Sintesi del contenuto:

Il progetto consiste in un’applicazione sviluppata in flutter il cui scopo è quello di registrare delle attività (corsa/auto/bici) tramite il gps del cellulare, creare il tragitto del percorso su una mappa ed inviare i dati su un database in un raspberry in modo da poterli elaborare e graficare.

Linguaggi:

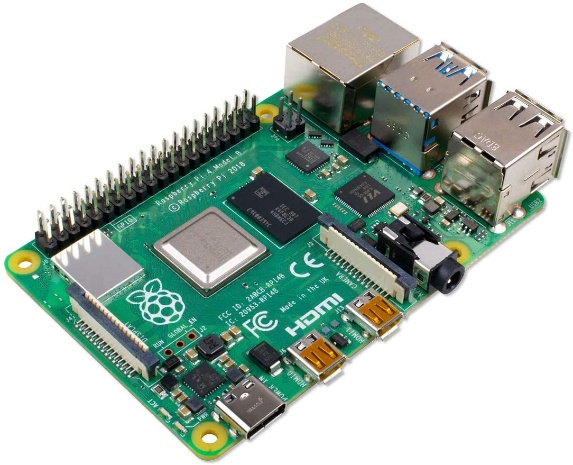
* **SQL:** Usato per la gestione del database.
* **PHP:** Usato per la gestione di richieste login/registrazione/salvataggio dati/scaricamento dati.
* **Dart:** Principale linguaggio usato per lo sviluppo dell’applicazione, è molto simile a java ed è stato ideato da Google per lo sviluppo di applicazioni web.
* **Flutter:** È il framework messo a disposizione da Google che si basa su dart. Il suo funzionamento è basato principalmente sul “nesting” di widgets: ogni applicazione sviluppata in Flutter parte da un widget padre, il quale conterrà n widgets figli, i quali a loro volta ne conterranno degli altri. Un widget è un qualsiasi elemento grafico visibile o non, creato dal programmatore o presente di default nelle librerie.



Strumenti:

**strumenti fisici**:

* Raspberry PI 4 (Host per il database e comunicazione client-server)
* Realme 8 pro (Usato per testare l’app)



**strumenti digitali**:

* [MobaXTerm](https://mobaxterm.mobatek.net/) (Usato per connettersi via ssh al raspberry e per creare tunnels alle varie porte).
* [AndroidStudio](https://developer.android.com/studio) (IDE usato per programmare in dart l’applicazione)
* [PhpMyAdmin](https://www.phpmyadmin.net/) (Gestione database)
* [No-ip](https://www.noip.com/it-IT) (hostname per ip dinamici)
* [OpenStreetMap](https://www.openstreetmap.org/) (host della mappa del mondo)
* Apache (webserver)
* Grafana (per l’analisi dei dati)

Obiettivi:

L’obiettivo del progetto è quello di creare un’applicazione che possa registrare delle attività sportive, disegnarle su una mappa e salvarle su un database per poter successivamente analizzare i dati.

Un obiettivo aggiunto del progetto è stato imparare ad usare Flutter e Dart per via dei vantaggi rispetto ad usare solo Java, uno dei quali è la possibilità di creare applicazioni in contemporanea sia per IOS che per Android senza dover riscrivere l’applicazione da capo. Purtroppo in questo caso, non avendo a disposizione un mac per la compilazione ed un ios per il testing, non è stato possibile verificare il funzionamento dell’app su tale piattaforma.

Struttura e Realizzazione (1):

**Lato Client**:

L’app è formata da diverse componenti:

* **Login\_screen/register\_screen**: Sono le due pagine iniziali, si occupano della ricezione e dell’invio delle credenziali con il server. Più precisamente, ricevono l’input dell’utente, lo criptano e lo inviano in forma di richiesta POST al server.
* **HomePage\_screen**: È la pagina principale dell’applicazione, nella quale si può visualizzare la lista delle attività (dell’utente corrente) presenti nel database, rinominarle, eliminarle o crearne di nuove. Subito dopo il login, appena prima della creazione della HomePage, viene fatta una richiesta al server di tutte le attività dell’utente in modo da poterle visualizzare in colonna (sono le varie istanze di activity\_row.dart).
* **Activity\_screen**: È la schermata che si apre dopo aver premuto un’ActivityRow, al momento dell’inizializzazione della classe, verrà inoltrata una richiesta al server per lo scaricamento di tutte le informazioni dell’attività (lat,lang,alt,speed,date\_time) e verrà visualizzato il tragitto nella mappa. È inoltre presente un floatingButton per poter centrare la posizione della mappa sulla la posizione dell’attività.
* **RegisterActivity\_screen:** Questa classe si occupa della registrazione delle nuove attività, è composta dalla mappa del mondo e 3 floatingButtons, uno per iniziare/fermare la registrazione, un altro per centrare la posizione della mappa sulle coordinate dell’utente e l’ultimo per caricare i dati raccolti sul server.
* **User\_Data:** È la classe che si occupa di detenere tutte le informazioni (eccetto la password) dell’utente corrente.

Le classi contenute nel package “components” sono tutte oggetti che vengono istanziati molteplici volte, mentre la classe functions.dart contiene funzioni generali come encrypt(String s), dectypt(String s), ecc…

Struttura e Realizzazione (2):

**Lato Server**:

* **Register.php:** Riceve i parametri POST username, mail, password, li analizza per verificare che rispettino il formato richiesto (lunghezza minima password 5, massima 32, ecc..), controlla che l’utente non sia già registrato e in tal caso, lo aggiunge al database “hashando” la password. Infine invia una mail di verifica tramite mailSender.py.
* **MailSender.py**: Riceve in input la mail del destinatario ed il messaggio. Tramite un profilo gmail creato per il progetto, ed abilitato per poter mandare mail in automatico, invia la mail contenente il codice di verifica dell’account appena registrato.
* **Login.php**: Riceve mail e password dal client, controlla se ci sono corrispondenze nel database e ritorna il risultato della query.
* **Activate.php**: Attende che un utente clicchi il messaggio di conferma nella casella di posta, se succede, imposta l’utente come “activated” in modo da permettergli di registrare nuove attività nell’app.
* **getActivity/getCoordinates**: Ricevono in input user\_id e activity\_id per poter ritornare rispettivamente la lista delle attività e la lista di CoordPoints richiesti.
* **RegisterActivity**: Riceve in input le informazioni di un’attività ed una lista di coordPoints, organizzati in forma (#lat\*long\*alt\*speed\*time) dove “#” indica l’inizio di una nuova coordPoint e “\*” separa i dati della singola riga e li aggiunge al database.
* **RenameActivity/DeleteActivity**: Funzionano in modo simile tra loro, servono per rinominare o eliminare un’attività dal database.
* **Credentials.php:** È un “util” creato per non dover riscrivere le credenziali del database in ogni file php richiamando invece *require\_once(credentials.php);*

MYSQL:

Come database è stato usato MariaDB con PHPMyAdmin e vi sono state create tre tabelle SQL:

* user(id, mail, username, password, activation\_code)
* activity(id, activity\_name, date\_time, user\_id)
* coord\_point(id, activity\_id, latitude, longitude, altitude, apeed, date\_time)

Grafana:

Tramite grafana si può costruire grafici sui dati raccolti, ad esempio per visualizzare i luoghi dove le attività vengono svolte, è stata usata la seguente query:

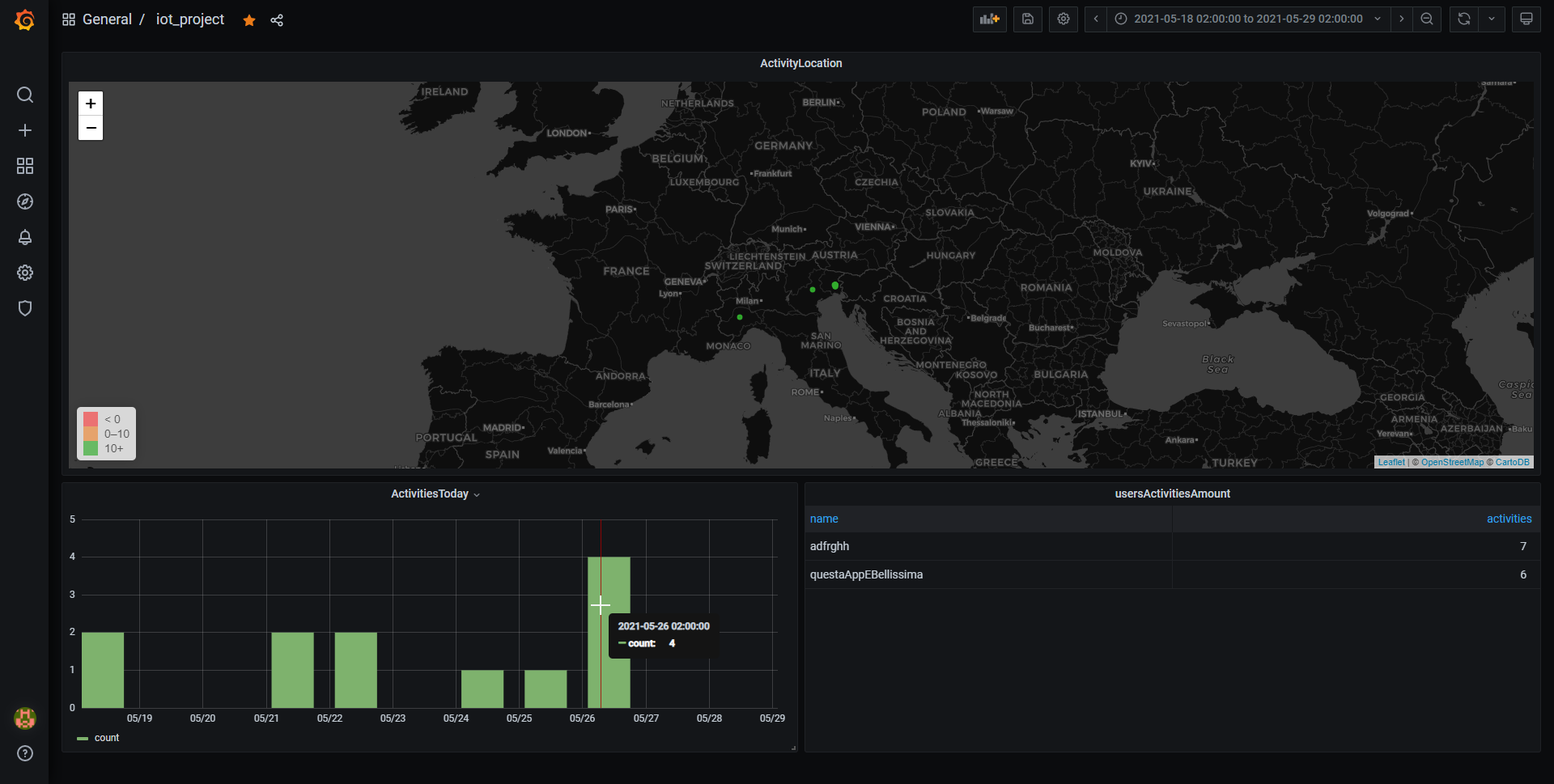
*SELECT activity.id as value, activity.activity\_name, coord\_point.latitude, coord\_point.longitude, user.username as name*

*FROM activity*

*INNER JOIN user ON activity.user\_id=user.id*

*INNER JOIN coord\_point ON activity.id=coord\_point.activity\_id*

*GROUP BY value*



È stato deciso di usare grafana come un “admin control panel”, per visualizzare dati generali relativi all’uso dell’app (come le attività medie in un giorno, il numero totale di attività registrate, ecc..) in quanto l’analisi delle singole attività richiederebbe un numero molto elevato di grafici e sarebbe sicuramente più conveniente usare un’alternativa lato client.

Osservazioni e Conclusioni:

Come per qualsiasi applicazione, le migliorie applicabili a livello di grafica, sicurezza ed efficienza, sono svariate, sicuramente però, nel caso dovesse venir pubblicata, sarebbe necessario rivedere il sistema di sicurezza di comunicazione: attualmente si basa sul criptare tutto il contenuto delle richieste POST tramite una funzione all’interno del programma. Il modo migliore sarebbe sostituire questo sistema con un protocollo ssl. Infine anche il codice dell’applicazione deve esser offuscato in modo non essere leggibile dopo un possibile decompiling.

Questo progetto mi ha permesso di approfondire le conoscenze in quanto programmazione, comunicazione client-server e IoT in generale, imparando allo stesso tempo ad usare un Raspberry, Dart, Flutter e Grafana.