

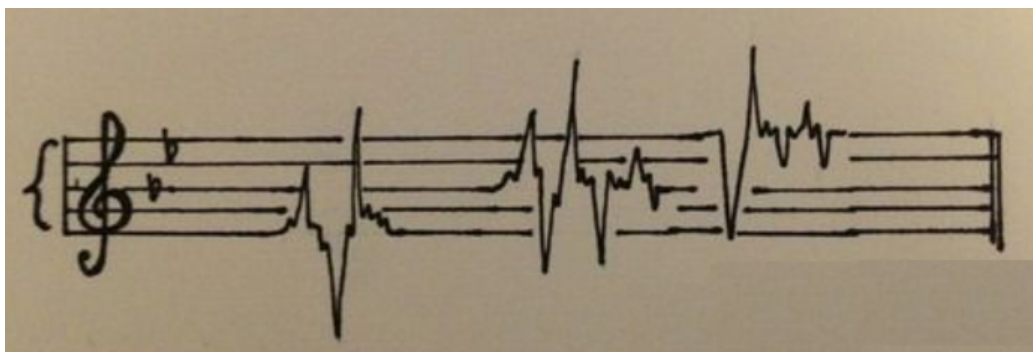


ISEL – INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA
ADEETC – ÁREA DEPARTAMENTAL DE ENGENHARIA DE
ELECTRÓNICA E TELECOMUNICAÇÕES E DE COMPUTADORES

LEIM

LICENCIATURA EM ENGENHARIA INFORMÁTICA E MULTIMÉDIA
UNIDADE CURRICULAR DE PROJETO

Heart Rate Playlist



Tomaz Azinhal (41619)

Orientadores

Professor André Lourenço

Professor Rui Jesus

Julho, 2017

Resumo

Com a popularização de plataformas de *streaming* de músicas, o número de músicas disponíveis aumentou imenso. Uma das razões para este aumento deve-se à disponibilização de *playlists* com novas músicas do mesmo estilo.

Muitas destas *playlists* são criadas por utilizadores com base no seu conhecimento musical ou estilo de música. Assim, uma *playlist* pode ser uma maneira de partilhar os gostos musicais de cada um de nós.

A personalização automática de *playlists* é um conceito aplicado pelas empresas que oferecem estes serviços, dando a descobrir múltiplas novas músicas e poupando tempo a utilizadores da criação das suas *playlists* ou da procura de uma que lhes agrade.

Esta personalização é feita com base no que já foi ouvido pelo utilizador são sugeridas novas músicas com características semelhantes.

Hoje em dia dá-se cada vez mais importância à personalização de conteúdo, pois é uma maneira excelente de angariar utilizadores, uma das razões da popularidade do *Spotify*. Algo extremamente pessoal, e impossível de replicar é a reacção de um utilizador a algo, quer seja uma música, uma imagem, ou um filme. Estas reacções podem ser obtidas através dos batimentos cardíacos em cada momento. Se um utilizador gosta de uma música irá ter uma reacção com batimentos cardíacos diferentes do que se não gostar da música.

Com base nestes conceitos e tendo por objetivo a criação de *playlists* mais personalizada, é apresentado este projeto - a criação de *playlists* baseada na reacção de utilizadores a músicas.

Agradecimentos

Ao Rui Jesus e André Lourenço, pelas suas orientações, disponibilidade e sugestões de alternativas quando me deparei com dificuldades.

Aos colaboradores da CardioID, pela sua ajuda e disponibilização de dispositivos para testar o meu código.

À minha família por me apoiar e manter focado nos meus objetivos.

Aos meus amigos por me manterem motivado, dando-me opiniões, sugestões e críticas construtivas para ir melhorando o projeto.

Índice

Resumo	i
Agradecimentos	iii
Índice	v
Lista de Tabelas	vii
Lista de Figuras	ix
1 Introdução	1
1.1 Estrutura do Relatório	2
1.2 Enquadramento Teórico	3
2 Trabalho Relacionado	5
2.1 Personalização de Conteúdo	5
2.2 Streaming de Música	5
2.3 Sensores Biométricos	6
3 Modelo Proposto	9
3.1 Requisitos	9
3.2 Fundamentos	10
3.2.1 Cliente	11
3.2.2 Servidor	12
3.3 Abordagem	13
4 Implementação do Modelo	17
4.1 Servidor	17
4.1.1 Interface com <i>Spotify</i>	18

4.1.2	Base de dados	20
4.1.3	Servidor Web	20
4.2	Cliente	21
4.2.1	Interface <i>BLE</i>	22
4.2.2	Interface com o Servidor	24
4.2.3	Interface com a aplicação nativa do Spotify	24
4.2.4	Funcionamento Geral	24
5	Validação e Testes	27
6	Conclusões e Trabalho Futuro	29
A	OAuth2 - Spotify	31
	Bibliografia	35

Lista de Tabelas

3.1	Dispositivos relevantes.	11
-----	----------------------------------	----

Lista de Figuras

2.1	Logo do <i>Spotify</i>	6
2.2	Logo do <i>Google Music</i>	6
2.3	Fitbit Charge 2	6
2.4	Mio Alpha 2.	7
3.1	Diagrama do lado do cliente.	12
3.2	Diagrama do lado do servidor.	13
3.3	Diagrama geral.	15
4.1	Diagrama relacional da Base de dados.	21
4.2	Flowchart de uma conexão BLE.	23
4.3	Flowchart geral da aplicação.	25
5.1	Pedido no Postman com a reação de um utilizador.	28
5.2	Utilizador com uma <i>playlist</i> criada.	28
A.1	Diagrama do método <i>Client Credentials Flow</i>	32
A.2	Diagrama do método <i>Authorization Code Flow</i>	33

Capítulo 1

Introdução

O termo *playlist* é um termo bastante conhecido hoje em dia, que significa um conjunto de músicas que partilham uma ou várias características. Este termo é a base do projeto.

A criação automática de *playlists* é atualmente uma ferramenta utilizada por empresas que oferecem *streaming* de música como um serviço para ganharem e manterem uma *user base* estável. Serviços como *Spotify* [Daniel Ek, 2007], por exemplo, criam uma *playlist* todas as semanas para os seus utilizadores. Desta maneira os utilizadores têm conteúdo novo para explorar todas as semanas e podem descobrir a sua nova banda favorita.

Neste documento é proposto um sistema de criação de *playlists* utilizando o ritmo cardíaco para classificar as reações de utilizadores enquanto são ouvidas as músicas, tornando as *playlists* ainda mais pessoais.

A monitorização fisiológica permite influenciar a forma como uma pessoa age e como deve começar a agir para manter um estado de saúde saudável. A integração de sensores biométricos em dispositivos como relógios e bandas de peito facilita esta monitorização do estado de saúde de um utilizador. Estes dispositivos são conhecidos como *smartwatches* e normalmente estão integrados com aplicações móveis que permitem a extração de dados recolhidos pelo dispositivo.

Uma das características exibida na criação de *playlists* automática pelos serviços de *streaming* é que as músicas inseridas em cada *playlist* são baseadas em músicas que um utilizador indicou que gostou. Estes serviços também devem ter em conta o género das músicas que o utilizador ouve, tal com outras características (e.g. energia, “danceabilidade”, tempo, valência, etc. . .).

Como pode ser observado, as características tidas em conta são relacionadas com músicas e não com os utilizadores, o que significa que utilizadores que ouvem as mesmas músicas possam ter a mesma *playlist*.

Diferentes utilizadores podem gostar de uma música no dia a dia, mas caso exista um evento que influencie o gosto de um utilizador, este evento não pode ser tido em conta pelo algoritmo de criação de *playlists*. Eventos como acordar, sair do trabalho ou fazer uma viagem grande influenciam os gostos de cada um. Possivelmente um ouvinte de Rock e Hip Hop prefira ouvir outro tipo de música quando está a acordar. Este tipo de preferências é complicada de detetar, e não é tido em conta pelo algoritmo de criação de *playlists*.

Para complementar a criação de *playlists* não só com características extraídas de músicas, pode-se usar a informação extraída de sensores de quando um utilizador ouve uma música, influenciando as músicas utilizadas como *seeds*. Assim cada *playlist* de cada utilizador fica mais pessoal.

Tendo em conta o tipo de sensores existentes hoje em dia, a característica utilizada para influenciar as *seeds* é o ritmo cardíaco. Mais especificamente a *heart rate variability*, que é calculada com a diferença entre batimentos cardíacos. É bastante diferente de usar o número médio de batimentos cardíacos por minuto (*bpm*) porque o que diferencia uma reação a uma música é a diferença de um batimento cardíaco para outro e não a variação da média.

1.1 Estrutura do Relatório

Este relatório está organizado de acordo com a seguinte estrutura proposta:

- No capítulo um, é realizada a *Introdução* ao trabalho, apresentando as motivações, conceitos principais e o enquadramento teórico.
- No capítulo dois, é identificado o *Trabalho Relacionado*.
- No capítulo três, é apresentado o *Modelo Proposto* que complementa a fase de conceito e pré-produção.
- No capítulo quatro, é descrita a *Implementação do Modelo*, onde são apresentados as tecnologias implementadas e explicados os métodos para a produção do modelo proposto.

- No capítulo cinco, são abordados os testes e avaliações realizados ao longo do projeto.
- No capítulo seis, é exposta a *Conclusão* e análise de *Trabalho Futuro*.

1.2 Enquadramento Teórico

Ao longo deste projeto irá ser mencionado múltiplas vezes o termo “reação” e batimentos cardíacos. Estes termos significam a mesma coisa neste contexto, que é a **Heart Rate Variability**, ou **HRV**. O HRV é a diferença de tempo entre batimentos cardíacos consecutivos, designados de intervalos RR, medido em milisegundos. O conceito mais conhecido de **Heart Rate**, mede o número médio de batimentos por minuto. Ao contrário da HRV, não precisa de valores exatos, pois é uma média, e por esta razão não pode ser utilizado para medir reações. Reações são instantes muito curtos que podem ser detetados apenas com a HRV, e não com o número médio de batimentos cardíacos.

Heart Rate Variability e Heart Rate são inversamente proporcionais [Moore, 2014], ou seja, quando estamos em repouso, o número de batimentos cardíacos deve ser baixo. Isto quer dizer que o intervalo entre batimentos deve ser maior. Quando se está num estado ativo, o intervalo entre batimentos é menor, e a Heart Rate mede um valor mais elevado.

A Heart Rate Variability é usada como uma ferramenta de medição da atividade do sistema nervoso autónomo. É possível utilizar a HRV para detetar stress psicológico, digestivo e físico. Apesar destes usos, é difícil medir a HRV durante exercício. A precisão dos dispositivos utilizados dependem muito do tipo de *wearable* utilizado, e os dispositivos mais precisos são normalmente mais caros.

Durante este projeto deve-se ter em conta que a medição da HRV é feita em repouso, para se ter a maior precisão possível das reações dos utilizadores.

Capítulo 2

Trabalho Relacionado

A personalização de conteúdo não é um conceito novo, quer seja de música, imagens ou produtos, se temos preferências então podemos ser alvo de conteúdo personalizado. Certamente já foi notaram que quando fazemos uma pesquisa por um hotel ou por preços para uma viagem de férias, quando nos dirigimos a outro *website* que tenha publicidade, essa publicidade estará eventualmente relacionada com hotéis ou preços de uma agência de viagens.

2.1 Personalização de Conteúdo

A *Google*, *Facebook*, e outros, utilizam mecanismos de caracterização dos seus utilizadores para oferecerem conteúdo personalizado a cada um dos seus utilizadores. É neste prisma que parte do trabalho se insere, na personalização de conteúdo baseado nas reações de utilizadores a músicas.

2.2 Streaming de Música

Um dos serviços de *streaming* mais utilizados neste momento, é o *Spotify*. Para além de ter uma base de dados com um elevado número de músicas, serviços de recomendação e personalização de conteúdo, tem também uma componente social, tornando-o num dos serviços mais popular.

Outros serviços de *streaming* semelhantes ao *Spotify* são *Google Music* e *Apple Music*. Com poucas diferenças entre eles, design e pequenos *add-ons*, estes são os outros grandes serviços de *streaming* de música.



Figura 2.1: Logo do *Spotify*.



Figura 2.2: Logo do *Google Music*.

Como já foi referido, estas *aplicações de streaming* utilizam algoritmos “estáticos” para fazer a personalização. Em contraste, este trabalho irá tentar implementar uma componente “dinâmica” em conjunto com a aplicação de *streaming* escolhida.

2.3 Sensores Biométricos

Normalmente, os *smartwatches* oferecem sensores de temperatura, localização e sensores de monitorização cardíaca. Dispositivos como *Fitbit*, *Mio Alpha* e *Mio Slice*, oferecem estes sensores que permitem que se saiba o contexto do utilizador.



Figura 2.3: Fitbit Charge 2



Figura 2.4: Mio Alpha 2.

Apesar de estes sensores terem todas estas capacidades, não se utilizou nenhum destes sensores pela falta sensores capazes de analisar intervalos RR. Por esta razão, utilizou-se um dispositivo com sensores capazes de retirar os intervalos desejados desenvolvidos pela *CardioID*.

Capítulo 3

Modelo Proposto

Dados os conceitos e ideias nos quais este projeto se baseia, é necessário fazer o levantamento de requisitos para que a aplicação desenvolvida seja capaz de criar recomendações para utilizadores registados na aplicação. Após os requisitos estarem identificados, serão abordados os fundamentos usados para este projeto, identificando as tecnologias e serviços usados.

3.1 Requisitos

Esta secção tem como intuito formular os requisitos da aplicação a ser desenvolvida.

Este projeto tem como objetivo final o desenvolvimento de uma aplicação que crie automaticamente *playlists*. Ao contrário dos algoritmos atuais de criação de *playlists*, esta aplicação/algoritmo deverá ter em conta a reação de um utilizador a cada música que ele ouve.

Use case: Um utilizador quer uma experiência mais personalizada para ouvir música que vá em encontro não só ao tipo de música que ele gosta, mas também em como se sente. Quando uma pessoa não se sente bem, ou teve uma experiência menos positiva, isso irá influenciar a sua disposição e, por sua vez, o seu gosto musical.

O utilizador necessita de um dispositivo que consiga medir os seus batimentos cardíacos. Após a medição dos batimentos cardíacos, os batimentos cardíacos devem ser enviados para um servidor. O servidor que recebe esta informação deve guardá-la.

Periodicamente devem ser criadas *playlists*. Isto é feito através da análise

das reações de cada utilizador, e escolhidas as músicas às quais o utilizador reagiu de uma maneira mais positiva, criar uma *playlist* com características semelhantes.

Use case: Ao ser criada a *playlist* automaticamente, o utilizador deve poder avaliar a *playlist* para que seja avaliada a qualidade do algoritmo. Desta maneira, futuras *playlists* mantêm ou sobem na qualidade das músicas recomendadas.

Deve ser feita a integração com uma plataforma de *streaming* para que se possa já ter uma base de dados de músicas, com as respectivas características. Esta plataforma deve ter uma API que deva poder ser utilizada para criar as respectivas *playlists* para os utilizadores.

3.2 Fundamentos

Como pode ser identificado nos requisitos, um utilizador irá utilizar esta aplicação para ter uma experiência mais personalizada. Para cada utilizador é preciso guardar as suas reações para posterior análise. Escolheu-se implementar um modelo cliente-servidor, onde o cliente é a aplicação que o utilizador irá utilizar. O cliente deve também comunicar com o dispositivo que mede os batimentos cardíacos de cada utilizador para os poder enviar para o servidor. O servidor vai receber a informação de cada utilizador e guardá-la numa base de dados.

Podem ser identificados os componentes principais para cumprir os requisitos:

- Um cliente para enviar as reações dos utilizadores;
- Um servidor web para receber os pedidos(reações) dos utilizadores;
- Uma base de dados para guardar a informação relativa aos utilizadores;
- Uma interface com a plataforma de *streaming*.

Escolheu-se o *Spotify* como plataforma de *streaming*. O *Spotify* disponibiliza a sua plataforma em múltiplos sistemas operativos - a mais importante para este trabalho é a aplicação Android. O *Spotify* tem uma Web API bastante bem documentada e SDK para desenvolver as interfaces necessárias.

Para o primeiro requisito houve uma pesquisa para encontrar um dispositivo que desempenhasse as funções necessárias. Os requisitos do dispositivo são conseguir retirar os intervalos do sensor, ser confortável de usar (e.g., relógio), e uma interface para se conseguir extrair os dados do dispositivo.

Nome	SDK	API	Tipo
Jawbone UP3	Não	Não	Smartwatch
Polar H10	Não	Não	Cheststrap
Wahoo TICKR	Não	Não	Cheststrap
Zephyr HxM	Sim	Sim	Cheststrap
Garmin Premium	Não	Não	Cheststrap
Mio Link	Não	Não	Smartwatch
Mio Slice	Não	Não	Smartwatch

Tabela 3.1: Dispositivos relevantes.

Como se pode ver na 3.1, nenhum dos dispositivos encontrados garante as especificações pelo que foi necessário recorrer a outro dispositivo. O dispositivo escolhido não é confortável mas para extrair a informação necessária é fácil implementar a interface. O dispositivo escolhido foi desenvolvido pela *CardioID*. O protocolo de comunicação com o dispositivo é *Bluetooth Smart*, ou melhor conhecido por *Bluetooth Low Energy (BLE)*.

3.2.1 Cliente

Para este projeto decidiu-se utilizar Android para implementar a interface do lado do cliente. Android permite a ligação ao dispositivo com os sensores biométricos, pois os dispositivos Android têm o hardware necessário (a partir da API 18) para comunicar através de *BLE*.

O cliente Android a ser desenvolvido deve ser capaz de fazer pedidos HTTP, enviando a informação de cada utilizador para um servidor. A informação que é enviada deve identificar o utilizador, a música que o utilizador ouvia e a sua reação. Podem também ser enviados outros dados como as horas a que foi a reação e a localização do utilizador.

O cliente tem implementa as seguintes funções:

- Conectar-se ao dispositivo com sensores biométricos;
- Obter a informação dos sensores;
- Uma interface com a plataforma de *streaming*, para obter informação de cada música;
- Enviar a informação para o servidor.

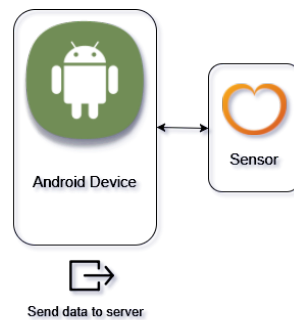


Figura 3.1: Diagrama do lado do cliente.

3.2.2 Servidor

O lado do servidor deve implementar um *Webserver* para receber pedidos dos clientes, nomeadamente o registo de reações. Estas reações devem ser guardadas numa base de dados para posterior análise. Em paralelo, à medida que vão sendo adicionadas reações, as músicas às quais os utilizadores vão reagindo devem ser analisadas com o auxílio da API do *Spotify*. Após haver uma base de dados decente de reações para cada utilizador, são analisadas periodicamente todas as reações e feitas as *playlists*.

As tecnologias implementadas para fazer o lado do servidor foram *Python* [Python3.6,] e *MySQL*. SQL significa *Structured Query Language*, e é usado para comunicar com bases de dados. *MySQL* é a linguagem standard para manter bases de dados relacionais. Foi escolhida esta linguagem porque é a linguagem com a qual eu estou mais confortável e a que eu tenho mais experiência. É potente e tem um número imenso de bibliotecas que facilitam no desenvolvimento de aplicações.

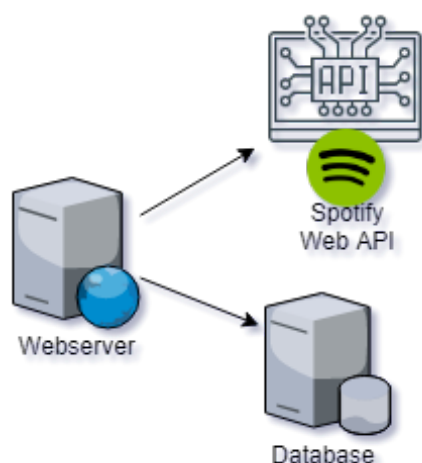


Figura 3.2: Diagrama do lado do servidor.

3.3 Abordagem

Foi realizada uma abordagem “bottom-up”. Isto significa que foram identificados todos os blocos a serem desenvolvidos e só depois de estarem interligados é que a aplicação final pode funcionar. É diferente de uma abordagem “top-to-bottom”, pois na segunda abordagem é implementado um prototipo muito simples inicialmente e a partir desse ponto são separados em blocos cada uma das funcionalidades. Esta foi a abordagem escolhida pois após a primeira análise dos requisitos foi decidido fazer implementações de partes do sistema de modo a ganhar experiência com as tecnologias utilizadas e maior capacidade crítica na sua integração. Uma abordagem “top-to-bottom” não permite este tipo de desenvolvimento.

Em primeiro lugar, foi experimentada a API do *Spotify*, dado ser esta a plataforma mais relevante para o trabalho final. Utilizou-se uma biblioteca em Python que implementa muitos dos *endpoints* necessários para comunicar com a API do *Spotify*, chamada “Spotipy”. Para utilizar a API do *Spotify* é necessário registar a aplicação para que se tenha uma chave de acesso à API e serem medidos os pedidos feitos por cada aplicação.

Em paralelo com o estudo da API do *Spotify*, foi criada a base de dados MySQL para poder guardar os dados de cada utilizador incluindo as reações e músicas. A API do *Spotify* permite retirar metadados relacionados com cada música, esta informação deve ser guardada na base de dados para que o

sistema tenha a maior independência possível da API do *Spotify*. Após criada a base de dados criaram-se os comandos para comunicar com a base de dados em Python. Após os comandos necessários para comunicar entre a API do *Spotify* e a base de dados, foi criado o Webserver para receber pedidos dos clientes com informação quanto ao seu registo na aplicação e ao registo de reações de utilizadores autenticados. Este Webserver é criado como uma “microframework” de Python chamada “Flask”. É com esta framework que se vai criar os endpoints necessários para receber a informação dos utilizadores.

A aplicação Android utiliza BLE para comunicar com o sensor que mede os intervalos RR. Após retirar informação do sensor, deve ser enviada a informação da reação, mais informação relativa ao contexto em que o utilizador se insere, como a data e a hora e a localização do utilizador. Este contexto pode ser utilizado para filtrar reações por hora ou por localização, e.g., quando o utilizador está no local de trabalho, ou de manhã quando está a tomar o pequeno almoço. Desta maneira, pode-se introduzir outra variável para o conteúdo ser ainda mais personalizável. Este contexto deve ser guardado enquanto um utilizador está a ouvir uma música e enviado quando o utilizador muda de música ou deixa de utilizar a aplicação.

Para enviar pedidos para os *endpoints* desenvolvidos através de Android, existe um módulo chamado *HttpOk*, mas é um módulo já antigo. Um módulo popular e recomendado para fazer pedidos HTTP é o *Retrofit*. Tem funcionalidades muito úteis e a sua implementação é fácil de entender. Destacam-se as seguintes funcionalidades do *Retrofit*:

1. Conversão automática de e para JSON;
2. Mecanismo para enviar novamente o pedido caso o primeiro falhe;
3. Fazer pedidos em background.

A partir destes requisitos e tendo em conta a abordagem tomada, a arquitetura desenhada para este projeto é a exibida no diagrama 3.3.

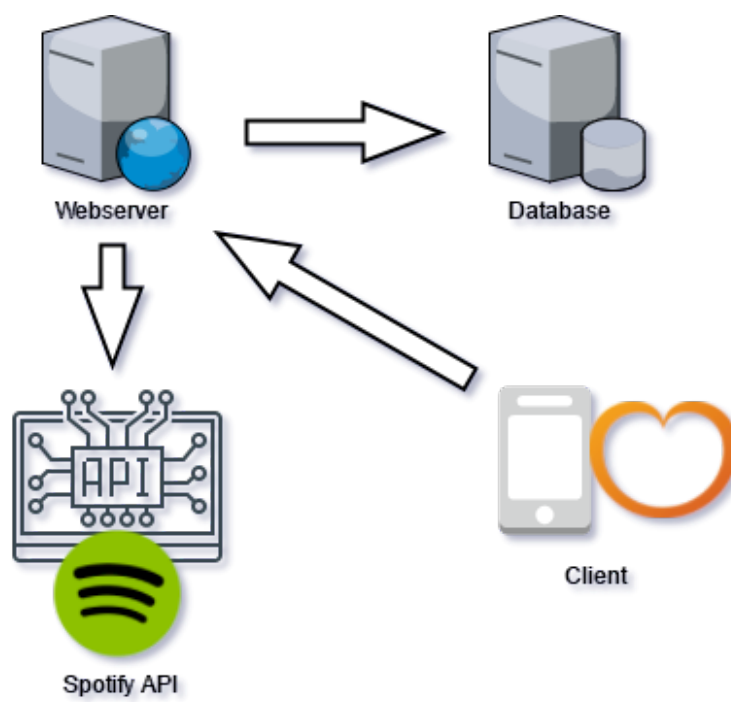


Figura 3.3: Diagrama geral.

Capítulo 4

Implementação do Modelo

Neste capítulo é descrito a implementação do modelo proposto. É aqui que vai ser apresentado ao detalhe todos os blocos implementados, bem como as dificuldades que foram aparecendo na implementação de cada um dos blocos. Vai ser seguida a ordem de apresentação introduzida na secção 3.3. Começando, assim, pela implementação do servidor e de seguida na implementação do cliente Android.

Os blocos desenvolvidos foram implementados por uma ordem específica pois foram descobertas dependências entre blocos que só conseguiriam ser implementados após a compleção de outros blocos. As dependências entre blocos ao longo da implementação foram as seguintes:

- O servidor web só pôde ser completo quando se soube o tipo de dados que viriam do cliente Android;
- O bloco que comunica com a base de dados só pôde ser completo quando se soube que dados vinham da interface com o Spotify;
- O bloco que comunica com a base de dados só pôde ser completo quando se soube que dados viriam do servidor web;

4.1 Servidor

Como descrito na secção 3.3 foi feita uma implementação “bottom-up”. Começando pelo servidor, e tendo em conta as restrições e requisitos do lado do cliente, foram criados os seguintes blocos de implementação:

1. Interface com o Spotify;
2. Base de Dados;
3. Servidor Web.

4.1.1 Interface com *Spotify*

Como já foi descrito nos fundamentos a biblioteca utilizada chama-se *Spotipy* [Lamere, 2014] e a sua função é facilitar o uso dos *end-points* disponibilizados pelo Spotify da sua Web API [Spotify, b]. A existência deste módulo facilitou este projeto pois não seria necessário criar todas funções para utilizar os *end-points* e tratar de possíveis erros que venham da sua utilização. A utilização deste módulo passou pela compreensão do método de autenticação *OAuth2*, apresentado em mais detalhe no apêndice A.

De seguida vão ser descritas as classes utilizadas, bem como as funções utilizadas de cada classe. As primeiras duas classes servem maioritariamente para obter *access tokens* e autenticar utilizadores. A última classe será a mais utilizada após serem obtidos os *access tokens*.

spotipy.oauth2.SpotifyClientCredentials : Esta classe serve para obter um *access token* através do ID da aplicação, estando assim limitados os pedidos que podem ser feitos a informação não referente a utilizadores.

spotipy.oauth2.SpotifyOAuth : Esta classe é usada para obter *access tokens* de utilizadores que se querem autenticar. Com este *access token* pode-se gerir a informação das *playlists* de cada utilizador e se descrito no *scope* da aplicação, outros dados.

spotipy.Spotify : É nesta classe que se faz interação com os *end-points* que têm a informação relevante a músicas e utilizadores. Recebe vários parâmetros, entre os quais um *access token* ou a classe com o tipo de autenticação utilizada. Se o *access token* fornecido for de um utilizador (*spotipy.Spotify(auth=access_token)*), então é possível aceder à informação desse utilizador. O métodos utilizados ao longo do projeto são apresentados de seguida:

- **me()**: Este método retorna a informação pessoal de um utilizador, como por exemplo o seu email, ID do *Spotify*, data de criação de conta, playlists, etc. . . Este método é maioritariamente para confirmar que a gestão feita pela aplicação está a ser feita corretamente.
- **user_playlist_create()**: Este método cria uma *playlist* para um utilizador com o nome e descrição passados como argumentos. É necessário instanciar a classe *spotipy.Spotify* com a autenticação do utilizador cujo ID se quer criar a *playlist*. Ao ser criada a *playlist* com sucesso é retornado o ID da *playlist* para depois ser referenciada quando se quiserem adicionar músicas.
- **user_playlist_add_tracks()**: A função deste método é a adição de músicas a playlists. Devem ser passados como parâmetros o ID do utilizador, o ID da playlist desse utilizador à qual devem ser adicionadas as músicas e as músicas a serem adicionadas.
- **audio_features()**: Esta função é utilizada para se retirar mais informação de cada música que foi ouvida. Deste modo, tem-se mais informação de cada música e podem ser usados mais características para influenciar a criação de playlists. Atributos como a “energia”, “*danceability*”, tom, etc. . . Este método é usado maioritariamente para não se ter de aceder à API do Spotify se for necessário acesso a estes dados, estando mais independente da informação do Spotify e permitindo escalabilidade à aplicação.
- **recommendations()**: Como o nome indica este método retorna uma lista de recomendações. Esta lista é baseada numa lista de músicas passada como argumento, e um algoritmo do Spotify irá analisar e retornar um conjunto de músicas baseadas nos argumentos. As músicas passadas como argumento são normalmente chamadas de “seed”. Também é possível passar como argumento o ID de gêneros de música (Rock, Jazz, Hip Hop, etc).

Estes são os métodos e classes utilizados na implementação da interface da API do Spotify.

4.1.2 Base de dados

A base de dados relaciona utilizadores, *playlists* e músicas (ver figura 4.1). Os utilizadores são registados na base de dados após autorizarem a aplicação a aceder aos seus dados dentro do *scope* definido. Quando isto acontece, é criada uma entrada com o ID do utilizador, o *access token* e o *refresh token* recebidos, bem como a data de expiração do *access token*.

Em adição a estas 3 tabelas, existe a tabela que guarda o HRV e o respetivo contexto das reações dos utilizadores. Esta tabela ficou designada de “reação”, pois guarda o HRV e o contexto. Uma reação faz referência a uma música e ao utilizador do qual foi obtido a reação. A outra tabela existente é a tabela de recomendações, onde estão guardadas as referências das músicas que são usadas nas *playlists*, bem como as referências para as *playlists*. Esta tabela é criada pela maneira como são implementadas *playlists* numa base de dados, que são um conjunto de referências para músicas (ver figura 4.1).

As músicas vão sendo adicionadas à medida que os utilizadores vão tendo reações a músicas, enviando a referência para a música que ouviram. Quando as *playlists* forem criadas a partir das recomendações do *Spotify*, estas são adicionadas às músicas e de seguida referenciadas nas recomendações. Desta maneira obtém-se uma tabela de músicas que todos os utilizadores já ouviram ou foram recomendadas para serem ouvidas nas *playlists*.

4.1.3 Servidor Web

O servidor web foi desenvolvido em Python com o auxílio da *microframework* Flask[Ronacher, 2010]. Esta framework dispõe utensílios, bibliotecas e tecnologias necessárias para construir uma aplicação web. É simples de implementar, e com poucas linhas de código é possível criar um servidor que responda a pedidos HTTP. O servidor implementado recebe dois pedidos, um para registar utilizadores à aplicação, e outro para receber reações de utilizadores.

- **new_user:** Este pedido vem do seguimento de um utilizador se querer registar na aplicação. Quando um utilizador quer autorizar a aplicação a aceder aos seus dados pessoais, é acedido um *end-point* para o utilizador autorizar a aplicação. Para este *end-point* são passados como argumentos o ID da aplicação, o segredo da aplicação e um “redirect URL”. Para este URL é enviado o código que deve ser trocado pelo

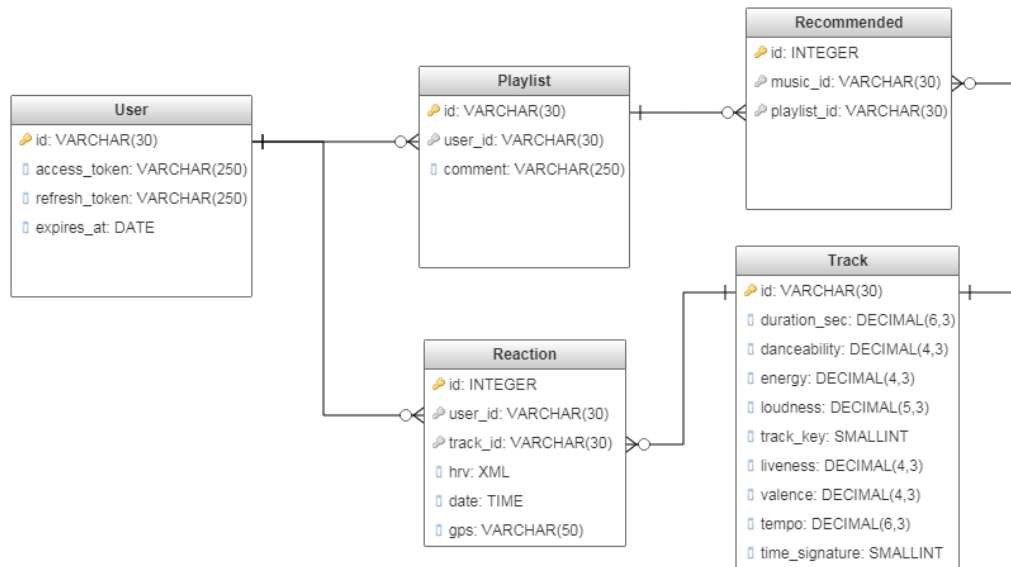


Figura 4.1: Diagrama relacional da Base de dados.

access token e pelo *refresh token* do utilizador autenticado. O URL passado como argumento deve ser o da máquina que está à espera de receber esta informação para a poder tratar conforme. Neste caso, será o servidor web desta aplicação.

- **new_reaction:** Este é utilizado pelo cliente para enviar a informação do contexto quando um utilizador ouve, ou acaba de ouvir, uma música na aplicação nativa do *Spotify*. Um utilizador ouve uma música e em paralelo são registados os batimentos cardíacos do utilizador.

4.2 Cliente

O cliente é a aplicação em Android que vai servir para retirar as medições do dispositivo com os sensores de HR e enviar essa informação para o servidor. Dito isto, é possível identificar as duas principais componentes no desenvolvimento do cliente - a interface com o dispositivo com os sensores (Bluetooth Low Energy)[Bluetooth SIG, 2017] e a interface que comunica com o servidor.

4.2.1 Interface *BLE*

O dispositivo com os sensores tem um adaptador BLE para poder comunicar e é identificado com um *Universally Unique Identifier* (**UUID**). Todos os dispositivos Bluetooth têm um UUID que serve para serem identificados univocamente. Estes dispositivos podem também ser identificados (apesar de não univocamente) através do tipo de serviços que disponibilizam (serviços *GATT*). Estes serviços também contêm um UUID pelos quais podem ser identificados univocamente, isto é, um dispositivo não pode ser diferenciado de outro que tenha os mesmos serviços através dos serviços, mas os serviços são identificados univocamente no mesmo dispositivo. Estes serviços rangem desde medições de ritmo cardíaco, temperatura, bateria, data e hora, alerta de notificações, informação do dispositivo, entre muitos outros ...

Cada um destes serviços GATT podem ter uma ou várias características, que também podem ser identificadas univocamente. É através destas características que se retira informação.

No contexto específico do trabalho implementado, tem-se o sensor de Heart Rate que tem um serviço GATT de Heart Rate. Este serviço tem múltiplas características, como a característica que tem os *beats per minute*(**bpm**) de um utilizador e a característica que lê a diferença entre batimentos cardíacos. A que interessa é a com a qual se obtém o HRV, e para obter estas medições é preciso configurar o dispositivo para enviar a informação do HRV, isto é feito configurando os descritores do dispositivo.

Este sistema de comunicação com dispositivos BLE é feito maioritariamente com *callbacks*. Como é o caso da descoberta de dispositivos, é utilizado um comando para começar a procura de dispositivos, quando um dispositivo com as características procuradas, é chamado um *callback* com informação do dispositivo encontrado. A mesma lógica é feita com os serviços e com as características. O descritor acima mencionado serve para indicar ao dispositivo com os sensores que envie uma notificação ao dispositivo Android que a característica à qual ele foi subscrito foi mudada. Deste modo, não é lida a característica cujo valor se quer saber e sim é notificado este valor ao dispositivo Android.

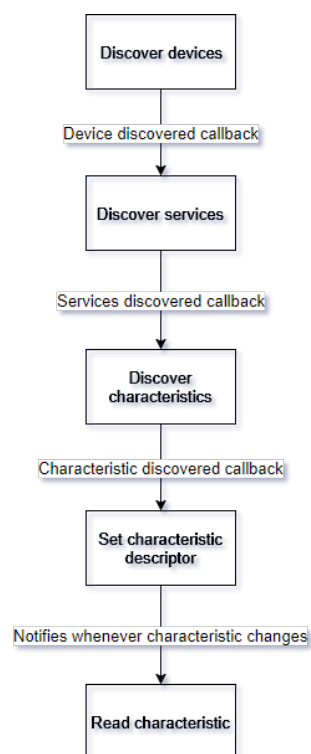


Figura 4.2: Flowchart de uma conexão BLE.

4.2.2 Interface com o Servidor

A interface com o servidor é feita com o auxílio da biblioteca *Retrofit*. Como descrito nos Requisitos, a informação da reação deve ser enviada para o servidor através de um *endpoint* específico do servidor. Este *endpoint* é identificado na aplicação Android, em conjunto com o endereço do servidor. Para utilizar este serviço são declarados os argumentos que devem ser passados no pedido, e o tipo de pedido que é feito.

Este *endpoint* utiliza o método **POST** e requisita múltiplos argumentos:

- O ID do utilizador;
- O ID da música;
- A localização do utilizador;
- A data e a hora à qual o utilizador ouviu a música;
- O HRV do utilizador a esta música.

4.2.3 Interface com a aplicação nativa do Spotify

A aplicação nativa do Spotify tem uma opção para auxiliar developers de aplicações que queiram utilizar a sua API. Esta opção faz com que aplicações no mesmo dispositivo Android consigam saber qual o estado de playback e qual a música que está a ser tocada. Esta informação é passada com o envio de *Broadcasts* por parte da aplicação nativa. Para capturar esta informação é preciso criar um *Broadcast Receiver*. Os broadcasts vêm de fontes específicas, se não for indicado a fonte à qual se está à escuta então não se irá capturar o broadcast. Por esta razão é preciso indicar qual a ação pela qual se está a filtrar no *AndroidManifest*.

Após receber o *broadcast* esperado, são retiradas as informações adequadas e guardadas para depois serem tratadas e enviadas para o servidor.

4.2.4 Funcionamento Geral

Aqui serão explicados todos os processos detalhadamente e como interagem com os outros blocos na aplicação do cliente. Através do flowchart

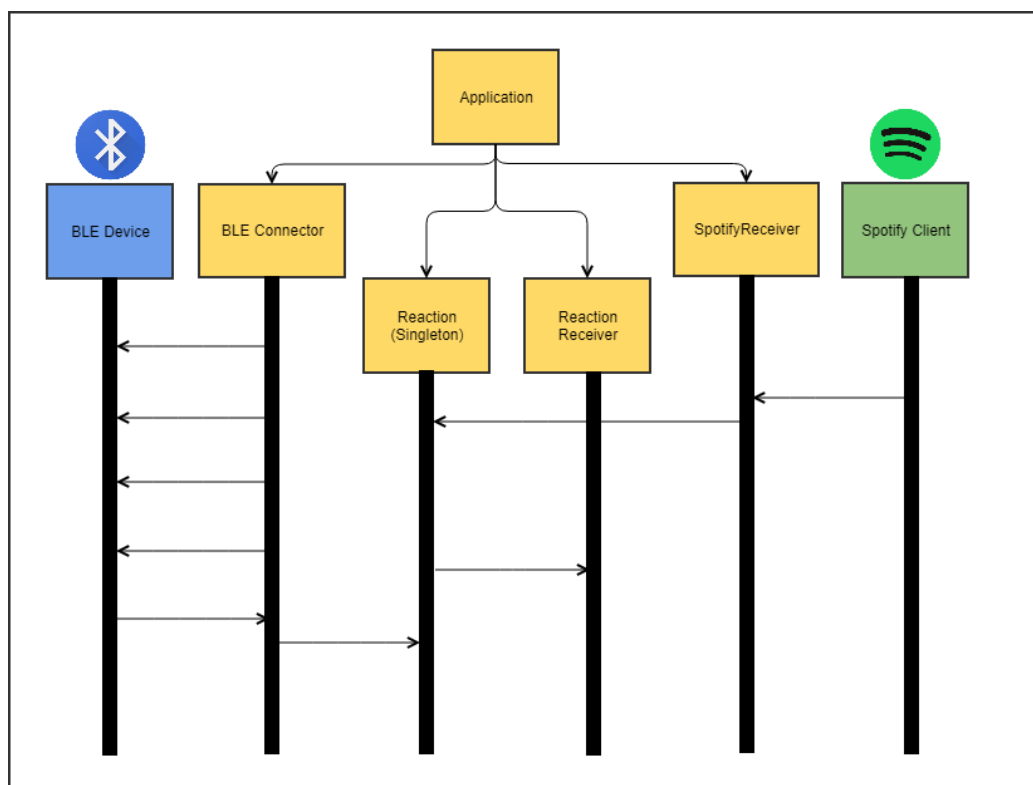


Figura 4.3: Flowchart geral da aplicação.

4.3 consegue-se perceber como devem ser cumpridos os requisitos e a ordem pelo qual foram implementados.

A implementação da interface que comunica com o dispositivo com os sensores foi desenvolvida primeiro. Quando a aplicação é lançada é feita uma verificação das capacidades de BLE do dispositivo Android, e caso seja, pedido que seja ligada a interface de Bluetooth. De seguida começa-se a seguir o flowchart 4.2 para ser feita a ligação ao dispositivo e poder ser retirada a informação dos sensores. Feito, isto, o dispositivo Android é notificado e recebe informação de HR e sempre que este for atualizado.

Para guardar a informação recebida dos sensores e do Spotify foi criada uma classe do tipo *Singleton*, designada de *Reaction*. Foi escolhida esta solução porque é fácil de implementar, apenas sendo preciso alguns cuidados, dado que só poderá haver uma instância desta classe em qualquer momento. Esta classe guarda o utilizador, a música que está a ser tocada, se a música está a tocar no momento e a hora. A localização não foi implementada por dificuldades na compreensão da API da localização.

Como já foi referido, a interface com o Spotify é feita com um Broadcast Receiver. Este recetor, quando recebe a informação de que o utilizador mudou de música, atualiza a instância de *Reaction* com a música que está a ser tocada. O mesmo é feito para se a música está em playback ou não. Se estiver a ser tocada, então devem ser guardados os dados recebidos do sensor para depois poderem ser enviados, se a música não estiver a ser tocada então os dados dos sensores deverão ser descartados pois não constituem uma reação referente à música que está na instância.

Uma reação deve ser enviada quando a música que um utilizador está a ouvir muda para outra. Quando isto acontece, é feito um *broadcast* com esta informação, tal como a aplicação do Spotify faz, para a informação ser enviada. Tem-se um *ReactionReceiver* que está à escuta deste específico *broadcast*. Após ser retirada a informação relevante, cria-se um pedido ao servidor utilizando Retrofit.

Capítulo 5

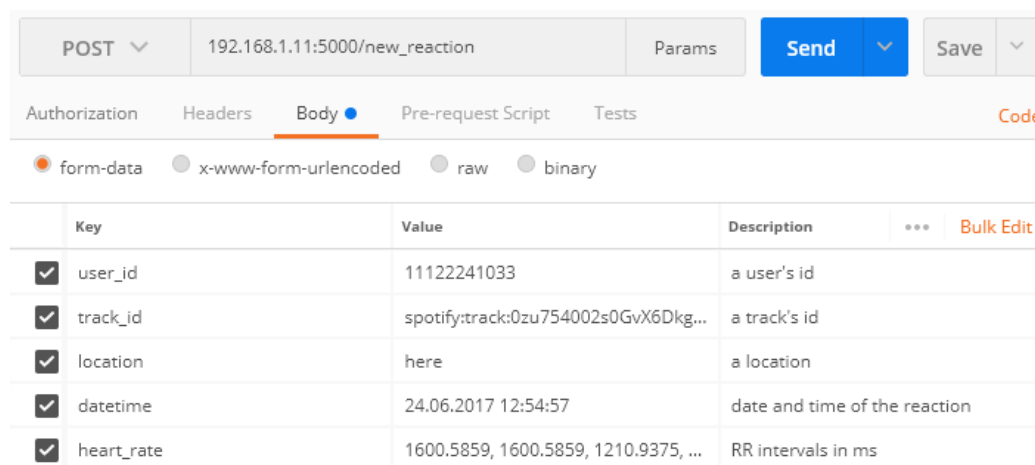
Validação e Testes

Neste capítulo vão ser enunciados os testes que foram sendo feitos ao longo projeto de modo a manter a coerência entre blocos e discernir implementações úteis de implementações menos úteis. Na implementação feita, “bottom-up”, sentiu-se necessidade de interligar os blocos através de testes mais simples, para se confirmar que as implementações estão a seguir o rumo correto.

Os primeiros testes feitos foram confirmações ao sistema de autenticação do Spotify, visto que este não é fácil de compreender. Foi criada uma conta de Spotify de testes que ainda não tivesse músicas e preferências associadas. Esta conta deu acesso à sua informação à aplicação desenvolvida. Após o utilizador dar acesso à aplicação, apareceu na base de dados a informação do utilizador em conjunto com os respetivos *tokens*.

Uma das ferramentas usadas para testar todos os *endpoints*, tanto do Spotify como os criados ao longo do projeto, foi o Postman [Postdot Technologies, 2012]. O Postman permite fazer pedidos a *endpoints* facilmente e guardar todos os pedidos e respostas num histórico.

A confirmação de que as playlists foram criadas é feita na plataforma do *Spotify*, sendo criada uma playlist com o nome “HRP-date”, onde *date* é o dia em que foi criada a playlist como mostrado na figura 5.2.



POST 192.168.1.11:5000/new_reaction Params Send Save

Authorization Headers **Body** Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	user_id	11122241033	a user's id		
<input checked="" type="checkbox"/>	track_id	spotify:track:0zu754002s0GvX6Dkg...	a track's id		
<input checked="" type="checkbox"/>	location	here	a location		
<input checked="" type="checkbox"/>	datetime	24.06.2017 12:54:57	date and time of the reaction		
<input checked="" type="checkbox"/>	heart_rate	1600.5859, 1600.5859, 1210.9375, ...	RR intervals in ms		

Figura 5.1: Pedido no Postman com a reação de um utilizador.

Figura 5.2: Utilizador com uma *playlist* criada.

Capítulo 6

Conclusões e Trabalho Futuro

A personalização de conteúdo é um tema bastante importante hoje em dia e pode ser o que separa serviço de sucesso de um serviço sem sucesso. Utilizadores sentem-se tentados a utilizar serviços cujo conteúdo é feito à sua medida, mantendo os utilizadores leais a uma plataforma em vez de outra. Este projeto é considerado uma prova de conceito de que a personalização de conteúdo pode ser levada um passo mais longe e ter em conta os gostos imediatos de um utilizador, sendo medida a HRV desse utilizador. O desafio ao longo deste projeto foi a integração de várias tecnologias e plataformas recentes.

Como prova de conceito, existem muitas componentes que devem ser melhoradas e implementadas com mais bases teóricas e testes extensos. De seguida destacam-se melhorias técnicas que podem ser feitas à aplicação desenvolvida:

1. Implementar um *load balancer* para que o servidor consiga atender múltiplos pedidos em simultâneo;
2. *Deploy* do servidor numa máquina fora da WLAN dos clientes (atualmente os clientes têm de estar na mesma WLAN);
3. Aquisição de um domínio para o servidor poder ser acedido publicamente;
4. Implementação de uma base de dados no cliente para o caso de o cliente não ter acesso à Internet (possivelmente com Firebase);

5. Utilizar o SDK do Spotify para fazer login e autorizar a aplicação no cliente;
6. Implementação das bibliotecas para aceder à localização do utilizador;
7. Avaliação das playlists criadas para futuras melhorias no algoritmo de recomendações.

Para aplicar um algoritmo de recomendações baseado no HRV e nas respetivas características, seria preciso aprofundar o conhecimento sobre quais as características existentes e os seus significados. Só desta maneira se poderia tirar conclusões quanto à emoção que um utilizador sente em relação a uma música.

O desenvolvimento de um projeto desde o cliente até ao servidor serviu como excelente experiência de como poderá ser o seguimento de um projeto desde a sua raiz ao seu lançamento.

Apêndice A

OAuth2 - Spotify

OAuth2 é uma framework de autenticação que permite que aplicações obtenham acesso limitado a contas de utilizadores. É permitida à aplicação que o utilizador faça login no serviço através da sua plataforma. Assim, esta aplicação consegue aceder à informação do utilizador e maneja-la conforme quiser (dentro dos acessos permitidos pelo utilizador). Existem vários *flows* possíveis para utilizar a API do *Spotify*[Spotify, a], por isso é preciso ter em consideração os usos de cada um.

- *Client Credentials Flow*(**CCF**): Usado para autenticar a aplicação desenvolvida para obter um limite mais elevado de pedidos a ser feitos pela aplicação. Este método não inclui autenticação pelo que não se pode aceder a informação de utilizadores.
- *Authorization Code Flow*(**ACF**)[drshrey, 2013]: Este método é o adequado para aplicações que correm durante longos períodos de tempo e onde o utilizador só se deve autenticar uma única vez. É com este método que se acede e gere a informação de um utilizador autenticado.
- *Implicit Grand Flow*: Usado por aplicações implementadas somente com *JavaScript* e que vão correr no browser.

Todas as aplicações que requisitam acesso a APIs que implementam OAuth2 devem ter atributos que as identifiquem univocamente. Isto é feito para que o serviço que disponibiliza a API consiga medir os pedidos e saiba que aplicação anda a fazer o quê. Estes atributos são um **Client ID**, e um **Client Secret**. É com estes atributos que vão ser feitas as autenticações dos clientes e da aplicação.

Neste projeto os métodos mais utilizados são o **CCF** e o **ACF**. O **CCF** é mais básico de usar - a aplicação requisita um *access token* com o qual vai fazer pedidos. Ao fazer isto, mais um limite maior para fazer pedidos do que se não o tivesse. Isto é útil para o caso em que se queira fazer extração de dados de músicas em *bulk*, ou seja, num grande grande conjunto.

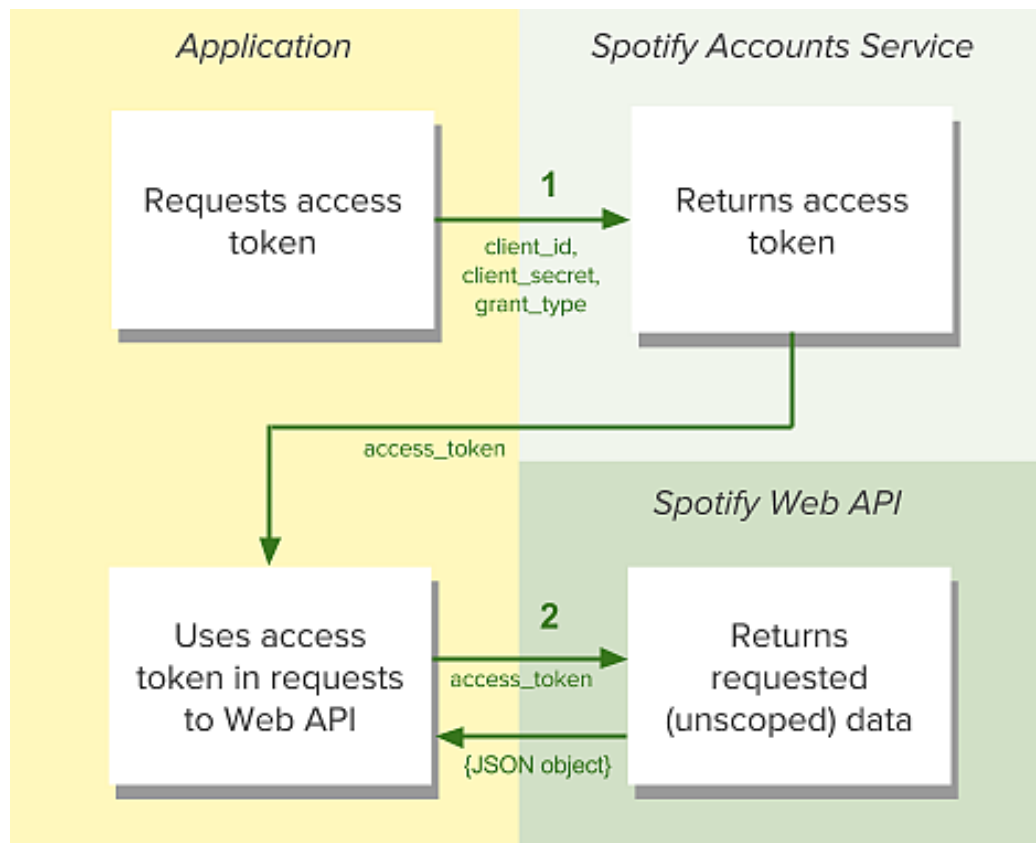


Figura A.1: Diagrama do método *Client Credentials Flow*.

O **ACF** é utilizado para aceder à informação de utilizadores após eles se autenticarem na aplicação que requer o acesso. O diagrama na figura A.2 ilustra os quais os passos a seguir para conseguir aceder à informação de cada utilizador.

Quando uma aplicação pede acesso a um utilizador, tem de especificar quais os dados que vai requerir o acesso, pode-se considerar isto o equivalente às permissões de uma aplicação móvel. Isto chama-se o *scope*. Uma aplicação que tente aceder a informação fora do seu *scope* não a vai receber.

Quando um utilizador fazer a autenticação numa aplicação, a aplicação recebe um *access token*, um *refresh token* e a data e hora de expiração do *access token*. Quando uma aplicação tenta aceder à informação de um utilizador através de um *access token* já expirado, não vai receber a informação pedida. Neste caso, é preciso atualizar o *access token*. Isto é feito fazendo um pedido de atualização utilizando o *refresh token* disponibilizado quando o utilizar se autentica pela primeira vez.

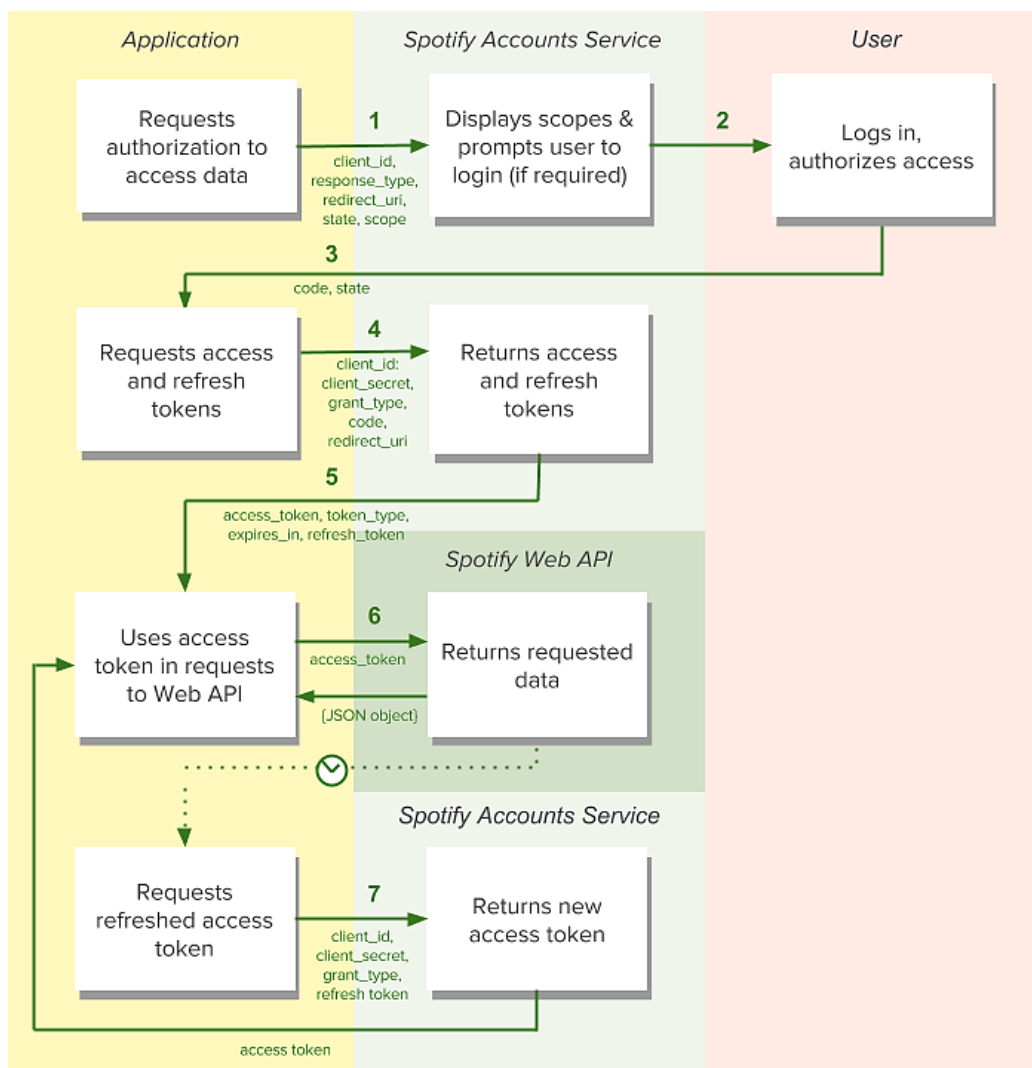


Figura A.2: Diagrama do método *Authorization Code Flow*.

Bibliografia

- [Anicas, 2014] Anicas, M. (2014). An introduction to oauth2. <https://www.digitalocean.com/community/tutorials/an-introduction-to-oauth-2>.
- [Aviles, 2016] Aviles, K. (2016). Android: A ble scanner app. https://www.youtube.com/channel/UCF7ybfgeryzKaQVSc8UM_g.
- [Bluetooth SIG, 2017] Bluetooth SIG, I. (2017). Gatt xml. <https://www.bluetooth.com/specifications/gatt>.
- [Daniel Ek, 2007] Daniel Ek, M. L. (2007). Spotify. <https://www.spotify.com/>.
- [drshrey, 2013] drshrey (2013). Acf walkthrough. <https://github.com/drshrey/spotify-flask-auth-example>.
- [GenMyModel, 2013] GenMyModel (2013). Modeling platform. <https://www.genmymodel.com/>.
- [Google,] Google. Google samples github repository. <https://github.com/googlesamples>.
- [Google e Alliance, 2015] Google e Alliance, O. H. (2015). Android api guide. <https://developer.android.com/guide/index.html>.
- [Gupt, 2014] Gupt, M. (2014). Android service and broadcastreceiver example. <http://www.truiton.com/2014/09/android-service-broadcastreceiver-example/>.
- [InfoQ, 2013] InfoQ (2013). Developing bluetooth smart applications for android tutorial. <https://www.youtube.com/watch?v=x1y4tEHDwk0>.

- [Koelsch e Jäncke, 2015] Koelsch, S. e Jäncke, L. (2015). Music and the heart. *European Heart Journal Advance Access*.
- [Lamere, 2014] Lamere, P. (2014). Spotipy library. <https://spotipy.readthedocs.io/en/latest/>.
- [M. e S.J., 2015] M., S. e S.J., X. (2015). Design and implementation of long-term single-lead ecg monitor. In *Journal of Biosciences and Medicines*, volume 3, p. 18–23. Strategic Decision Group.
- [Moore, 2014] Moore, J. (2014). Heart rate variability vs. heart rate. <https://hrvcourse.com/heart-rate-variability-vs-heart-rate/>.
- [Postdot Technologies, 2012] Postdot Technologies, I. (2012). Postman. <https://www.getpostman.com/>.
- [Python3.6,] Python3.6. Python programming language. <https://docs.python.org/3.6/>.
- [Ronacher, 2010] Ronacher, A. (2010). Flask. <http://flask.pocoo.org/>.
- [Semiconductor,] Semiconductor, N. Nordic semiconductor’s official github repository. <https://github.com/NordicSemiconductor>.
- [Spotify, a] Spotify. Interactive api console. <https://developer.spotify.com/web-api/console/>.
- [Spotify, b] Spotify. Spotify developer. <https://developer.spotify.com/>.
- [Spotify, c] Spotify. Web api tutorial. <https://developer.spotify.com/web-api/tutorial/>.
- [Square, 2013] Square, I. (2013). Retrofit. <http://square.github.io/retrofit/>.
- [Studio, 2014] Studio, T. A. (2014). Android bluetooth low energy tutorial. <http://toastdroid.com/2014/09/22/android-bluetooth-low-energy-tutorial/>.

-
- [Wang e Huang, 2014] Wang, H.-M. e Huang, S.-C. (2014). Musical rhythms affect heart rate variability: Algorithm and models. Technical report, The Department of Electrical Engineering, National Chiao Tung University, 1001 University Road, Hsinchu 30010, Taiwan.