Music Buddies

Computação Móvel - Flutter

Universidade de Aveiro

Grupo 3

Pedro Mateus - 88858

Samuel Duarte - 89222

Motivação

A motivação para este tema vem da paixão de ambos elementos pela música aliada a um conhecimento e experiência prévia da API do Spotify. A ideia era criar uma aplicação que permitisse ao utilizador partilhar as suas músicas rapidamente num encontro físico ao invés de enviar links por mensagens, isto de forma a promover o contacto entre pessoas que as redes sociais nos retiraram. A API do Spotify juntamente com pacotes de Flutter tornaram a integração de música fácil e cómoda.

Requisitos e funcionalidades desenvolvidas

Na fase de brainstorming do projeto foi pensado em encontrar pessoas que estavam a usar a app em simultâneo e detectá-las através de bluetooth para de seguida marcar encontros. No entanto e após um momento de reflexão concluímos que ter uma *third-party application* a usufruir de dados do spotify não revelava muita praticidade. Então pensamos que seria mais prático ter uma forma fácil e rápida de disponibilizar as músicas num caso de encontro, através de *QR Codes.*

Posto isto, as funcionalidades desenvolvidas são então:

- ☐ *Log-In* com a conta do Spotify;
- ☐ Google Maps integrado na App;
- ☐ Disponibilização das playlists e álbuns do utilizador e de playlists de topo do Spotify;
- ☐ Geração de *qr codes* para todas as playlists e álbuns presentes na aplicação;
- ☐ Leitor de *qr codes* integrado na App;
- ☐ *Player* integrado na aplicação para reprodução dos *qr codes* lidos (com ligação e controlo do *player* oficial do Spotify).

Tendo em conta os pensamentos iniciais das primeiras *milestones* foram tomadas decisões de remover algumas funcionalidades e tecnologias (bluetooth), apesar de a aplicação ficar mais reduzida também ficou mais simples e intuitiva, bem como mais fluida. Sendo assim, ficamos contentes com o resultado mesmo tendo cortado algumas das decisões iniciais.

Arquitetura e opções técnicas

The widget tree rationale

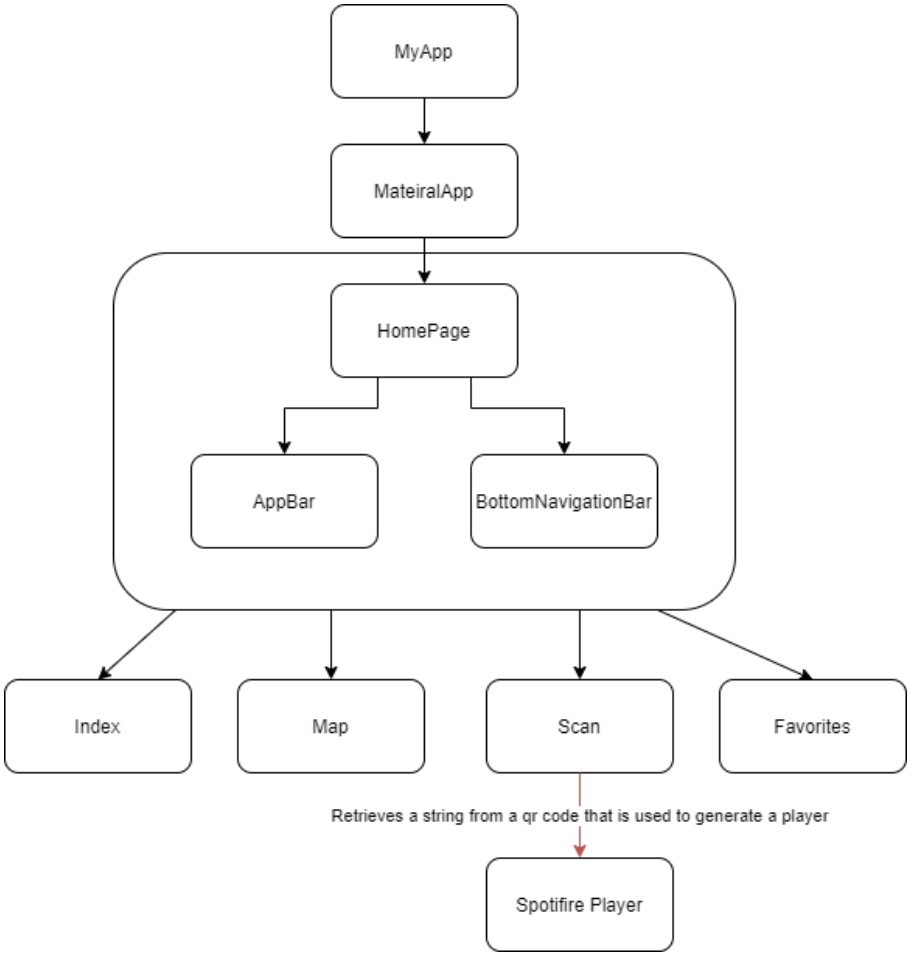
A aplicação é constituída por 5 páginas:

1. página inicial (index.dart);
2. página do mapa (map.dart);
3. página de *scan* de *qr codes* (scan.dart);
4. página dos favoritos (favorites.dart);
5. página do *player* que apenas está acessível após leitura de um *qr code* (spotifire.dart).

Para além destes ficheiros .dart são ainda usados mais três para concluir a navegação dentro da aplicação:

1. ficheiro global.dart, inclui apenas funções de pedidos à API do Spotify, esta página é de seguida importada pelas outras que fazem uso das funções;
2. ficheiro home_page.dart, inclui a AppBar com o nome de cada página e foto de utilizador e a barra de navegaçáo no fundo do ecrã. Todas as páginas da aplicação fazem uso desta página sendo esta o “esqueleto”;
3. ficheiro main.dart, ficheiro que contém a *main* do projeto onde é a *Hive* é instanciada, onde é dada a ordem de correr a App e apenas chama a HomePage.

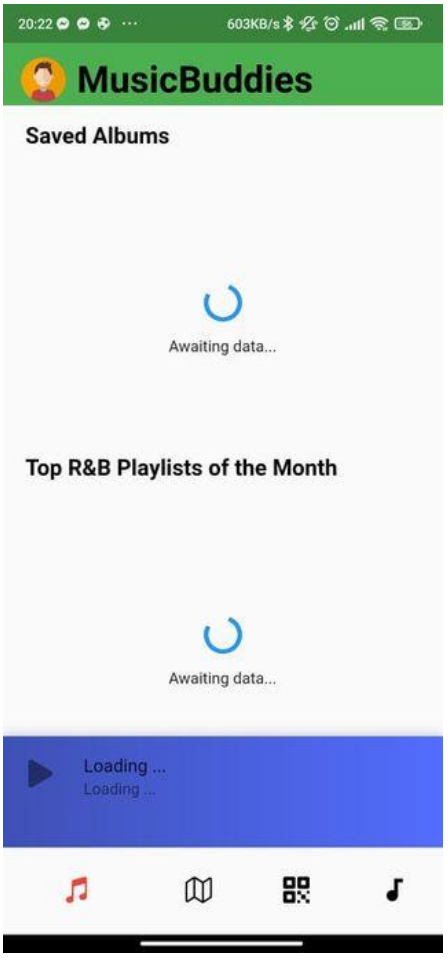
Sobra apenas um ficheiro .dart, sendo este o favorite_model.dart (que de seguida gera o ficheiro favorite_model.g.dart), aqui é feita a declaração da classe das músicas favoritas para posterior inserção na box ‘favorites’ da *Hive.*



Cenário de suporte desacoplado

Para o correto funcionamento da aplicação foram feitas alterações na pasta Android, não só para a API do Spotify como para a API da Google para o funcionamento do mapa. Sendo assim pressuposto que a aplicação não funcionará num dispositivo IOS pois as alterações correspondentes teriam que ser feitas na pasta ios.

Caso a aplicação seja usada sem acesso a internet esta está preparada para continuar a funcionar sem erros, apesar de que não ter qualquer tipo de dados apresentando apenas CircularProgressIndicator() até que volte a ter conexão, tal como pode ser visto na seguinte imagem:



Soluções de gestão de dados

A aplicação apenas possui persistência referente a cada dispositivo através do pacote *Hive.* O utilizador pode adicionar músicas favoritas a partir do player integrado da aplicação, com isto pode depois visualizar as músicas adicionadas numa página dedicada e consultar o nome da música, o álbum a que corresponde e o dia em que adicionou aos favoritos, para além disto pode também remover dos favoritos.

Isto porque os dados que são disponibilizados são provenientes da API do Spotify vindo assim de pedidos feitos na inicialização da aplicação ou recursos leves gerados em runtime (*qr codes*). A atualização de dados durante o uso da aplicação era possível mas seriam pedidos à API do Spotify (PUT/DELETE) e isso seria redundante com o que já estava a ser feito no intuito curricular.

Fontes/sensores externos

Esta aplicação usa:

- ☐ API do Spotify para disponibilizar dados (existem pacotes Flutter para este propósito);
- ☐ API da Google para disponibilizar um mapa fluido integrado na aplicação;
- ☐ Pacote não oficial do Spotify que replica um *player* para que o utilizador possa reproduzir música dentro da aplicação **(força a aplicação a não ter null-safety),** este *player* liga-se ao do Spotify permitindo ao utilizador controlar a música a partir da sua conta em qualquer dispositivo (televisão, computador, etc...);
- ☐ Faz uso da câmara do telemóvel para leitura de *qr codes;*

Objetivos concluídos e dificuldades

Conseguimos com sucesso fazer uma aplicação com forte integração da aplicação Spotify com casos de uso intuitivos de partilha de conteúdo. Apesar de simples, foram usados vários conceitos de Flutter tanto no lado estético da aplicação como no *back-end.* As maiores dificuldades apresentadas pelo projeto foram:

- o facto de os dados da aplicação terem como origem uma API exterior, isto leva a problemas de não ter os dados imediatamente disponíveis implicando um uso forte da funcionalidade ***Future*** do Flutter;
- garantir que a aplicação não retornava erros caso acontecesse algo com a API (como aconteceu numa das aulas) ou caso o utilizador levantasse outros problemas como não ter acesso à internet, não dar as autorizações necessárias para o funcionamento correto da aplicação, entre outros;

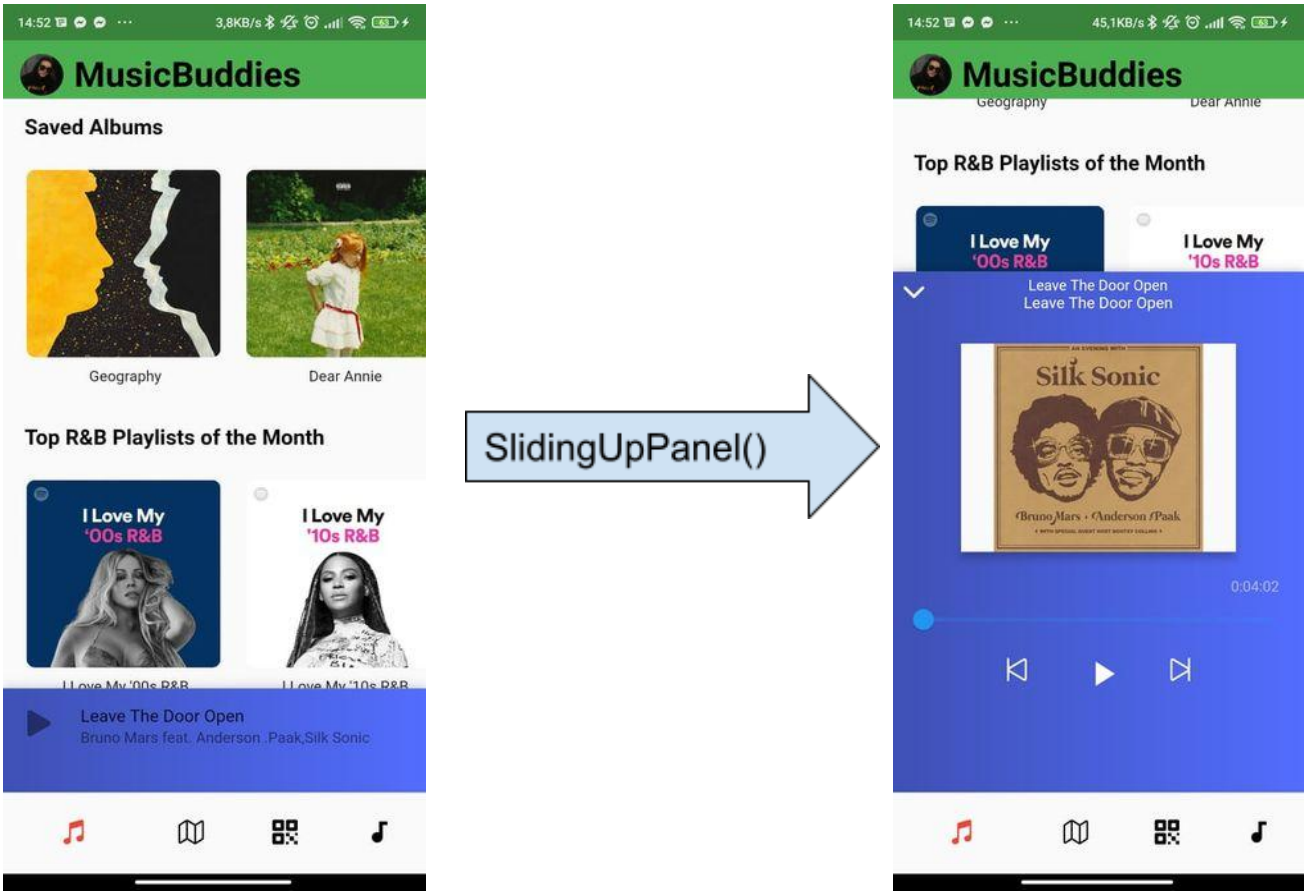
- a *Hive* não funcionar corretamente com classes que podem ser *nullable.* Da experiência que tivemos, apesar de esta permitir que se guardem objetos *nullable* estão são convertidos internamente para não *nullable,* sendo que apesar de guardados corretamente quando os dados são retirados de novo causa incoerência o que leva a erros (isto já usando os *TypeAdapters* que deveriam agilizar este processo);
- as documentações de alguns destes pacotes (como o *player*) não contêm muitos exemplos práticos para além de exemplos gerais o que levou a um requerimento maior de tempo despendido de forma a entender como funcionar com os mesmos.

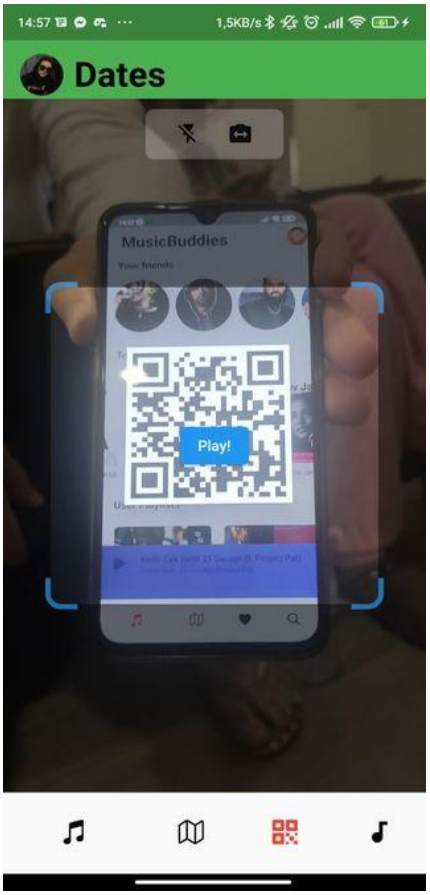
Contribuições

Ambos os membros contribuíram de forma igual para o projeto, **50%**. O membro Pedro Valente trabalhou na integração do mapa através da API da Google, na integração do *player* interno da aplicação e em *Hive* para gestão de dados. O membro Samuel Duarte trabalhou no uso da API do Spotify, desde *Log-In,* geração de *tokens* para os pedidos e manipulação dos dados retirados. Ambos os membros entenderam e trabalharam de forma igual nos conceitos básicos e não tão básicos de Flutter bem como no *design* da aplicação.

Manual

Interações de algumas das páginas, as páginas com funcionamento mais “óbvio” não estão aqui demonstradas.





A partir do qr code gerado pela aplicação é construída uma página com o player

Adicionar aos favoritos