



# Flutter

## GREENWALK

Introdução à Computação Móvel

3º Ano LEI



universidade  
de aveiro

### ABSTRACT

Implementação da aplicação Greenwalk de tracking de atividade física, qualidade do ar e partilha de percursos pedonais. Aplicação em flutter no âmbito da cadeira de ICM

Pedro Marques

92926

# Motivation

Com este trabalho tinha como objetivo implementar um sistema que permitisse gravar percursos realizados a pé. Igualmente pretendíamos conseguir gravar dados sobre a qualidade do ar no percurso bem como partilhar o mesmo através de fotografias. Esta aplicação teve como ideia base a aplicação já existente designada Ciclogreen. Esta tem como objetivo fomentar a mobilidade sustentável e diminuir as emissões de gases poluentes. A aplicação ainda fornece recompensas de acordo com a quantidade de quilómetros viajados. Como ideia inicial, foram definidas algumas funcionalidades chave que deveriam ser implementadas na aplicação. Assim sendo, esperavam-se as seguintes funcionalidades:

- **GPS:** Utilizado para obter a localização do utilizador.
- **Firestore Real Time Database:** Utilizado para guardar percursos e dados de utilizadores.
- **Firestore Storage:** Para guardar imagens.
- **GoogleMap:** Para representar o percurso de uma atividade
- **MoorDatabase:** Uma cache dos percursos realizados por um utilizador
- **Retrofit2:** Utilizado para fazer uma chamada à API para ir buscar a qualidade do ar
- **Step Counter:** Para contar passos dados.

Este projeto foi previamente implementado para android em java e foi realizado no âmbito da cadeira de Introdução à Computação Móvel.

## The Solution

A solução possui todas as funcionalidades que eram pretendidas bem com a adição de um sistema de notificações que a cada 45 minutos relembra a qualidade do ar. Para além disso, a aplicação usanda ainda a câmara para tirar fotografias.

A aplicação conta com um sistema de login e registo que implicam o fornecimento do género sexual. Este valor será mais à frente utilizado para calcular a distância percorrida por um utilizador.

O ficheiro **home.dart** inicializa a aplicação. Assim sendo é responsável por reencaimhar o utilizador para **login/registo** ou para a página principal da mesma, a página **Feed**. O UI principal é composto por 3 páginas: **Feed**, **Activity** e **Profile**. Existem mais duas páginas: **Authentication** e **ActivityDetails**. A classe **MainViewModel** possui dados que são fundamentais ao funcionamento da aplicação e que são iguais em toda a aplicação. Existem 3 entidades principais **User1**, **Activity** e **AirData**.

Em análise, todas as funcionalidades foram implementadas embora com algumas alterações. Para além disso foi implementado um sistema de notificações que alerta a cada 15 minutos sobre a qualidade do ar e quando um utilizador faz upload de uma nova atividade.

- **GPS:** Utilizado para obter a localização do utilizador.
- **Firestore Real Time Database:** Utilizado para guardar percursos realizados, dados de utilizadores e tokens.
- **Firestore Storage:** Para guardar imagens uploaded por utilizadores.
- **GoogleMaps e PolyLines:** Para representar o percurso de uma atividade
- **HiveDatabase:** Database local de percursos realizados, serve como cache quando há falta de internet.
- **HttpRequest:** Utilizado para fazer uma chamada à API weatherbit.io para ir buscar a qualidade do ar. A conversão de json para objeto é realizada na classe AirData
- **Pedometer:** Biblioteca que utiliza o sensor Acelerómetro para contar passos.
- **ImagePicker:** É possível escolher imagens da galeria ou utilizar a câmara.

- **Firestore Cloud Messaging:** Sistema de notificações quando ocorre upload de uma nova atividade
- **Flutter Local Notifications:** A cada 15 minutos avisa sobre a qualidade do ar.

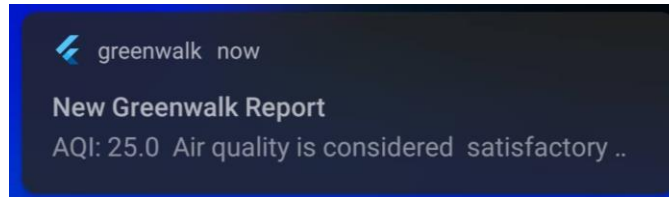


Figura 1 - Sistema de Notificações a Cada 15 Min Sobre a Qualidade do Ar (Feed.dart)

## BlocPattern

O UI principal (Feed, Activity, e Profile) utilizam o bloc pattern para a atualização constante do seu estado. Existem assim 4 classes: DataBloc, LocAQIBloc, TimerBloc e ActivityBloc. Para partilhar multiplos dados através de uma única stream foram criadas 3 classes:

### LocAQIBlocData

```
class LocAQIBlocData {
  double aqi;
  String tip;
  Color color;
  Position currentPosition;
  LocAQIBlocData(this.aqi, this.tip, this.color, this.currentPosition);
}
```

*AQI*- Índice de qualidade do ar

*Tip* – String com sugestão sobre procedimentos a ter e gravidade da qualidade do ar.

*Color* – cores diferentes para patamares distintos do índice de qualidade do ar.

*currentPosition* – posição do utilizador.

### TrackData

```
class TrackData {
  double totalDistance = 0;
  int steps = 0;
  int avgAQI = 0;
  double avgSpeed = 0;
  Duration latestTime = new Duration();
  List<LatLng> locations = List();
  TrackData();
}
```

*totalDistance* – distância total percorrida no percurso, calculada de acordo com o género sexual e os passos dados pelo utilizador.

*Steps* – passos dados pelo utilizador.

*avgAQI* – índice de qualidade do ar medio no percurso

*avgSpeed* – velocidade média do utilizador

*latestTime* – tempo no cronómetro.

*Locations* – lista de pontos percorridos no percurso

## ActivityDetail

```
class ActivityDetail{
  LocAQIBlocData locAQIBlocData;
  TrackData trackData;
  ActivityDetail(this.locAQIBlocData, this.trackData);
}
```

locAQIBlocData – localização e qualidade do ar no momento

trackData – dados sobre o percurso em execução até ao momento

## DataBloc

Responsável pela atualização de dados. Este Bloc vai buscar os dados à base de dados local. De seguida, recorre à Real Time Database do Firebase para verificar que estão todos presentes. Se não estiverem, recolhe-os e guarda-os localmente de modo a que, da próxima vez, não seja necessária a conexão à base de dados remota.

```
StreamController< List<Activity>> allActivities = StreamController< List<Activity>>.broadcast();
StreamController< List<Activity>> allPublicActivities = StreamController< List<Activity>>.broadcast();
```

## LocAQIBloc

Obtém duas das informações fundamentais à aplicação. Este bloc fornece uma stream contínua com a localização e a qualidade do ar no momento no local, através da utilização de objetos do tipo LocAQIBlocData. Assim sendo, esta informação é utilizada no sistema de notificações, na página Feed e na Activity.

```
StreamController<LocAQIBlocData> aqiStreamController = StreamController<LocAQIBlocData>.broadcast();
Stream get getAQI => aqiStreamController.stream;
```

A qualidade do ar é obtida através de um http request . Este request à API weatherbit.io retorna vários dados em formato json. Estes dados são convertidos num objeto do tipo AirData. No final, só nos interessa o índice de qualidade do ar.

```
var response =
await http.get('https://api.weatherbit.io/v2.0/current/airquality?'
  'lat=$latitude&lon=$longitude&key=d367b455453f495d88622c24e902bc4e');
```

Este request envia como parâmetros a latitude e longitude do utilizador. Igualmente, o parâmetro key corresponde a uma chave de utilização necessária para realizar a chamada à API. A subscrição atual só permite 500 chamadas diárias à API

## TimerBloc

Responsável por atualizar o tempo que decorre desde o início de uma atividade. Assim sendo, a stream que este bloco fornece é de apenas um valor int. Poderia ter utilizado um ValueNotifier em vez do BlocPattern mas devido à forma como decidi organizar o ActivityBloc, esta opção tornou-se mais vantajosa.

```
StreamController<Duration> timerCount = StreamController<Duration>.broadcast();
Stream get getTime => timerCount.stream;
```

## ActivityBloc

Utilizado para gravar um certo percurso. Quando o utilizador carrega no ícone de play na página Activity, este bloc recebe as streams provenientes do TimerBloc cujo timer é devidamente inicializado e do LocAQIBloc. A contagem de passos é dada pela biblioteca Pedometer que utiliza o acelerómetro. A cada passo dado é criado um novo objeto do tipo TrackData com novos dados. Assim, produz uma stream contínua que fornece objeto do tipo ActivityDetail. Por fim, a página Activity é constantemente atualizada de acordo com os dados recebidos pelo ActivityBloc.

***A distância percorrida é calculada por cálculo matemático utilizando o número de passos dados. A velocidade é determinada utilizando os valores de distância e tempo decorrido***

```
StreamController<ActivityDetail> activityStreamController = StreamController<ActivityDetail>.broadcast();  
Stream get getActivityDetail => activityStreamController.stream;
```

## ChangeNotifier

### MainViewModel

Utilizado para guardar o utilizador que está a utilizar a aplicação no momento. Para além disso possui ainda alguns métodos importantes.

**getUser(String email)** – Faz uma ligação à Firebase e retorna o user de acordo com o email providenciado.

```
Future<void> getUser(String email) async {...}  
Future<void> updateUser() async {...}  
Future uploadImageToFirebase(File _imageFile) async {...}  
List<String> tokens = new List<String>();  
Future<List> getTokens() async {...}  
FirebaseMessaging _firebaseMessaging = FirebaseMessaging();  
void saveToken() async{...}
```

**updateUser()** – Atualiza o user atual na base de dados firebase.

**uploadImageToFirebase(File \_imageFile)** – Método que permite fazer o upload de uma imagem para o Firebase e guardar a sua referência com o utilizador como foto de perfil.

**getTokens()** – Retorna todos os tokens de Firebase Cloud Messaging de utilizadores que são guardados na base de dados Firebase.

**saveToken()** – Guarda o token de Firebase Cloud Messaging do utilizador na base de dados Firebase.

## Hive Database

Hive é a biblioteca responsável pela base de dados local. Como já tinha referido antes, é unicamente utilizada como cache de atividades, pelo que existem duas entidades.

Os ficheiros **LatLng.g.dart** e **ActivityClass.g.dart** são automaticamente gerados pelo `floor_generator` e criam alguns métodos e adaptadores para podermos gravar os dados na caixa Hive.

```
@HiveType()
class Activity {
  @HiveField(0)
  String date;
  @HiveField(1)
  int time = 0;
  @HiveField(2)
  int steps = 0;
  @HiveField(3)
  int AQI;
  @HiveField(4)
  double avgSpeed = 0;
  @HiveField(5)
  double distance = 0;
  @HiveField(6)
  String user_email;
  @HiveField(7)
  String id;
  @HiveField(8)
  List<String> images = new List<String>();
  @HiveField(9)
  List<LatLng> coordinates = List<LatLng>();
  @HiveField(10)
  bool isPrivate = true;
  Activity(this.date);
}
```

```
@HiveType()
class LatLng {
  @HiveField(0)
  double latitude;
  @HiveField(1)
  double longitude;
  LatLng(this.latitude, this.longitude);
}
```

## Firebase

Como já disse antes, usamos o firebase para guardar vários dados tais como atividades realizadas, dados de utilizadores, imagens e tokens de Firebase Cloud Messaging de utilizadores.

Sempre que o cliente acaba uma atividade e esta é gravada na base de dados, uma mensagem é enviada a todos os restantes utilizadores da aplicação. Esta mensagem informa sobre nova atividade de um dado cliente bem como a distância total percorrida pelo mesmo.

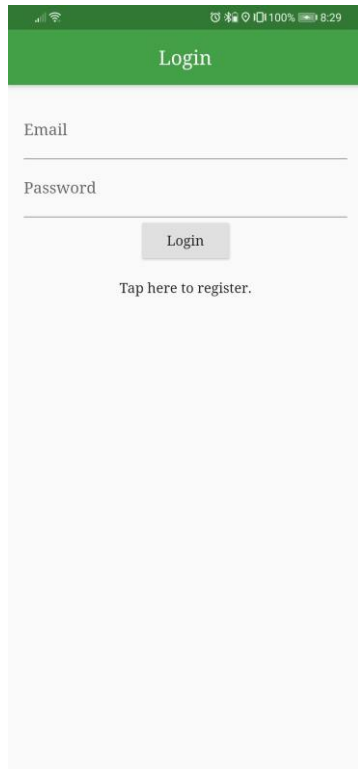
As funções **saveToken** e **getTokens** no `MainViewModel` são utilizadas para encontrar e gravar os destinatários destas notificações.

```
FirebaseMessaging _firebaseMessaging = FirebaseMessaging();

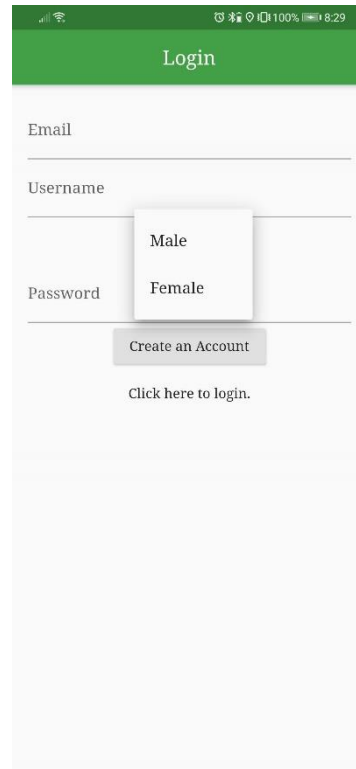
Future<Map<String, dynamic>> sendAndRetrieveMessage(token) async {
  await _firebaseMessaging.requestNotificationPermissions(
    const IOSNotificationSettings(
      sound: true, badge: true, alert: true, provisional: false),
  );
  debugPrint("-----");
  debugPrint(token);
  await http.post(
    'https://fcm.googleapis.com/fcm/send',
    headers: <String, String>{
      'Content-Type': 'application/json',
      'Authorization':
        'key=AAAA5057R3k:APA91bPvW8w0Tq0WHP_-pA2ISj60T6LwPHBCur7Wp2210P3uP6037H6chyQ5Awy7fGgLyOrx2eCl_HfgoDwa01-RmEx0w0KMD-v3w3cStPBXkgZ2DwBuT8u9c3934teXTNc10Ht',
    },
    body: jsonEncode(
      <String, dynamic>{
        'notification': {
          'body': 'Activity ' +
            curActivity.distance.toString() +
            "km in " +
            curActivity.time.toString() +
            " seconds",
          'title': 'New Activity From ' + currentUser.username
        },
        'priority': 'high',
        'to': token,
      },
    ),
  );
}
```

## Tutorial

Como já tinha referido antes, ao iniciar a aplicação somos confrontados com o login do utilizador. Caso não tenha criado uma conta, basta carregar em *tap here to register* para se registar. É fácil voltar à pagina de login ao clicar no botão 'Click here to login' nesta segunda página. Antes de criar uma conta, é necessário o preenchimento de todos os campos. Assim sendo, é necessário fornecer email, username, password e género sexual.



The image shows a mobile app login screen. At the top is a green header with the word "Login" in white. Below the header are two input fields: "Email" and "Password". Under the "Password" field is a grey button labeled "Login". At the bottom of the screen, there is a link that says "Tap here to register."



The image shows a mobile app registration screen. At the top is a green header with the word "Login" in white. Below the header are four input fields: "Email", "Username", "Password", and a gender selection dropdown menu with options "Male" and "Female". Under the "Password" field is a grey button labeled "Create an Account". At the bottom of the screen, there is a link that says "Click here to login."

Após clicar em 'Login' ou 'Create a new account', caso seja a primeira vez de utilização da aplicação, o utilizador é obrigado a dar permissões. Assim a aplicação tem que conseguir aceder à **Camera, Location e Activity Recognition**. Caso estas permissões não sejam autorizadas a aplicação não funcionará corretamente.

De seguida, o utilizador é confrontado com a página de feed. Esta página é uma das 3 mais importantes na aplicação. No canto superior direito temos um botão de logout que reinicia a aplicação. A qualquer momento, é possível navegar entre as páginas Feed, Activity e Profile através do menu de navegação.



## Feed

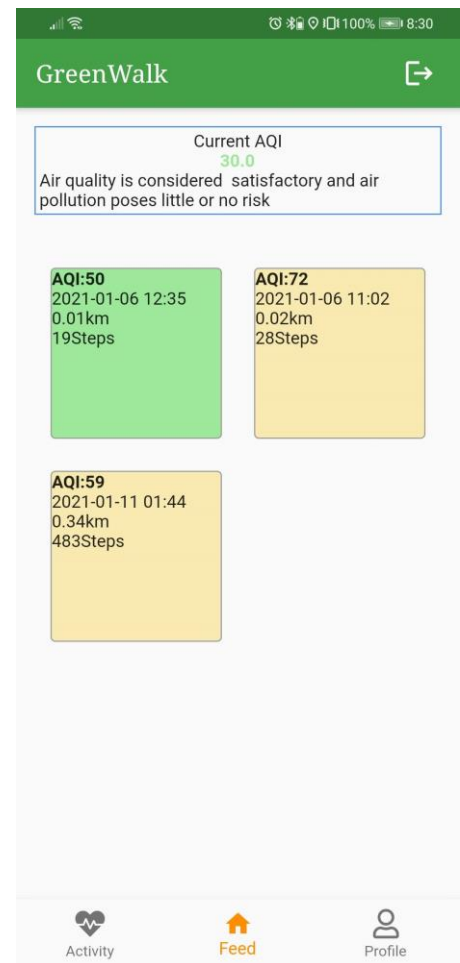
É nesta página que estão disponíveis todas as atividades públicas dos diferentes utilizadores. Estas atividades são representadas em cada quadrado.

Estes quadrados possuem cores diferentes consoante a média da qualidade do ar no percurso.

Em cada quadrado são apresentadas 4 informações sobre o percurso realizado. Podemos então observar o índice médio da qualidade do ar, a data de realização, a distância percorrida e o número de passos dados no percurso. Estes quadrados só representam percursos que foram tornados públicos pelo seu criador.

Igualmente, se carregarmos em qualquer um destes, somos encaminhados para a página ActivityDetails que contém informações mais detalhadas sobre a atividade.

Por cima das publicações, um pequeno retângulo apresenta informações sobre o índice de qualidade do ar no momento e uma dica de saúde.



## Profile

Nesta página podemos aceder aos dados sobre um utilizador. De cima para baixo, podemos observar uma foto de perfil, o username, email, género sexual e uma lista de atividades realizadas pelo utilizador. Este utilizador já tem uma foto de perfil selecionada no entanto, caso não tivesse, apareceria o texto 'Touch to add picture'. Ao clicar neste texto aparece um pop-up a perguntar se o utilizador quer tirar uma fotografia ou selecionar da galeria. Caso queira alterar a foto, basta carregar na fotografia atual e aparecerá o mesmo pop-up.

Tal como na página feed, é-nos apresentada uma lista de caixas referentes a diferentes percursos realizados pelo utilizador. Igualmente, são apresentadas as mesmas informações que os da página feed e, ao clicar nestes quadrados, somos encaminhados para a página ActivityDetails

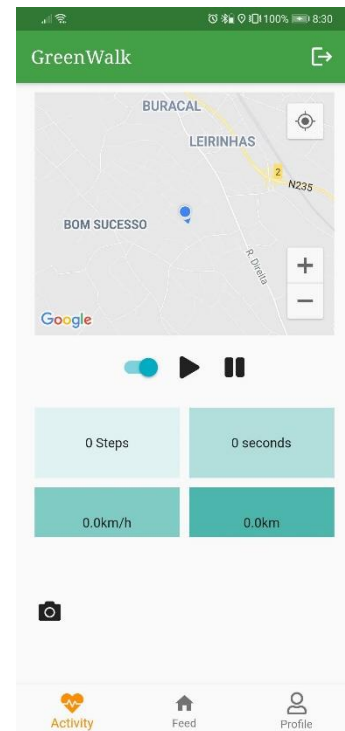
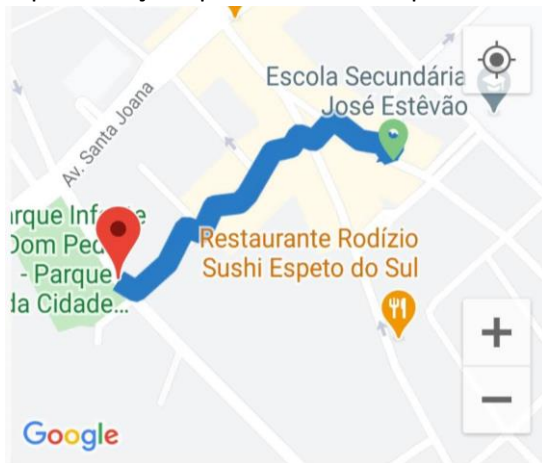




## Activity

É nesta página que se realiza a gravação de percursos realizados. Como podemos ver, temos um mapa que apresenta a localização atual do utilizador. Imediatamente abaixo temos dois botões e um switch. O switch permite alterar entre os estados públicos e privados do percurso. Esta alteração é apresentada por um pop-up que aparece a representar o estado atual. Os botões de play e pause, respetivamente, iniciam e terminam a gravação de um percurso. Após ser iniciada a gravação, a seguinte

representação apresentará no mapa.



Aqui, podemos ver uma polyline que representa o caminho percorrido. Igualmente, o marcador vermelho representa o ponto final do percurso e o verde o inicial. Esta descrição dos marcadores é visível ao carregar em cada um deles.

Os quadrados azuis são 5 e representam diferentes informações sobre a atividade. Esta lista é deslizável pelo que a 5ª caixa se encontra escondida na imagem. Podemos então ver o número de passos dados, tempo decorrido em segundos, velocidade média em km/h, distância total percorrida em km e a média da qualidade do ar.

Em baixo, o ícone de câmara permite tirar fotografias utilizando a câmara e estas são apresentadas imediatamente à direita, como uma lista. Quando clicadas, a imagem é zoomificada. Tal como acontece em **ActivityDetails**.

***Um percurso só é guardado se a distância for diferente de 0 e se o tempo decorrido for maior que 10 segundos.***

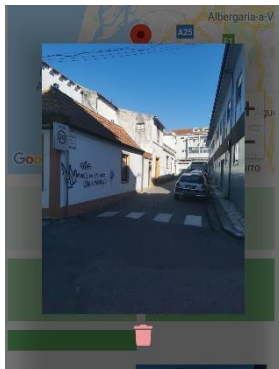
## ActivityDetails

Finalmente, esta página estática possui uma descrição detalhada de um percurso.

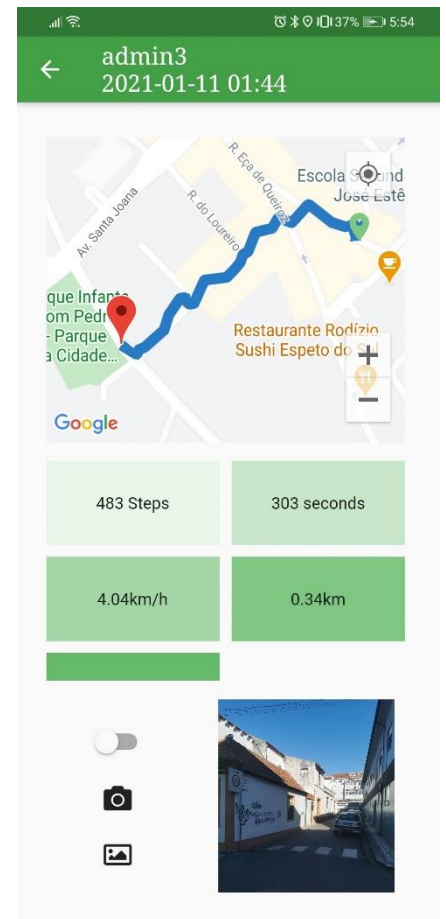
Por cima do mapa, na AppBar, é apresentado o nome do utilizador que a criou bem como a data de sua criação. Como já foi referido, esta página é demonstrada ao carregar numa caixa na página Feed ou Profile.

Tal como foi descrito na página Activity, está representado no mapa o percurso realizado, que teve início no marcador verde e terminou no vermelho. Igualmente, temos acesso aos dados como passos dados, tempo decorrido em segundos, distância percorrida, velocidade média e índice de qualidade média do ar.

Por baixo da lista temos novamente 2 botões e 1 switch. O switch só está disponível para o dono do percurso. Este permite alterar a atividade entre pública e privada. Para além disso os botões de camera e galeria permitem adicionar uma foto à atividade quer através da câmara quer da galeria respetivamente. À direita temos uma lista com todas as fotos associadas à atividade.



Ao clicar numa destas fotos é apresentada uma versão zoomificada da imagem. Caso a atividade seja do utilizador atual é possível eliminar uma imagem clicando no caixote do lixo abaixo da fotografia. Caso contrário este ícone não aparece.



## Conclusão

Com este trabalho consegui com sucesso implementar uma aplicação de tracking de atividade física e de qualidade do ar. Para além do que era suposto ser integrado, adicionei ainda um sistema de notificações. Consegui implementar tudo o que queria e considero que a aplicação ficou de fácil compreensão. No entanto, embora não soubesse como o fazer, gostava de ter desenvolvido melhor a UI.

Trabalho realizado por



Pedro Marques

92926