, chiamati **risorse**, su cui si basa lo sviluppo delle applicazioni.
Questa astrazione si basa su alcuni concetti molto comuni in un ambiente IoT:

- **THING**: si tratta di una "cosa" (una persona, un ambiente, un oggetto, etc.) per la quale si sta creando la misura. In questa tesi, la thing è la persona che trasporta il microcontrollore.
- **FEATURE**: si tratta della dimensione che si sta misurando (temperatura, umidità, peso, etc.). In questo progetto è stata misurata la posizione della persona che trasporta il microcontrollore.
- **DEVICE**: si tratta di un dispositivo (che può essere hardware o software) adibito alla misura della dimensione (feature) su una determinata cosa (thing). In questa tesi, il device è il microcontrollore.
- **MEASUREMENT**: si tratta di una misura effettuata da un dispositivo (device) per una certa dimensione (feature) su una determinata cosa (thing). Essa contiene, oltre tutte le informazioni sulla misura effettuata, anche informazioni riguardo la data e l'ora di inizio e fine della misura, la thing sulla quale è stata eseguita la misura, il device che l'ha rilevata, la feature che rappresenta e il valore della misurazione attraverso un array di samples.

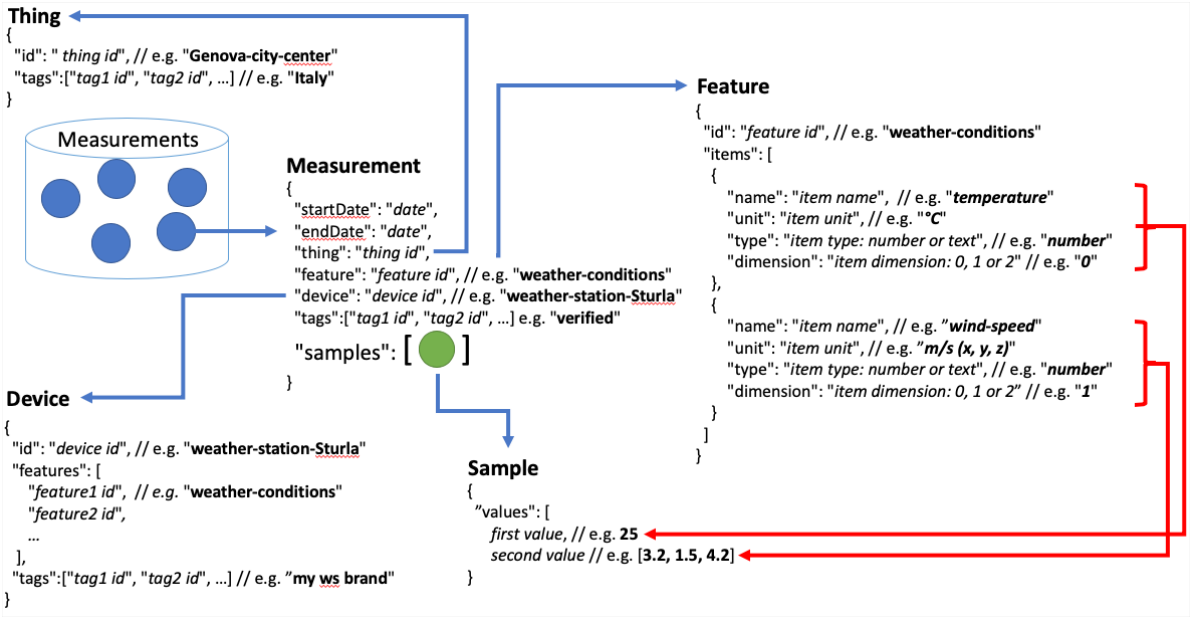

Figura 6: Schematizzazione risorse

I concetti di Measurement e Feature permettono **astrazione rispetto ai valori** memorizzati e richiesti dal cloud. Questo è possibile soltanto grazie al fatto che i valori all'interno delle Measurement devono corrispondere alla descrizione rappresentata dalla Feature: ogni volta che Measurify riceve una Measurement, viene utilizzato lo schema della Feature per controllare che i valori siano strutturati correttamente e in caso di successo, tali valori sono archiviati all'interno di un database che l'utente potrà interrogare e modificare attraverso richieste HTTP. Questo lavoro è svolto da un'API di tipo RESTful che, posta tra l'interfaccia utente e il database, si occupa della gestione delle richieste e dell'invio delle relative risposte.

2.3. Postman

L'applicazione **Postman** è molto utile per effettuare richieste HTTP all'esterno di un browser: esso consente di specificare manualmente il verbo, gli headers e il body della richiesta e analizzare la risposta ricevuta dal server. Al fine di effettuare richieste all'API di Measurify è necessario autenticarsi con le corrette credenziali attraverso una richiesta con verbo POST al seguente URL:

```
http://students.atmosphere.tools/v1/login
```

Le credenziali devono essere specificate all'interno del body della richiesta e sono le seguenti:

```
{
  “username” : “heatmap-user-username”,
  “password” : “heatmap-user-password”,
  “tenant” : “measurify-heatmap”
}
```

In un modello di comunicazione client-server è spesso necessario che il server debba identificare il client per questioni di sicurezza. Questo può avvenire attraverso delle credenziali che, teoricamente, dovrebbero essere specificate in ogni richiesta inviata dal client al server, ma questo comporta alta probabilità che tali informazioni siano rubate da client malevoli. Per ovviare questa problematica sono state sviluppate diverse tecniche di verifica tra cui il modello di autenticazione **token-based** utilizzato anche da Measutify: se il server verifica che le credenziali specificate dal client sono corrette, allora la risposta del server contiene un token che il client potrà utilizzare nelle richieste successive per identificarsi senza la necessità di specificare nuovamente le credenziali. Generalmente, è previsto che il token abbia una durata di validità molto limitata in modo che, un possibile client esterno che ne entra in possesso abbia un tempo molto limitato per eseguire azioni malevole.
Nel caso di Measurify, il token ha durata di 30 minuti.

Se l'autenticazione ha successo, il body della risposta ricevuta da Measurify contiene una stringa formattata in JSON e uno dei suoi membri è il **token** di identificazione. Il client dovrà utilizzare tale informazione nelle richieste successive per essere identificato dal server e, in particolare, deve essere specificato attraverso l'header “Authorization”.

In figura 7 si può vedere un esempio di risposta di Measurify in seguito a una corretta richiesta di autenticazione.


```
{
    "user": {
        "_id": "61497dcb4c9b88761ea310cf",
        "username": "heatmap-user-username",
        "type": "provider"
    },
    "token_expiration_time": "30m",
    "token": "JWT
        eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.
        eyJ1c2VyIjp7InN0YXR1cyI6ImVuYWJsZWQiLCJfaWQiOi
        ...
        zyWewcHi-Yim2RHbrWKlfXJnHLTllAC61Gtzv-tVxvM"
}
```


Figura 7: Esempio di risposta con token

Postman presenta una tab, chiamata **Test**, in cui può essere specificato del codice JavaScript che verrà mandato in esecuzione a seguito della risposta del server a una particolare richiesta inviata. In questo modo è possibile usufruire delle variabili globali di Postman per salvare automaticamente il token e utilizzarlo in altre richieste:

```
postman.setEnvironmentVariable(“token”, JSON.parse(responseBody).token);
```

Attraverso la riga di codice appena mostrata, il valore del token è memorizzato nella variabile globale “token”.
All'interno dell'ambiente di Postman, le variabili globali vengono utilizzate mediante il nome racchiuso all'interno di doppie parentesi graffe.
In figura 8 viene mostrato come la variabile globale appena definita possa essere utilizzata in una richiesta.

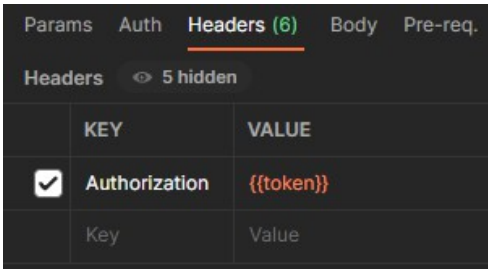

Figura 8: Utilizzo automatico del token

2.4. Flutter

La continua evoluzione dei dispositivi mobile ha spinto gli sviluppatori a comprendere in modo approfondito i meccanismi e le linee guida di sviluppo di applicazioni per i principali sistemi operativi e piattaforme così da poter raggiungere quanti più utenti possibile. Nonostante lo sviluppo di applicazioni sia diventato più semplice con il passare del tempo, resta comunque difficile per un unico sviluppatore riuscire a realizzare e mantenere nel tempo un'app nativa per diversi sistemi operativi e piattaforme.
In questo contesto nasce **Flutter**, un framework open-source e cross-platform per la creazione di interfacce native per iOS e Android. Per implementare codice tramite Flutter si utilizza un linguaggio chiamato **Dart**.

I componenti principali di Flutter sono:

- **Flutter engine**: scritto principalmente in C++ e Dart, fornisce supporto per il rendering a basso livello. Uno degli aspetti più apprezzati del Flutter engine, grazie al codice scritto in Dart, è quella di poter effettuare un “hot-reload” del codice e quindi iniettare immediatamente nell'applicazione in esecuzione le modifiche effettuate in modo da poter visionare all'istante i cambiamenti senza la necessità di un riavvio completo.

- **Foundation library**: scritta in Dart, fornisce classi e funzioni elementari che vengono utilizzate per costruire applicazioni che si basano su Flutter, come le API per comunicare con l'engine.

- **Widgets**: la progettazione dell'interfaccia utente prevede l'assemblaggio e/o la creazione di widgets, ovvero, descrizioni dell'user interface che permettono di realizzare grafici, testo, forme e animazioni. E' possibile creare widgets complessi attraverso la combinazione di widget più semplici.

Ogni progetto Android prevede un file chiamato **AndroidManifest.xml** il cui compito è quello di fornire delle informazioni essenziali riguardo l'applicazione a Android build tools, al so Android e a Google Play.
Tra le altre cose, questo file manifesto contiene obbligatoriamente le seguenti informazioni:

1. Il nome del package dell'applicazione che viene utilizzato dagli Android Build Tools per determinare la locazione del codice durante il build del progetto. Quando l'applicazione viene creata, questo valore viene rimpiazzato con un ID utilizzato per identificare univocamente l'applicazione su Google Play.

2. I permessi che è necessario fornire all'applicazione in modo che essa possa accedere a sezioni protette del sistema operativo o altre applicazioni. Ma anche i permessi che le altre applicazioni devono possedere al fine di accedere alle informazioni dell'applicazione che si sta sviluppando.

3. Le funzionalità hardware e software di cui l'applicazione ha bisogno. Queste informazioni determinano quali dispositivi possono scaricare l'applicazione da Google Play.

Nell'esempio sviluppato in questa tesi è stato necessario modificare questo file per specificare la key utilizzata per la comunicazione con i servizi di Google Maps. In particolare, è stata aggiunta la seguente riga:

```
<meta-data android:name="com.google.android.geo.API_KEY"
           android:value="[YOUR KEY HERE]"/>
```

Secondariamente, ma di eguale importanza per un progetto Flutter, è il file chiamato **pubspec.yaml** in cui vengono specificate le dipendenze del progetto. In questo file sono state aggiunte le seguenti dipendenze:

- **google_maps_flutter_heatmap**: plugin che permette la creazione di heatmaps.
- **http**: plugin che fornisce la possibilità di creare delle richieste HTTP.
- **flutter_spinkit**: plugin che fornisce delle animazioni per creare schermate di caricamento.
- **intl**: plugin che fornisce dei metodi di formattazione per le stringhe.

2.5. Arduino IDE

L'**Arduino IDE** è un software open-source che permette la stesura del codice sorgente ma anche la sua compilazione e il caricamento su una determinata scheda semplicemente cliccando sull'apposito bottone.

Un altro aspetto positivo dell'utilizzo del software Arduino è la presenza di una grande varietà di librerie che rendono semplice e veloce l'utilizzo dei dispositivi hardware. Nel caso di questo progetto sono state utilizzate molteplici librerie per semplificare lo sviluppo dello sketch.

I programmi sviluppati tramite Arduino IDE sono scritti in un linguaggio derivato dal **C++**, in cui all'utilizzatore è richiesto di definire soltanto due funzioni:

- void setup() : funzione invocata soltanto una volta all'inizio dell'esecuzione
- void loop() : funzione invocata ripetutamente fino a quando la scheda è alimentata

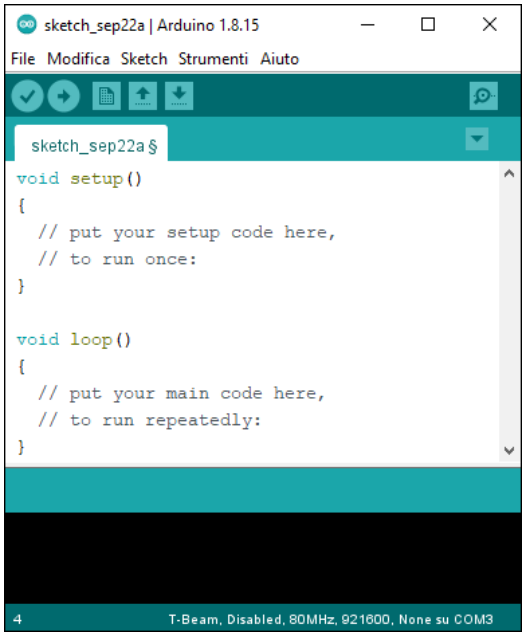

Figura 9: Interfaccia Arduino IDE

L'Arduino IDE è predisposto per essere utilizzato soltanto con le schede Arduino, tuttavia, esistono librerie che permettono l'utilizzo di altre tipologie di schede. Ad esempio, per utilizzare le schede ESP32 è necessario modificare un'impostazione relativa al gestore delle schede, ovvero, specificare il seguente URL:

`https://dl.espressif.com/dl/package_esp32_index.json`

3. Sperimentazione e Risultati

3.1. Sistema embedded

Come già detto in precedenza, allo scopo di programmare il microcontrollore è stato realizzato uno sketch Arduino le cui le due funzioni principali sono **setup()** e **loop()**. In questo capitolo si andrà a discutere nel dettaglio l'implementazione di queste due funzioni.

Le librerie utilizzate nello sketch sono le seguenti:

- **WiFi** : libreria che permette alle schede di collegarsi a internet attraverso la connessione Wi-Fi.
- **HTTPClient** : libreria utilizzata per inviare richieste HTTP a un server web.
- **TinyGPS++** : libreria che mette a disposizione i dati in uscita dal rilevatore GPS.
- **ArduinoJson** : libreria che fornisce funzionalità per la serializzazione e deserializzazione JSON.

3.1.1. Fase di setup

In questa fase vengono inizializzate **due porte seriali** che saranno successivamente utilizzate per la comunicazione della board con il dispositivo al quale essa è collegata tramite cavo USB e con il modulo GPS.

- La prima porta seriale viene utilizzata soltanto per eseguire operazioni di logging e verificare che il software sviluppato funzioni correttamente.

- La seconda porta seriale viene invece utilizzata per la comunicazione tra il microcontrollore e il GPS: è necessario specificare i pin utilizzati dal modulo per la scrittura sulla porta seriale. Questa informazione la si trova direttamente sul dispositivo.

Figura 10: Pin utilizzati dal GPS

3.1.2. Fase di loop

Nella fase di loop vengono eseguite le istruzioni che descrivono il reale comportamento del dispositivo. Di seguito è riportato un diagramma di flusso che descrive i passi concettuali più importanti.

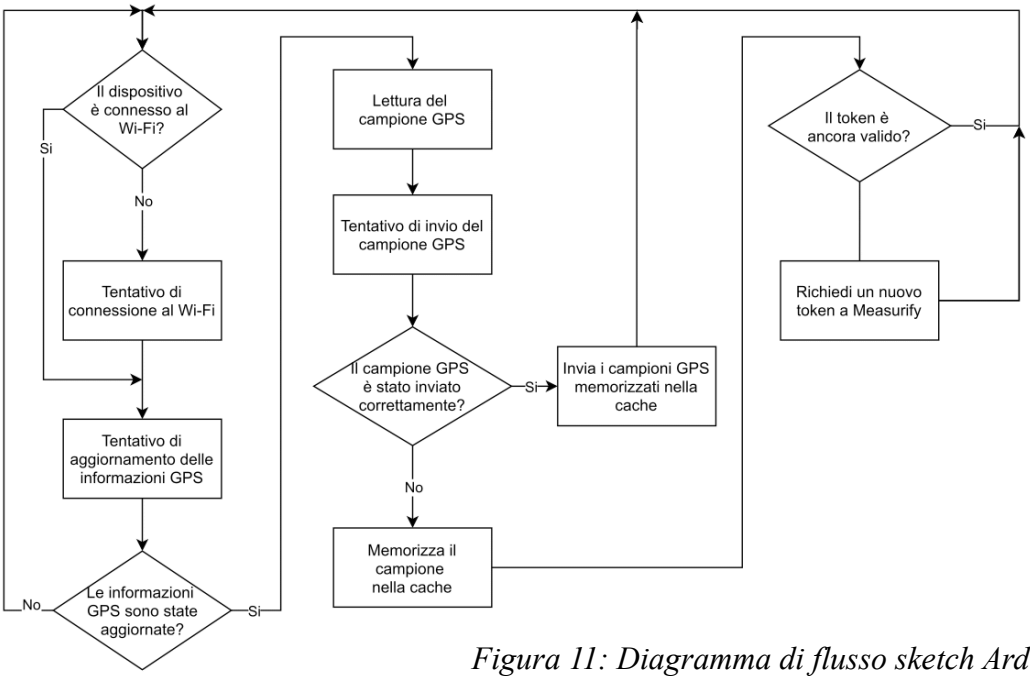

Figura 11: Diagramma di flusso sketch Arduino

I passaggi concettuali mostrati dal diagramma di flusso non sono sufficienti per capire il funzionamento effettivo del software eseguito dal microcontrollore, quindi verranno ora spiegati più dettagliatamente.

- **Verifica della connessione Wi-Fi**: la libreria WiFi contiene una funzione chiamata *WiFi.status()*, essa restituisce un valore intero grazie al quale è possibile stabilire se il dispositivo è correttamente connesso alla rete Wi-Fi. Questo passaggio è semplificato dalla definizione di alcune macro che mappano i possibili valori restituiti dalla funzione a una stringa contenente alcune parole significative. Una volta verificato che il dispositivo sia effettivamente connesso alla rete, si passa al prossimo step, altrimenti, si tenta di stabilire una connessione attraverso la funzione *WiFi.begin(ssid, password)* .

- **Aggiornamento dei dati GPS**: la libreria TinyGPS++ mette a disposizione delle funzionalità che possono essere utilizzate per decodificare i dati scritti sulla porta seriale dal modulo GPS e memorizzarli in alcune variabili. Di particolare importanza sono l'istruzione *gps.encode(Serial1.read())* che si occupa di leggere i dati scritti sulla porta seriale specificata e li interpreta, e la funzione *gps.location.isUpdated()* permette di stabilire se la lettura dei dati è completa.
  L'intera procedura è racchiusa in una funzione che restituisce un valore booleano in base al corretto aggiornamento dei dati. Questo deve avvenire entro un tempo limite specificato: può verificarsi che il dispositivo non abbia copertura GPS e quindi non possa prelevare le informazioni dai satelliti.

- **Prelevamento effettivo dei dati GPS**: se il passaggio precedente andasse a buon fine, le variabili all'interno della libreria TinyGPS++ sarebbero aggiornate correttamente e sarebbe possibile leggerne i valori attraverso una variabile di tipo struct chiamata gps.

- **Richiesta del token a Measurify**: per poter memorizzare un campione sul cloud è necessario effettuare una richiesta HTTP a Measurify e fornire un token. Quest'ultimo viene ottenuto soltanto in successione a un'operazione di login tramite un'ulteriore richiesta HTTP e, in particolare, questa deve avere verbo POST e nel body devono essere specificati username, password e tenant. Se la richiesta di login viene accettata, la risposta conterrà, all'interno di uno dei campi del body, il token desiderato.
  Dal punto di vista dell'implementazione, il body della risposta contiene una stringa in formato JSON che viene interpretata attraverso la libreria ArduinoJson, in questo modo prelevare il token dalla stringa risulta semplice.

- **Memorizzazione del campione nella cache**: durante l'esecuzione del codice si possono verificare numerose problematiche: il dispositivo può perdere la connessione Wi-Fi, la richiesta di autenticazione può fallire, la stringa in formato JSON può non essere interpretata correttamente, etc. Per fronteggiare queste eventualità è stata predisposta una cache che permette di memorizzare i campioni che non è possibile inviare per i più svariati motivi.
  La cache è stata realizzata attraverso un array la cui dimensione è calcolata in base alla memoria disponibile e alla dimensione dei dati da memorizzare. Nel caso specifico del dispositivo di questa tesi, la dimensione dell'array è tale da contenere 4166 campioni GPS prima che si verifichi overflow. Siccome i campioni sono prelevati con un intervallo di circa 5 secondi, è possibile che il dispositivo rimanga per quasi 6 ore senza connessione prima di iniziare a scartare campioni.

- **Invio dei campioni memorizzati all'interno della cache e del campione corrente**: se tutti i passaggi precedenti sono stati portati a termine senza errori è possibile inviare sia tutti gli eventuali campioni salvati nella cache sia il campione che è stato prelevato nell'iterazione corrente.

3.2. Applicazione smartphone

La seconda parte della tesi consiste nella realizzazione di un'applicazione smartphone con lo scopo di usufruire dei campioni di geolocalizzazione, raccolti dalla board ESP32, per realizzare una heatmap e mettere in evidenza alcune statistiche. L'applicazione è stata sviluppata basandosi sul framework Flutter tramite Android Studio. Le operazioni di debugging sono state effettuate attraverso uno smartphone.

3.2.1. Login

La prima schermata visualizzata all'avvio dell'applicazione è una schermata di login: è richiesto di inserire delle credenziali (username, password e tenant) in modo che possa essere effettuata una richiesta di autenticazione su Measurify.

Questa schermata è composta da tre TextFormField. Ognuno di essi comprende un riquadro rettangolare, un testo e un'icona rappresentante l'informazione da inserire. Quando uno dei campi testuali viene selezionato per inserire dei caratteri, il colore degli elementi sopra elencati varia e il testo viene spostato in alto a sinistra in modo che esso sia sempre visibile. Una funzione particolare è quella della seconda icona in quanto essa funge anche da bottone: normalmente, la password inserita è nascosta tramite pallini che vengono visualizzati al posto dei caratteri inseriti. Una pressione del bottone permette la visualizzazione della password e una seconda pressione permette di oscurala nuovamente.

Nella parte inferiore della schermata è presente un bottone che, una volta premuto, memorizza le informazioni inserite in alcune variabili e invia la richiesta di login a Measurify. Le informazioni vengono salvate dall'applicazione in modo che sia possibile effettuare nuovamente, e in maniera automatica, un'altra richiesta di autenticazione allo scadere del token.

Può verificarsi che il server rifiuti la richiesta di login.
Ad esempio, perché le credenziali inserite non sono corrette.
In questo caso, viene visualizzato un messaggio di errore nella parte inferiore dello schermo.

Se invece tutto procede correttamente e quindi la richiesta di autenticazione va a buon fine, la risposta presenta un token che potrà essere utilizzato, al posto dell'autenticazione, per le richieste successive. Per questo motivo è necessario che il valore del token venga memorizzato in una variabile.

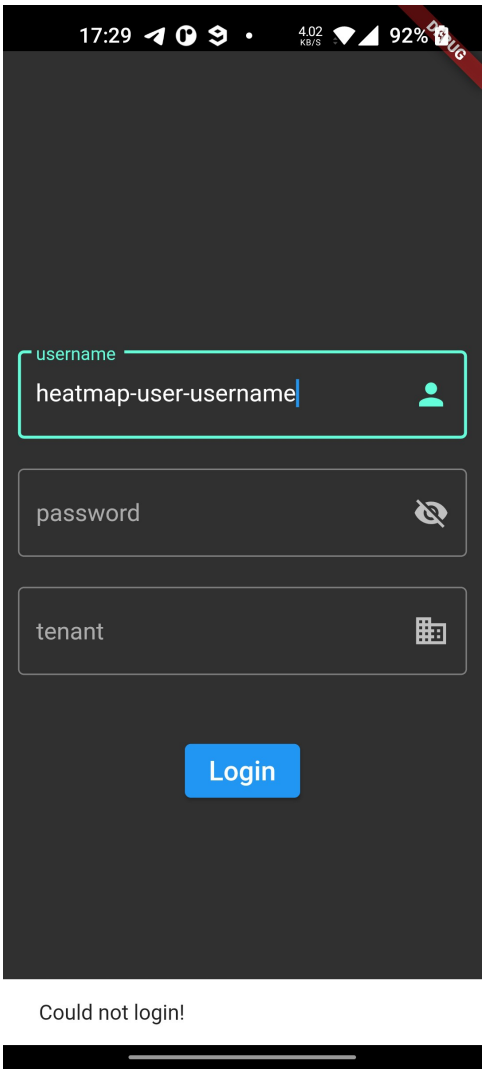

Figura 12: Schermata di login

3.2.2. Dashboard

Se la richiesta di autenticazione ha successo, la schermata di login viene rimpiazzata dalla dashboard. Questa schermata svolge la funzione di contenitore per tre sotto-schermate: calendario, heatmap e statistiche.
La dashboard è realizzata tramite l'utilizzo di un widget chiamato Barrel, esso contiene:
- Nella parte superiore dello schermo una AppBar che mostra il nome dell'applicazione.
- Nella parte inferiore dello schermo una BottomNavigationBar.

La BottomNavigationBar contiene tre bottoni raffiguranti un'icona e il nome di una delle sotto-schermate. Quando uno dei bottoni viene premuto, esso rimane selezionato con un colore differente e la dashboard viene aggiornata con il contenuto della sotto-schermata selezionata. Inizialmente è visualizzato il calendario.

3.2.3. Calendario

Lo scopo della schermata del calendario è quello di permettere all'utente di selezionare una data. A seguito di questa azione, viene formulata una richiesta e inviata a Measurify. In questo modo, l'applicazione richiede tutti i campioni memorizzati nella data specificata dall'utente. Inizialmente è impostata la data corrente.

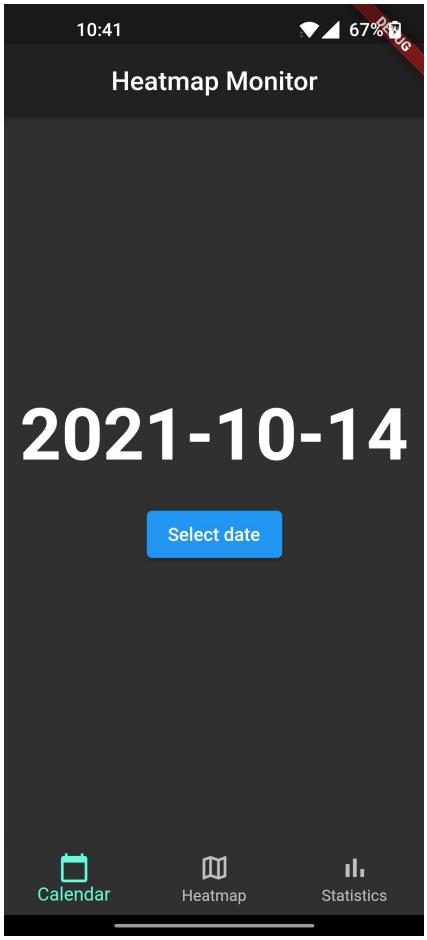

Figura 13: Schermata del calendario

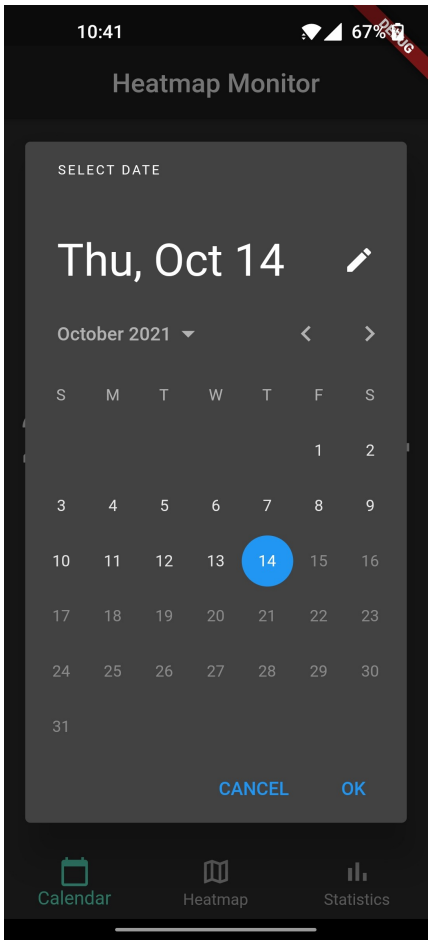

Figura 14: Overlay aperto

Nella parte centrale dello schermo è presente un widget che mostra all'utente la data attualmente selezionata. Sempre nella parte centrale dello schermo, è presente un bottone che, una volta premuto, apre un overlay tramite il quale è possibile scorrere i mesi e selezionare la data desiderata. Questa azione è controllata tramite una semplice regola che impedisce all'utente di selezionare una data superiore a quella corrente.

3.2.4. Caricamento

A seguito della risposta ricevuta da Measurify, contenente i campioni desiderati, viene visualizzata una schermata di caricamento per la necessità di eseguire della computazione locale.
Se questa schermata non esistesse, il software sarebbe costretto ad entrare in una situazione di stallo fino al termine della computazione e, a causa di ciò, l'utente potrebbe avere l'impressione che l'applicazione abbia smesso di funzionare correttamente. Quindi, lo scopo di questa schermata è comunicare all'utente, tramite un feedback visivo, che l'applicazione sta continuando a funzionare correttamente.

La computazione locale prevede i seguenti passaggi:

1. I campioni GPS sono contenuti all'interno del body della risposta tramite una stringa in formato JSON. Per la trasduzione da stringa a oggetti software viene utilizzata una libreria Dart chiamata 'dart:convert' che mette a disposizione la funzionalità desiderata. L'oggetto restituito da tale funzione è una mappa contente delle stringhe rappresentanti i campioni.

2. Prima di memorizzare i nuovi campioni è necessario scartare quelli precedenti. Per la memorizzazione viene utilizzata una lista, chiamata 'data', che è possibile ripulire completamente attraverso il tipico metodo delle liste “data.clear()” .

3. I campioni sono contenuti in un elemento della mappa a cui è possibile accedere attraverso la chiave “docs”: viene restituita una lista di stringhe ciascuna rappresentante un campione GPS.

4. Viene eseguito un ciclo su tutti gli elementi della lista appena ricavata e, ad ogni iterazione, si tenta di convertire la stringa rappresentante il campione a un oggetto di classe Sample.
   Se questo non risultasse possibile, il campione verrebbe scartato e si passerebbe al successivo.
   Se invece risultasse possibile ricavare l'oggetto, esso verrebbe aggiunto alla lista 'data'.

5. A questo punto, si è ottenuta una lista contenente degli oggetti che rappresentano i campioni GPS. Questi vengono utilizzati per calcolare le statistiche che verranno poi mostrate nella schermata apposita. Calcolare le statistiche in questo punto permette di eseguire la computazione soltanto una volta.

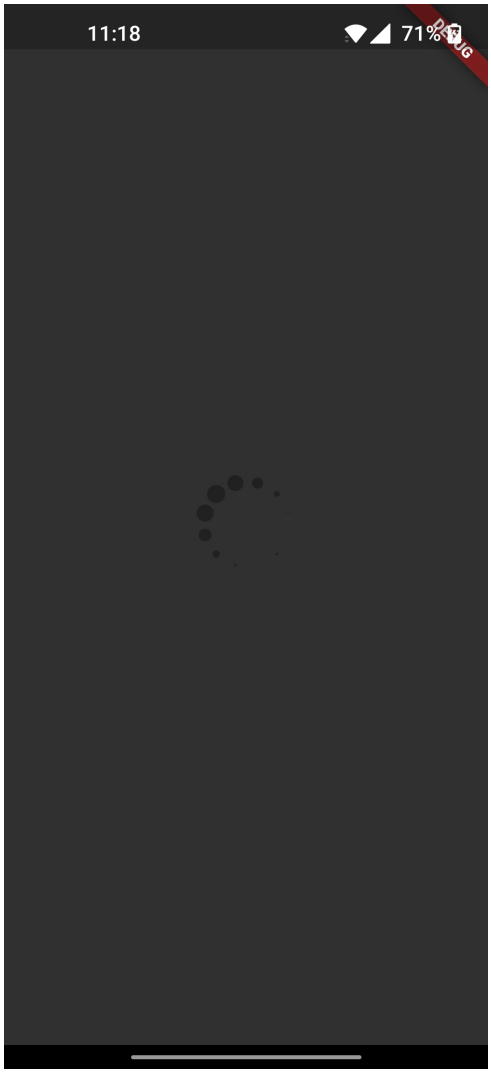

Figura 15: Schermata di caricamento

Al termine delle operazioni appena descritte, la schermata di caricamento viene chiusa e viene riaperta automaticamente la schermata del calendario nella quale sarà visualizzata la data appena selezionata.
Grazie al fatto che la computazione è stata eseguita in background durante il caricamento, non si possono verificare rallentamenti delle prestazioni quando si naviga da una sotto-pagina all'altra.

3.2.5. Heatmap

In questa sezione viene analizzata la schermata principale dell'applicazione, ovvero, la schermata in cui è visualizzata la heatmap sulla quale vengono mostrati i campioni prelevati dalla scheda ESP32. La realizzazione prevede l'utilizzo di un plugin di Flutter chiamato 'google_maps_flutter_heatmap' che fornisce un widget apposito per la creazione di questa tipologia di mappe. Sono necessari alcuni passaggi preliminari:

1. Affinché il plugin possa essere importato correttamente nel progetto, è necessario specificarne il nome e la versione all'interno del file pubspec.yaml presente nella directory del progetto.

2. Alcuni servizi forniti da Google Maps necessitano l'utilizzo di una key in base alle funzionalità di cui si vuole usufruire. Tale key deve essere specificata all'interno del file AndroidManifest.xml altrimenti si verificano errori nel momento in cui si tenta di visualizzare la heatmap.

L'intera sotto-pagina è composta da un unico widget chiamato GoogleMap e per il quale è necessario specificare alcuni parametri importanti:

- **mapType**: tramite questo parametro si specifica il tipo di mappa che si vuole visualizzare all'interno del widget. Ne esistono di quattro tipi:
  - roadmap: mostra la mappa “standard” in cui vengono visualizzate soltanto le strade e alcune informazioni sui punti di interesse.
  - satellite: mostra una mappa ricavata dalle immagini satellitari di Google Earth.
  - hybrid: mostra una mappa ottenuta dall'unione della mappa stradale e delle mappe satellitari.
  - terrain: mostra una mappa fisica basata sulle informazioni di altitudine del terreno.

  In questa tesi, è stata impostata la rodmap.

- **initialCameraPosition**: questo parametro è un oggetto di tipo CameraPosition e viene utilizzato per specificare la posizione iniziale della mappa. Per poter creare un oggetto di questo tipo è necessario passare come argomenti al costruttore una posizione geografica e un valore decimale che corrisponde allo zoom iniziale della mappa.

- **heatmaps**: questo è il parametro più significativo perché contiene una lista di oggetti di tipo Heatmap che andranno ad essere visualizzati sulla mappa.

Figura 16: Schermata heatmap

L'oggetto Heatmap che viene passato in ingresso al widget GoogleMap attraverso l'ultimo parametro, è creato all'inizializzazione della schermata all'interno del metodo “initState()”. A questo scopo, la lista di oggetti Sample, creata durante la schermata di caricamento, viene mappata in una lista di oggetti di tipo WeightedLatLng in modo che possa essere passata come parametro al costruttore della Heatmap.
Il costruttore dell'oggetto prende in ingresso anche altri parametri, meno significativi ma tuttavia utili a definire la grafica con cui la heatmap viene disegnata all'interno della schermata del widget.

3.2.6. Statistiche

La schermata delle statistiche è l'ultima schermata realizzata per l'applicazione. Essa mostra alcune statistiche che vengono ricavate attraverso computazione locale semplicemente invocando un'apposita funzione e memorizzando i valori ottenuti in una variabile.

Nella parte centrale dello schermo è presente un widget chiamato Column il cui compito è quello di essere un raccoglitore di altri widget e visualizzarli in verticale.
Le statistiche vengono mostrate attraverso un widget chiamato StatisticsField e realizzato appositamente per questo scopo: esso comprende due stringhe ai due lati opposti dello schermo che rappresentano rispettivamente il nome e il valore assunto dalla statistica.
Si nota come una Column possa contenere qualsiasi tipologia di widget, infatti, le StatisticsField sono raggruppate e separate attraverso un altro widget chiamato Divider che viene visualizzato attraverso delle linee grigie orizzontali.

Le statistiche mostrate in questa pagina sono:

- **Total Samples**: il numero totale di campioni prelevati nella data selezionata. Per ottenere questo valore si legge la dimensione della lista dei campioni attraverso l'apposito metodo.
- **Valori massimi e minimi**: vengono mostrati i valori estremi di latitudine, longitudine e altitudine. Questi valori sono ottenuti semplicemente attraverso un ciclo for-in su tutti i campioni.
- **Travelled Distance**: la distanza percorsa dell'utente a partire dal primo campione fino all'ultimo. Per ottenere questo valore è stata calcolata la distanza tra ogni coppia di campioni attraverso la formula di Haversine che, tramite approssimazione della forma della Terra, permette di calcolare la distanza tra due coordinate geografiche espresse in latitudine e longitudine.
- **Trip Time**: intervallo tempo trascorso tra il primo e l'ultimo campione.

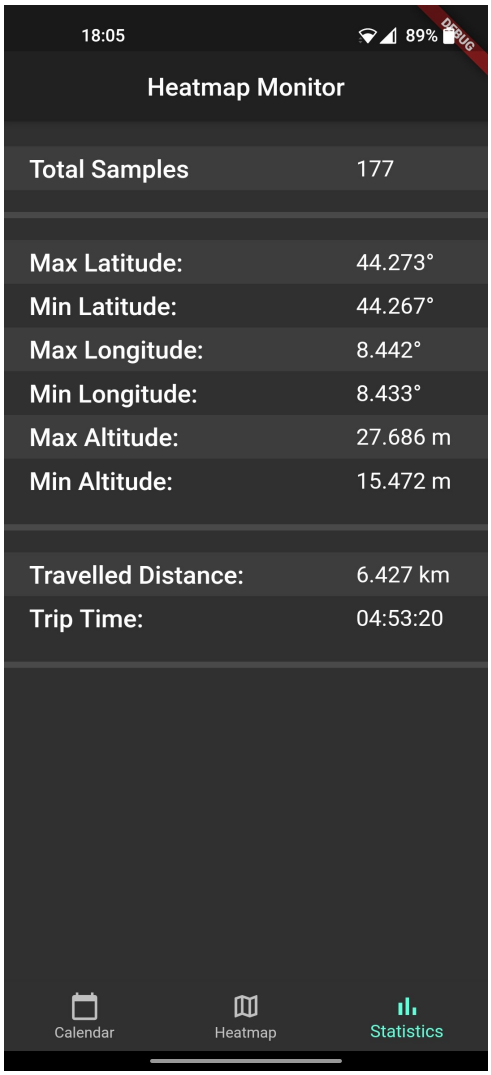

Figura 17: Schermata statistiche

4. Contributo personale e considerazioni conclusive

L’obbiettivo di questa tesi era quello di progettare e implementare un sistema embedded allo scopo di rilevare e memorizzare su cloud informazioni sulla posizione dell’utente che trasporta il dispositivo e, in un secondo momento, utilizzare tali dati di geolocalizzazione per ottenere una heatmap degli spostamenti.

Al fine della programmazione del microcontrollore, l'utilizzo dell'IDE Arduino ha facilitato lo sviluppo di codice grazie a un'ottima documentazione, un'ampia quantità di materiale disponibile sul web e una community disponibile ad aiutare gli utenti in difficoltà. Tuttavia, lo strumento che più ha facilitato la programmazione è stato è il gestore delle librerie facente parte dell'ambiente di sviluppo. Questo componente permette, con pochi e semplici passaggi, di cercare e installare librerie per qualsiasi funzionalità si desideri. Molto probabilmente la realizzazione di questa tesi non sarebbe stata possibile senza la presenza delle librerie che sono state utilizzate per la connettività Wi-Fi, l'utilizzo del modulo GPS integrato, la gestione delle richieste e risposte HTTP e la conversione da stringhe formattate JSON in variabili di facile utilizzo.

Data la necessità di realizzare un'applicazione per dispositivi mobile dotata di interfaccia grafica, è stato necessario basarsi su un framework cross-platform, come Flutter, che rendesse semplice, intuitivo e veloce lo sviluppo di software. A tal scopo, è stata eseguita una ricerca e uno studio della documentazione del framework. Per la stesura effettiva del codice è stato utilizzato Android Studio e, nonostante l'ambiente di sviluppo metta a disposizione un emulatore di dispositivi mobile, si è preferito utilizzare un dispositivo smartphone fisico per testare l'applicazione a causa di problematiche dello strumento in alcune situazioni.
Inoltre, si è dedicata particolare attenzione all'esperienza dell'utente. Uno degli elementi chiave per questo scopo è la schermata di caricamento, il cui compito è quello di comunicare all'utente che l'applicazione sta continuando a funzionare correttamente ed è necessario attendere soltanto qualche secondo per ottenere il risultato richiesto. Altrettanto importante è l'utilizzo di un interfaccia grafica minimalista che permette di comprendere velocemente e in maniera intuitiva come utilizzare l'applicazione.

Un punto cruciale nello sviluppo del sistema IoT è stato quello di capire come formulare correttamente le richieste HTTP e inviarle alle API di Measurify in modo da poter memorizzare e prelevare i campioni. La problematica è stata risolta tramite l'utilizzo di librerie sia nello sviluppo del sistema embedded, sia nello sviluppo dell'applicazione. In quest'ultimo caso in particolare, una libreria di Flutter, chiamata “http”, permette allo sviluppatore di realizzare una comunicazione con il server in maniera molto intuitiva.

Riguardo a un possibile sviluppo futuro, si potrebbe affrontare la problematica della gestione dei consumi del sistema embedded progettato. Un miglioramento attuabile consiste nel non inviare singoli campioni al cloud ma memorizzarne un determinato numero nella cache e poi inviarli in gruppo in un secondo momento.

Le possibili applicazioni di questo progetto sono molteplici:

- Monitorare giornalmente gli spostamenti delle persone afflitte da malattie degenerative come le diverse tipologie di demenza, tra cui anche il morbo di Alzhaimer.

- Supponendo che ogni persona sia fornita del sistema embedded, sarebbe possibile realizzare delle heatmap per determinare il grado di affollamento, in un determinato luogo, in tempo reale oppure allo scopo di analisi a posteriori.

- Favorire analisi statistiche in ambito sportivo dato che, oltre alla generazione della heatmap, è possibile estrarre dai campioni alcune informazioni di interesse come già mostrato nella schermata apposita.

Complessivamente, sono molto soddisfatto del risultato ottenuto tramite questa tesi sia dal punto di vista della formazione personale e delle capacità acquisite, sia dal punto di vista del prodotto realizzato. In particolar modo, sono contento di essere venuto a conoscenza di Flutter perché lo reputo uno strumento molto interessante per la sua grande potenzialità in ambito di sviluppo di applicazioni mobile: realizzare un'interfaccia è semplice, veloce e penso che anche con poca pratica si possano ottenere grandi risultati.

5. Riferimenti bibliografici

- Measurify, https://measurify.org/
- Android Studio, https://developer.android.com/studio
- Flutter, https://flutter.dev/
- Dart, https://dart.dev/
- Arduino, https://www.arduino.cc/
- Microcontrollore ESP32, https://www.espressif.com/en/products/socs/esp32
- LILYGO T-Beam, http://www.lilygo.cn/prod_view.aspx?TypeId=50033&Id=1237&FId=t3:50033:3
- IoT, Materia del corso di studi: “Approccio Makers alla progettazione elettronica”