



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

Habemus Festa



Davide Gouveia (41541)

Tiago Silva (43965)

Orientador

Professor Doutor Pedro Fazenda

Julho, 2021

Resumo

Socialização é a assimilação de hábitos que caracterizam o indivíduo ao seu grupo social. É o processo através do qual o indivíduo se torna membro funcional de uma comunidade, assimilando a sua cultura. É um processo contínuo que é realizado através da comunicação.

A comunicação interpessoal pode ser o maior pesadelo para pessoas que sofrem de timidez e/ou introversão. Para muitas destas pessoas, as redes sociais acabam por ser uma forma de contornar este “obstáculo”.

Há mais de uma década que as redes sociais têm vindo a crescer e a tornar-se parte do dia-a-dia da maioria das pessoas. Coloca-se a questão de que o tipo de relação estabelecida pessoalmente é semelhante ao tipo de relação estabelecida virtualmente. O contato visual é fundamental para o processo de empatia e para a capacidade de relacionamento interpessoal.

Atualmente existem eventos que promovem as relações interpessoais. Nas empresas, quando organizam festas de verão, jantares de natal, bootcamps, etc. Em algumas festas académicas eram entregues aos alunos e às alunas, parafusos e porcas (de diferentes tamanhos), respetivamente. Era uma forma de incentivar ao contato entre os alunos com o pretexto de verificarem se eram “compatíveis” com os parafusos/porcas que cada um tinha.

Desta forma, surge a aplicação Habemus Festa, uma aplicação móvel que permite ao utilizador visualizar os vários eventos que irão decorrendo ao longo do ano e que incentiva o mesmo a comunicar com pessoas novas, através de um sistema de recompensas motivado pelo relacionamento interpessoal.

Abstract

Socialization is the assimilation of habits that characterize the individual to their social group. It is the process through which the individual becomes a functional member of a community, assimilating its culture. It is an ongoing process that is carried out through communication.

Interpersonal communication can be the biggest nightmare for people who suffer from shyness and/or introversion. For many of these people, social networks end up being a way to get around this “obstacle”.

For over a decade, social media has been growing and becoming part of most people’s daily lives. The question arises that the type of relationship established personally is similar to the type of relationship established virtually. Eye contact is fundamental to the process of empathy and the ability to interpersonal relationships.

There are currently events that promote interpersonal relationships. In companies, when they organize summer parties, Christmas dinners, boot-camps, etc. At some academic parties, screws and nuts (of different sizes) were given to students, respectively. It was a way to encourage contact between students with the pretext of checking if they were “compatible” with the screws/nuts that each one had.

Thus, the Habemus Festa application appears, a mobile application that allows users to view the various events that will take place throughout the year and encourages them to communicate with new people, through a reward system motivated by interpersonal relationships.

Agradecimentos

A realização deste projeto final de curso contou com um apoio e incentivo bastante importantes, aos quais o grupo gostaria deixar os seguintes agradecimentos.

Ao nosso orientador Professor Pedro Fazenda, por aceitar a nossa proposta de projeto, pelo tempo dispendido, paciência e disponibilidade durante todo este percurso.

Eu, Tiago Silva, agradeço aos meus pais, pelo apoio que me prestaram durante todo o meu percurso académico, aos meus amigos, pelas suas sugestões e a ajuda constante que me deram ao longo do desenvolvimento do projeto e por fim ao meu colega e amigo Davide Gouveia, por me ter apresentado esta ideia de projeto, que penso continuar o seu desenvolvimento no futuro.

Eu, Davide Gouveia, agradeço aos meus pais e à minha namorada, por todo o apoio prestado, não apenas durante o projeto, mas por todo o percurso académico até à presente data. Quero também agradecer ao meu colega e amigo Tiago Silva por participar comigo neste projeto, que pela nossa perspectiva foi alcançado com sucesso.

Índice

Resumo	i
Abstract	iii
Agradecimentos	v
Índice	vii
Lista de Tabelas	ix
Lista de Figuras	xi
Lista de Acrónimos	xv
1 Introdução	1
2 Arquitetura e Tecnologias Utilizadas	5
2.1 Firebase	6
2.2 API	7
2.3 Geofire	7
2.4 Código QR	7
2.4.1 QRGenerator	7
2.4.2 Code Scanner	8
2.5 PhotoView	8
2.6 Glide	8
3 Modelo Proposto	9
3.1 Requisitos	9
3.1.1 Requisitos Funcionais	9

3.1.2 Requisitos Não Funcionais	10
3.2 Casos de utilização	10
3.3 Modelo Iniciais	11
4 Implementação do Modelo	15
4.1 Autenticação e Registo	15
4.1.1 Email e Password	15
4.1.2 Google Sign-in	18
4.2 Definições	20
4.3 Utilizador	20
4.3.1 Sistema de localização GPS	21
4.3.2 QR Codes	25
4.3.3 Listar Eventos	32
4.3.4 Perfil	47
4.3.5 Terminar Sessão	53
4.4 Colaborador	54
4.4.1 Criar/Remover Eventos	54
4.4.2 Criar/Remover Produtos	65
4.4.3 Sistema de Transação	69
4.4.4 Adicionar Colaboradores	72
4.4.5 Lista de Transações	75
5 Validação e Testes	77
5.1 Contexto	77
5.2 Participantes	78
5.3 Teste da aplicação	81
5.4 Respostas dos inquiridos	82
6 Conclusões e Trabalho Futuro	85
Bibliografia	87
A Formulário de Usabilidade	1
B Modelo EA RealTime Database do Firebase	11

Lista de Tabelas

3.1 Requisitos funcionais	10
3.2 Requisitos não funcionais	10

Lista de Figuras

2.1	Diagrama de blocos	6
3.1	Mockups de acesso geral	11
3.2	Mockups de acesso do utilizador	12
3.4	Mockups de acesso do colaborador	14
4.1	Registo por email	16
4.2	Ecrã principal	17
4.3	Definições	20
4.4	”Eventos-Users” no RealTime Database	22
4.5	Entrada no evento	24
4.6	Código QR do utilizador	26
4.7	Scan código QR	27
4.8	”Links” no RealTime Database	29
4.9	Listagem dos eventos mais próximos do utilizador	33
4.10	Página do evento	41
4.12	Página do utilizador	48
4.13	Código QR do utilizador	49
4.14	Caixa de diálogo do ”Merge with email”	50
4.15	Link da conta com o Google Sign In	53
4.16	Criar Evento	55
4.17	Imagen do evento carregada	57
4.18	”tags” no RealTime Database	57
4.19	Escolha da data e hora do evento	58
4.20	Remover Evento	62
4.21	Criar Produto	66
4.22	”Produtos” no RealTime Database	67

4.23 Pasta dos eventos no Storage	67
4.24 Remover Produto	68
4.26 Adicionar Colaboradores	72
5.1 Intervalo de idades dos participantes	78
5.2 Género dos participantes	78
5.3 Estado cívil dos participantes	79
5.4 Tempo de utilização do telemóvel	80
5.5 Frequência de “saídas”	80
5.6 Tipo de eventos	80
5.7 Criar novo registo	81
5.8 Iniciar sessão com novo registo	81

Lista de Código

1	Pedido do IdToken para o utilizador	18
2	Criação da credencial e autenticação do utilizador	18
3	Utilização do Geofire	22
4	Verificação da entrada de um utilizador num evento	23
5	Geração de um código QR	25
6	Validação do código QR examinado	28
7	Validação do scan entre utilizadores	30
8	Creditação de pontos	31
9	Obtenção dos eventos mais próximos do utilizador	33
10	Criação do marcador dentro do mapa	35
11	Definição do <i>listener</i> do marcador	36
12	Obtenção dos dez eventos com mais presenças registadas	37
13	Obtenção dos dez eventos mais próximos de se realizarem	39
14	Registo da presença ou não presença do utilizador num determinado evento	41
15	Obtenção dos eventos que o utilizador confirmou a sua presença	43
16	Filtragem dos eventos	45
17	método changeFilter(String tag) dos tipos de pesquisa	46
18	método hideMarkers(String tag) do <i>SearchMap</i>	46
19	Merge with Email	51
20	Terminar sessão	54
21	Criação do Intent e inicialização da atividade	55
22	OnActivityResult	56
23	Validação dos pârametros das datas	59
24	Criação do <i>SearchMap</i> e da visualização do mapa da Google	60
25	Listener de queries da barra de pesquisa	61
26	Listener de queries da barra de pesquisa	63

27	Validação do código QR examinado	70
28	Validação do colaborador na base de dados	72

Lista de Acrónimos

API Interface de Programação de Aplicações

SDK Kit de desenvolvimento de software

QR Quick Response

SMS short message Service

GIF Graphics Interchange Format

URL Uniform Resource Locator

GPS Global Positioning System

URI Uniform Resource Identifier

DB Base de dados

ID Identificador único

JPEG/JPG Joint Photographic Experts Group

PNG Portable Network Graphics

BMP Bitmap

APK Android Package

FEIM Fórum de Engenharia Informática e Multimédia

ISEL Instituto Superior de Engenharia de Lisboa

LEIM Licenciatura de Engenharia Informática e Multimédia

Capítulo 1

Introdução

Inicialmente, quando surgiu a ideia do projeto Habemus Festa, o grupo investigou acerca de outras aplicações idênticas, ou até mesmo relacionadas com o mesmo propósito. Existem aplicações que apresentam uma listagem de eventos a decorrer num determinado período de tempo ou lugar, mas atualmente na grande maioria das vezes são divulgados através das redes sociais. Relativamente a aplicações relacionadas com este aspeto, é atualmente dominado pelas redes sociais. Desta forma, pensamos não haver uma aplicação diretamente relacionada com aquela que o grupo se propôs a desenvolver, visto conseguir “juntar” estes dois pontos fortes.

O principal foco deste projeto assenta na ideia de promover o relacionamento interpessoal. Desta forma, a aplicação apresenta um sistema de visualização de eventos sociais, como por exemplo concertos, festivais de música, eventos académicos, etc. e, a sua respetiva calendarização. A pesquisa destes eventos é realizada a partir da visualização diretamente da API “Google Maps”, onde também está dividido por secções - “Mais próximos”, “Tendências”, “Por data” e “Meus eventos”.

De forma a promover o aspetto anteriormente mencionado, o relacionamento interpessoal, a aplicação motiva os utilizadores a interagir entre si de forma presencial durante o decorrer destes eventos, através da implementação de um sistema de pontos que permitirá ao utilizador usufruir como “moeda de troca”. Estes pontos são obtidos de duas formas: aquando o utilizador assiste ao evento a sua entrada no recinto é validada, e ser-lhe-ão atribuídos um determinado número de pontos; através da leitura do código QR pessoal entre utilizadores. Esta troca atribui pontos às duas entidades em questão,

sendo esta só permitida apenas uma única vez. Com a obtenção destes pontos, é possível aos utilizadores trocá-los por produtos selecionados em pontos de venda que patrocinem a aplicação. Para a troca ser realizada, o utilizador apenas tem de apresentar o seu código QR na aplicação, e o colaborador efetua uma leitura desse mesmo código retirando assim os respetivos pontos.

Acerca do colaborador, a aplicação está dividida em duas vertentes de utilização: O utilizador, como referido anteriormente e sendo entendido como qualquer indivíduo que tenha efetuado registo na aplicação; e o colaborador, sendo aquele que é pertencente à organização de um ou mais evento(s). Este pode criar/remover eventos, criar/remover produtos, retirar pontos a um utilizador, adicionar novos colaboradores e verificar o histórico de transações efetuadas.

Qualquer utilizador que tenha serviço para venda de consumo no evento e queira patrocinar a aplicação, deverá entrar em contato com os proprietários da aplicação de forma a obter o primeiro registo enquanto colaborador. Qualquer outro colaborador que possa vir a ser adicionado a este patrocinador, poderá ser criado por si. A troca de pontos estará presente para utilização em todos os serviços patrocinadores da aplicação no respetivo evento, tais como comidas, bebidas e merchandising. A acumulação de pontos é ilimitada e serão válidos para cada evento, querendo dizer com isto que poderão também ser transportados para o evento seguinte acumulando oportunidade de troca por qualquer um dos serviços anteriormente descritos.

Num evento não é mandatório que todos os colaboradores estejam agregados ao serviço da nossa aplicação, pelo que poderão encontrar colaboradores com produtos passíveis de troca de pontos e outros com troca de pontos e com pagamento de valor acrescido.

Todo e qualquer serviço de colaborador que não pretenda fazer parte da troca de pontos, não poderá ser aceite como colaborador oficial da Habemus Festa. Sendo a aplicação desenvolvida em foro de processo escolar, não tem qualquer tipo de legislação ainda agregada visto tratar-se de um projeto académico de fim de curso.

Ao longo deste documento serão apresentados todos os objetivos, processos, tecnologias utilizadas, e testes realizados. De modo a facilitar a sua leitura. Segue a estruturação do mesmo:

No capítulo 2, é mencionado o trabalho relacionado com o nosso projeto.

A implementação das diferentes ferramentas e bibliotecas externas para diferentes implementações.

No capítulo 3, o modelo proposto é apresentado, contendo uma análise dos requisitos funcionais e não funcionais da aplicação, e os casos de utilização. Conta também com os modelos iniciais da aplicação, ou *Mockups*.

O capítulo 4 descreve a implementação do projeto organizado por módulos, onde cada secção corresponde às componentes principais.

No capítulo 5, são realizados os testes de validação da aplicação.

Por fim, no capítulo 6, são apresentadas as conclusões obtidas pelo grupo de trabalho, e possível futuro da aplicação.

Capítulo 2

Arquitetura e Tecnologias Utilizadas

O desenvolvimento da aplicação foi realizado sob a plataforma *Android* através do *Android Studio* [*Android Studio, 2013*], e pela plataforma *Firebase* [*Firebase, 2010*], ambas pertencentes à *Google*. Para controlo e gestão de versões da aplicação, foi utilizada a plataforma *GitHub* [*Tom Preston-Werner, 2008*].

No entanto, ao longo do desenvolvimento do projeto foram utilizadas várias ferramentas e bibliotecas externas de forma a serem criadas as várias funcionalidades pretendidas para a aplicação. Neste capítulo iremos detalhar todas as bibliotecas e API, Interface de Programação de Aplicações, utilizadas na aplicação.



Figura 2.1: Diagrama de blocos

2.1 Firebase

Firebase [2.1] é uma plataforma desenvolvida pela *Google* para a criação de aplicações móveis e para a web. Esta plataforma permite ligar um projeto de *Firebase* a um projeto de *Android Studio*, fazendo a adição do *SDK* básico do *Firebase*. Dos múltiplos tipos de autenticação disponíveis no *Firebase* escolhemos o *email* e o da *Google*, para o utilizador ter acesso a mais opções de escolha para o tipo de autenticação que pretende utilizar.

2.2 API

API, é um conjunto de rotinas e padrões estabelecidos por um *software* para a utilização das suas funcionalidades por aplicativos que não pretendem envolver-se em detalhes da implementação do *software* e apenas querem usar os seus serviços. Neste projeto fez-se uso de uma API da *Google*, *Places*[*Google Places, 2015*], *Google Maps*[*Google Maps, 2013*] e *Google Sign-In*[*Google Sign-In, 2013*], que permite obter informação dos vários locais dos mapas da *Google* e também utilizarmos um outro método de autenticação.

2.3 Geofire

Geofire[*Geofire, 2014*] é uma biblioteca *open-source* para *Java* que permite guardar e fazer *queries* a um conjunto de chaves associadas a localizações geográficas. Esta biblioteca guarda e faz *queries* da informação das localizações no *RealTime Database* do *Firebase*, de forma rápida e eficiente.

2.4 Código QR

Um código QR [2.4] é um tipo de código de barras bidimensional utilizado maioritariamente pelos dispositivos móveis. Um código de barras é uma etiqueta óptica legível por máquina que contém informação do objeto ao qual está anexado. Um código QR pode ser convertido em texto, um endereço URI, um número de telefone, uma localização georreferenciada, um e-mail, um contato ou um SMS. Decidimos utilizar códigos QR na nossa aplicação devido à sua utilidade nos dispositivos móveis.

Para criar e utilizar um código QR, utilizámos duas bibliotecas: *QRGenerator* e *Code Scanner*.

2.4.1 QRGenerator

QRGenerator[*QRGenerator, 2016*] é uma biblioteca para *Android* que permite criar códigos QR e guardá-los como imagens, tornando o processo de visualização destes numa aplicação *Android* mais eficiente.

2.4.2 Code Scanner

Code Scanner[*Code Scanner, 2017*] é uma biblioteca para *Android*, baseada na biblioteca *Zxing*[*Zxing, 2018*] da *Google*, que permite fazer o *scan* de códigos QR para obter a sua informação. Esta biblioteca serve para complementar a anterior, de forma a pudermos criar e gerar códigos QR na aplicação.

2.5 PhotoView

PhotoView[*Photo View, 2018*] é uma biblioteca *open-source* para *Android* que permite o utilizador fazer *zoom* de uma imagem na aplicação. Esta funcionalidade será usada porventura nas imagens (panfletos) dos eventos, que permitirá ao utilizador uma melhor visualização dos mesmos.

2.6 Glide

Glide[*Glide, 2013*] é uma *framework open-source* de carregamento de imagens e gestão de mídia para *Android* que envolve descodificação de mídia, armazenamento em cache de memória e disco e pool de recursos numa interface simples e fácil de usar. *Glide* permite realizar a procura, decodificação e exibição de fotos, imagens e ficheiros GIF animados. Esta *framework* irá permitir o carregamento de imagens da *FirebaseStorage* do *Firebase*, através de um endereço URL.

Capítulo 3

Modelo Proposto

De forma a conseguir desenvolver um modelo capaz de satisfazer os requisitos, é necessário apontá-los e analisá-los previamente. Desta forma, neste terceiro capítulo iremos abordar o problema de modo a transmitir uma visão clara sobre a motivação do modelo proposto. Segue-se então uma análise dos requisitos funcionais e não-funcionais da aplicação, secção 3.1, juntamente com os casos de utilização, secção 3.2. Para terminar, apresentamos os modelos iniciais, *Mockups*, secção 3.3.

3.1 Requisitos

Requisitos são descrições das necessidades ou propósitos de um produto, que neste caso, uma aplicação móvel para a plataforma *Android*. É um conjunto de condições que se dividem em funcionais, e não funcionais.

3.1.1 Requisitos Funcionais

Requisitos funcionais descrevem sucintamente os objetivos que a aplicação pretende cumprir. Aqueles que o utilizador consegue identificar na aplicação, e que na sua perspetiva, o sistema faz.

Função	Categoria
Registo e login	Evidente
Listar eventos	Evidente
Filtrar eventos	Evidente
Criar/Remover eventos	Evidente
Scan de códigos QR	Evidente
Troca de pontos	Evidente
Inserir/Remover produtos	Evidente

Tabela 3.1: Requisitos funcionais

3.1.2 Requisitos Não Funcionais

Requisitos não funcionais descrevem outros parâmetros necessários ao funcionamento da aplicação, não fazendo parte do objetivo geral da mesma.

Função	Categoria
Suporte de diferentes tipos de utilizadores	Invisível
Confirmação de localização por GPS	Invisível
Criação de códigos QR	Invisível
Sistema de recompensas	Invisível
Base de dados que suporte a aplicação	Invisível
Comunicação da aplicação com uma base de dados	Invisível

Tabela 3.2: Requisitos não funcionais

3.2 Casos de utilização

O caso de utilização é um tipo de classificador que representa uma unidade funcional coerente pelo sistema com um ou mais atores. No contexto do nosso projeto, o caso principal de utilização é quando um utilizador inicia a aplicação, regista-se, ou inicia sessão, e consegue verificar todos os eventos possíveis. Ou numa outra situação, em que o colaborador inicia sessão, cria um novo evento e posteriormente associa produtos a esse mesmo evento. São ambos exemplos de casos de utilização, praticados por utilizadores com níveis de acesso diferentes.

3.3 Modelo Iniciais

Inicialmente, antes da implementação da aplicação, de forma a ser possível planear toda a estrutura da interface gráfica, foram criados esboços ou Mockups através de um *software* - *Mockplus Classic* [Mockplus Classic, 2019]. Um *Mockup* é um modelo de um projeto que, neste caso é utilizado para a demonstração do *design* que se pretende alcançar futuramente no desenvolvimento da aplicação. Assim sendo, encontra-se abaixo na figura 3.1 os *mockups* do ecrã inicial da aplicação, definições e registo de novo utilizador.

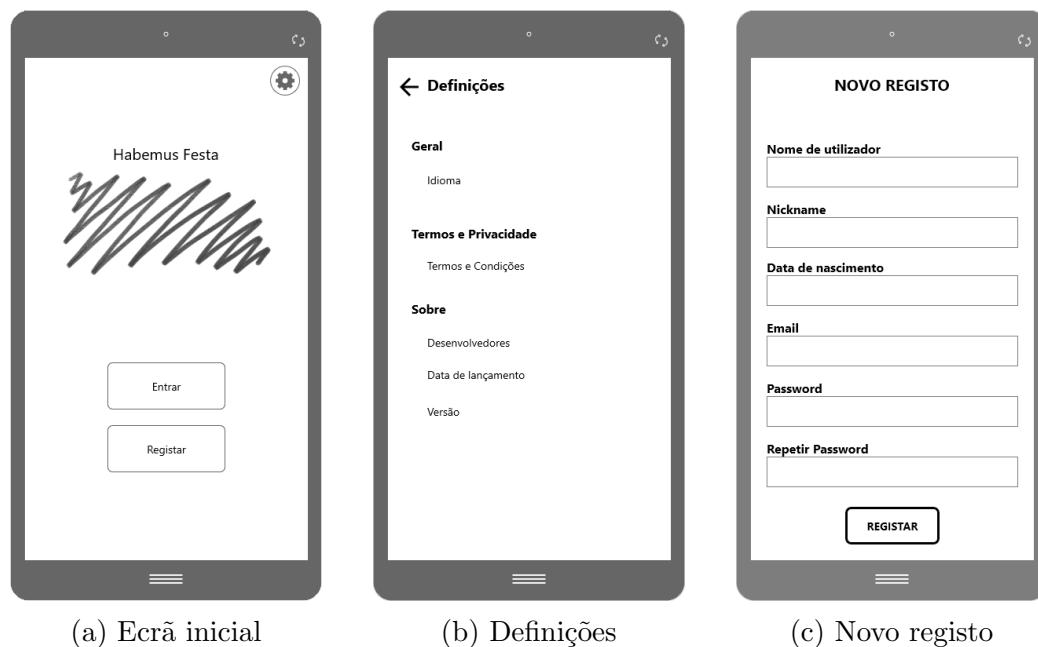


Figura 3.1: Mockups de acesso geral

Na figura seguinte (3.2), podemos observar o ecrã de navegação de utilizador após inicio de sessão. Juntamente com este menu, estão presentes todas as opções que o utilizador pode tomar a partir do mesmo, sendo estes:

- Listagem de eventos
- Scan de utilizadores
- Ver código QR
- Perfil
- Definições - o mesmo que o anterior (figura 3.1(b))
- Terminar sessão - redireciona para o ecrã principal



Figura 3.2: Mockups de acesso do utilizador

De seguida, observamos um caso semelhante, mas desta vez com o inicio de sessão por parte de um colaborador, onde podemos notar que após o inicio de sessão, o menu de navegação apresenta opções diferentes daquelas que o utilizador obtém, sendo estas:

- Novo evento
- Novo produto
- Remover Evento
- Remover produto
- Retirar pontos
- Transações
- Definições - o mesmo que o anterior (figura 3.1(b))
- Terminar sessão - redireciona para o ecrã principal



(a) Menu de colaborador

(b) Criar evento

(c) Novo produto

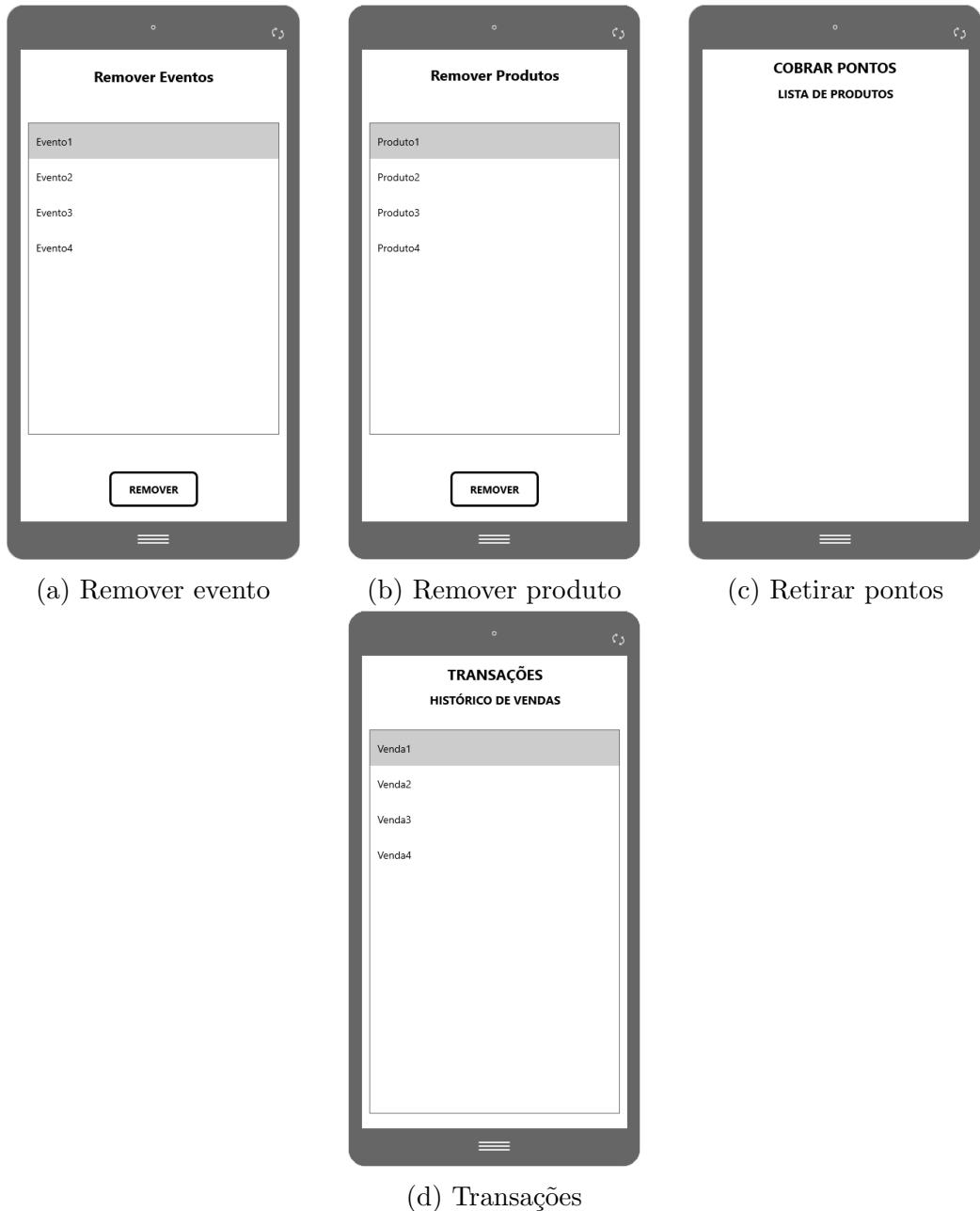


Figura 3.4: Mockups de acesso do colaborador

Capítulo 4

Implementação do Modelo

Neste capítulo iremos desenvolver a forma de como foram implementadas as várias funcionalidades do projeto, tal como fazer uma análise de qual as melhores escolhas para a implementação destas. Na secção 4.1, iremos detalhar a forma como foi feito o registo e autenticação na aplicação, na secção 4.2, as definições da aplicação, na secção 4.3, as várias funcionalidades do utilizador e por fim, na secção 4.4, as várias funcionalidades do colaborador.

4.1 Autenticação e Registo

Nesta secção iremos abordar os dois tipos de registo e *login* utilizados na aplicação: Email e Password e Google Sign-in.

4.1.1 Email e Password

O *FirebaseAuth* é uma classe do *Firebase* que cria um *entry point* com o sistema de autenticação *Firebase*, como também nos permite ter acesso à informação do utilizador corrente. Esta classe irá ser utilizada não só no registo e autenticação como também nas outras funcionalidades da aplicação.

No processo de registo do utilizador são pedidos os seguintes dados ao utilizador: nome, *username*, data de nascimento, *email* e *password*.

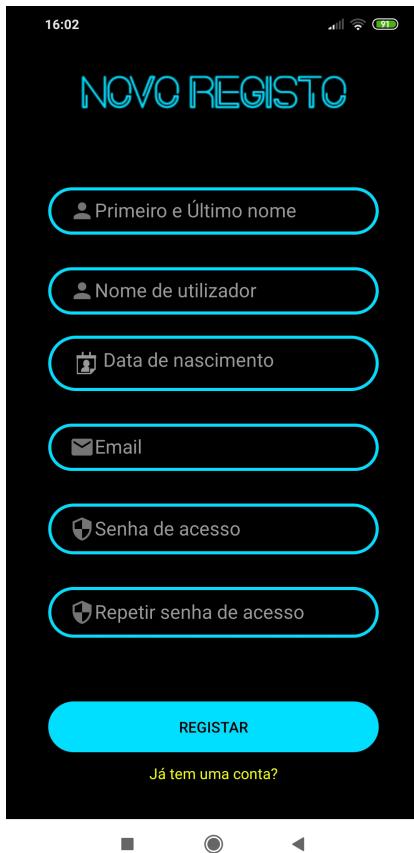


Figura 4.1: Registo por email

O registo via *email* e *password* é feito através do método `createUserWithEmailAndPassword(email, password)` da classe `FirebaseAuth`, que faz uma tentativa de registo de uma nova conta de utilizador com o *email* e *password* providenciados. Caso o registo seja realizado com sucesso, irão ser guardados os dados introduzidos neste processo de registo dentro de uma referência no *RealTime Database* do *Firebase*.

As restrições implementadas no processo de registo são:

- nome do utilizador não pode ter mais de 255 caracteres;
- *username* tem de ter entre 3 a 12 caracteres;
- *email* e data de nascimento obrigatórios e têm de ser válidos;
- *password* tem de ter entre 6 a 15 caracteres;

A autenticação é feita através do método `signUserWithEmailAndPassword(email, password)` da classe `FirebaseAuth`, que faz uma tentativa de autenticação do utilizador na aplicação com recurso ao `email` e `password` providenciados. Caso consiga, irá ser redirecionado para uma destas páginas: **HomeUserLogin**, caso seja um utilizador normal; **HomeAdminLogin**, caso seja um colaborador.



Figura 4.2: Ecrã principal

4.1.2 Google Sign-in

Os processos de registo e login são efetuados no *MainActivity*, com recurso a 3 classes do *Google Sign-in*: *GoogleSignInOptions*, *GoogleSignInClient* e *GoogleSignInAccount*.

O método *signInWithCredential(credential)* da classe *FirebaseAuth* é utilizado para autenticar o utilizador na aplicação, com recurso a uma credencial. Para obtermos esta credencial precisamos de um *token* providenciado pela API *Google Sign In*.

No código seguinte, começamos por declarar uma variável do tipo *GoogleSignInOptions* para pudermos obter o *token* ”**IdToken**”, e de seguida, criamos um cliente que interage com a API *Google Sign In*.

Código 1: Pedido do IdToken para o utilizador

```
//Google Sign-In
mGoogleSignInOptions = new GoogleSignInOptions
    .Builder(GoogleSignInOptions.DEFAULT_SIGN_IN)
    .requestIdToken(getString(R.string.google_web_client_id))
    .requestEmail()
    .build();
mGoogleSignInClient = GoogleSignIn.getClient(this,
    mGoogleSignInOptions);
```

Após o processo de autenticação e validação de dados feito pela própria API da *Google*, criamos a credencial e autenticamos o utilizador na aplicação. Caso o utilizador ainda não esteja registado no *Firebase*, este irá ser automaticamente adicionado e autenticado.

Código 2: Criação da credencial e autenticação do utilizador

```
AuthCredential credential = GoogleAuthProvider.getCredential(
    idToken, null);

mAuth.signInWithCredential(credential)
    .addOnCompleteListener(this, new
        OnCompleteListener<AuthResult>() {
            @Override
```

```
public void onComplete(@NonNull Task<
    AuthResult> task) {
    if (task.isSuccessful()) {
        // Sign in success, update UI with
        // the signed-in user's information
        Log.d(TAG, "signInWithCredential:
            success");

        mGoogleSignInAccount = GoogleSignIn.
            getLastSignedInAccount(
            MainActivity.this);
        // Signed in successfully.
        String email = mGoogleSignInAccount.
            getEmail();
        ...
    }
}
```

4.2 Definições

As definições, tanto para o utilizador como para o colaborador, são idênticas. A alteração do idioma da aplicação está adaptado ao mesmo idioma do sistema do dispositivo. Neste caso, a aplicação está adaptada às linguas português e inglês. É também possível verificar as políticas de privacidade, desenvolvedores e versão da aplicação.

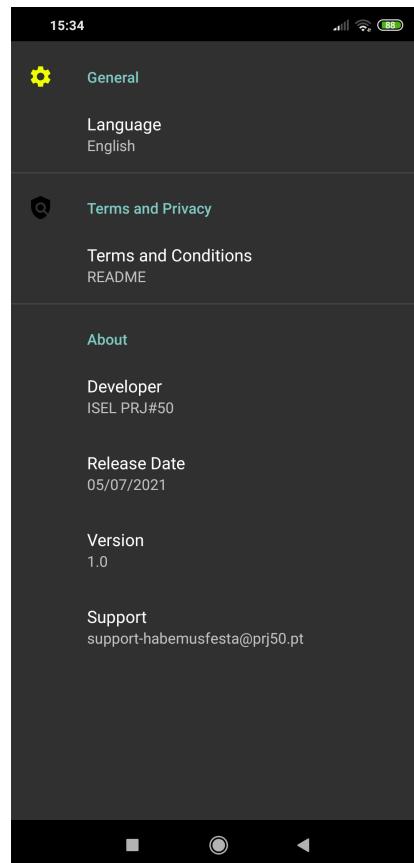


Figura 4.3: Definições

4.3 Utilizador

Nesta secção iremos abordar a forma como foram implementadas as funcionalidades do utilizador, começando pelo sistema de localização GPS do utilizador, o scan de códigos qr, a listagem de eventos, o perfil do utilizador e o *logout* da aplicação.

4.3.1 Sistema de localização GPS

A nossa ideia inicial foi criar um sistema que conseguisse detetar a localização geográfica do utilizador para confirmar a sua entrada nos eventos e também para verificar, no caso de *scan* de códigos QR, se ambos os utilizadores estavam dentro do evento ou não. Para tal, fez-se uso da classe *IntentService*.

IntentService

IntentService é uma classe que estende de *Service*, proveniente das bibliotecas do *Android*, e que tem por objetivo realizar um serviço pedido pela aplicação. Este serviço corre dentro de uma *Worker Thread*, separada da *Main Thread*, que não entra em conflito com nenhuma das outras funcionalidades da aplicação. Dentro deste serviço iremos então criar o processo de localização da posição do utilizador. Para localizarmos o utilizador via GPS, temos de criar um cliente (*FusedLocationProviderClient*) que interaja com a API *Fused Location Provider*[*Fused Location Provider, 2012*] da *Google*.

Através desta API iremos conseguir obter dados referentes à localização do utilizador, tais como o endereço da rua, a latitude e a longitude. Feita a localização do utilizador, iremos verificar se o utilizador se encontra dentro de um evento ou não.

Para tal, utilizamos a biblioteca *Geofire*[*2.3*] que, como foi mencionada nos capítulos anteriores, permite guardar e fazer *queries* a um conjunto de chaves associadas a localizações geográficas. Cada chave irá corresponder ao identificador único de cada evento.

Geofire

A classe *Geofire* dispõe de dois métodos cruciais para a verificação de entrada nos eventos: *queryAtLocation()* e *addGeoQueryEventListener()*.

queryAtLocation(geoLocation, radius) é um método que permite fazer *queries* à *RealTime Database* do *Firebase*, com base na localização do utilizador (*geoLocation*) e um determinado raio de ação (*radius*).

addGeoQueryEventListener() é um método do tipo *listener* que nos indica quais os eventos que se encontram dentro do raio de ação definido anteriormente.

O exemplo seguinte ilustra o processo de utilização destes métodos.

Código 3: Utilização do Geofire

```

DatabaseReference ref = FirebaseDatabase.getInstance().
    getReference("Eventos-Locs");
geoFire = new GeoFire(ref);

GeoQuery geoQuery = geoFire.queryAtLocation(new GeoLocation(
    currentUser.getLocation().getLatitude(), currentUser.getLocation().getLongitude()
), 0.6f); //600m

geoQuery.addGeoQueryEventListener(new GeoQueryEventListener() {
    @Override
    public void onKeyEntered(String key, GeoLocation location)
    {
        ...
    }
})
}

```

Com o método `onKeyEntered(String key, GeoLocation location)` do `listener` obtemos o identificador único (`key`) e a localização geográfica de cada evento (`location`), para de seguida podermos confirmar se o utilizador já entrou no evento ou não, pois o processo de entrada num evento credita 10 pontos na conta do utilizador, e como nós só queremos que isso aconteça 1 vez por evento, temos de confirmar na base de dados.

No *RealTime Database* temos definido um nó (**Eventos-Users**) que guarda todas as entradas dos utilizadores nos vários eventos. A figura seguinte ilustra a forma como este nó está definido na base de dados.

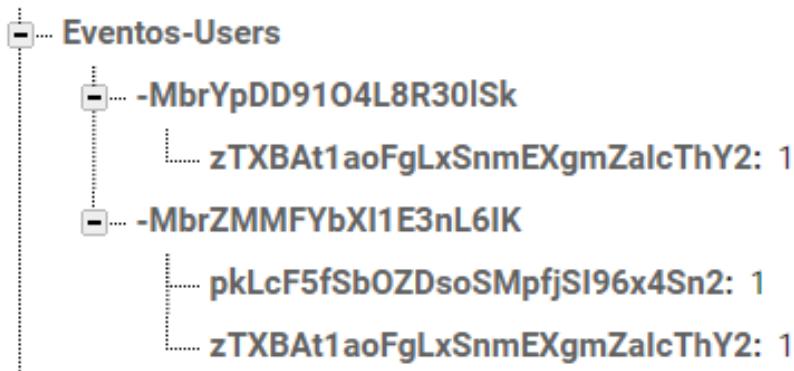


Figura 4.4: "Eventos-Users" no RealTime Database

O primeiro ramo do nó corresponde ao identificador único de cada evento e o segundo corresponde ao identificador único de cada utilizador.

Para verificar se o utilizador já entrou no evento ou não, basta adicionar um *listener* no ramo correspondente ao identificador único do evento, como se pode observar no exemplo seguinte.

Código 4: Verificação da entrada de um utilizador num evento

```
refEventosUsers = FirebaseDatabase.getInstance().getReference()
    .child("Eventos-Users").child(key).child(mAuth.getUid());
valueEventListener = new ValueEventListener(){
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot
    ) {
        if(shouldContinue) {
            if (snapshot.exists()) {
                shouldContinue = false;
                Log.d("GPS", "User already entered the event"
                    );
            } else {
                FirebaseDatabase.getInstance().
                    getReference("Eventos").child(key).
                    addListenerForSingleValueEvent(new
                    ValueEventListener() {
                        @Override
                        public void onDataChange(@NonNull
                            DataSnapshot snapshot) {
                            if (snapshot.exists()) {
                                Event e = snapshot.getValue(
                                    Event.class);
                                Log.d("GPS", e.getNome());
                                showEventDialog(e);
                            }
                        }
                    ...
                }
            }
        ...
    }
}
```

Caso o utilizador já tenha entrado no evento, o serviço pára, pois não necessita de estar constantemente a verificar a sua posição. Caso contrário, serão creditados 10 pontos na conta do utilizador que será escrito na base de dados um ramo semelhante ao visto anteriormente.



Figura 4.5: Entrada no evento

4.3.2 QR Codes

Para a utilização de códigos QR na aplicação dispomos de 2 bibliotecas: **QR Code Generator** e **Code Scanner**.

QR Code Generator

QR Code Generator é uma biblioteca para *Android* que nos permite criar códigos QR e guardá-los como imagens. A classe *QRGEncoder* cria um código QR com a informação da variável ”**uid**”. Esta variável (**uid**) corresponde ao identificador único de cada utilizador. De seguida utilizamos o método *getBitmap()* do *QRGEncoder* para obtermos o *Bitmap* do código QR gerado. Por fim, resta só definir o *Bitmap* no *ImageView* para podermos mostrar o código QR na aplicação. A figura seguinte ilustra a forma de como são gerados.

Código 5: Geração de um código QR

```
private static void generateQRCode(ImageView qrCodeImage,
    String uid, int width){

    QRGEncoder qrgEncoder = new QRGEncoder(uid, null,
        QRGContents.Type.TEXT, width);
    try {
        // Getting QR-Code as Bitmap
        Bitmap bitmap = qrgEncoder.getBitmap();
        // Setting Bitmap to ImageView
        qrCodeImage.setImageBitmap(bitmap);
    } catch (IllegalArgumentException e) {
        Log.v(TAG_QR_CODE, e.toString());
    }
}
```

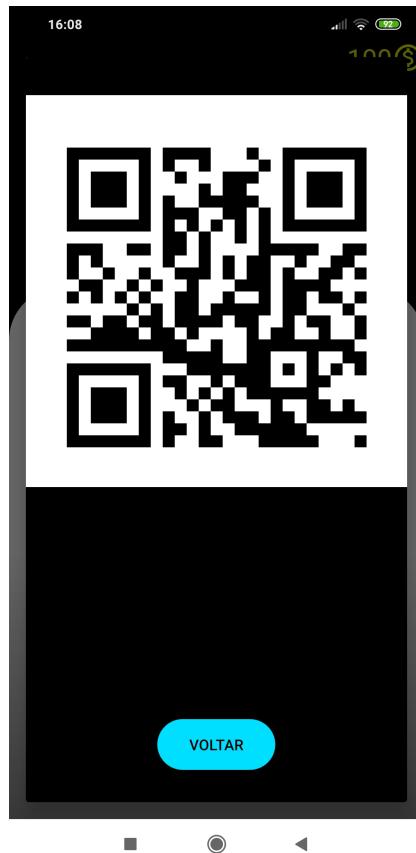


Figura 4.6: Código QR do utilizador

Code Scanner

Code Scanner é uma biblioteca para *Android* que permite ao utilizador fazer *scan* de um código QR. A informação contida num código QR pode ser convertida em texto, um endereço URI, um número de telefone, uma localização georreferenciada, um *e-mail*, um contato ou um SMS. Na nossa aplicação os códigos QR são convertidos em *Strings*.

Para o utilizador fazer *scan* de um código QR, tem de primeiro permitir o uso de câmera na aplicação. Sem esta permissão a funcionalidade não pode ser usada.

Caso tenha permitido o uso de câmera, a aplicação irá abrir a câmera do telemóvel, e o utilizador tem de simplesmente apontar a câmera para o código QR que quer examinar.



Figura 4.7: Scan código QR

O processo de examinação do código QR passa por três validações. A primeira consiste em saber se ambos os utilizadores estão no mesmo evento. A segunda se o código QR examinado é válido. E a terceira se o *scan* deste código QR já foi feito pelo utilizador previamente.

Para termos acesso à localização do utilizador que fez o scan do código QR, utilizamos o serviço descrito na secção Sistema de localização GPS[4.3.1].

Porventura, não basta ter apenas a confirmação da localização do utilizador no evento, precisamos também de arranjar maneira de haver comunicação entre o *IntentService* e a *Activity* corrente. Para tal, utilizamos a classe *BroadcastReceiver*.

BroadcastReceiver é uma classe que recebe e trata de mensagens (*intents*) que são enviadas tanto pelas *Activity* como pelos *IntentService*. As mensagens trocadas entre eles são do tipo *Intent*, em que iremos introduzir o *id* do

utilizador, o *id* do código qr examinado e o *id* do evento em que o utilizador está.

Agora, para validar a localização do utilizador ao qual pertence o código QR, iremos verificar se o seu identificador único consta dentro do nó do evento no *RealTime Database* do *Firebase*. O exemplo seguinte ilustra a forma como fazemos esse processo.

Código 6: Validação do código QR examinado

```
private void checkIfUsersAreLinked(String uid_user, String
    uid_friend, String event_id, String date){

    DatabaseReference ref = FirebaseDatabase.getInstance().
        getReference();
    ref.child("Users").addSingleValueEventListener(new
        ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot snapshot) {
            if(snapshot.exists() && uid_friend != null) {
                if (snapshot.hasChild(uid_friend)) {
                    handleLink(event_id, uid_user, uid_friend,
                        date);
                } else {
                    Toast.makeText(HomeUserLogin.this, "Error:
                        This is not a valid QR code.",
                        Toast.LENGTH_SHORT).show();
                    return;
                }
            }
        }

        @Override
        public void onCancelled(@NonNull DatabaseError error) {
            Toast.makeText(HomeUserLogin.this, "Error: Please
                try again.",
                Toast.LENGTH_SHORT).show();
            return;
        }
    });
}
```

Se o identificador único constar na base de dados, significa que o código QR examinado é válido. Caso contrário, será apresentada uma mensagem na aplicação que informa o utilizador que o código QR não é válido.

Agora, basta apenas verificar se o utilizador já fez scan do código QR previamente.

Dado que o *scan* de um código QR credita 10 pontos na conta de cada utilizador envolvido, para evitar o abuso excessivo desta funcionalidade, a última validação irá impedir o utilizador de fazer *scan* deste código QR mais do que uma vez por festa.

Para tal, iremos examinar o nó na *RealTime Database* que guarda todas as ocorrências de *scan* de códigos QR (denominado de "Links"). Cada filho deste nó contém o *id* do evento, a data de quando ocorreu o *scan* do código e os *ids* dos utilizadores envolvidos. A figura seguinte ilustra como este nó se encontra definido na base de dados.



Figura 4.8: "Links" no RealTime Database

O identificador único de cada nó-filho é gerado através do método `push()` da classe `DatabaseReference`.

O código seguinte irá ilustrar a forma como é feita esta última validação.

Código 7: Validação do scan entre utilizadores

```
 DatabaseReference ref = FirebaseDatabase.getInstance().  
     getReference();  
 ref.child("Links").orderByChild("uid_user_1").equalTo(  
     first_user).addListenerForSingleValueEvent(new  
     ValueEventListener() {  
         @Override  
         public void onDataChange(@NonNull DataSnapshot snapshot) {  
             Link mLink;  
             if(snapshot.exists()) {  
                 for (DataSnapshot child : snapshot.getChildren()) {  
                     Link l = child.getValue(Link.class);  
                     if (l.getUid_user_2().equals(second_user) && l.  
                         getEvent_id().equals(event_id)) {  
                         Toast.makeText(HomeUserLogin.this, "Error:  
                             You have already linked in this event.",  
                             Toast.LENGTH_SHORT).show();  
                         return;  
                     }  
                 }  
             }  
             else {  
                 mLink = new Link(first_user, second_user, event_id,  
                     date);  
                 ref.child("Links").push().setValue(mLink);  
                 Toast.makeText(HomeUserLogin.this, "User linked!",  
                     Toast.LENGTH_SHORT).show();  
                 handleRewards(first_user, second_user);  
             }  
         }  
         @Override  
         public void onCancelled(@NonNull DatabaseError error) {  
             Toast.makeText(HomeUserLogin.this, "Error: Please try  
                 again.",  
                 Toast.LENGTH_SHORT).show();  
         }  
     });
```

Se o utilizador já tiver feito *scan* deste código QR, a aplicação irá informá-lo em como não pode fazer *scan* do mesmo código QR mais do que uma vez. Caso contrário, iremos proceder para a creditação de pontos na conta dos utilizadores envolvidos. O código seguinte ilustra este processo.

Código 8: Creditação de pontos

```
private void handleRewards(String first_user, String second_user){
    DatabaseReference ref = FirebaseDatabase.getInstance().
        getReference();
    ref.child("Users").child(first_user).
        addListenerForSingleValueEvent(new ValueEventListener()
    {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot
        ) {
            if(snapshot.exists()){
                User u = snapshot.getValue(User.class);
                int pontos = u.getPontos();
                pontos += 10;
                ref.child("Users").child(first_user).child("pontos").
                    setValue(pontos);
            }
        }
    });

    @Override
    public void onCancelled(@NonNull DatabaseError error)
    {}
});

ref.child("Users").child(second_user).
    addListenerForSingleValueEvent(new ValueEventListener()
{
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot
    ) {
        if(snapshot.exists()){
            User u = snapshot.getValue(User.class);
            int pontos = u.getPontos();
            pontos += 10;
```

```
        ref.child("Users").child(second_user).child("pontos").setValue(pontos);
    }
}

@Override
public void onCancelled(@NonNull DatabaseError error)
{
}
});
```

4.3.3 Listar Eventos

A listagem de eventos foi criada a partir de uma *Tabbed Activity*, que separa os diferentes tipos de pesquisa (“**Mais Próximos**”, “**Tendências**”, “**Por Data**” e “**Meus Eventos**”) por várias abas (*tabs*). A figura seguinte ilustra a visualização desta atividade, na aba “**Mais Próximos**”.

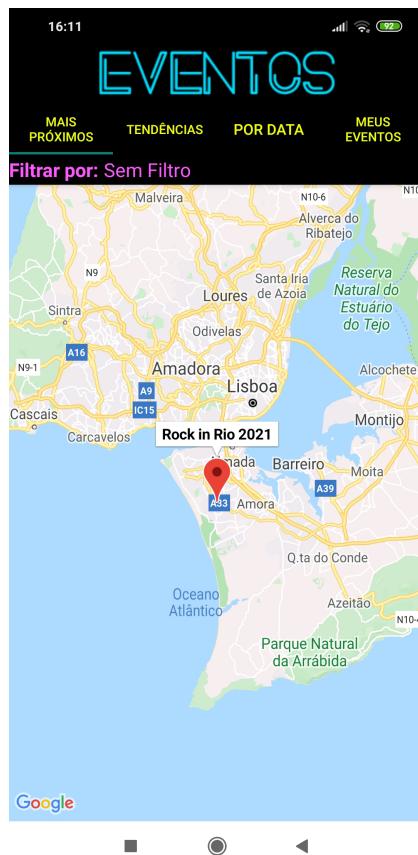


Figura 4.9: Listagem dos eventos mais próximos do utilizador

Estas abas porventura são *Fragments* que contêm um *FragmentLayout*, de forma a poder mostrar o mapa disponibilizado pela *API Google Maps*. Dentro de cada mapa irão ser adicionados marcadores nas localizações de cada evento.

Mais Próximos

Na aba "Mais Próximos" são mostrados os eventos que estão a cerca de 3 quilómetros (3km) do utilizador. Este procedimento é feito com os métodos *queryAtLocation(GeoLocation center, double radius)* e *addGeoQueryEventLister(final GeoQueryEventListener listener)* da classe *Geofire*, que foram descritos anteriormente na secção 4.3.1. O código seguinte ilustra este processo.

Código 9: Obtenção dos eventos mais próximos do utilizador

```
private void getNearestLocations(Double lat, Double lon){
    DatabaseReference ref = FirebaseDatabase.getInstance().
       getReference("Eventos-Locs");
    geoFire = new GeoFire(ref);
    geoQuery = geoFire.queryAtLocation(new GeoLocation(lat, lon
        ), 3.0f); //3000m
    geoQuery.addGeoQueryEventListener(new GeoQueryEventListener()
    {
        @Override
        public void onKeyEntered(String key, GeoLocation
            location) {
            DatabaseReference ref = FirebaseDatabase.getInstance
                ().getReference();
            ref.child("Eventos").child(key).
                addListenerForSingleValueEvent(new
                    ValueEventListener() {
                    @Override
                    public void onDataChange(@NonNull DataSnapshot
                        snapshot) {
                        if(snapshot.exists()){
                            Event e = snapshot.getValue(Event.class);
                            events.put(new String[]{key, e.getNome(),
                                e.getEvent_type()}, location);
                        }
                    }
                    @Override
                    public void onCancelled(@NonNull DatabaseError
                        error) {}
                });
            System.out.println(String.format("Key %s entered the
                search area at [%f,%f]", key, location.latitude,
                location.longitude));
        }
        ...
        @Override
        public void onGeoQueryReady() {
            searchMap = new SearchMap(true, currentTag, events);
            FragmentManager fragmentManager =
                getParentFragmentManager();
            fragmentManager.beginTransaction().replace(R.id.map,
```

```

        searchMap).commit();
        System.out.println("All initial data has been loaded
            and events have been
            fired!");
    }
    ...
});
}

```

O método *onKeyEntered(String key, GeoLocation location)* fornece o identificador único (*key*) e a localização (*location*) de cada evento que esteja no raio de 1 quilómetro (1km) do utilizador. O método *onGeoQueryReady()* é chamado quando o método anterior acabar, ou seja, quando os dados dos eventos forem todos carregados. Dentro deste método iremos inicializar uma variável *SearchMap* para poder mostrar os vários eventos já recolhidos num mapa fornecido pela *API Google Maps*.

No método *addLocation(String event_id, String event_title, String event_type, Double lat, Double lon)* do *SearchMap* definimos os marcadores que irão indicar a localização de cada evento dentro do mapa. O código seguinte ilustra este processo.

Código 10: Criação do marcador dentro do mapa

```

public void addLocation(String event_id, String event_title,
    String event_type, Double lat, Double lon){
    LatLng latLng = new LatLng(lat, lon);
    Marker marker = map.addMarker(new MarkerOptions().position(
        latLng));
    marker.setTag(new String[]{event_id, event_type});
    marker.setTitle(event_title);
    mapMarkers.add(marker);
    map.animateCamera(CameraUpdateFactory.newLatLngZoom(latLng
        ,10));
}

```

Também é definido um título e uma *tag* em cada marcador. O título corresponderá ao nome do evento enquanto que a *tag* é um *array* que irá conter o identificador único do evento e o tipo de evento. Com o método *setOnMarkerClickListener(GoogleMap.OnMarkerClickListener() listener)* do *GoogleMap*, o utilizador ao pressionar um marcador no mapa é redirecionado para a página do evento correspondente. O código seguinte ilustra este processo.

Código 11: Definição do *listener* do marcador

```
map.setOnMarkerClickListener(new GoogleMap.  
    OnMarkerClickListener() {  
        @Override  
        public boolean onMarkerClick(Marker marker) {  
            if(marker.isVisible()) {  
                if (isInfoWindowShown && ((String[]) currentMarker.  
                    getTag())[0].equals(((String[]) marker.getTag())[0])) {  
                    if (marker.getTag() != null) {  
                        Intent intent = new Intent(getActivity(),  
                            EventPage.class);  
                        String[] tags = (String[]) marker.getTag();  
                        intent.putExtra("event_id", tags[0]);  
                        isInfoWindowShown = false;  
                        startActivity(intent);  
                        return true;  
                    }  
                } else {  
                    marker.showInfoWindow();  
                    currentMarker = marker;  
                    isInfoWindowShown = true;  
                }  
            }  
            return true;  
        }  
    }  
});
```

Tendências

A aba "Tendências" mostra os dez (10) eventos com mais presenças confirmadas até ao momento. Este número de presenças aumenta quando um utilizador prime o botão "Vou" dentro da página do evento.

O código seguinte ilustra o processo de obtenção dos dez eventos com mais presenças registadas.

Código 12: Obtenção dos dez eventos com mais presenças registadas

```
private void getTrendingEvents() { //its going to select the
    top 10 most voted events
    DatabaseReference ref = FirebaseDatabase.getInstance().
       getReference();
    ref.child("Eventos").orderByChild("likes").limitToLast(10).
        addListenerForSingleValueEvent(new ValueEventListener()
    {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot
        ) {
            if (snapshot.exists()) {
                events_size = snapshot.getChildrenCount();
                for (DataSnapshot child : snapshot.getChildren()
                ) {
                    Event e = child.getValue(Event.class);
                    geoFire = new GeoFire(ref.child("Eventos-Locs
                    "));
                    LocationCallback locationCallback = new
                        LocationCallback() {
                        @Override
                        public void onLocationResult(String key,
                            GeoLocation location) {
                            events.put(new String[]{key, e.
                                getNome(), e.getEvent_type()},
                                location);
                            count++;
                            if(count == events_size){
                                count = 0L;
                                searchMap = new SearchMap(true,
                                    currentTag, events);
                                FragmentManager fragmentManager =
                                    getParentFragmentManager();
                            }
                        }
                    };
                }
            }
        }
    });
}
```

```
        fragmentManager.beginTransaction()
            .replace(R.id.map_2, searchMap)
            .commit();
    }
}
@Override
public void onCancelled(DatabaseError
    databaseError) {
    Toast.makeText(getActivity().
        getApplicationContext(), R.string.
        events_location_not_found, Toast.
        LENGTH_SHORT);
}
);
geoFire.getLocation(e.getEvent_id(),
    locationCallback);
}
}
}
@Override
public void onCancelled(@NonNull DatabaseError error)
{}
```

O método `getLocation(String key, LocationCallback callback)` do `Geofire` obtém as coordenadas geográficas de um dado evento (`key`) que estão dentro do `RealTime Database` do `Firebase`. Após obtermos toda a informação destes dez eventos iremos inicializar uma variável `SearchMap`, sendo este processo idêntico ao da secção anterior.

Por Data

”Por Data” mostra os dez (10) eventos mais próximos de se realizarem, ou seja, irá ordenar os eventos pela sua data de início. O código seguinte ilustra este processo.

Código 13: Obtenção dos dez eventos mais próximos de se realizarem

```
private void getMostRecentEvents() { //its going to select the
    top 10 closest upcoming events
    DatabaseReference ref = FirebaseDatabase.getInstance().
        getReference();
    ref.child("Eventos").orderByChild("data_inicial").
        limitToLast(10).addListenerForSingleValueEvent(new
        ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot
        ) {
            if (snapshot.exists()) {
                events_size = snapshot.getChildrenCount();
                for (DataSnapshot child : snapshot.getChildren()
                ) {
                    Event e = child.getValue(Event.class);
                    geoFire = new GeoFire(ref.child("Eventos-Locs
                    "));
                    LocationCallback locationCallback = new
                    LocationCallback() {
                    @Override
                    public void onLocationResult(String key,
                        GeoLocation location) {
                        events.put(new String[]{key,e.getNome
                            (), e.getEvent_type()}, location);
                        count++;
                        if(count == events_size){
                            count = 0L;
                            searchMap = new SearchMap(true,
                                currentTag, events);
                            FragmentManager fragmentManager =
                                getParentFragmentManager();
                            fragmentManager.beginTransaction()
                                .replace(R.id.map_3, searchMap)
                                .commit();
                        }
                    }
                    @Override
                    public void onCancelled(DatabaseError
                        databaseError) {
```

```
        Toast.makeText(getApplicationContext(),
            R.string.events_location_not_found, Toast.
            LENGTH_SHORT);
    }
};

geoFire.getLocation(e.getEvent_id(),
    locationCallback);
}
}

@Override
public void onCancelled(@NonNull DatabaseError error)
{}


});
```

O processo de obtenção da localização de cada evento e a visualização destes num mapa é igual ao da secção anterior.

Meus Eventos

E por fim, ”Meus Eventos” mostra os eventos no qual o utilizador confirmou a sua presença, ou seja, premiu o botão ”Vou” na página do evento. Antes de passarmos para a explicação de como esta secção funciona, iremos explicar a funcionalidade dos botões na página do evento.



Figura 4.10: Página do evento

A página do evento dispõe de dois botões, "Vou" e "Não Vou". Estes ao serem premidos fazem com que a aplicação guarde a presença ou a não presença do utilizador nesse evento. O código seguinte ilustra este processo.

Código 14: Registo da presença ou não presença do utilizador num determinado evento

```
private void handleLikeSystem(String reactionType) {
    Intent givenIntent = getIntent();
    DatabaseReference ref = FirebaseDatabase.getInstance().
       getReference();
    ref.child("Eventos").child(givenIntent.getStringExtra(
        "event_id")).addListenerForSingleValueEvent(new
        ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot
        ) {
```

```

        if (snapshot.exists()) {
            Event e = snapshot.getValue(Event.class);
            int likes = e.getLikes();
            int value = 0;
            switch (reactionType) {
                case "ADD":
                    likes += 1;
                    value = 1;
                    yesBtn.setClickable(false);
                    noBtn.setClickable(true);
                    break;
                case "REMOVE":
                    likes = likes == 0 ? 0 : likes - 1;
                    value = -1;
                    yesBtn.setClickable(true);
                    noBtn.setClickable(false);
                    break;
                default:
                    break;
            }
            e.setLikes(likes);
            ref.child("Eventos").child(givenIntent.
                getStringExtra("event_id")).setValue(e);
            ref.child("Users-Likes").child(mAuth.getUid()).
                child(givenIntent.getStringExtra("event_id"))
                .setValue(value);
            likesTxt.setText(Integer.toString(likes));
        }
    }
    @Override
    public void onCancelled(@NonNull DatabaseError error)
    {}
});
}
}

```

No fim deste processo guardamos uma referência na base de dados em como o utilizador confirma a sua presença ($value = 1$) ou a sua não presença ($value = -1$) num determinado evento. Com isto em mente, dentro de "Meus Eventos" iremos visualizar na base de dados quais os eventos que o utilizador

confirmou a sua presença (*value = 1*). O código seguinte ilustra este processo.

Código 15: Obtenção dos eventos que o utilizador confirmou a sua presença

```
private void getMyEvents() { //its going to select all the
    confirmed user events
DatabaseReference ref = FirebaseDatabase.getInstance().
    getReference();
ref.child("Users-Likes").orderByChild(mAuth.getUid()).
    addListenerForSingleValueEvent(new ValueEventListener() {
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
    if(snapshot.exists()){
        events_size = snapshot.getChildrenCount();
        for (DataSnapshot child : snapshot.getChildren()) {
            if((Long)child.getValue() == 1L) {
                ref.child("Eventos").child(child.getKey()).
                    addListenerForSingleValueEvent(new
                    ValueEventListener() {
@Override
public void onDataChange(@NonNull
                    DataSnapshot snapshot) {
                if (snapshot.exists()) {
                    Event e = snapshot.getValue(Event.
                        class);
                    geoFire = new GeoFire(ref.child(""
                        Eventos-Locs"));
                    LocationCallback locationCallback =
                        new LocationCallback() {
                    @Override
                    public void onLocationResult(
                        String key, GeoLocation
                        location) {
                        events.put(new String[]{key, e
                            .getNome(), e.getEvent_type
                            ()}, location);
                        count++;
                        if (count == events_size) {
                            count = 0L;
                            searchMap = new SearchMap(
                                true, currentTag,
                                events);
                        }
                    }
                }
            }
        }
    }
}
```


Como também nas secções anteriores, o seu processo de obtenção da localização de cada evento e a visualização destes num mapa é idêntico.

Filtrar Eventos

Os filtros com os tipos de evento foram disponibilizados na aplicação através da utilização de um *Spinner*. Ao selecionar um destes filtros, os eventos não correspondentes irão desaparecer do mapa, ficando escondidos até o utilizador selecionar o filtro correspondente. O código seguinte ilustra este processo.

Código 16: Filtragem dos eventos

```
viewPager.addOnPageChangeListener(new ViewPager.OnPageChangeListener() {
    @Override
    public void onPageScrolled(int position, float positionOffset, int positionOffsetPixels) {}

    @Override
    public void onPageSelected(int position) {
        Fragment f = sectionsPagerAdapter.getCurrentFragment();
        if(f != null) {
            switch (f.getClass().getSimpleName()) {
                case "NearByEvents":
                    ((NearByEvents) f).changeFilter(currentTag);
                    break;
                case "RecentEvents":
                    ((RecentEvents) f).changeFilter(currentTag);
                    break;
                case "TrendingEvents":
                    ((TrendingEvents) f).changeFilter(currentTag);
                    ;
                    break;
                case "MyEvents":
                    ((MyEvents) f).changeFilter(currentTag);
                    break;
                default:
                    System.out.println(f.getClass().getSimpleName());
                    break;
            }
        }
    }
})
```

```

    }
    sectionsPagerAdapter.setCurrentFilter(currentTag);
}
@Override
public void onPageScrollStateChanged(int state) {}
});

```

As classes de cada tipo de procura (“NearByEvents”, “RecentEvents”, “TrendingEvents”, “MyEvents”) contêm um método *changeFilter(String tag)*, que faz com que os marcadores dos eventos que não correspondem ao filtro selecionado fiquem escondidos dentro do mapa. Os códigos seguintes ilustram este processo.

Código 17: método *changeFilter(String tag)* dos tipos de pesquisa

```

public void changeFilter(String tag){
    if(searchMap != null) {
        this.currentTag = tag;
        searchMap.hideMarkers(tag);
    }
}

```

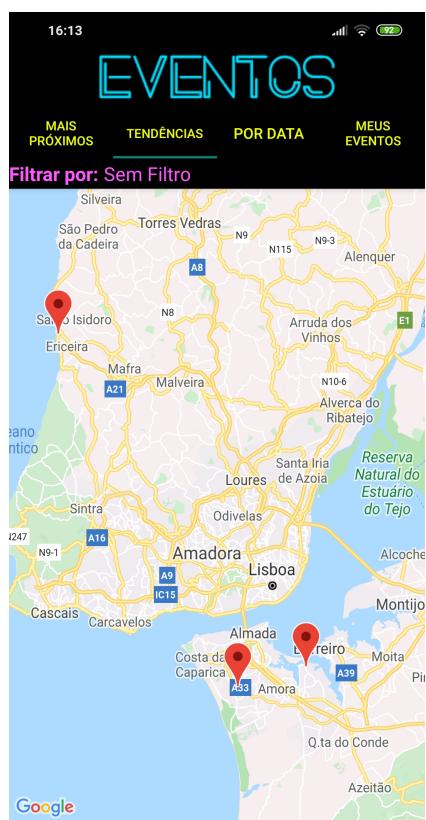
Código 18: método *hideMarkers(String tag)* do *SearchMap*

```

public void hideMarkers(String tag){
    for(Marker m: mapMarkers){
        String marker_tag = ((String[])m.getTag())[1];
        if(!marker_tag.equals(tag) && !tag.equals("0")){
            m.setVisible(false);
        }else{
            m.setVisible(true);
        }
    }
}

```

A figuras seguintes mostram o antes e o depois do uso do filtro ”Música”, no tipo de procura ”Tendências”.



(a) ”Tendências” sem filtro



(b) ”Tendências” com filtro ”Música”

4.3.4 Perfil

O perfil do utilizador mostra alguma da informação introduzida aquando do registo do utilizador, como o nome, *username*, data de nascimento e *email*.

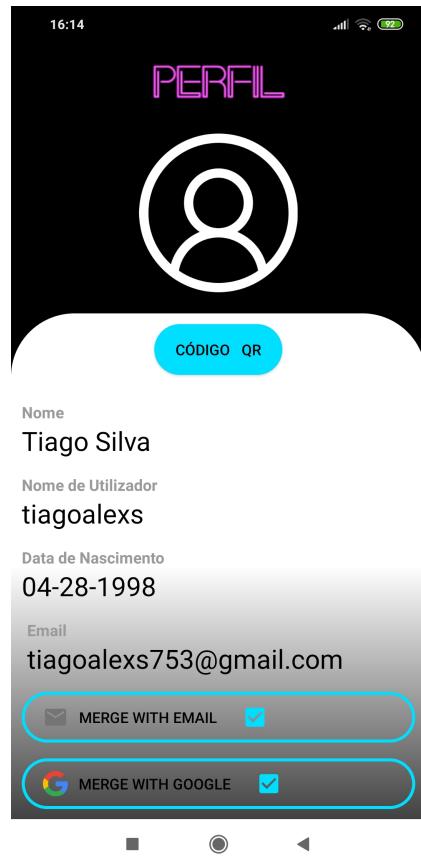


Figura 4.12: Página do utilizador

O utilizador também pode visualizar o seu próprio código QR ao premir o botão ”Código QR”.



Figura 4.13: Código QR do utilizador

Para além disso, também dispõe de 2 funcionalidades: **Merge With Email** e **Merge With Google**.

Merge With Email

Esta funcionalidade faz com os utilizadores que tenham feito o registo via *Google Sign In* possam fazer o seu processo de autenticação também por *email* e *password*. O utilizador ao premir o botão "*Merge With Email*", ativa uma caixa de diálogo (*Dialog*) com um parâmetro "*password*" por preencher.

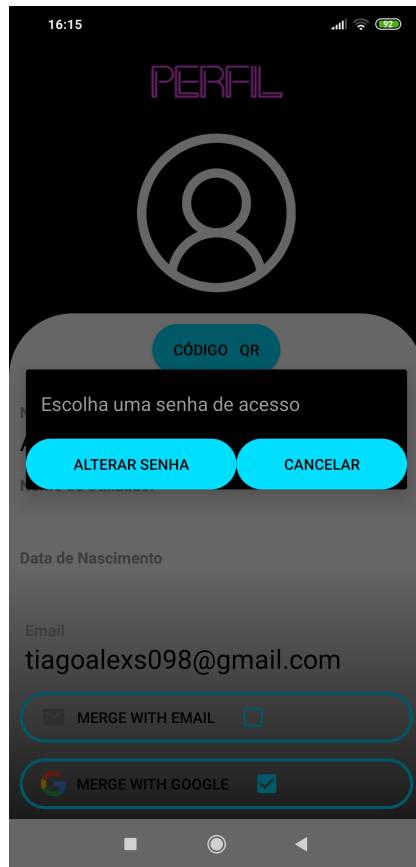


Figura 4.14: Caixa de diálogo do ”Merge with email”

Após o utilizador introduzir a *password* desejada, iremos verificar se a mesma contém entre 6 a 15 caracteres (tal como fizemos no processo de registo na subsecção *Email and Password*[4.1.1].

Caso a verificação seja validada, iremos utilizar o método *linkWithCredential(AuthCredential credential)*, que, à semelhança do método *signInWithCredential(AuthCredential credential)* visto anteriormente, também necessita de uma credencial que é fornecida por um *authentication provider*. Neste caso, iremos precisar da credencial do autenticador de *email* e *password*. O código seguinte ilustra o processo de obtenção da credencial e utilização do método *linkWithCredential(AuthCredential credential)*.

Código 19: Merge with Email

```
private void linkAuthType(String authType, String password,
    String token){
    AuthCredential credential;
    String auth_provider;
    switch (authType){
        case "GOOGLE":
            //Google Sign-In
            credential = GoogleAuthProvider.getCredential(token,
                null);
            auth_provider = "google_auth";
            break;
        case "EMAIL":
            credential = EmailAuthProvider.getCredential(
                emailTxt.getText().toString(), password);
            auth_provider = "normal_auth";
            break;
        default:
            credential = null;
            auth_provider = null;
            break;
    }
    if(credential!=null) {
        mAuth.getCurrentUser().linkWithCredential(credential)
            .addOnCompleteListener(this, new OnCompleteListener<
                AuthResult>() {
                @Override
                public void onComplete(@NonNull Task<AuthResult>
                    task) {
                    if (task.isSuccessful()) {
                        DatabaseReference ref = FirebaseDatabase.
                            getInstance().getReference().child("Users").
                            child(mAuth.getUid());
                        ref.child(auth_provider).setValue((int)1)
                            ;
                        Log.d(TAG, "linkWithCredential:success");
                        Toast.makeText(UserProfile.this, "
                            Successfully linked your account!",
                            Toast.LENGTH_SHORT).show();
                    } else {
                }
            }
        );
    }
}
```

```
        Log.w(TAG, "linkWithCredential:failure",
              task.getException());
        Toast.makeText(UserProfile.this, "Link
              failed. Try again.",
              Toast.LENGTH_SHORT).show();
    }
}
});  
}
```

Caso o *linkWithCredential(AuthCredential credential)* for concluído com sucesso, o utilizador poderá iniciar a sessão na aplicação via *email* e *password*.

Merge With Google

Esta funcionalidade faz com que os utilizadores que tenham feito o registo por email e password possam fazer o seu processo de autenticação pela *API Google Sign In*. A única coisa que muda em contraste com a funcionalidade interior é que em vez de aparecer uma caixa de diálogo a perguntar pela *password*, irá mostrar uma interface própria da API da *Google* em que o utilizador tem de introduzir o seu *email* da *Google* e a sua *password*. O resto do processo é idêntico.

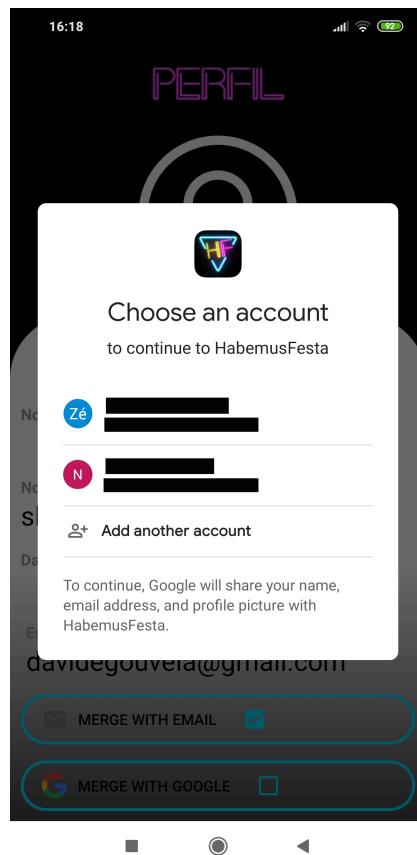


Figura 4.15: Link da conta com o Google Sign In

Após o processo ser concluído, o utilizador já poderá utilizar o *Google Sign In* para iniciar a sessão na aplicação.

4.3.5 Terminar Sessão

O terminar sessão é feito através do método `signOut()` das classes `FirebaseAuth` e `GoogleSignInClient` (no caso de este ter feito o processo de autenticação pelo *Google Sign In*). Para além da sessão do utilizador ser terminada, o serviço de localização GPS também será parado. O código seguinte ilustra todo este processo.

Código 20: Terminar sessão

```

private void signOut(){
    GPSTracker.shouldContinue = false;
    //user logout
    mAuth.signOut();
    mGoogleSignInClient.signOut()
        .addOnCompleteListener(this, new OnCompleteListener<
            Void>() {
            @Override
            public void onComplete(@NonNull Task<Void> task) {
                //Go back to Login Screen
                Intent intent = new Intent(HomeUserLogin.this,
                    MainActivity.class); // start ResultActivity
                startActivityForResult(intent, 1);
            }
        });
}

```

4.4 Colaborador

Nesta secção iremos abordar como foram implementadas as funcionalidades do colaborador, começando pela criação e remoção de eventos, criação e remoção de produtos, sistema de transação de pontos, e por fim, listagem de transações ocorridas no evento.

4.4.1 Criar/Remover Eventos

Esta secção está dividida em 2 partes: a primeira irá explicar a funcionalidade de criar eventos; a segunda explica o processo de remoção de eventos.

Criar Eventos

Para criar um evento, o colaborador tem de preencher uma série de pârametros obrigatórios na aplicação. Esses pârametros são: o nome do evento, a imagem do evento (que pode ser, por exemplo, o panfleto do evento), o tipo de evento, as datas de início e fim do evento e a localização do evento. A descrição do evento é optativa.



Figura 4.16: Criar Evento

Para escolher a imagem do evento, tem de se criar um *Intent* com um determinado conjunto de parâmetros e utilizar o método *startActivityForResult(Intent intent, int requestCode)*. O código seguinte ilustra o processo de criação do *Intent* e de inicialização da atividade.

Código 21: Criação do Intent e inicialização da atividade

```
private void chooseImage(){
    Intent intent = new Intent();
    intent.setType("image/*");
    intent.setAction(Intent.ACTION_GET_CONTENT);
    startActivityForResult(intent, IMAGE_REQUEST);
}
```

O método *setType(String type)* define o tipo de ficheiro que o *intent* pode

aceitar. No nosso caso definimos que só podem ser aceites imagens (`type = "image/*"`). As imagens podem ter qualquer tipo de extensão (.gif, .jpeg, .bmp, .jpg, .png, etc.).

Com o `setAction(String action)`, definimos que o utilizador pode selecionar ficheiros que tenha dentro do seu telemóvel (`action = Intent.ACTION_GET_CONTENT`). Como anteriormente tinhemos feito uma restrição quanto ao tipo de ficheiro que pode ser retornado, o utilizador só pode selecionar imagens.

E por fim, inicializamos a atividade que irá retornar uma imagem.

O método `onActivityResult(int requestCode, int resultCode, Intent data)` retorna o resultado de cada atividade.

Dentro deste método iremos verificar se o utilizador selecionou uma imagem com sucesso. Para tal, o `requestCode` tem de ser igual a `IMAGE_REQUEST` e o `resultCode` igual a `RESULT_OK`.

Código 22: OnActivityResult

```
@Override
protected void onActivityResult(int requestCode, int resultCode,
    , @Nullable Intent data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == IMAGE_REQUEST && resultCode == RESULT_OK
        && data != null && data.getData() != null){
        event_image_uri = data.getData();
        try {
            Bitmap yourBitmap = MediaStore.Images.Media.
                getBitmap(this.getContentResolver(),
                event_image_uri);
            Bitmap resized = Bitmap.createScaledBitmap(
                yourBitmap, event_image_view.getWidth(),
                event_image_view.getHeight(), true);
            event_image_view.setImageBitmap(resized);
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

De seguida iremos converter a imagem escolhida pelo utilizador num `Bitmap` para poder ser visualizado na aplicação.



Figura 4.17: Imagem do evento carregada

Os tipos de evento são apresentados com recurso à classe *Spinner*, de modo a aparecerem em formato *dropdown* na aplicação. Os tipos de evento disponíveis encontram-se definidos dentro do *RealTime Database* do *Firebase*, dentro do nó ”tags”. A figura seguinte ilustra a forma como foram definidos.

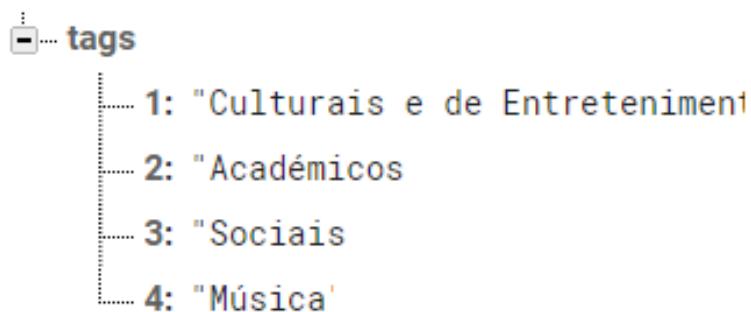


Figura 4.18: ”tags” no RealTime Database

O valor de cada nó-filho de "tags" corresponde a um tipo de evento.

A escolha das datas de início e fim de um evento são feitas com recurso a uma caixa de diálogo (*Dialog*). Dentro dessa caixa, o utilizador terá ao seu dispor um calendário e um selecionador de tempo. O calendário é mostrado com recurso a um *CalendarView*, enquanto que o tempo é mostrado através de um *TimerPicker*. A figura seguinte ilustra a caixa de diálogo.

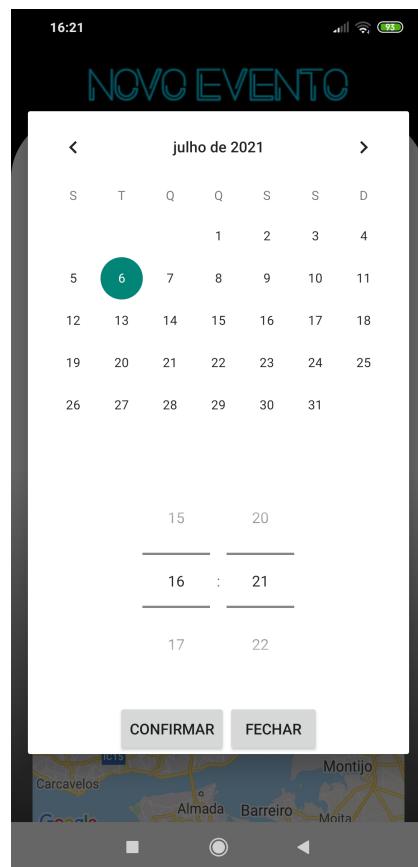


Figura 4.19: Escolha da data e hora do evento

Após inserir a data e hora do evento serão feitas algumas validações. A primeira validação verifica se os pârametros foram escolhidos, a segunda verifica se a data de início introduzida não ocorre depois da data de fim. O código seguinte ilustra este passo.

Código 23: Validação dos pârametros das datas

```
calendarConfirmBtn.setOnClickListener(new View.OnClickListener
() {
    @Override
    public void onClick(View v) {
        if(curDate.length() > 0){
            String time = String.format("%02d:%02d", picker.
                getCurrentHour(), picker.getCurrentMinute());
            if(dateType.equals("init")){
                if(event_end_date.getText().toString().length()
                    > 0){
                    if(!checkIfDatesAreValid(curDate+" | "+time,
                        event_end_date.getText().toString())){
                        Toast.makeText(EventActivity.this, "Error
                            : Time for end date must be superior
                            to init date", Toast.LENGTH_SHORT).
                            show();
                    }
                }
                event_init_date.setText(curDate+" | "+time);
                isInitDateTimeComplete = true;
            }else{
                if(event_init_date.getText().toString().length()
                    > 0){
                    if(!checkIfDatesAreValid(event_init_date.
                        getText().toString(), curDate+" | "+time)
                    ){
                        Toast.makeText(EventActivity.this, "Error
                            : Time for end date must be superior
                            to init date", Toast.LENGTH_SHORT).show
                            ();
                    }
                }
            }
        }
    }
})
```

```

        event_end_date.setText(curDate+" | "+time);
        isEndDateTimeComplete = true;
    }
    mDialog.dismiss();
}
});
}
);
});
```

Agora resta apenas escolher a localização do evento.

Neste passo fazemos uso das API *Google Maps* e *Places* e também de um *FrameLayout* para mostrar um mapa na aplicação com uma barra de pesquisa associada, de modo a que o utilizador possa introduzir uma morada ou um local. O código seguinte ilustra este passo.

Código 24: Criação do SearchMap e da visualização do mapa da Google

```

FragmentManager fragmentManager = getSupportFragmentManager();
SearchMap fragment = new SearchMap();
fragmentManager.beginTransaction().replace(R.id.testFragment,
    fragment).commit();
```

”testFragment” é um identificador único associado ao *FrameLayout* que vai servir de ”placeholder” para o mapa.

SearchMap é uma classe que estende de *Fragment* e que tem por objetivo mostrar na aplicação um mapa da *Google* com ou sem barra de pesquisa, e realizar as *queries* associadas.

SearchView é um *widget* que fornece uma interface ao utilizador para realizar *queries* e submeter pedidos a um determinado *search provider*. Esta classe servirá para criar a nossa barra de pesquisa.

Esta classe dispõe de um método *setQueryTextListener(OnQueryTextListener listener)* que permite a realização de *queries* ao *search provider*, que no nosso caso será o *Geocoder*.

Geocoder é uma classe das bibliotecas do *Android* que realiza *geocoding* e *geocoding reverse*. *Geocoding* é o processo de transformar locais, moradas e informação sobre um determinado sítio em coordenadas geográficas como latitude e longitude.

Dentro do método `setOnQueryTextListener(OnQueryTextListener listener)` do `SearchView` iremos instanciar a classe `Geocoder` e utilizar o método `getFromLocationName(String locationName, int maxResults)` para adquirir uma variável do tipo `Address` que contém a latitude e longitude do determinado local. O código seguinte ilustra o processo de realização de *queries*.

Código 25: Listener de queries da barra de pesquisa

```
searchView.setOnQueryTextListener(new SearchView.  
    OnQueryTextListener() {  
        @Override  
        public boolean onQueryTextSubmit(String query) {  
            String location = searchView.getQuery().toString();  
            List<Address> addressList = null;  
            if(location!=null || !location.equals("")){  
                Geocoder geocoder = new Geocoder(getApplicationContext());  
                try {  
                    addressList = geocoder.getFromLocationName(  
                        location, 1);  
                    if(addressList == null || addressList.size()==0)  
                    {  
                        Toast.makeText(getApplicationContext(),  
                            "No location  
                            found with that name.",  
                            Toast.LENGTH_SHORT).show();  
                        return false;  
                    }  
                    address = addressList.get(0);  
                    LatLng latLng = new LatLng(address.getLatitude()  
                        , address.getLongitude());  
                    map.addMarker(new MarkerOptions().position(  
                        latLng).title(location));  
                    map.animateCamera(CameraUpdateFactory.  
                        newLatLngZoom(latLng,10));  
                }catch (IOException e){  
                    e.printStackTrace();  
                }  
            }  
            return false;  
        }  
    }
```

```
@Override  
public boolean onQueryTextChange(String newText) {  
    return false;  
}  
});
```

Após obtermos a latitude e longitude do local, criamos um marcador com essas coordenadas e centramos o mapa nesse local. Por fim basta o utilizador confirmar a criação do evento.

Remover Eventos

Nesta secção é mostrado ao colaborador o conjunto de eventos ao qual este pertence. A figura seguinte ilustra esta secção.



Figura 4.20: Remover Evento

A imagem e o nome do evento são definidos dentro de um *LinearLayout* com orientação horizontal. Para evitar que algum evento não seja mostrado dentro da área de ecrã do utilizador, decidimos inserir os eventos dentro de um *ScrollView*. Assim, os eventos ficam dentro de uma área pré-definida e o utilizador pode deslizar com o dedo no ecrã para visualizar os eventos que estão mais em baixo e que não aparecem no ecrã.

Para um utilizador remover um evento, basta premir o evento pretendido. O processo de remoção de um evento irá eliminar alguns nós no *RealTime Database* do *Firebase* como também imagens dentro de *Storage* que sejam pertencentes ao mesmo. A figura seguinte ilustra este processo.

Código 26: Listener de queries da barra de pesquisa

```
private void removeEvent(String event_id){

    DatabaseReference ref = FirebaseDatabase.getInstance().
        getReference();
    StorageReference storageRef = FirebaseStorage.getInstance()
        .getReference();

    ref.child("Eventos").child(event_id).removeValue();
    ref.child("Eventos-Users").child(event_id).removeValue();
    ref.child("Eventos-Locs").child(event_id).removeValue();
    ref.child("Eventos-Imgs").child(event_id).removeValue();
    ref.child("Produtos").child(event_id).removeValue();

    //Images from event
    storageRef.child(event_id).delete();

    //delete from Users-Likes
    ref.child("Users-Likes").addSingleValueEventListener(new
        ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot
        ) {
            if(snapshot.exists()){
                for (DataSnapshot child : snapshot.getChildren()
                ) {
                    String user_id = child.getKey();
                    if(((HashMap<String, Integer>)child.getValue
                    ()).containsKey(event_id)){

```

```
        ref.child("Users-Likes").child(user_id).
            child(event_id).removeValue();
    }
}
Toast.makeText(RemoveEventsActivity.this, R.
    string.remove_event_success,
    Toast.LENGTH_SHORT).show();
}
}
@Override
public void onCancelled(@NonNull DatabaseError error)
{}
});

//delete from Eventos-Collabs
ref.child("Eventos-Collabs").orderByChild("evento_id").
    equalTo(event_id).addListenerForSingleValueEvent(new
ValueEventListener() {
    @Override
    public void onDataChange(@NonNull DataSnapshot snapshot
    ) {
        System.out.println(snapshot);
        if(snapshot.exists()){
            for (DataSnapshot child : snapshot.getChildren()
            ) {
                child.getRef().removeValue();
            }
            finish();
            startActivity(getIntent());
            Toast.makeText(RemoveEventsActivity.this, "Event
                deletion complete!",
                Toast.LENGTH_SHORT).show();
        }
    }
}

@Override
public void onCancelled(@NonNull DatabaseError error)
{}
});
}
```

Primeiramente iremos começar por eliminar o evento em si (“Eventos”), seguido da sua localização (“Eventos-Locs”), da referência para as imagens (“Eventos-Img”) e os produtos associados (“Produtos”). De seguida, eliminaremos todas as imagens do *Storage* que estejam dentro da pasta do evento. E por fim, iremos eliminar a referência dos colaboradores que pertencem ao evento (“Eventos-Collabs”).

No final se for tudo bem sucedido, irá ser apresentado ao utilizador uma mensagem de sucesso.

4.4.2 Criar/Remover Produtos

Esta secção também se encontra dividida em 2 subsecções. A primeira irá abordar o processo de criação de produtos. A segunda explica o processo de remoção.

Criar Produtos

Para criar um produto, o colaborador tem de preencher uma série de pârametros obrigatórios na aplicação. Esses pârametros são: o nome do produto, uma imagem que o represente, o evento ao qual está associado e o número de pontos que o produto custa.



Figura 4.21: Criar Produto

Os eventos disponíveis para associar a um produto estão em conformidade com os eventos ao qual o colaborador pertence. Estes são mostrados através de um *Spinner*, que, como visto anteriormente, apresenta os resultados em formato *dropdown*.

O processo de escolha e amostra de imagem do produto é idêntica ao processo descrito para a imagem do evento na secção Criar Eventos[4.4.1].

Após o colaborador confirmar a criação do produto, irá ser verificado se todos os parâmetros foram introduzidos. Caso sejam validados, irá ser guardada a informação do produto no *RealTime Database* do *Firebase*. Também será guardado a imagem escolhida no *Storage* do *Firebase*, dentro da pasta do evento. A figura seguinte ilustra a forma como ficou definido o produto na base de dados.

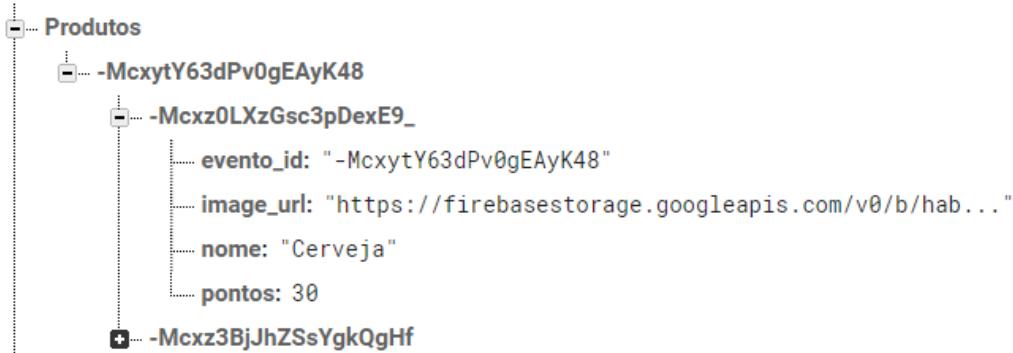


Figura 4.22: ”Produtos” no RealTime Database

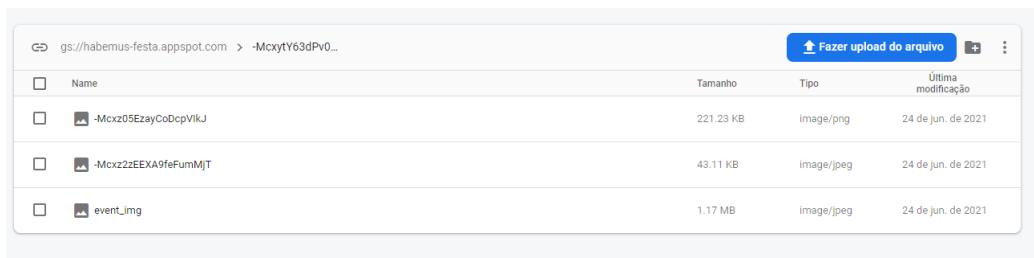


Figura 4.23: Pasta dos eventos no Storage

O primeiro nó-filho de ”Produtos” corresponde ao identificador único de cada evento. Dentro desse nó, estará um nó-filho correspondente ao identificador único de cada produto, com toda a informação inserida no processo de criação do produto. `image_url` corresponderá ao URL, Localizador Uniforme de Recursos, do produto.

Remover Produtos

Nesta secção, a informação dos produtos é disposta no ecrã de forma idêntica ao que foi demonstrado na secção Remover Eventos[4.4.1]. A informação de cada produto (nome e imagem) é mostrada dentro de um *LinearLayout*, que por sua vez está dentro de um *ScrollView*.



Figura 4.24: Remover Produto

Para remover um produto o colaborador tem de simplesmente o selecionar. Caso o processo de remoção seja bem sucedido, a aplicação irá notificá-lo com uma mensagem de sucesso.

4.4.3 Sistema de Transação

Na secção do sistema de transação de pontos, se o colaborador estiver envolvido em mais que um evento, irá aparecer um menu *dropdown* com os eventos a que este pertence. Ao selecionar um evento, a lista de produtos mostrada no ecrã é alterada de forma a que apareça os produtos referentes a esse evento. A figura seguinte ilustra esta secção na aplicação.



Para o colaborador iniciar a cobrança de pontos, este tem de primeiramente selecionar o produto escolhido pelo utilizador. De seguida, o utilizador tem de mostrar o seu código QR ao colaborador de forma a que este o possa examinar. Durante o processo de exame do código QR irão ser feitas duas validações. A primeira irá verificar se o código QR é válido, ou seja, se pertence a um utilizador. A segunda verifica se o utilizador tem pontos

suficientes para efetuar a troca pelo produto. O código seguinte ilustra o processo das validações.

Código 27: Validação do código QR examinado

```
private void handleTransaction(String uid, String event_id,
    String product, String points){
    DatabaseReference ref = FirebaseDatabase.getInstance().
        getReference();
    ref.child("Users").child(uid).
        addListenerForSingleValueEvent(new ValueEventListener()
    {
        @Override
        public void onDataChange(@NonNull DataSnapshot snapshot
        ) {
            if(snapshot.exists()){
                User u = snapshot.getValue(User.class);
                if(u.getPontos() - Integer.parseInt(points) < 0)
                {
                    Utils.showTransactionStatus(
                        getApplicationContext(),"FAILED", mDialog
                    );
                }else {
                    int userPoints = u.getPontos() - Integer.
                        parseInt(points);
                    ref.child("Users").child(uid).child("pontos")
                        .setValue(userPoints);
                    Calendar calendar = Calendar.getInstance();
                    SimpleDateFormat dateFormat = new
                        SimpleDateFormat("MM/dd/yyyy");
                    String date = dateFormat.format(calendar.
                        getTime());
                    date = date.replaceAll("/", "-");
                    int hours = calendar.get(Calendar.HOUR_OF_DAY
                    );
                    int minutes = calendar.get(Calendar.MINUTE);
                    String time = String.format("%02d:%02d",
                        hours, minutes);
                    Transaction transaction = new Transaction(
                        date, time, mAuth.getUid(), uid, product,
                        points);
                    ref.child("Transacoes").child(event_id).child
```

```
        (date).push().setValue(transaction);
        Utils.showTransactionStatus(
            getApplicationContext(), "COMPLETE",
            mDialog);
        return;
    }
}
else{
    Toast.makeText(TransactionActivity.this, "Error:
        username not found.",
        Toast.LENGTH_SHORT).show();
}
}

@Override
public void onCancelled(@NonNull DatabaseError error) {
    Toast.makeText(TransactionActivity.this, "Error:
        Please try again.",
        Toast.LENGTH_SHORT).show();
}
});
}
```

Durante a confirmação sobre o atual número de pontos na conta do utilizador, irá aparecer uma caixa de diálogo (*Dialog*) no ecrã do colaborador, que o indicará se a compra foi bem sucedida (no caso do utilizador ter o número necessário de pontos) ou se a compra falhou (se o utilizador não tiver o número necessário de pontos).

Caso tenha sido tudo validado, resta fazer a remoção de pontos na conta do utilizador.

4.4.4 Adicionar Colaboradores

Nesta secção definimos dois *Spinners*: um que contém os vários eventos ao qual este colaborador pertence e outro que contém todos os utilizadores/colaboradores da aplicação.

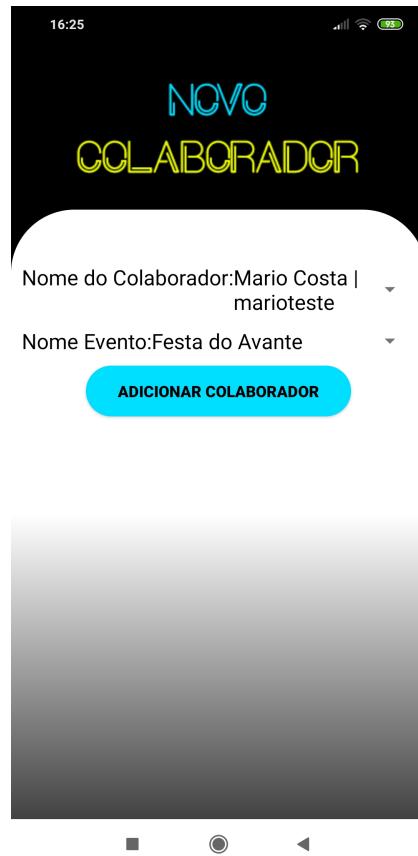


Figura 4.26: Adicionar Colaboradores

Para adicionar um utilizador/colaborador a um evento basta selecionar o evento e o utilizador/colaborador pretendido. Irá ser feita uma validação para verificar se o utilizador/colaborador já pertence ao evento que foi escolhido. O código seguinte ilustra este processo.

Código 28: Validação do colaborador na base de dados

```
//check if collab is already linked to the chosen event
ref.child("Eventos-Collabs").orderByChild("collab_id").equalTo(
    collab_id).addSingleValueEventListener(new
    ValueEventListener() {
```

```
@Override
public void onDataChange(@NonNull DataSnapshot snapshot) {
    if(snapshot.exists()) {
        for (DataSnapshot child : snapshot.getChildren()) {
            EventsCollabs ecTest = child.getValue(
                EventsCollabs.class);
            if(ecTest.getEvento_id().equals(event_id)){
                addCollabConfirmBtn.setClickable(true);
                canContinue = false;
                Toast.makeText(getApplicationContext(), R.
                    string.toast_colaborator_double_added,
                    Toast.LENGTH_LONG).show();
                break;
            }
        }
        if(canContinue) {
            //add reference to DB
            ref.child("Eventos-Collabs").push().setValue(ec)
                ; //change user_type(if needed) to "Admin"
            ref.child("Users").child(collab_id).
                addListenerForSingleValueEvent(new
                    ValueEventListener() {
                        @Override
                        public void onDataChange(@NonNull
                            DataSnapshot snapshot) {
                            if (snapshot.exists()) {
                                User u = snapshot.getValue(User.class
                                    );
                                if (u.getUser_type().equals("Normal"))
                                    {
                                        ref.child("Users").child(collab_id
                                            ).child("user_type").setValue(
                                                "Admin");
                                    }
                                canContinue = true;
                                addCollabConfirmBtn.setClickable(true
                                    );
                                Toast.makeText(getApplicationContext()
                                    (), R.string.
                                    toast_colaborator_event_added,
                                    Toast.LENGTH_LONG).show();
                            }
                        }
                    })
        }
    }
}
```

```
        }
    }

    @Override
    public void onCancelled(@NonNull DatabaseError error) {}
);

} else{
    canContinue = true;
}
} else{
    //add reference to DB
    ref.child("Eventos-Collabs").push().setValue(ec);

    //change user_type(if needed) to "Admin"
    ref.child("Users").child(collab_id).
        addListenerForSingleValueEvent(new
            ValueEventListener() {
        @Override
        public void onDataChange(@NonNull DataSnapshot
snapshot) {
            if (snapshot.exists()) {
                User u = snapshot.getValue(User.class);
                if (u.getUser_type().equals("Normal")) {
                    ref.child("Users").child(collab_id).
                        child("user_type").setValue("Admin
");
                }
                canContinue = true;
                addCollabConfirmBtn.setClickable(true);
                Toast.makeText(getApplicationContext(), R
                    .string.toast_colaborator_event_added,
                    Toast.LENGTH_LONG).show();
            }
        }
    }
}

@Override
public void onCancelled(@NonNull DatabaseError
error) {}
);
}
```

```
    }
    @Override
    public void onCancelled(@NonNull DatabaseError error) {}
});
```

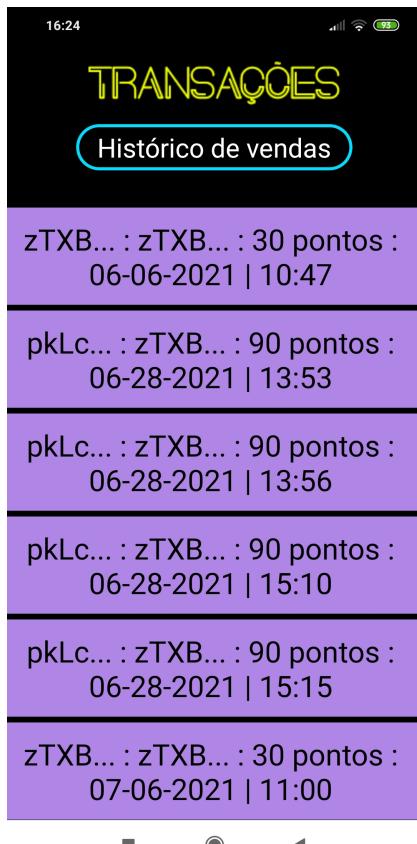
No caso de ter sido selecionado um utilizador, este passará a ser um colaborador.

4.4.5 Lista de Transações

Dentro desta componente definimos um *CardView* que irá conter a informação de cada transação de um determinado evento, ordenada pela sua data de criação. O colaborador só tem acesso às transações dos eventos a que pertence.

No caso do colaborador estar envolvido em mais que um evento, é mostrado um menu *dropdown* na aplicação, com recurso a um *Spinner*, com os nomes dos eventos ao qual este pertence.

O colaborador ao selecionar um destes eventos atualiza a lista de transações para estarem em conformidade com o evento escolhido.



(a) Transações (colaborador com 1 evento)



(b) Transações (colaborador com 2 ou mais eventos)

Capítulo 5

Validação e Testes

De forma a testar a aplicação e respetivas funcionalidades e tendo como objetivos a experiência do utilizador face à sua utilização e opinião acerca de possíveis alterações no visual ou funcionalidades, foi criado um questionário com o intuito de avaliar os diferentes inquiridos. Neste questionário está presente uma hiperligação referente ao ficheiro APK gerado, de forma a que estes tenham acesso à aplicação.

5.1 Contexto

Este questionário é composto por um conjunto de questões acerca do inquirido relativamente a dados demográficos, dados tecnológicos e preferências do tipo lazer. Segue-se com um conjunto de instruções relativas à aplicação estando estas divididas tanto para o utilizador como para o colaborador, associadas a campos de resposta que averiguam o sucesso do utilizador face a essas instruções. Por fim, são apresentadas um conjunto de questões de resposta aberta, para que o utilizador possa dar a sua opinião relativamente ao questionário e à aplicação.

Um dos grandes objetivos deste questionário era compreender se o conjunto de participantes envolvidos conseguia concluir todas as questões com sucesso de forma a que nós pudessemos perceber que alterações deveriam ser feitas para melhorar estes aspetos que viesssem a ser retirados. Também foi importante receber o *feedback* dos inquiridos relativamente a futuras melhorias na aplicação.

5.2 Participantes

Para este questionário obtiveram-se dez resultados, de dez utilizadores. Relativamente às idades, podemos observar que metade dos inquiridos centram-se na faixa etária entre os 18-32 anos, sendo os 32-40 o segundo maior grupo e os restantes variam com idades inferiores a 18 anos, e superiores a 50 anos. Do conjunto dos participantes, obteve-se uma amostra de seis indivíduos do sexo masculino e quatro do sexo feminino. Relativamente ao estado civil dos participantes em causa, obteve-se uma amostra de sete solteiros e três casados.

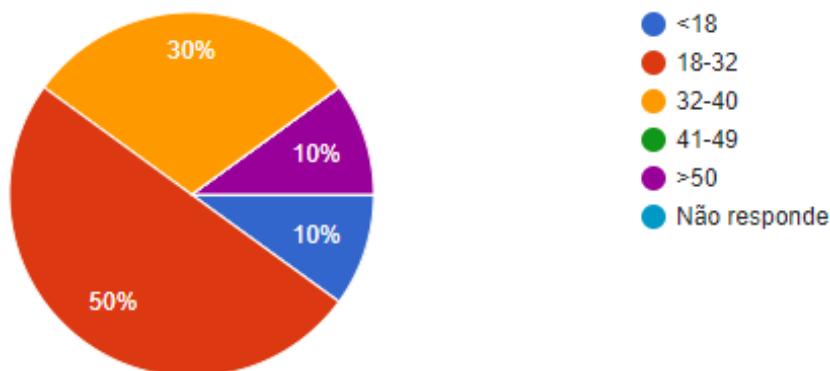


Figura 5.1: Intervalo de idades dos participantes

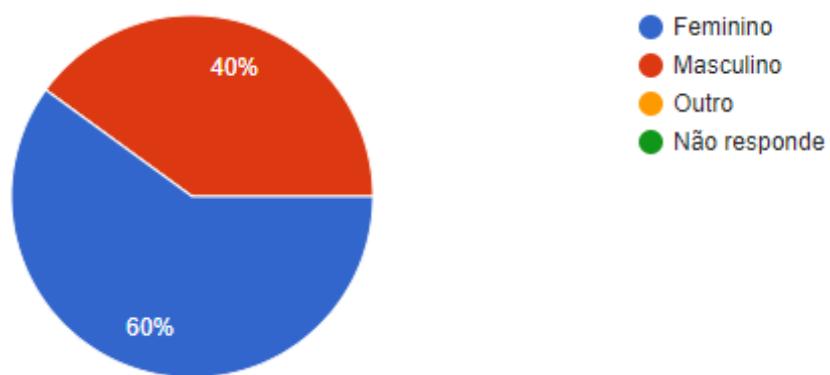


Figura 5.2: Género dos participantes

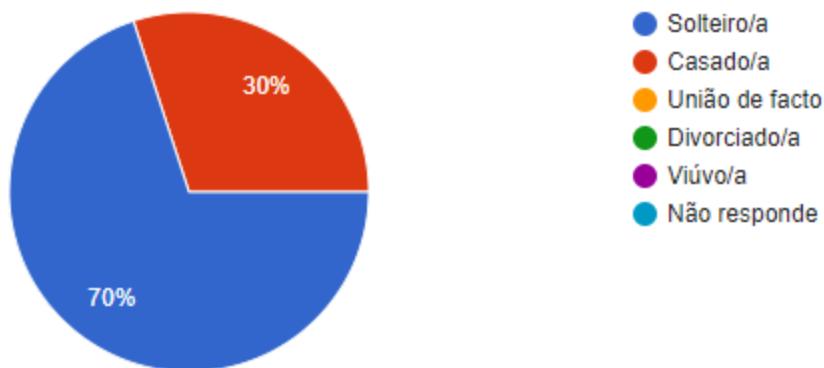


Figura 5.3: Estado civil dos participantes

Relativamente aos resultados da questão sobre o tempo de utilização médio de telemóvel por dia (figura 5.4), podemos observar que a grande maioria centra-se entre as duas a quatro horas diárias, seguidas de menos de duas horas, e finalmente segue-se o intervalo de quatro a seis horas.

Na figura 5.5, seguem-se os resultados acerca da frequência com o que o inquirido costuma sair em termos de lazer com referência aos já mencionados anteriormente no capítulo 1. “Entre 1 a 2 vezes por mês” e “Menos de 1 vez por mês” são os que apresentam maior incidência, seguido de “Entre 3 a 4 vezes por mês” e por fim “Mais do que 1 vez por semana”. Finalmente, no que toca ao tipo de eventos que os inquiridos preferem, os mais selecionados foram os festivais e os concertos, e de seguida, as festas académicas e *sunset*s. De notar que nesta questão era possível selecionar mais do que uma escolha. Assim como o tipo de eventos mais selecionados, o motivo que leva os inquiridos a assistir este tipo de eventos é na grande maioria das vezes para relaxar/descontrair, ou assistir a um determinado artista, seguindo-se de fazer novas amizades.

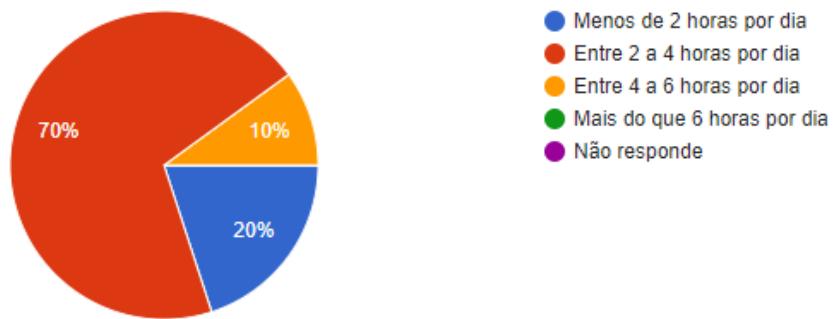


Figura 5.4: Tempo de utilização do telemóvel

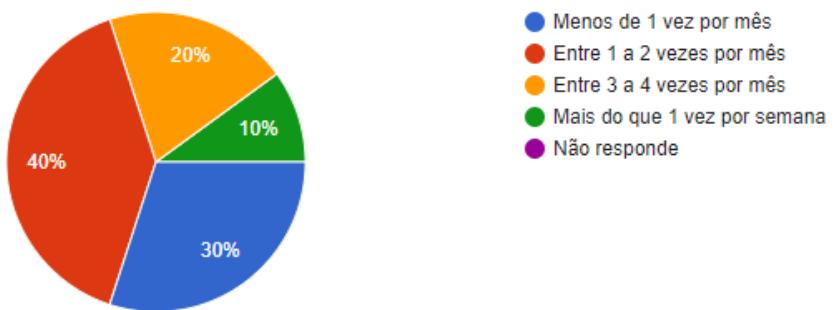


Figura 5.5: Frequência de “saídas”

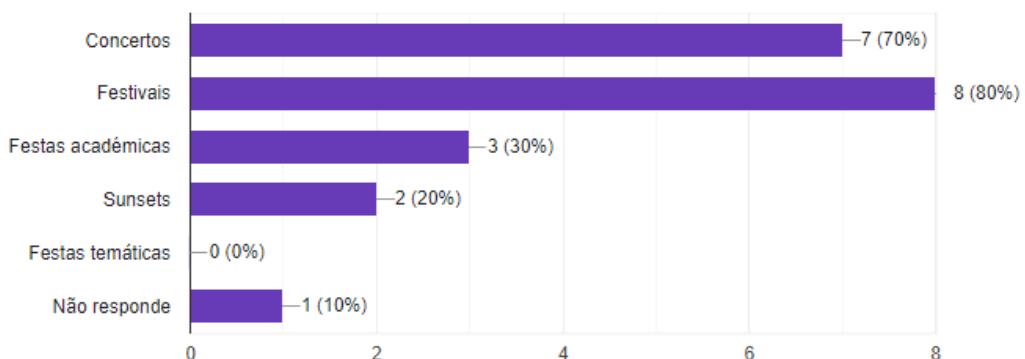


Figura 5.6: Tipo de eventos

5.3 Teste da aplicação

Relativamente aos testes efetuados na aplicação pelos inquiridos, eram dadas tarefas que obrigavam à navegação pela aplicação, estando estas divididas conforme o tipo de conta com que se iniciasse sessão. Inicialmente era pedido ao inquirido que criasse uma nova conta, e que iniciasse sessão com a mesma. Nestes dois pontos surgiram algumas dificuldades na realização destas tarefas, visto que existia um problema com a sincronização dos pedidos à base de dados no ato do novo registo, que levava à dificuldade da concretização da segunda tarefa, como se pode observar na figura 5.8 onde apenas metade dos inquiridos conseguiram concluir estes dois passos.

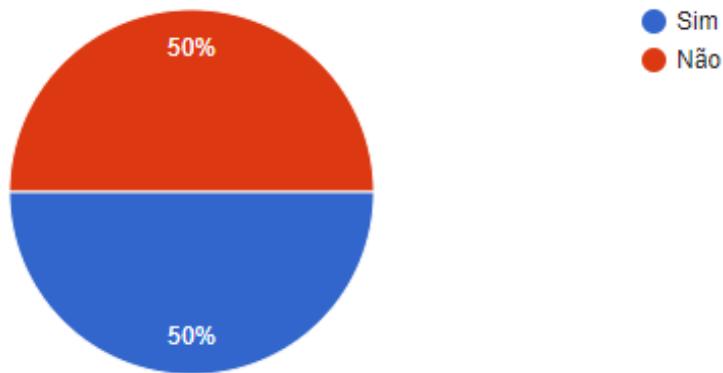


Figura 5.7: Criar novo registo

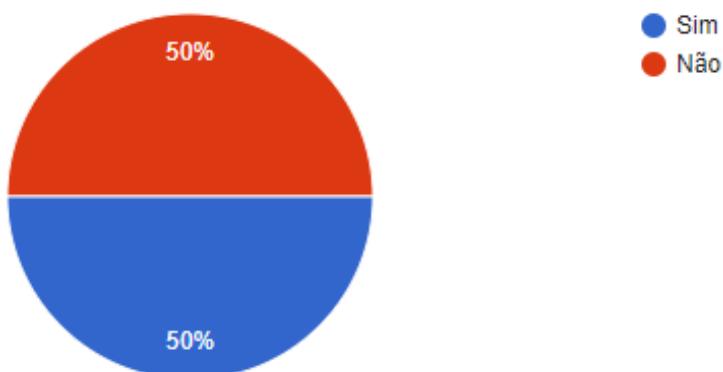


Figura 5.8: Iniciar sessão com novo registo

De seguida era pedido ao inquirido que realizasse multiplas operações enquanto utilizador, tais como verificar os eventos “Por data”, quantos eventos estavam presentes no separador “Meus eventos”, selecionar a opção “Vou” num dos eventos à escola pelo inquirido e verificar novamente o separador “Meus eventos”, verificar o perfil e confirmar se os dados coincidiam, e por fim verificar a versão da aplicação nas definições. Estes testes na sua grande maioria foram concluídos com sucesso. O problema surgiu na questão de verificar novamente o separador “Meus eventos” após selecionar “Vou” num dos eventos à escolha, onde não foram apresentados quaisquer resultados, devendo-se ao facto da taxa de atualização dos valores da base de dados não ser rápida o suficiente para o utilizador notar esta alteração na listagem de eventos. Terminada a primeira fase de tarefas relativas ao utilizador, passamos então para a segunda fase onde o inquirido inicia sessão com a conta de colaborador. O tipo de tarefas aqui pedidas requeria ao inquirido que criasse um novo evento, criar um produto associado a esse evento, remover o produto e o evento anteriormente criado pela respetiva ordem, e por fim, verificar o número de transações presentes no histórico de transações. Aqui, para “surpresa” nossa, considerámos que todas as tarefas foram concluídas com sucesso, visto que nas questões anteriormente mencionadas nesta fase eram avaliadas com uma escala de “1..10”, “Muito difícil..Muito fácil” respectivamente, e que todos os resultados obtiveram em média o valor 9.

5.4 Respostas dos inquiridos

Relativamente às questões de resposta aberta aos inquiridos, a dificuldade que estes tiveram para a conclusão do questionário obteve um excelente resultado com uma nota média de nove em dez valores. As dificuldades incidiram maioritariamente na criação de um novo registo e inicio de sessão com o mesmo, e no campo da localização da criação de um novo evento, registaram-se anotações de que este deveria ser mais intuitivo. Em relação ao visual da aplicação no seu geral, obtivemos também uma média de 9, onde as críticas centraram-se mais no aspetto de algumas cores, nomeadamente do menu de navegação do utilizador/colaborador e no tamanho de algumas *textviews* face ao tamanho de alguns botões. Por fim os inquiridos, na sua maioria responderam “sim” às questões relativamente à utilização da aplicação num futuro

próximo, e se a recomendariam aos seus amigos/as.

Capítulo 6

Conclusões e Trabalho Futuro

Ao terminar este projeto, o grupo pode assumir que foram cumpridos os objetivos que haviam sido traçados inicialmente. Entre eles, está o desenvolvimento de uma aplicação para a plataforma Android que, ajuda a promover a interação social entre as pessoas, mesmo tratando-se de uma aplicação para smartphones. A ideia à partida parece um pouco controversa, mas trata-se de um produto que ajuda os utilizadores a iniciar uma conversação “cara a cara”, motivados pela troca de perfis, sendo esta uma vantagem para ambas as partes.

Nos testes realizados com os inquiridos, foram tomadas medidas relativamente aos problemas detetados pelos mesmos, tais como o registo de um novo utilizador, e inicio de sessão do mesmo. Alguns aspetos relacionados com a *interface* também foram alterados de forma a ir ao encontro das críticas mencionadas no questionário.

Consideramos este questionário um dos elementos mais importantes na fase final do nosso projeto, porque faz com que tenhamos uma perspectiva diferente daquela que tínhamos anteriormente, e ao mesmo tempo a descobrir determinados problemas na aplicação que ainda não tínhamos obtido dificuldades.

Como trabalho futuro, faltará registar uma patente e submeter à legislação em vigor. À aplicação poderão ser adicionados mais campos na área de registo de novo utilizador e de eventos, como também elementos personalizáveis às definições de forma a melhorar a sua utilização. Também serão consideradas melhorias provenientes dos testes de usabilidade. Em conjunto com as melhorias, outra ideia será criar parcerias com empresas que normal-

mente estão presentes em eventos, tais como empresas de bebidas, alimentos, merchandise, entre outros. Uma outra implementação, seria a remoção automática de colaboradores e seus eventos associados no final de cada evento adicionado por estes. Desta forma, qualquer colaboração, seria renovada a cada evento que viesse a decorrer. Finalmente, para otimizar a aplicação e também “acabar” com o dinheiro físico e sua utilização nos eventos, a ideia seria criar uma carteira virtual em que cada utilizador pudesse carregar a sua conta com valor monetário por determinar, para adquirir produtos latentes e não passíveis de troca de pontos.

Para terminar esperamos que o nosso projeto corresponda às expetativas e que demonstre o trabalho e o conhecimento que desenvolvemos ao longo do curso, tendo aplicado aqui muitos desses aspectos. Pensamos que o desenvolvimento desta aplicação nos permitiu ter uma ideia diferente acerca de um desenvolvimento de projeto numa escala maior e, tendo participado no FEIM elucidou-nos um pouco mais aproximando-nos da realidade do que será a apresentação do projeto como fim de curso, representando assim o fim da licenciatura e, esperamos, o início de uma nova etapa.

Bibliografia

- [Android Studio, 2013] Android Studio (2013). Android studio. <https://developer.android.com/studio>.
- [Code Scanner, 2017] Code Scanner (2017). Code scanner. <https://github.com/yuriy-budiyev/code-scanner>.
- [Firebase, 2010] Firebase, G. (2010). Firebase. <https://firebase.google.com/>.
- [Fused Location Provider, 2012] Fused Location Provider (2012). Fused location provider. <https://developers.google.com/location-context/fused-location-provider>.
- [Geofire, 2014] Geofire (2014). Geofire. <https://github.com/firebase/geofire-java>.
- [Glide, 2013] Glide (2013). Glide. <https://github.com/bumptech/glide>.
- [Google Maps, 2013] Google Maps (2013). Google maps. <https://developers.google.com/maps/documentation/android-sdk/start>.
- [Google Places, 2015] Google Places (2015). Google places. <https://developers.google.com/maps/documentation/places/web-service/overview>.
- [Google Sign-In, 2013] Google Sign-In (2013). Google sign-in. <https://developers.google.com/identity/sign-in/android/start-integrating>.
- [Mockplus Classic, 2019] Mockplus Classic (2019). Mockplus classic. <https://www.mockplus.com/?home=1>.

[Photo View, 2018] Photo View (2018). Photo view. <https://github.com/Baseflow/PhotoView>.

[QRGenerator, 2016] QRGenerator (2016). Qrgenerator. <https://github.com/androidmads/QRGenerator>.

[Tom Preston-Werner, 2008] Tom Preston-Werner, Chris Wanstrath, S. C. P. J. H. (2008). Github. <https://github.com/>.

[Zxing, 2018] Zxing (2018). Zxing. <https://github.com/zxing/zxing>.

Apêndice A

Formulário de Usabilidade

Habemus Festa

Questionário de usabilidade

*Obrigatório

Caracterização

Habemus Festa é uma aplicação móvel (Android) com o tema de eventos e socialização. O principal foco deste projeto assenta na ideia de promover o relacionamento interpessoal. Desta forma, a aplicação apresenta um sistema de visualização de eventos sociais, como por exemplo concertos, festivais de música, eventos académicos, etc. e, a sua respetiva calendarização.

A aplicação motiva os utilizadores a interagir entre si de forma presencial durante o decorrer destes eventos, através da implementação de um sistema de pontos que permitirá ao utilizador usufruir como moeda de troca. Estes pontos são obtidos de duas formas:

- Aquando o utilizador assiste ao evento a sua entrada no recinto é validada, e ser-lhe-ão atribuídos um determinado número de pontos;
- Através da leitura do código QR pessoal entre utilizadores. Esta troca atribui pontos às duas entidades em questão, sendo esta só permitida apenas uma única vez.

Com a obtenção destes pontos, é possível aos utilizadores trocá-los por produtos selecionados em pontos de venda que patrocinem a aplicação. Para a troca ser realizada, o utilizador apenas tem de apresentar o seu código QR na aplicação, e o colaborador efetua uma leitura desse mesmo código retirando assim os respetivos pontos.

Acerca do colaborador, a aplicação está dividida em duas vertentes de utilização:

- O utilizador, como referido anteriormente e sendo entendido como qualquer indivíduo que tenha efetuado registo na aplicação; - O colaborador, sendo aquele que é pertencente à organização de um ou mais evento(s). Este pode criar/remover eventos, criar/remover produtos, retirar pontos a um utilizador, adicionar novos colaboradores e verificar o histórico de transações efetuadas.

Download da Aplicação

Link: <https://we.tl/t-hTYxJHEuDF>

1. Idade *

Marcar apenas uma oval.

- <18
- 18-32
- 32-40
- 41-49
- >50
- Não responde

2. Género *

Marcar apenas uma oval.

- Feminino
- Masculino
- Outro
- Não responde

3. Estado civil *

Marcar apenas uma oval.

- Solteiro/a
- Casado/a
- União de facto
- Divorciado/a
- Viúvo/a
- Não responde

4. Com que frequência costuma sair? *

Marcar apenas uma oval.

- Menos de 1 vez por mês
- Entre 1 a 2 vezes por mês
- Entre 3 a 4 vezes por mês
- Mais do que 1 vez por semana
- Não responde

5. Que tipo de eventos prefere? *

Marcar tudo o que for aplicável.

- Concertos
- Festivais
- Festas académicas
- Sunsets
- Festas temáticas
- Não responde

Outra: _____

6. Por que razão gosta de ir a eventos? *

Marcar tudo o que for aplicável.

- Relaxar/Descontrair
- Fazer novas amizades
- Assistir a um determinado artista
- Passar o tempo
- Não responde

Outra: _____

7. Com que frequência usa o telemóvel? *

Marcar apenas uma oval.

- Menos de 2 horas por dia
- Entre 2 a 4 horas por dia
- Entre 4 a 6 horas por dia
- Mais do que 6 horas por dia
- Não responde

8. Qual o tipo de utilização mais frequente com o telemóvel? *

Marcar tudo o que for aplicável.

- Chamadas
- SMS
- Email
- Redes sociais
- Jogos
- Notícias
- Não responde

Outra: _____

Desafio

Nas próximas questões, será necessária a utilização da aplicação para a sua resposta.

Após a instalação, é necessário conceder as permissões necessárias no seu smartphone para o bom funcionamento da aplicação. É também necessário o acesso à internet, e o sistema de localização ativo no seu dispositivo. Após iniciar a aplicação, siga as questões abaixo ordenadamente.

Por favor, seja o mais sincero possível relativamente às suas respostas.

9. 1. Criar um novo registo - Conseguiu com sucesso? *

Marcar apenas uma oval.

- Sim
- Não

10. 2. Iniciar sessão com o registo anterior - Conseguiu com sucesso? *

Marcar apenas uma oval.

- Sim
- Não

Caso a resposta à questão 1. tenha sido "Não", inicie sessão com as seguintes credenciais:

Utilizador: usertest | Password: projeto50

11. Aceda aos eventos. Quantos eventos estão presentes na listagem dos Eventos por Data? *

Marcar apenas uma oval.

- Nenhum
- 1
- 2
- 3

12. Ainda na listagem de eventos, quantos estão disponíveis no separador dos meus eventos? *

Marcar apenas uma oval.

- Nenhum
- 1
- 2
- Mais do que 2

Aceda a um dos eventos à sua escolha e selecione "Vou".

13. Volte à listagem de eventos, e no separador dos meus eventos, verifique quantos estão disponíveis. *

Marcar apenas uma oval.

- Nenhum
- 1
- 2
- Mais do que 2

14. Aceda ao seu perfil. Verifique se os dados estão corretos. *

Marcar apenas uma oval.

- Sim
- Não

15. Aceda às definições. Qual o número da versão da aplicação? *

Termine sessão. A partir deste ponto iniciará sessão enquanto Colaborador. Volte a iniciar sessão com os seguintes dados:

Utilizador: colabtest | Password: projeto50

16. Crie um novo evento ao seu critério. Avalie a dificuldade na tarefa. *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Muito difícil Muito fácil

17. Crie um novo produto associado ao evento que criou anteriormente. Avalie a dificuldade na tarefa. *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Muito difícil Muito fácil

18. Remova o produto criado anteriormente. Avalie a dificuldade na tarefa *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Muito difícil Muito fácil

19. Remova o evento criado anteriormente. Avalie a dificuldade na tarefa *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Muito difícil Muito fácil

20. Verifique o histórico de transações. Quantas transações consegue observar? *

Marcar apenas uma oval.

Nenhuma

1

3

5

Pode terminar sessão e encerrar a aplicação.

As próximas questões servem para nos ajudar a melhorar a aplicação no futuro. Agradecemos a sua resposta.

21. Relativamente ao questionário de usabilidade. Avalie a dificuldade que teve para o concluir. *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Muito difícil Muito fácil

22. Caso tenha sentido dificuldades, enumere-as.

23. Relativamente ao visual da aplicação. Como o classifica? *

Marcar apenas uma oval.

1 2 3 4 5 6 7 8 9 10

Nada apelativo Muito apelativo

24. Quais as alterações que faria relativamente ao visual?

25. Utilizaria esta aplicação? Se sim, em que momentos utilizaria?

26. Recomendaria aos seus amigos/as?

Este conteúdo não foi criado nem aprovado pela Google.

Google Formulários

Apêndice B

Modelo EA RealTime Database do Firebase

