



Università degli Studi di Parma
Dipartimento di Ingegneria e Architettura

Progetto di Programmazione Sistemi Mobili

E-Motion Application

Davide Reverberi

Indice

Scopo generale	3
Requisiti dell'applicazione	4
Guida all'installazione e all'uso	6
Avvio e Gestione del Web Service	6
Installazione dell'applicazione.....	7
Funzionamento di E-Motion App.....	9
Tecnologie Utilizzate	12
Implementazione del Bluetooth	12
Database MongoDB.....	14
OpenWeather API	14
Architettura Software	15
Struttura dell'applicazione	15
Struttura del server	18
Link Utili	20

Scopo generale

L'applicazione E-Motion è sviluppata con l'obiettivo di fornire agli utenti un'interfaccia intuitiva e funzionale per il controllo a distanza di un robot specializzato nella pulizia dei pannelli solari. Essa intende migliorare e rendere più facili ed efficienti le operazioni di pulizia, contribuendo così all'ottimizzazione della produzione di energia solare e alla riduzione dei costi di manutenzione.

Obiettivi principali:

1. Controllo del robot

Fornire agli utenti la possibilità di controllare direttamente il movimento dei cingoli del robot tramite slider interattivi. Inclusione di pulsanti funzionali per avviare, interrompere e gestire le operazioni di pulizia.

2. Gestione degli utenti

Fornire funzionalità per consentire la registrazione e la successiva autenticazione degli utenti che intendono utilizzare l'applicazione.

3. Consultazione delle previsioni meteo

Rendere possibile agli utenti registrati e autenticati di consultare in tempo reale le previsioni meteorologiche della città di interesse per ottenere informazioni relative alle ore successive al momento della consultazione. Il fine ultimo è quello di pianificare le operazioni di pulizia in modo ottimale.

Nel complesso, l'applicazione è progettata per essere utilizzata dai proprietari di impianti solari di piccole-medie dimensioni, dai tecnici della manutenzione e da altri operai del settore. Essa è inoltre sviluppata per garantire facilità di utilizzo attraverso un'interfaccia immediata e semplice, accessibile mediante smartphone in pochi semplici passaggi.

E-Motion rappresenta quindi uno strumento innovativo e strategico di supporto per permettere un maggior controllo nelle operazioni di manutenzione dei pannelli solari e per una pianificazione efficiente delle stesse operazioni, in base al meteo.

Requisiti dell'applicazione

Si specifica che lo sviluppo di E-Motion è mirato alla realizzazione di un modello prototipale, non adatto al rilascio pubblico, per via di alcuni aspetti legati alla sicurezza delle informazioni e dell'implementazione del Web Service associato, non attualmente pronto per l'esposizione sulla rete internet.

Requisiti funzionali

1. Controllo del movimento

L'applicazione deve fornire due slider interattivi che assumono valori da 0 (fermo) a 100 (potenza erogata massima) e li trasmettono ai relativi motori, uno per il controllo del movimento verticale (avanti/indietro) del robot e l'altro per il controllo orizzontale (destra/sinistra). Entrambi gli slider devono ritornare alla posizione neutra a metà (valore 50) quando l'utente li rilascia. Si prevede anche un pulsante dedicato allo spegnimento di emergenza dei motori.

2. Controllo delle operazioni di pulizia

L'applicazione deve fornire due pulsanti per il controllo delle operazioni di pulizia: un pulsante dedicato al rilascio di acqua e l'altro dedicato all'attivazione delle spazzole del robot.

3. Gestione degli utenti

L'applicazione deve garantire che l'utente sia in grado di accedere ai controlli del robot in seguito ad un'apposita autenticazione. Dev'essere inoltre possibile la registrazione di nuovi utenti, permettendo loro di creare account personali identificati da username, email e password.

4. Consultazione delle previsioni metereologiche

L'utente deve poter accedere ad un'interfaccia dedicata al recupero delle informazioni metereologiche relative alla città inserita. L'applicazione deve mostrare le previsioni delle ore successive in modo chiaro e comprensibile, per facilitare l'utente nella pianificazione delle operazioni di pulizia.

Requisiti non funzionali

1. Prestazioni generali

Gli elementi grafici della schermata dei controlli devono rispondere quasi istantaneamente alle interazioni con l'utente, con al massimo un ritardo di mezzo secondo. I pulsanti che gestiscono le richieste di login, registrazione e meteo devono rispondere in tempi ragionevoli, non superiori ai 20 secondi, e che si riporti all'utente l'errore, in caso di operazione non a buon fine.

2. Prestazioni del Web Service

Il web service associato ad E-Motion deve rispondere alle richieste http che riceve in modo rapido, di conseguenza è necessario non appesantire con informazioni non richieste o elaborazioni eccessive le chiamate API al servizio meteo, così come la consultazione del database online per recuperare le informazioni relative ai dati degli account utente.

3. Utilizzo del Bluetooth

L'utente deve poter accedere all'interfaccia di controllo solamente se autenticato e se connesso con successo al dispositivo Bluetooth associato al robot. L'autenticazione avviene solamente dopo che all'utente è stato notificato lo stato della connessione con il dispositivo Bluetooth.

4. Usabilità ed accessibilità

L'applicazione dev'essere intuitiva e facile da utilizzare, con una navigazione chiara tra le schermate. Almeno metà degli utenti che dispongono in tutto il mondo di smartphone con sistema operativo Android, devono poter scaricare l'applicazione ed utilizzarla efficacemente.

5. Manutenibilità e scalabilità

Deve essere prevista una struttura modulare per consentire l'aggiunta di nuove funzionalità che non vada ad impattare sulle componenti esistenti. Il prototipo realizzato deve essere predisposto per l'introduzione dei protocolli di sicurezza e per la distribuzione online del web server realizzato, al fine di un futuro rilascio pubblico. L'applicazione deve essere in grado di gestire un numero crescente di utenti senza degradare di prestazioni.

Guida all'installazione e all'uso

Tutti i file specificati nei paragrafi successivi sono contenuti nella cartella del progetto insieme a questa relazione oppure scaricabili [qui](#) dalla repository ufficiale di GitHub del progetto. Per arrivare all'applicazione funzionante è necessario prima avviare il server contenente il Web Service e poi successivamente installare l'applicazione sul dispositivo.

Avvio e Gestione del Web Service

L'avvio del server è necessario in quanto l'applicazione è sviluppata nella sua versione prototipale. Nella versione definitiva si intende rendere questo passaggio non più necessario, rendendo pubblica e raggiungibile su internet da tutti gli utenti una distribuzione del server.

Prima di avviare il server sul computer o sul generico dispositivo Host a disposizione, è necessario sottolineare che se si intende installare e provare l'applicazione su uno degli emulatori supportati disponibili su Android Studio, il server dev'essere avviato sullo stesso dispositivo su cui viene eseguita l'applicazione con l'IDE.

Per avviare il server in Windows, bisogna posizionarsi nel percorso in cui risiede il file Python del web service (WebService.py) e digitare i seguenti comandi dopo aver aperto la console:

```
$env:FLASK_APP = "WebService"  
flask --app WebService run --host 0.0.0.0 --port=5000
```

Nel caso si avesse UNIX come sistema operativo bisogna specificare solo il secondo comando.

Una volta avviato, il server sarà in ascolto su tutte le interfacce della rete di riferimento, quindi sarà individuabile e raggiungibile da tutti i dispositivi connessi alla stessa rete attraverso protocollo http.

Installazione dell'applicazione

L'applicazione è sviluppata per versioni di Android superiori o uguali a 12 (Snow Cone). Il tentativo di far funzionare E-Motion su dispositivi con versioni precedenti potrebbe portare a bug di varia natura o addirittura impossibilità di avviare l'app stessa.

Qualora non si disponesse di un dispositivo Android con una versione supportata, si può sfruttare uno tra i vari emulatori messi a disposizione su Android Studio, dopo aver importato ed eseguito correttamente il progetto rinominato come **"E-MotionApp-Emulator"** nell'IDE. Tuttavia, questa modalità di utilizzo è restrittiva e valida solo per una verifica generale dell'applicazione, per via dell'impossibilità di integrare le funzionalità Bluetooth (e quindi di controllo del robot) dell'emulatore.

Si identificano due modalità di installazione con un dispositivo fisico:

1. Installazione ed avvio dall'IDE di Android Studio

Dopo aver opportunamente collegato il dispositivo Android al computer con Android Studio installato, è necessario importare il progetto **"E-MotionApp"** nell'IDE ed avviarlo utilizzando come target il dispositivo personale.

Prima di eseguire l'applicazione è tuttavia necessario modificare nel file **WebService.kt** l'indirizzo IP che identifica il dispositivo su cui è stato avviato il server, andando a sostituire la stringa “**http://10.0.2.2:5000**” con “**http://<Ip-address>:5000**” dove **<Ip-Address>** dev'essere sostituito con l'IP del dispositivo, della stessa rete, su cui è stato avviato il server.

```
val retrofit = Retrofit.Builder() Retrofit.Builder
    .baseUrl(" http://192.168.1.56:5000 ") //Change this with the IP of the server Device
    .addConverterFactory(MoshiConverterFactory.create(moshi)) Retrofit.Builder
    .client(okHttpClient)
    .build()
```

Esempio in cui si è utilizzato l'indirizzo IP 192.168.1.56

Questo passaggio è necessario affinché il client (l'applicazione) sia in grado di raggiungere correttamente il server (computer o generico dispositivo su cui è in esecuzione il Web Service).

Se l'applicazione non dovesse avviarsi correttamente si rimanda al successivo paragrafo di funzionamento di E-Motion App, per la parte relativa ai permessi. Questa modalità di utilizzo dell'applicazione permette di verificare anche l'eventuale connessione Bluetooth con il robot (identificato dal generico modulo Bluetooth HC-05).

2. Installazione ed avvio dall'apk dell'applicazione

L'installazione dell'applicazione avviene attraverso il file apk contenuto nella cartella del progetto. Anche in questa modalità è possibile utilizzare il Bluetooth per la connessione con il modulo HC-05, se a disposizione dell'utente. Con quest'ultima versione dell'applicazione non sarà possibile però contattare il server per le operazioni di login, registrazione e quelle relative alle informazioni sul meteo, ma solamente verificare il funzionamento del Bluetooth.

Si riassumono di seguito le 3 modalità di installazione e le caratteristiche:

- Progetto **E-MotionApp-Emulator** : Per provare solo funzioni del server
- Progetto **E-MotionApp** : Per testare tutte le funzioni con proprio device
- Apk **E-MotionApp** : Per provare solo funzionalità con Bluetooth

Se non si ha a disposizione il modulo Bluetooth HC-05 da connettere, si consiglia la prima modalità (con emulatore), in quanto si prevede comunque la possibilità nell'applicazione di navigare in tutte le schermate anche senza connessione Bluetooth.

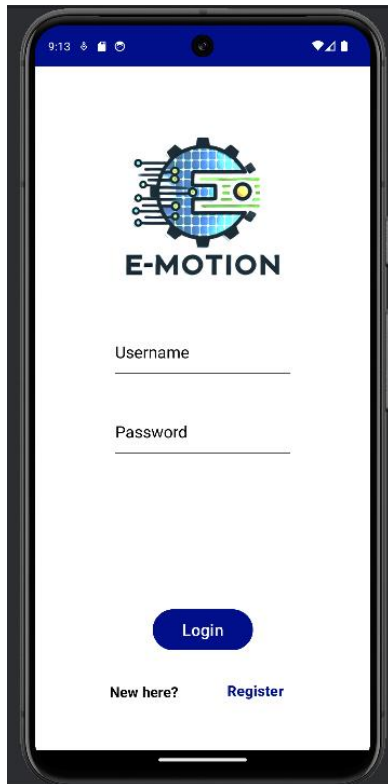
Funzionamento di E-Motion App

Una volta correttamente installata l'app sul dispositivo Android, è necessario verificare che siano stati accettati i permessi richiesti. Qualora all'avvio dell'applicazione non comparisse un pop-up automatico di richiesta dei permessi relativi al Bluetooth e della localizzazione e quest'ultima non riuscisse ad avviarsi, si deve procedere manualmente con l'abilitazione dei permessi dalle impostazioni del dispositivo, nella sezione apposita di gestione applicazioni, per l'applicazione E-Motion.

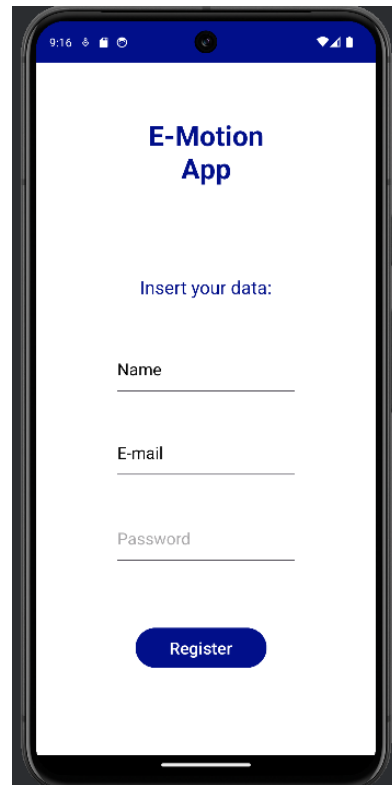
Schermata di login e di registrazione

Una volta avviata l'applicazione e verificati i permessi associati comparirà sul display la schermata di login (vedi immagine 1 sotto). Qui sarà possibile compilare i relativi campi con le proprie credenziali per poter accedere ai controlli del robot. Qualora invece non si avesse già creato un account personale sarà possibile farlo premendo sul pulsante “register” in fondo alla schermata. Si aprirà un'apposita interfaccia per la registrazione (Vedi immagine 2 sotto) in cui si dovranno inserire le proprie credenziali.

Nell'applicazione (tranne nella versione sviluppata per test sull'emulatore) non sarà possibile accedere all'interfaccia dei controlli senza che sia avvenuta la connessione con il modulo Bluetooth HC-05 (presente all'interno del robot, è quindi necessario prima effettuare l'associazione, poi avviare l'app.



1. Schermata di login



2. Schermata di registrazione

Per velocizzare i tempi di accesso ai controlli si può effettuare il login con le credenziali: username: *guest* e password: *guestpassword*

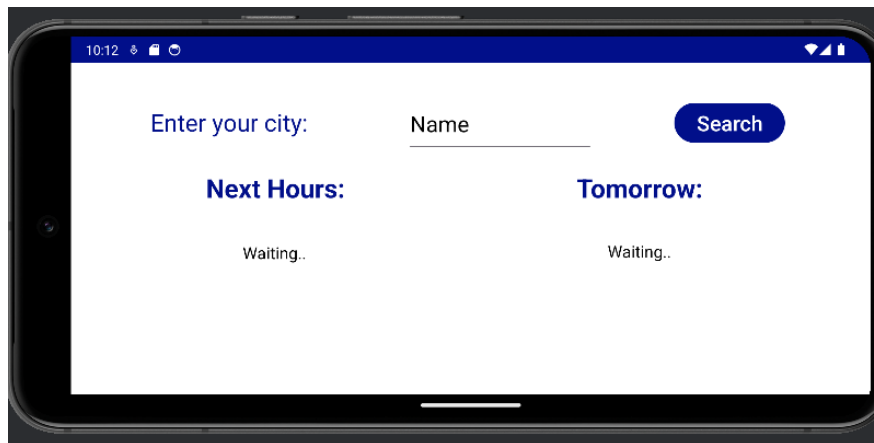
Schermata dei controlli e del meteo

Una volta loggati al sistema si accederà all'interfaccia dei controlli (vedere immagine 3 successiva) in cui sarà possibile manovrare il robot tramite i due slider interattivi, utilizzare i pulsanti di pulizia (clean), acqua (water) e interruzione dei motori (stop) e accedere all'interfaccia con le informazioni sul meteo attraverso l'apposito pulsante (weather).

Nell'interfaccia di consultazione delle previsioni metereologiche (vedere immagine 4 successiva) sarà richiesto di indicare la città di interesse per la quale si è interessati alle informazioni. Sarà la pressione del tasto apposito Search che in meno di pochi secondi (in base alla raggiungibilità e velocità del server) riporterà le informazioni negli slot sottostanti.



3. Schermata dei controlli



4. Schermata delle previsioni metereologiche

Ciascuna di delle quattro interfacce presentate prevede la gestione degli errori di varia natura che possono verificarsi. Per esempio, se le credenziali della schermata di login non fossero corrette oppure il server o il database non fossero raggiungibili, verrà riportato un errore e verrà impedito l'accesso alla schermata dei controlli. Analogamente, qualora non venissero compilati tutti i campi nella schermata di registrazione oppure lo username indicato fosse già esistente, si informerà l'utente con un opportuno messaggio riportato nella interfaccia grafica. Anche la ricerca di una città non disponibile tra quelle presenti nell'API meteo di riferimento genererà un errore che verrà comunicato graficamente all'utente.

Qualora si testasse l'applicazione con l'emulatore si potrà comunque accedere all'interfaccia dei controlli ed interagire con essi, anche senza connessione Bluetooth.

Tecnologie Utilizzate

Implementazione del Bluetooth

Per la comunicazione tra il robot e il dispositivo mobile si è scelto di utilizzare il protocollo Bluetooth. La connettività BT è stata implementata seguendo una serie di passaggi, dalla verifica dei permessi necessari al funzionamento, all'uso delle socket lato client e server per la comunicazione tra il dispositivo e il modulo HC-05 risiedente nel robot.

Attivazione e configurazione del Bluetooth

Dopo aver indicato nell'Android Manifest i permessi richiesti dall'applicazione, per inizializzare le risorse necessarie al funzionamento del Bluetooth, nella MainActivity del programma sarà un'istanza della classe di default `BluetoothAdapter()` a permettere tutte le interazioni con i componenti che gestiscono il Bluetooth.

Richiesta di attivare i permessi e connessione con i dispositivi associati

Qualora l'utente non avesse abilitato i permessi, nel software è prevista la richiesta automatica attraverso un pop up di avvertimento all'utente:

```
//Checking if the bluetooth permission is granted or not
if (bluetoothAdapter?.isEnabled == false) {
    if (ActivityCompat.checkSelfPermission(
        context: this,
        Manifest.permission.BLUETOOTH_CONNECT
    ) != PackageManager.PERMISSION_GRANTED
    ) {
        // If not granted, permission is requested to the user
        if (Build.VERSION.SDK_INT > 31) {
            ActivityCompat.requestPermissions(
                activity: this,
                arrayOf(android.Manifest.permission.BLUETOOTH_CONNECT),
                requestCode: 100
            )
            return
        }
    }
}
```

Verifica e richiesta dei permessi necessari

Una volta verificati i permessi, si procede con la rilevazione dei dispositivi associati sfruttando il metodo `bondedDevices()` dell'istanza della classe `bluetoothAdapter()`, già citata in precedenza. Tra tutti i dispositivi BT associati rilevati dal sistema, verrà individuato solamente quello di interesse (identificato dal nome HC-05) e ne viene estratto il suo indirizzo MAC, necessario per la successiva fase di connessione vera e propria.

```
//Permission granted --> Getting all the paired devices
val pairedDevices: Set<BluetoothDevice>? = bluetoothAdapter?.bondedDevices
pairedDevices?.forEach { device ->
    val deviceName = device.name
    val deviceHardwareAddress = device.address // MAC address
    Log.d( tag: "cnt", msg: "Device Name --> $deviceName, Device MAC --> $deviceHardwareAddress")
    if (deviceName == "HC-05") {
        deviceHMAC = device //Address of interest
        Log.d( tag: "cnt", msg: "HC-05 MAC: --> ${deviceHMAC!!.address}")
    }
}
```

Ottenimento di tutti i dispositivi associati e identificazione del modulo HC-05

La connessione con il dispositivo BT avviene invece in un thread separato, per evitare la congestione della MainActivity. Si stabilisce inizialmente il canale di comunicazione con le informazioni acquisite in precedenza, andando a definire un'istanza della classe `BluetoothSocket` che identifica la socket, poi si procede alla connessione con il metodo `connect()` e all'identificazione dello stream di output con il metodo `outputStream()`, sempre della classe `BluetoothSocket`. Queste operazioni vengono eseguite solamente se il dispositivo BT di interesse è stato rilevato tra tutti quelli associati, altrimenti viene riportato all'utente un messaggio di errore per impossibilità di identificare il modulo HC-05.

Scrittura nello stream di output

Una volta avviato il canale di comunicazione con il modulo Bluetooth HC-05, ogni qualvolta l'utente interagisce con i widget adibiti al controllo del robot, si va a richiamare la funzione `writeBytes()` della `mainActivity()`, il cui compito è quello di scrivere i bytes che riceve come parametro nello stream di comunicazione con il modulo BT.

```

// Call this function to send data to the remote device.
@ Davide Reverberi
fun writeBytes(bytes: ByteArray) {
    try {
        if (OutputStream == null){
            Log.d( tag: "error", msg: "OutputStream is null --> Can't communicate")
        }
        OutputStream?.write(bytes)    //Writing the Bytes in the stream
        //Thread.sleep(50)
    } catch (e: IOException) {
        Log.e( tag: "error", msg: "Error occurred when sending data", e)
    }
}
}

```

Dettaglio della funzione di scrittura dei bytes sullo stream

Database MongoDB

MongoDB è un database NoSQL che in E-Motion è stato utilizzato per gestire e salvare le informazioni relative agli account degli utenti. Questa scelta ha permesso di avere un sistema di autenticazione e gestione utenti robusto e scalabile. Ciascun account utente è salvato in formato JSON con le informazioni di nome, email e password e la comunicazione con il database è mediata dal Web Service associato all'applicazione, attraverso chiamate http dall'applicazione Android (client) al server stesso (Web Service), che a sua volta interroga il database.

OpenWeather API

L'applicazione Android sviluppata include una funzionalità per ottenere informazioni metereologiche utilizzando l'OpenWeather API. Questa API è stata utilizzata come fonte principale per ottenere dati metereologici precisi e in tempo reale, che vengono poi processati e inviati al client tramite il Web Service sviluppato appositamente. Quando un utente inserisce un nome di una città valido, alla pressione del pulsante Search nell'interfaccia per il controllo del meteo, viene generata una richiesta http di tipo GET al WebService. Quando quest'ultimo riceve la richiesta, viene utilizzata una chiave API specifica per autenticarsi e inviare una richiesta delle informazioni metereologiche all'OpenWeather API, che verranno poi elaborate e restituite al client.


Architettura Software

L'architettura complessiva dell'applicazione si suddivide in due componenti in grado di comunicare tra di loro: il client (app Android) e il server (web service).

Struttura dell'applicazione

Gestione della Main Activity

L'architettura software complessiva dell'app è sviluppata per fare in modo che tutte le operazioni vengano avviate e terminate da un'unica MainActivity. All'interno di essa vengono gestite tutte le procedure necessarie per tutto il ciclo di vita delle connessioni Bluetooth, dalla richiesta dei permessi alla connessione e invio dei dati al modulo. Analogamente, tutte le operazioni associate al servizio web sono gestite all'interno della stessa Activity. In particolare, nel metodo `onStart()` della MainActivity viene effettuato un binding il cui scopo è quello di “legare” il funzionamento del servizio di accesso alle informazioni elaborate sul server al ciclo di vita dell'Activity.



```
▲ Davide Reverberi
override fun onStart() {
    super.onStart()
    //Bind to local WebService
    val boundIntent = Intent(packageContext: this, WebService::class.java)
    bindService(boundIntent, connection, Context.BIND_AUTO_CREATE)
}
```

Dettaglio del metodo `onStart()`, in cui si esegue il “binding” con il servizio web

La navigazione tra le diverse interfacce grafiche previste è resa possibile dal navigation graph associato al file `activity_main.xml`, grazie ad un `FragmentManagerView`. Di conseguenza, anche in questo caso, tutta la grafica sviluppata dipende direttamente dal ciclo di vita dell'activity principale.

Per evitare di congestionare l'activity, nella struttura del software si ricorre molto spesso all'utilizzo di thread (come nel caso della connessione Bluetooth) o di coroutine.

In particolare, le operazioni di login e registrazione dell'utente, dovendo attendere una risposta dal server, devono essere gestite in un'unità di esecuzione separata da quella che gestisce l'aggiornamento dell'interfaccia grafica, per evitare di congestionare la grafica e compromettere l'esperienza dell'utente. Le coroutine ricoprono quindi un ruolo fondamentale nell'architettura generale.

Gestione della grafica e dell'interazione con l'utente

Come già anticipato, ciascuna delle quattro interfacce navigabili dall'utente si basa su un grafico di flusso preciso che viene ospitato nel file xml associato alla Activity principale, con connessioni e grafiche personalizzate a seconda dell'obiettivo di ciascuna schermata. Ciascuna di queste interfacce presenta un file di stile xml in cui vengono definiti gli elementi grafici, e un file kotlin contenente la classe che si occupa di gestire le interazioni con essi. Quando l'utente interagisce con un widget grafico, vengono attivate funzioni specifiche che si occupano di trasferire le informazioni e reagire agli input dell'utente in tempo reale.

Comunicazione con il server

Il servizio che si occupa delle interazioni con il web service vero e proprio utilizza la libreria Moshi per la serializzazione dei file JSON, Retrofit per supportare il networking e l'interazione con l'API del server, e la configurazione di un client OkHttpClient con impostazioni personalizzate per i timeout di connessione, lettura e invio di dati.

```
// Configure OkHttpClient with custom timeout settings --> Registration process may require several seconds
val okHttpClient = OkHttpClient.Builder()
    .connectTimeout(timeout: 30, TimeUnit.SECONDS) // Connection timeout
    .readTimeout(timeout: 30, TimeUnit.SECONDS)    // Read timeout
    .writeTimeout(timeout: 30, TimeUnit.SECONDS)   // Write timeout
    .build()
```

Si è scelto di utilizzare 30 secondi prima che venga rilevato un timeout

Il WebService definito nel file WebService.kt mette a disposizione tre metodi che permettono di richiamare le funzioni per contattare il server. L'interfaccia WebServiceAPI, definita nel file ServiceUtil.kt, si occupa di tradurre queste chiamate in richieste http di tipo GET e POST.

Di seguito vengono riportate le 3 possibili chiamate all'API del web service, dove sono visibili anche le strutture dati che contengono le informazioni da inserire nel body delle richieste POST.

```
interface WebServiceAPI {  
  
    @GET("weather")  
    suspend fun getWeather(@Query("city") city: String): Response<WeatherReport>  
  
    @POST("register")  
    suspend fun registerUser(@Body userProfile: UserProfile): Response<RegisterResponse>  
  
    @POST("login")  
    suspend fun loginUser(@Body userCredentials: UserCredentials): Response<LoginResponse>  
}
```

Design Patterns

Per lo sviluppo dell'applicazione si è ricorso ad alcuni design patterns utili. La maggior parte di essi sono stati utilizzati implicitamente, non tanto per scelta progettuale dell'applicazione, quanto più per come la logica delle API e delle classi di Android è stata realizzata.

Tra questi, il **Singleton** è forse il modello più ricorrente tra tutti a causa della logica con cui sono stati realizzati i vari metodi listener dei widget (pulsanti e seekBar). Ogni volta che l'utente interagisce con questi componenti, viene restituita un'unica istanza del listener, evitando così la creazione di più istanze multiple dello stesso oggetto. Anche la connessione con il web service utilizza un Singleton per garantire che vi sia una sola istanza della connessione, migliorando l'efficienza e la coerenza delle operazioni di rete.

Un altro design pattern utilizzato nello sviluppo del software è **Observer**. Questo modello di tipo comportamentale viene implicitamente utilizzato nell'interazione tra la connessione tra il servizio (ServiceConnection) e la MainActivity. Grazie al binding e a questo design pattern, quando il servizio si connette o disconnette, la MainActivity viene notificata tramite i metodi onServiceConnected e onServiceDisconnected (nel file MainActivity.kt).

```

/** Defines callbacks for service binding, passed to bindService(). */
private val connection = object : ServiceConnection {
    override fun onServiceConnected (className: ComponentName, service: IBinder) {
        // We've bound to service, cast the IBinder and get service instance.
        val binder = service as MyBinder
        customService = binder.getService()
        mBound = true
    }
    override fun onServiceDisconnected (arg0: ComponentName) {
        mBound = false
    }
}
}

```

Esempio di utilizzo del Singleton e dell'Observer nella connessione con il servizio

Nel codice è ricorrente anche l'utilizzo implicito del pattern **Adapter** per gestire la compatibilità tra le diverse librerie di Retrofit, Moshi e OkHttpClient. Questo modello permette la cooperazione tra queste librerie senza che venga modificato in alcun modo il loro codice interno.

Struttura del server

Flask e componenti

Il server è implementato utilizzando il framework Flask, un framework leggero adatto per lo sviluppo di applicazioni web in Python. Il server è stato inoltre sviluppato per essere facilmente estendibile, con l'aggiunta di eventuali nuove API web. Nel server sono stati integrati i componenti necessari per la connessione con il database MongoDB e per l'interazione con OpenWeatherAPI. Le chiavi utilizzate per accedere ai servizi sopra citati sono riportate nel file .env ma qualora si desiderasse creare un proprio server con database personale è necessario sostituirle con le proprie chiavi personali.

Endpoints API

Il server mette a disposizione tre endpoints API per permettere le operazioni di accesso, registrazione e consultazione delle informazioni metereologiche:

1. **/weather:** Endpoint che accetta una richiesta di tipo GET con il parametro "city" che specifica la città che dovrà essere considerata per elaborare le informazioni metereologiche. Vengono restituiti in risposta le informazioni sul meteo per la città richiesta.

2. /register: Endpoint di tipo POST che gestisce la registrazione degli utenti. Vengono elaborati i dati JSON e vengono salvate le credenziali sul database MongoDB se l'username non viene rilevato come già esistente.

3. /login: Endpoint di tipo POST che gestisce le operazioni di accesso all'applicazione da parte degli utenti. Anche in questo caso vengono elaborate le informazioni in formato JSON relative a username e password che si ricevono dal client. Il server si occupa di verificare eventuali corrispondenze presenti nel database e restituisce un messaggio di errore o conferma di autenticazione.

Un'ulteriore endpoint dedicato al test sull'attività del database è messo a disposizione:

4. /test_db_connection: Endpoint che si occupa di testare la connessione al database MongoDB. Restituisce un messaggio di successo se la connessione è stabilita con successo oppure in caso contrario un messaggio di errore.

Link Utili

1. Documentazione Android sul Bluetooth:

<https://developer.android.com/develop/connectivity/bluetooth?hl=it>

2. MongoDB Atlas:

<https://www.mongodb.com/it-it/atlas>

3. OpenWeather API:

<https://openweathermap.org/>

4. Flask:

<https://flask.palletsprojects.com/>

5. Repository GitHub del progetto:

<https://github.com/DaddaRev/E-Motion-App/tree/Web-Service>