Documentazione Tecnica

Applicazione Utente

Progetto Smart Office - IoT and 3D Intelligent Systems

Puzone Felicia

Arfaoui Bilel

Macellaro Vincenzo

Overview

L'applicazione utente è parte integrante del sistema e permette ai clienti del servizio di registrarsi alla piattaforma, di effettuare prenotazioni, di visualizzare i dati dei sensori e del meteo, di modificare i valori degli attuatori.

Framework

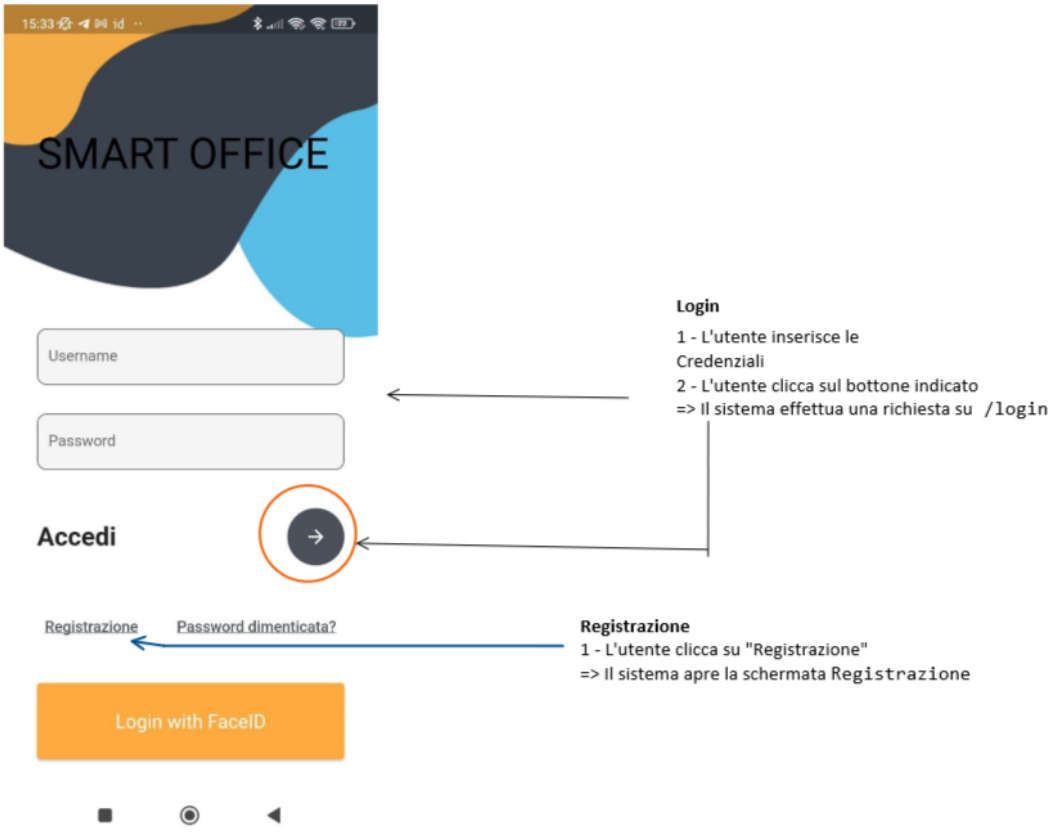
Per lo sviluppo dell'applicativo si è deciso di optare per il framework Flutter. **Flutter** è un framework open source sviluppato da Google allo scopo di costruire applicazioni cross-platform compilate nativamente, a partire da un unico codice. La motivazione dietro questa scelta risiede nella possibilità di poter compilare in un'unica soluzione app native **Android, iOS, Windows e Web based**. Le performance sono simili a quelle delle app native e l'utilizzo del **Material Design** dona all'esperienza utente un feeling moderno e pulito.

Flutter utilizza il linguaggio di programmazione **Dart**, un linguaggio object oriented C-like.

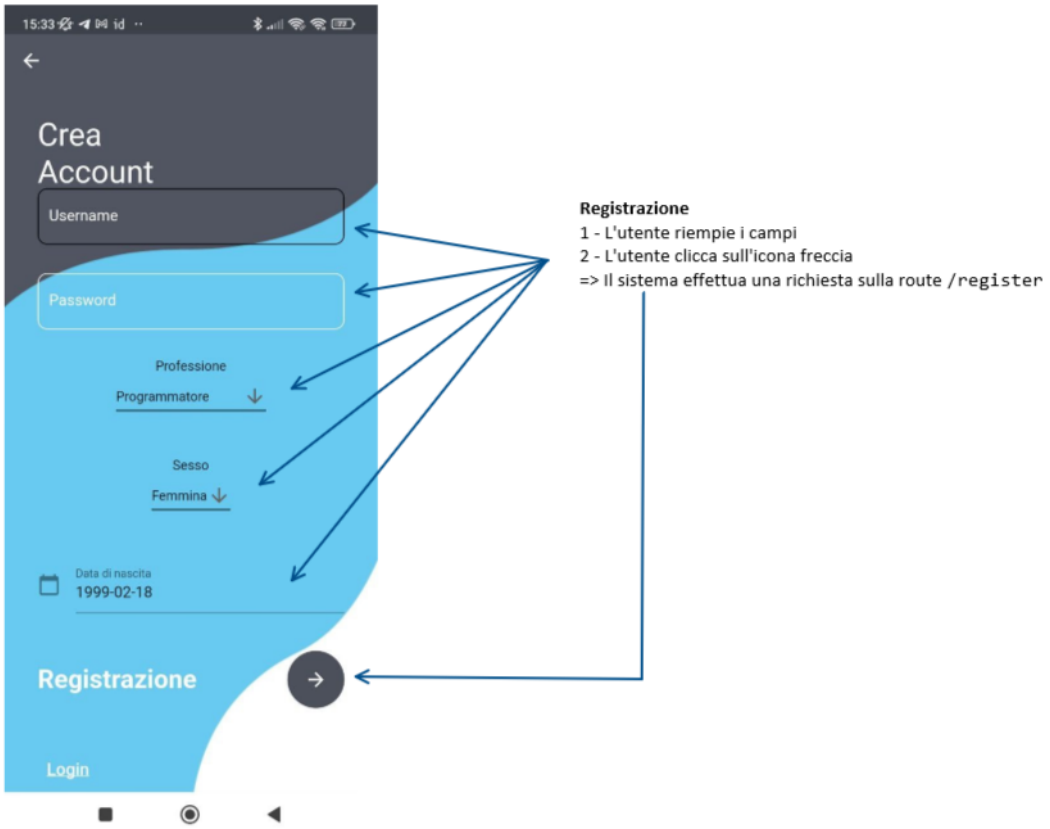
UX/UI

Visualizzazione del flusso di utilizzo dell'app da parte di un utente, e delle relative routine svolte dal sistema di conseguenza.

Schermata Login



Schermata Registrazione



Schermata Mappa

La schermata si apre con la visualizzazione immediata della porzione di mappa relativa alla posizione corrente dell'utente, che è visualizzata in maniera standard.

La visualizzazione della posizione corrente dell'utente fa uso della libreria `Geolocator`.

Affinchè fosse correttamente configurata, è necessario, in AndroidManifest.xml ed inserire il seguente permesso.

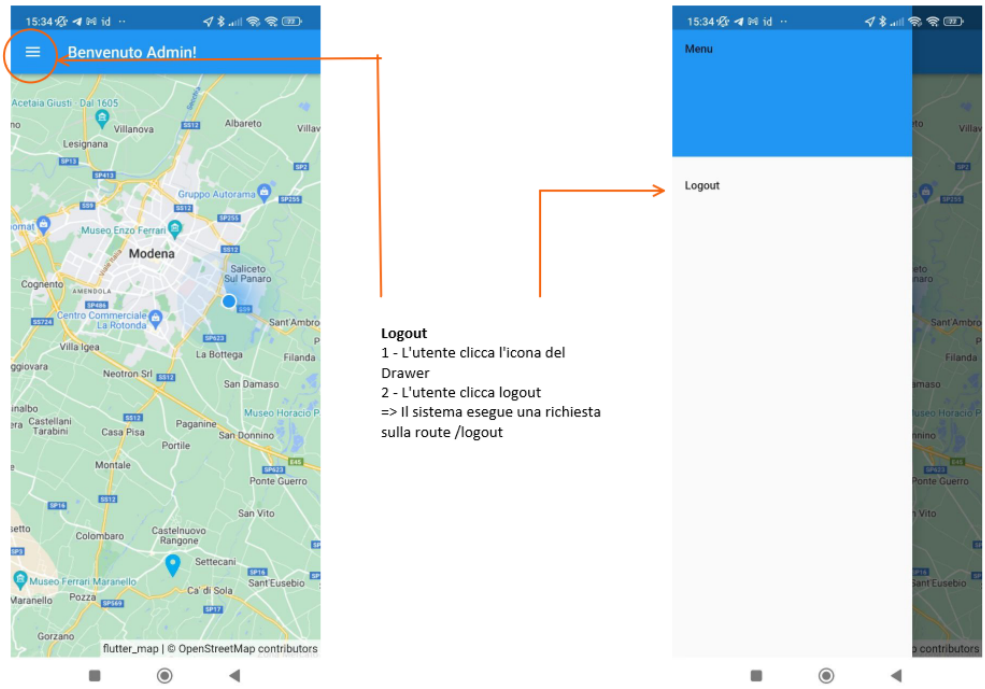
```
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION" />
```

E' inoltre stato fatto uso delle librerie Dart:

```
flutter_map_location_marker

flutter_map
```

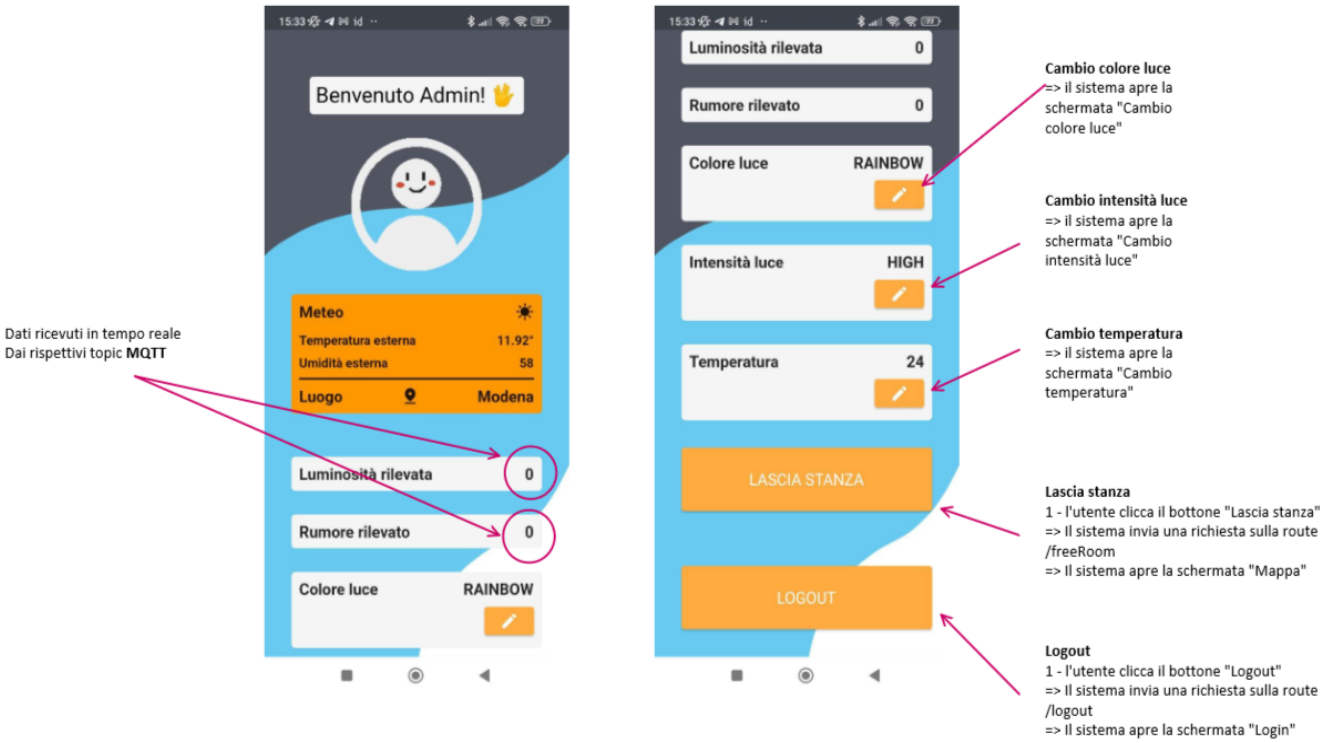
Per visualizzare correttamente i marker sulla mappa attraverso latitudine e longitudine, è risultato necessario il pacchetto `latlong2/latlong.dart`



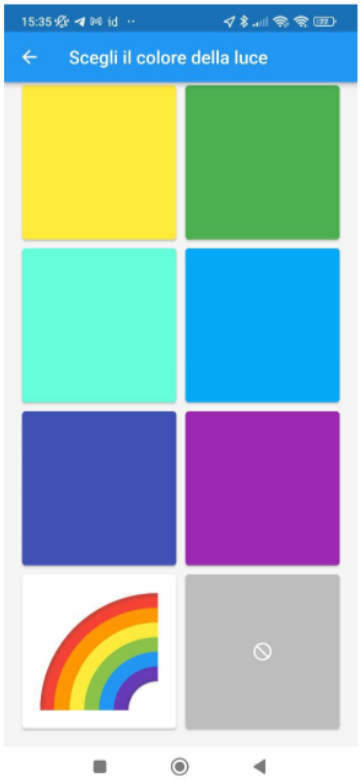
Schermata Home

In questa schermata l'utente visualizza

- Il proprio nome e la propria foto profilo
- I dati meteorologici e geografici aggiornati.
- I dati in tempo reale provenienti dai sensori della stanza occupata
- Sezioni dedicate al cambio dei parametri degli attuatori della stanza

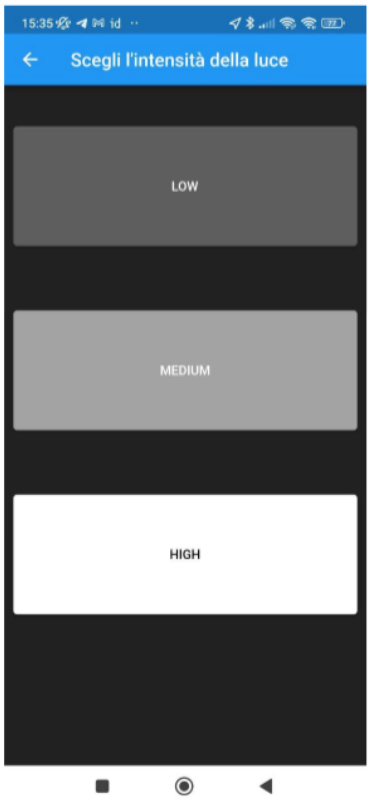


Schermata Cambio colore luce



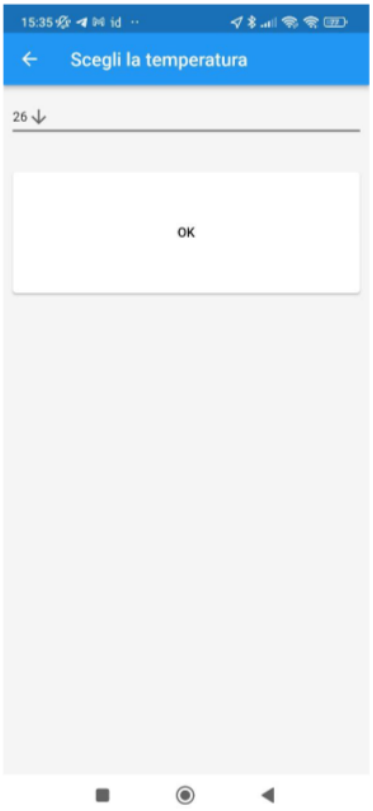
Cambio colore
1 - L'utente seleziona il colore desiderato (o NO COLOR)
=> Il sistema effettua una richiesta sulla route `/update` secondo le scelte effettuate

Schermata Cambio intensità luce



Cambio intensità
1 - L'utente seleziona l'intensità desiderata
=> Il sistema effettua una richiesta sulla route `/update` secondo le scelte effettuate

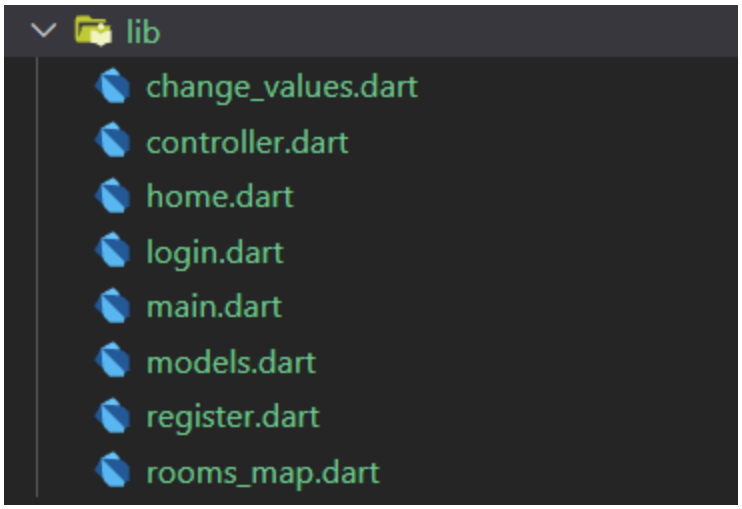
Schermata Cambio temperatura



Cambio temperatura
1 - L'utente seleziona dalla combobox la temperatura desiderata
=> Il sistema effettua una richiesta sulla route `/update` secondo le scelte effettuate

Specifiche tecniche

Il codice Dart è stato suddiviso seguendo il paradigma **Model-View-Controller**.



Il file `models.dart` contiene le classi necessarie all'applicazione.

Il file `controller.dart` contiene le funzioni necessarie all'invio delle richieste HTTP al server.

I restanti moduli contengono i widget e le strutture necessarie alla visualizzazione di pagine e componenti.

Models

Credentials

```
class Credentials {
  late String username;
  late String password;
  late String authToken;

  Credentials({required this.username, required this.password});
}
```

Classe contenente le informazioni utili di autenticazione.

UserSession

```
class UserSession {
  late int? id_edificio;
  late int? id_room;
  late String? city_name;
  late String username;

  UserSession({this.id_edificio, this.id_room, required this.username});

  factory UserSession.fromJson(dynamic json) {
    return UserSession(
      id_edificio: json['id_edificio'],
      id_room: json['id_room'],
      username: json['username'],
```

```
    );
  }
}
```

Classe contenente i dati relativi alla sessione di occupazione di una stanza da parte dell'utente. La classe contiene anche il metodo statico **fromJson**, utile a creare un'istanza di UserSession direttamente da risposta JSON.

Profession

```
class Profession {
  late int id;
  late String name;

  Profession({required this.id, required this.name});

  factory Profession.fromJson(dynamic json) {
    return Profession(
      id: json['id_profession'],
      name: json['name'],
    );
  }
}
```

Classe che specifica un oggetto di tipo professione. Una lista di Profession viene fetchata da DB in fase di registrazione utente, e visualizzata in una combobox. Anche in questo caso è presente il metodo fromJson.

Digital Twin

```
class DigitalTwin {
  var room_color;
  var room_brightness;
```

```
  int room_temperature;


  DigitalTwin(
      {required this.room_color,
      required this.room_brightness,
      required this.room_temperature});


  factory DigitalTwin.fromJson(dynamic json) {
    json = json['digitalTwin'];
    return DigitalTwin(
        room_color: json?['room_color'],
        room_brightness: json?['room_brightness'],
        room_temperature: json?['room_temperature']);
  }
}
```

Classe contenente gli attributi del digital twin e relativo json parser.

WeatherInfo

```
class WeatherInfo {
  late String city_name;
  late String ext_temp;
  late String ext_humidity;


  WeatherInfo(
      {required this.city_name,
      required this.ext_temp,
      required this.ext_humidity});
```

```
  void set cityName(String cityName) {
    this.city_name = cityName;
  }

  factory WeatherInfo.fromJson(dynamic json) {
    json = json['weather'];
    return WeatherInfo(
      city_name: json['city_name'],
      ext_temp: json['temperature'],
      ext_humidity: json['humidity'],
    );
  }
}
```

Classe contenente informazioni meteo e relativo parser json.

Global Values

```
class GlobalValues {
  static UserSession? userSession;
  static late DigitalTwin digitalTwin =
      DigitalTwin(room_color: '', room_brightness: '', room_temperature: 0);
  static late List<dynamic> listBuildings = [];
  static late WeatherInfo weatherInfo =
      WeatherInfo(ext_temp: '', ext_humidity: '', city_name: '');
  static late Credentials credentials = Credentials(username: '', password: '');
}
```

Classe globale che assimila tutte le informazioni sulla sessione corrente.

Controller

Sono qui mostrate tutte le funzioni partecipanti alla logica dell'applicazione.

HTTP

L'applicazione Android ha bisogno della configurazione di alcuni permessi per accedere ad internet. In `AndroidManifest.xml`, sono state aggiunte le righe:

```
<uses-permission android:name="android.permission.INTERNET" />

<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

I metodi che inviano richieste sono denominati con la keyword **async**. I metodi async ricevono in risposta un oggetto di tipo **Future<T>**.

```
//@login

Future<String> fetchUserSession(user, pwd) async
```

Esegue una richiesta sulla route /login con body username e password

```
//@selectRoom

Future<String> sendOccupationRequest(id_building) async
```

Esegue una richiesta sulla route /selectRoom con parametro id_building dell'edificio richiesto.

```
//@logout

Future<String> logout() async
```

Esegue una richiesta sulla route /logout

```
//@freeRoom

Future<String> freeRoom() async
```

Esegue una richiesta sulla route /freeRoom per liberare la stanza.

```
//@update

Future<String?> changeActuatorRequest(UpdateReq request) async
```

Esegue una richiesta sulla route /update con parametro un oggetto di tipo UpdateReq formato come segue

```
class UpdateReq {
  final String color_val;
  final String brightness_val;
  final int temp_val;

  UpdateReq(this.color_val, this.brightness_val, this.temp_val);

  UpdateReq.fromJson(Map<String, dynamic> json)
      : color_val = json['color_val'],
        brightness_val = json['brightness_val'],
        temp_val = json['temp_val'];

  Map<String, dynamic> toJson() => {
        'color_val': color_val,
        'brightness_val': brightness_val,
        'temp_val': temp_val
      };
}
```

Lo scopo è quello di richiedere al server di modificare lo stato di qualche attuatore.

```
//@register
Future<String> registerUser(user, pwd, sex, profession, birthDate) async
```

Esegue una richiesta sulla route /register. Utilizza i parametri username, password, sex, profession, birthDate acquisiti dal form registrazione. Lo scopo è quello di registrare un nuovo utente.

```
//@professions
```

```
Future<List<Profession>> fetchJobs() async
```

Esegue una richiesta sulla route /professions. Serve a fetchare la lista di professioni da mostrare in combobox all'avvio del form di registrazione utente.

MQTT

L'applicazione presenta delle routine per comportarsi come un **Client MQTT**. Ciò è ottenuto utilizzando il pacchetto Dart https://pub.dev/packages/mqtt_client

L'utilizzo di MQTT è impiegato per **visualizzare in tempo reale i dati inviati dai sensori** della stanza prenotata dall'utente.

```
Future<int> mqttConnect() async
```

Funzione che controlla la connessione al broker MQTT.

In questo caso le impostazioni sono state configurate come:

```
client.keepAlivePeriod = 20;
client.connectTimeoutPeriod = 2000; // milliseconds


final connMess = MqttConnectMessage()
    .withClientIdentifier('app_unique_id')
    .withWillTopic('appWillTopic')
    .withWillMessage('App disconnected')
    .startClean()
    .withWillQos(MqttQos.atLeastOnce);
```

Il subscribing ai topic è eseguito come da

```
String topicLightSensor =
    'smartoffice/building_$id_edificio/room_$id_room/sensors/light_sensor';
String topicNoiseSensor =
    'smartoffice/building_$id_edificio/room_$id_room/sensors/noise_sensor';
client.subscribe(topicLightSensor, MqttQos.atMostOnce);
client.subscribe(topicNoiseSensor, MqttQos.atMostOnce);
```

Il livello di **QoS** per la ricezione dei sensori è di tipo a**tMostOnce (1)**. Ciò poiché non è di vitale importanza perdere la ricezione di qualche dato relativo ai sensori nella visualizzazione in App.

Build

Per la build è stato generato un file **apk**.

Per la configurazione del manifest e della build (build.gradle) sono state scelte le seguenti impostazioni:

Manifest

```
package="it.unimore.iot.smartoffice"

        android:label="Smart Office - IoT Project"
```

Gradle

```
android {
    compileSdkVersion 33
    ndkVersion flutter.ndkVersion

    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    kotlinOptions {
        jvmTarget = '1.8'
    }
    sourceSets {
        main.java.srcDirs += 'src/main/kotlin'
    }
    defaultConfig {
```

```
        applicationId "it.unimore.iot.smartoffice"
        minSdkVersion 28
        targetSdkVersion flutter.targetSdkVersion
        versionCode flutterVersionCode.toInteger()
        versionName flutterVersionName
    }
```

Pubspec.yaml

In questo file è presente tutta la gestione delle dipendenze del progetto.

```
name: smart_office
description: Smart Office.
publish_to: 'none'
version: 1.0.0+1
environment:
  sdk: '>=2.18.2 <3.0.0'

dependencies:
  flutter:
    sdk: flutter

  cupertino_icons: ^1.0.2
  http: ^0.13.5
  flutter_emoji: ^2.4.0
  mqtt_client: ^9.7.4
  flutter_map: ^3.1.0
  latlong2: ^0.8.1
  flutter_map_location_marker: ^5.1.0+1
```

```
dev_dependencies:
  flutter_test:
    sdk: flutter


  flutter_lints: ^2.0.0


flutter:
  assets:
    - assets/
```