



Escola de Engenharia



Projeto prático de LI4

**Grupo 5**

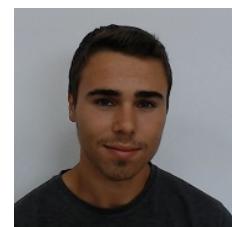
Pedro Parente, A85919

Luís Ramos, A83930

Luís Vila, A84439

Nuno Cunha, A85400

António Lindo, A85813



# Conteúdo

|     |   |    |
|-----|---|----|
| 1   | <b>Introdução</b>                                 | 4  |
| 1.1 | <b>Contextualização</b>                           | 4  |
| 1.2 | <b>Objetivos</b>                                  | 4  |
| 1.3 | <b>Levantamento de Requisitos</b>                 | 5  |
| 1.4 | <b>Tecnologias adotadas</b>                       | 5  |
| 1.5 | <b>Plano de Desenvolvimento</b>                   | 7  |
| 2   | <b>Modelação</b>                                  | 8  |
| 2.1 | <b>Modelo de Domínio</b>                          | 8  |
| 2.2 | <b>Diagrama de Classe</b>                         | 9  |
| 2.3 | <b>Diagrama de Use Cases</b>                      | 9  |
| 2.4 | <b>Modelo de Base de Dados</b>                    | 10 |
| 3   | <b>Especificação de Use Cases</b>                 | 10 |
| 4   | <b>Mockups</b>                                    | 11 |
| 5   | <b>BackEnd</b>                                    | 13 |
| 5.1 | <b>Base de Dados</b>                              | 15 |
| 6   | <b>BackOffice</b>                                 | 16 |
| 7   | <b>FrontEnd</b>                                   | 17 |
| 7.1 | <b>Ferramentas</b>                                | 17 |
| 7.2 | <b>Ligaçāo à Base de Dados</b>                    | 20 |
| 8   | <b>Dificuldades no desenvolvimento do projeto</b> | 22 |
| 8.1 | <b>Plano inicial</b>                              | 22 |
| 8.2 | <b>Pandemia</b>                                   | 22 |
| 8.3 | <b>Implementaçāo de Serviços de Localizaçāo</b>   | 22 |

|     |  |    |
|-----|--|----|
| 8.4 | <b>Funcionalidades por Implementar</b> | 23 |
| 9   | <b>Resultado Final</b>                 | 23 |
| 9.1 | <b>FrontEnd</b>                        | 23 |
| 9.2 | <b>BackOffice</b>                      | 27 |
| 10  | <b>Conclusão</b>                       | 30 |
| A   | <b>Use Cases</b>                       | 31 |

# **1 Introdução**

No âmbito da unidade curricular de Laboratórios de Informática IV, foi proposto desenvolver um projeto.

A aplicação desenvolvida foi denominada de ”Meeet”. Neste relatório é demonstrado todo o processo de desenvolvimento deste projeto, desde a motivação e levantamento de requisitos até à demonstração final, passando pela modelação e implementação.

O objetivo final passa por demonstrar que o projeto é viável, útil e super funcional.

## **1.1 Contextualização**

Muitas vezes deparamo-nos com situações em que temos a necessidade de marcar eventos, tanto em contexto profissional como recreacional. A marcação desses eventos nem sempre é fácil, tanto pela quantidade de pessoas envolvidas, que se traduz em muita confusão e spam, como pelos fracos softwares disponíveis. Para resolver todos estes problemas, decidimos desenvolver o ”Meeet”.

”Meeet” é uma aplicação que permite ajudar com a marcação e organização de eventos. Esta será fácil de usar, e irá incluir muitos extras de modo a facilitar a vida, não só ao criador do evento, mas também aos participantes!

## **1.2 Objetivos**

O objetivo desta aplicação é facilitar a criação de eventos. Para isto, apresentamos algumas ferramentas como a criação de evento, que é bastante simples e rápido. O utilizador também poderá criar grupo de amigos facilitando assim também a tarefa. Esses amigos são geridos pelo utilizador.

A alteração da data ou hora do evento poderá ser feita a qualquer altura (pelo utilizador), tendo apenas que fazer essa mudança manualmente, e o programa encarregar-se-á de notificar todos os convidados, poupando assim trabalho e tempo ao utilizador.

A aplicação irá conter um sistema de localização dos participantes, podendo

estes pedir partilha de localização entre eles, de modo a facilitar o encontro dos mesmos.

### 1.3 Levantamento de Requisitos

Ao planejar todo este projeto, levantou-se um conjunto de requisitos minimos que seriam possíveis realizar na aplicação que serão listados de seguida. A aplicação permitirá a autenticação dos seus utilizadores, não sendo possível utilizar a aplicação sem essa mesma autenticação. Cada utilizador poderá pesquisar por outros utilizadores, e se pretender, enviar pedido de amizade. Este terá também a opção de aceitar ou rejeitar os pedidos que receber. Cada utilizador terá a sua própria lista de amigos, podendo este criar grupos de amigos, facilitando assim o convite para um evento. Um utilizador poderá criar um evento, tendo este que preencher alguns campos. O utilizador escolhe depois da sua lista de conhecidos aqueles que quer convidar para o evento. Cada utilizador que é convidado para um evento recebe uma notificação e indica se irá ou não comparecer. A aplicação deverá permitir que os utilizadores partilhem as suas localizações uns com os outros, se assim o desejarem, para que seja mais fácil que todos se encontrem na altura de um evento, e para que saibam do paradeiro uns dos outros. Quando a marcação de um evento, a aplicação deverá permitir a consulta da informação meteorológica na altura do evento.

### 1.4 Tecnologias adotadas

#### Base de Dados

Para o seu desenvolvimento desta aplicação iremos recorrer a uma base de dados SQL, utilizando o software MySQL WorkBench para modelar, traduzindo depois esta linguagem para SQL, sendo esta *hosted* pelo *Azure*.

## **API RESTful/Back End**

A aplicação faz uso de uma API Restful par controlar o acesso a base de dados sendo que esta é implementada em C#, o hosting desta API é mais uma vez feito pelo *Azure*. São ainda chamadas na aplicação outras API externas

## **Front End**

O Front-End onde são feitas as chamadas para a API foi implementado na framework de Javascript React Native. Esta tecnologia desenvolvida pelo facebook (o que oferece garantias relativamente a sua longevidade) é capaz de gerar código nativo android que tem uma performance melhor relativamente a webapps, mas simplificando no entanto o desenvolvimento de interfaces gráficas. Este tipo de tecnologia tem como a sua grande vantagem a partilha da grande maioria do código entre Android e iOS, assim embora o grupo tenha desenvolvido uma aplicação android, seria relativamente simples expandir o seu mercado para iOS.

## **Backoffice**

O backoffice aparece neste projeto na forma de um site criado com o uso de ReactJS, a escolha e adaptação do grupo a este framework foi rápida uma vez que partilha várias semelhanças com React Native.

## 1.5 Plano de Desenvolvimento

Este projeto tinha como planeamento inicial o seguinte diagrama.



O projeto inicial tinha um planeamento para 3 meses, como é visível na imagem. Conforme o decorrer deste projeto, foram surgindo algumas dificuldades e alguns obstáculos, levando a alguns ajustes das datas no plano do desenvolvimento. Com o adiamento da entrega (em 2 semanas) foi dada oportunidade de melhorar alguns aspectos visuais do projeto, assim como a implementação de algumas funcionalidades extras.

## 2 Modelação

Para uma melhor gestão do projeto, representaram-se os seguintes modelos iniciais. De notar, que com a evolução do projeto, alguns destes modelos sofreram alterações, como será visível mais à frente neste relatório.

### 2.1 Modelo de Domínio

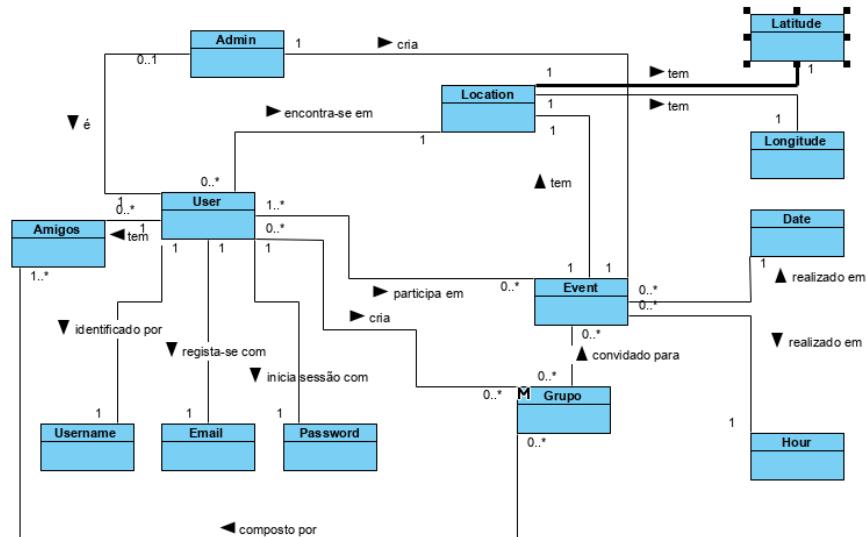


Figura 1: Modelo de Domínio

## 2.2 Diagrama de Classe

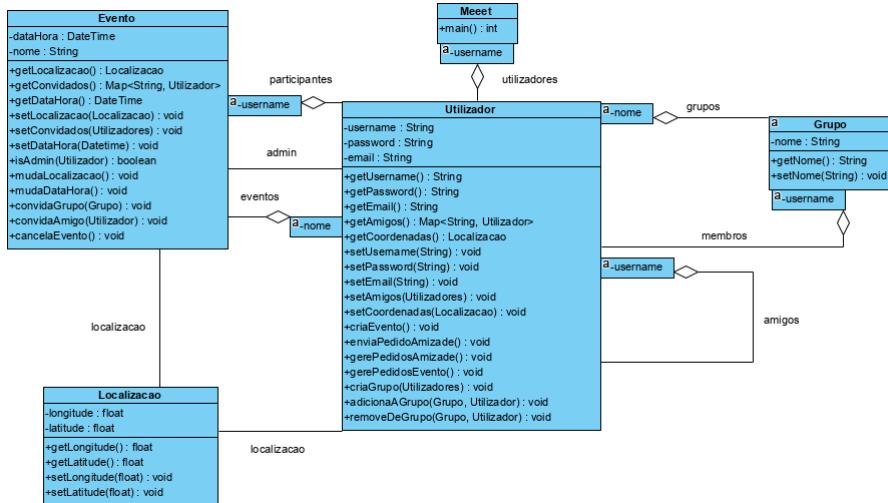


Figura 2: Diagrama de Classe

## 2.3 Diagrama de Use Cases

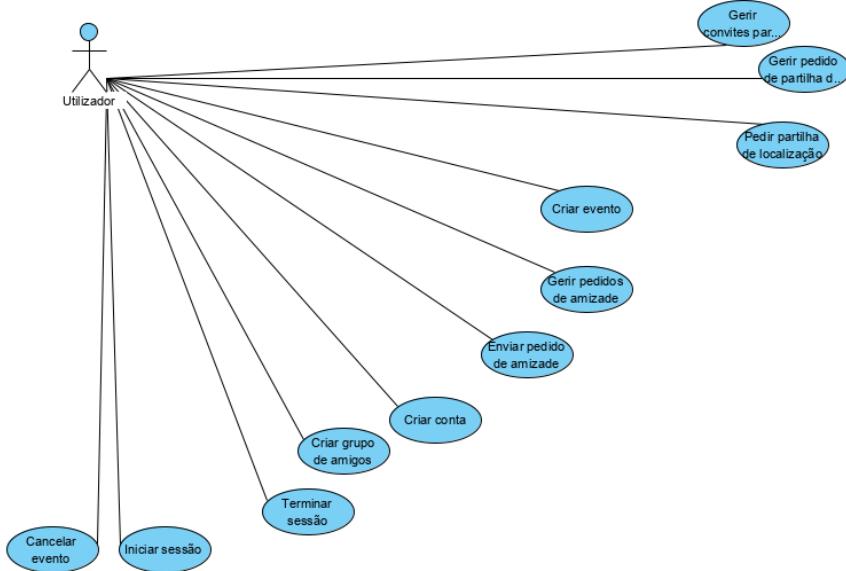


Figura 3: Diagrama de Use Cases

## 2.4 Modelo de Base de Dados

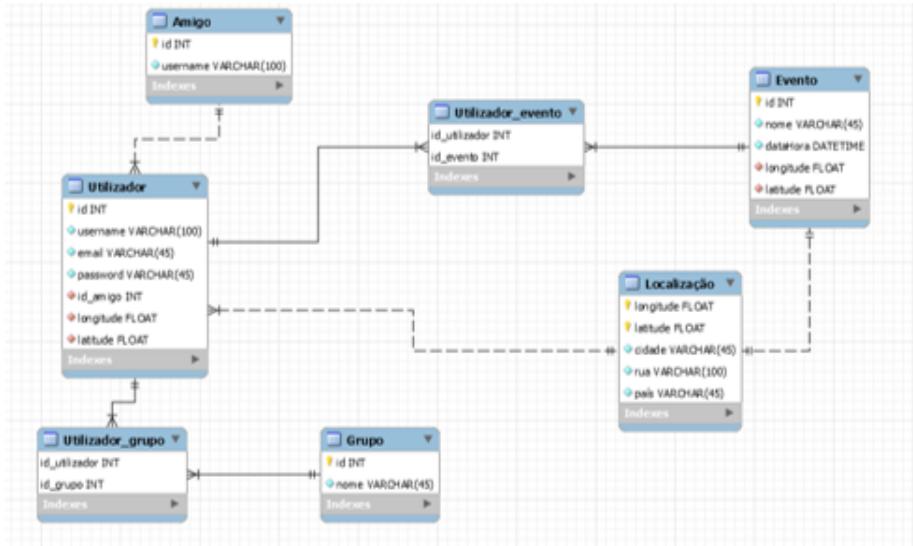


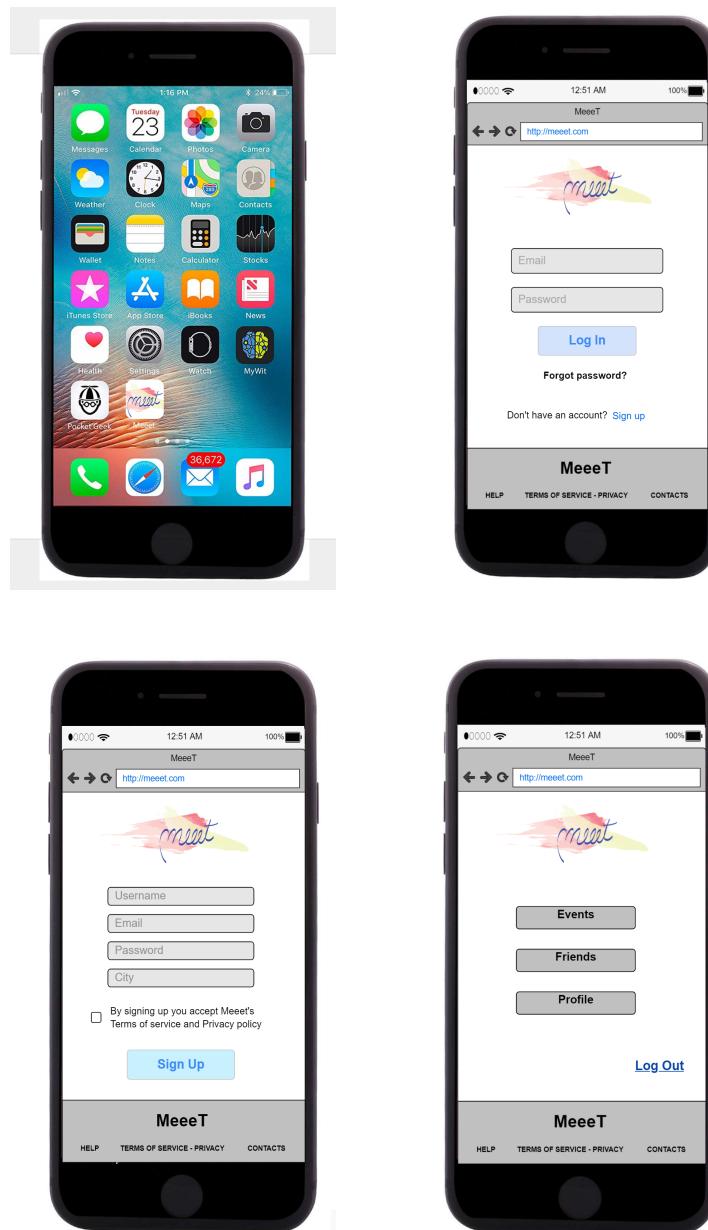
Figura 4: Modelo de Base de Dados Inicial

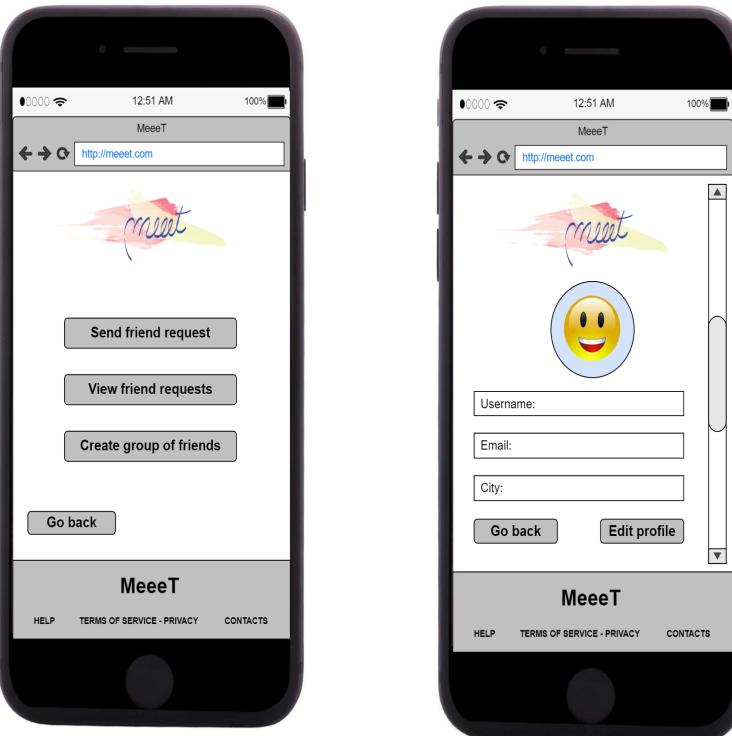
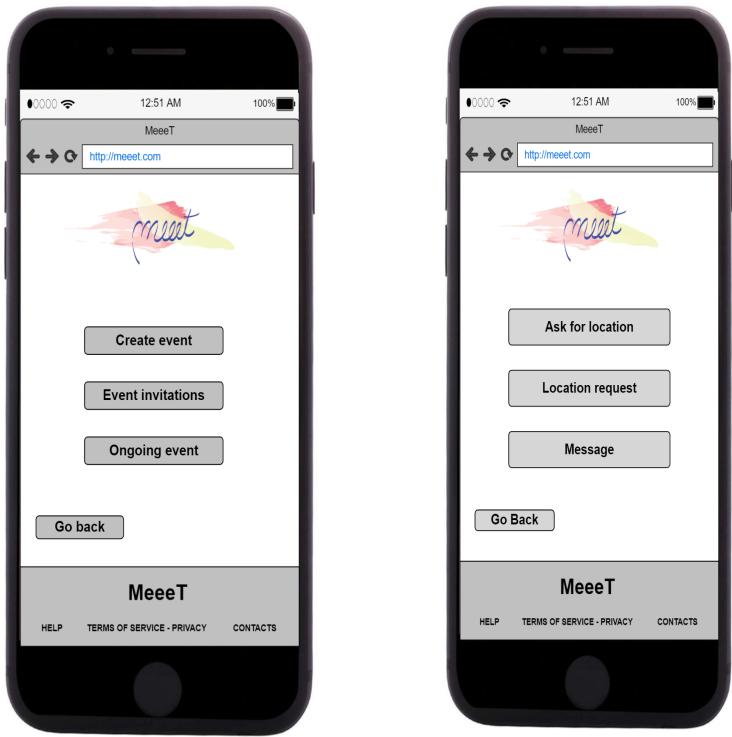
## 3 Especificação de *Use Cases*

Os use cases foram expandidos e detalhados e encontram-se disponíveis no Anexo A.

## 4 Mockups

Nesta secção serão demonstradas as mockups iniciais do projeto, as quais serviram de fio condutor para o desenvolvimento da interface da aplicação, de modo que o resultado final fosse algo parecido, ainda que durante o processo sofresse algumas alterações.





## 5 BackEnd

Para o desenvolvimento da aplicação, do lado do *backend*, a linguagem de programação que optamos por usar foi o *C#*. O nosso objetivo nesta fase do trabalho foi arquitetar uma *REST API (Representational State Transfer)*, criando assim um *Web service RESTful*. Assim, a nossa aplicação funcionará com a utilização de métodos *HTTP*, entre os quais principalmente *GET*, *PUT*, *POST* e *DELETE*. Para a implementação destes métodos, foi criada uma classe *MeeeTController.cs*. Um exemplo destes métodos é o seguinte, que retorna um *Utilizador*, pesquisando-o pelo seu *id*.

---

```
[Route("getUser/{id:int}")]
[HttpGet]
public Utilizador GetUser(int id)
{
    return _context.Utilizador.Find(id);
}
```

---

Estes métodos operam sobre informação que temos presente numa base de dados implementada utilizando o serviço da *Microsoft, Azure*. É utilizando este serviço que damos *host* ao código *backend* e à base de dados da aplicação, permitindo assim que a aplicação opere *online*. Para o funcionamento da aplicação, foi necessário estabelecer uma conexão entre o *backend* e a base de dados, o que permite à aplicação aceder à informação da base de dados, e permite ainda adicionar/remover informação conforme a necessidade. Para tal, a metodologia usada foi fazer *scaffold* das tabelas da base de dados para ficheiros *.cs* e para um ficheiro *context*, que contém toda a informação da base de dados, assim como também todas as relações entre as várias tabelas desta. Assim, criou-se uma variável *\_context* a partir da qual acedemos à informação da base de dados, evitando usar código *SQL*.

Nesta fase do projeto, uma das maiores dificuldades com que nos deparamos foi perceber que na chamada das funções no *frontend*, por cada *http request*, só

podemos ter um acesso à base de dados, pois cada *request* só pode aceder a uma tabela da base de dados de cada vez. Ou seja, se numa chamada de uma função quisesse aceder a informação na tabela de utilizadores, não podia na mesma chamada aceder à tabela de eventos.

A partir das tabelas da base de dados, foram geradas classes em ficheiros *.cs*. Um exemplo de uma destas classes é a classe *Evento.cs*, apresentada seguidamente:

---

```
using System;
using System.Collections.Generic;

namespace TodoApi.DB
{
    public partial class Evento
    {
        public Evento()
        {
            EventoHasRequests = new HashSet<EventoHasRequests>();
            UtilizadorEvento = new HashSet<UtilizadorEvento>();
            Votacao = new HashSet<Votacao>();
        }

        public int Id { get; set; }
        public string Nome { get; set; }
        public DateTime DataHora { get; set; }
        public float Longitude { get; set; }
        public float Latitude { get; set; }
        public int TipoEvento { get; set; }
        public int IdAdmin { get; set; }
        public string Descricao { get; set; }
        public int? IdadeMinima { get; set; }

        public virtual Utilizador IdAdminNavigation { get; set; }
        public virtual ICollection<EventoHasRequests> EventoHasRequests
        {
            get; set; }
        public virtual ICollection<UtilizadorEvento> UtilizadorEvento {
            get; set; }
        public virtual ICollection<Votacao> Votacao { get; set; }
    }
}
```

---

## 5.1 Base de Dados

Para a nossa base de Dados utilizamos um modelo relacional. Foi criado um modelo através do MySQL Workbench que permitisse armazenar todas as informações necessárias para as várias funções da aplicação. Ao longo do processo de implementação, a Base de Dados foi alterada várias vezes, uma vez que surgiam novas ideias ou simplesmente pelo facto de o que tínhamos não ser o mais ideal para o que precisávamos. Se observarmos a figura 4 (modelo incial da base de dados) percebemos o quanto simples ela era e para a implementação, por exemplo, de votações precisaríamos de mais entidades. Foram portanto adicionadas várias novas entidades que nos ajudassem a chegar ao objetivo final. A versão final da Base de Dados tem o seguinte aspeto:

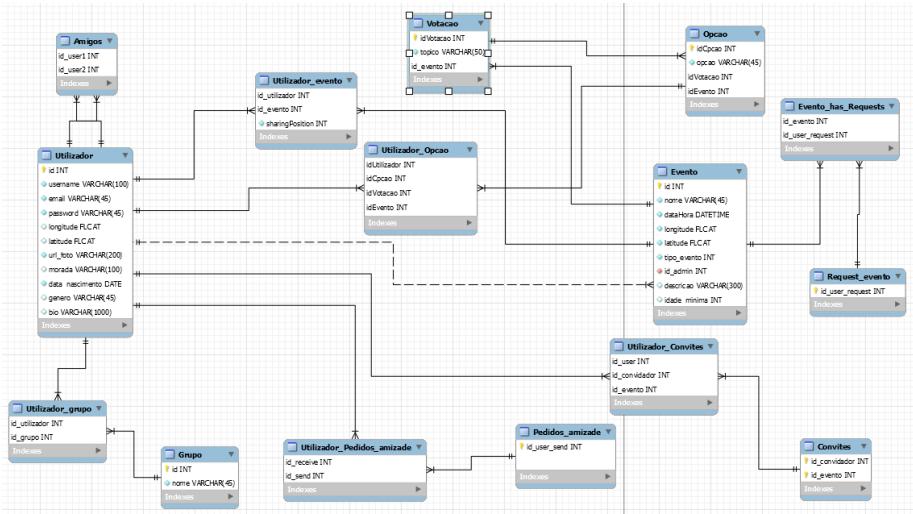


Figura 5: Versão Final da Base de Dados

As entidades principais entidades da Base de Dados são:

- **Utilizador** armazena informação relativa aos utilizadores da aplicação desde dados pessoais até ao email e password da sua conta;
- **Grupo** armazena informação relativa a cada grupo de utilizadores criado. Cada utilizador pode pertencer a vários grupos ou a nenhum;
- **Evento** armazena informação relativa a eventos criados por utilizadores,

desde informações básicas como nome, hora, data a outros tipos de informação como o tipo de evento, descrição etc;

- **Pedidos amizade** armazena informação relativa aos vários pedidos de amizade entre utilizadores;
- **Convites** armazena informação relativa aos vários convites para eventos feitos entre utilizadores desde o Id do utilizador que convidou ao Id do evento para o qual o convite foi feito;
- **Request Evento** armazena o Id de utilizadores que pedem para aderir a um dado evento;
- **Opcão** armazena informação relativa a uma dada opção de uma dada votação;
- **Votacao** armazena informação relativa a uma votação entre utilizadores, por exemplo para escolher uma hora para um evento;
- **Amigo** armazena pares de Ids de utilizadores amigos na aplicação;

Terminada a Base de Dados, esta foi depois transferida para o Azure de modo a ficar Online.

## 6 BackOffice

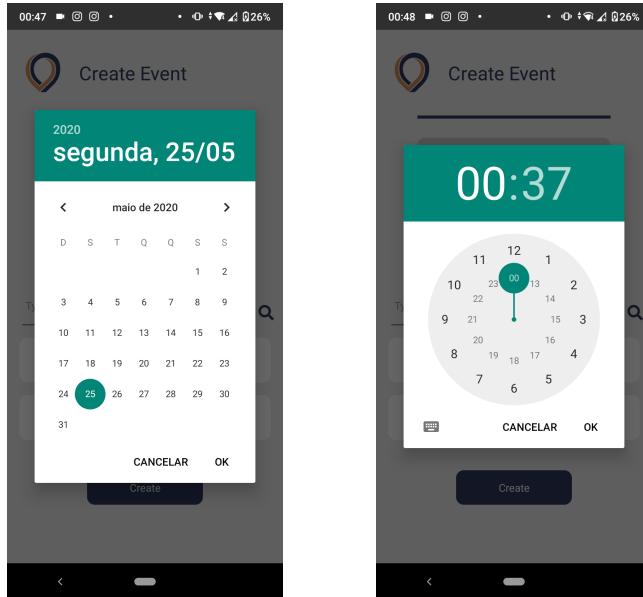
Quanto ao *backoffice*, desenvolvemos também um *site* a que damos *host* no serviço *Azure*, e este foi desenvolvido em *ReactJS*. A criação de um *backoffice* permitiu um acesso mais fácil à informação contida na base de dados, facilitando assim consultas, alterações e adições de informação.

## 7 FrontEnd

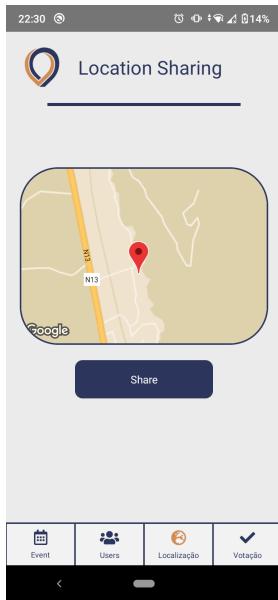
O desenvolvimento do *FrontEnd* foi todo realizado em *React Native*. Sendo esta uma framework de JS, algo que os elementos do grupo nunca tinham manejado, esta revelou algumas dificuldades ao inicio, mas à medida que o projeto foi evoluindo, essas dificuldades foram ultrapassadas.

### 7.1 Ferramentas

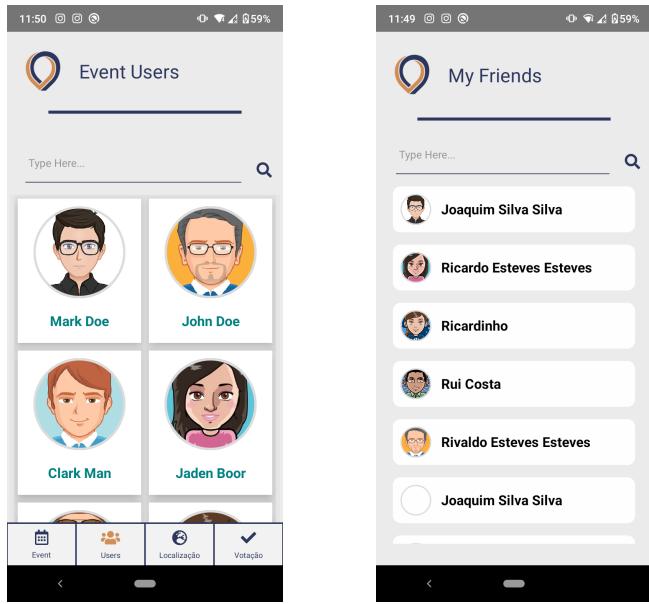
Esta linguagem permitiu o desenvolvimento das interfaces sendo possível tornar a aplicação bastante apelativa e com funcionalidades engraçadas e úteis, muitas destas já incluidas nas bibliotecas do *React Native*, sendo de fácil aplicação, como, por exemplo, o uso do calendário para ser possível escolher uma data, assim como o uso de um relógio para escolher uma hora.



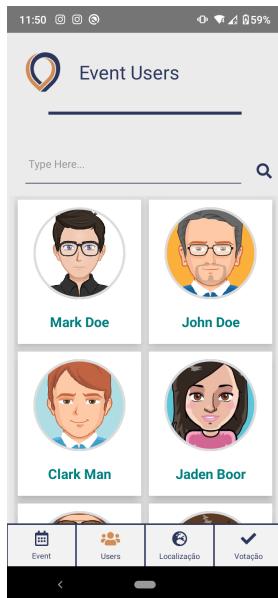
Outra ferramenta que foi possível implementar foi a partilha de localização entre utilizadores, através da API do google maps, permitindo assim que os utilizadores se consigam encontrar mais facilmente quando um evento está a decorrer.



Para ser possível trabalhar com uma lista de utilizadores, recorremos à funcionalidade **FlatList**. Esta potenciou um fácil, prático e elegante manejamento de um conjunto de utilizadores. Na imagem da direita, abaixo demonstrada , é possível visualizar uma lista bastante simples, demonstrando a foto de perfil do utilizador e o seu respetivo nome. Já na imagem da esquerda, a lista já está um pouco mais trabalhada, colocando os utilizadores '2 a 2' demonstrando na mesma a foto de perfil e o respetivo nome.



Uma funcionalidade que foi possível implementar foi a **Tab Navigation**. Através desta funcionalidade, foi possível implementar uma *tabnavigation* em que é possível alterar entre 4 *screens* diferentes, sendo estes todos referentes ao evento, como é visível no final da imagem abaixo.



## 7.2 Ligação à Base de Dados

A ligação do *front end* à **Base de Dados** (*back end*) é feito através da ferramenta **fetch**. Esta permite fazer diferentes funções, sendo as mais utilizadas neste projeto: **GET**, **POST**, **PUT** e **DELETE**.

Referente ao **GET**, esta função é bastante simples e tem como objetivo obter informação armazenada na base de dados. Para isso será necessário colocar o link desejado e esperar pela resposta. No seguinte exemplo, é possível verificar uma chamada à *back end* obtendo a informação de um determinado utilizador.

---

```
useEffect(() => {
  fetch(`https://meeet-projeto.azurewebsites.net/api/meeet/getUser/
    + global.userID)
    .then((response) => response.json())
    .then((json) => {
      setUserData(json);
    })
    .catch((error) => {
      console.error(error);
    })
    .finally(() => { setLoading(false) });
}, []);
```

---

Quanto ao **POST**, é necessário já ter preenchido o *body* anteriormente, sendo apenas depois necessário colocar o link desejado. No seguinte exemplo, é possível verificar um *post* à *back end* enviando a informação de um novo utilizador.

---

```
useEffect(() => {
  fetch(`https://meeet-projeto.azurewebsites.net/api/meeet/PostUser`,
    {
      method: 'POST',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(userInfo)
    });
}, []);
```

---

O **PUT**. Este tem como objetivo atualizar informação já existente na base de dados. Para isso é necessário já ter preenchido o *body* anteriormente, sendo apenas depois necessário colocar o link desejado. No seguinte exemplo, é possível verificar um *put* à *back end* enviando a nova informação de um utilizador já existente.

---

```
useEffect(() => {
  fetch(`https://meeet-projeto.azurewebsites.net/api/meeet/updateuser/' +
    + global.userID, {
      method: 'PUT',
      headers: {
        Accept: 'application/json',
        'Content-Type': 'application/json'
      },
      body: JSON.stringify(userInfo)
    })
    .catch((error) => {
      console.error(error);
    });
}, []);
```

---

Por último, o **DELETE**. Este apenas serve para apagar informação da base de dados. Para isso é necessário mandar no *body* o elemento que se pretende eliminar da BD.

## **8 Dificuldades no desenvolvimento do projeto**

O desenvolvimento do projeto não seguiu totalmente o plano e conceito desenhados inicialmente, devido a mudanças do foco da aplicação e também em circunstâncias externas à mesma o grupo foi levado a tomar decisões relativamente à sua implementação.

### **8.1 Plano inicial**

Uma das falhas do desenvolvimento deste projeto passou pela definição de um plano de trabalhos que não teve em conta épocas de exames e a disponibilidade do grupo durante diversos espaços de tempo, sendo que estes fatores diminuíram consideravelmente a produtividade dos elementos. Esta falha acabou por ter um impacto negativo na produção da aplicação.

### **8.2 Pandemia**

Devido às circunstâncias impostas pela situação de pandemia vivida no país o grupo foi forçado a alterar consideravelmente o seu método, sendo que o trabalho a distância e a paragem momentânea das actividades lectivas acabaram por dificultar o cumprimento do plano definido.

### **8.3 Implementação de Serviços de Localização**

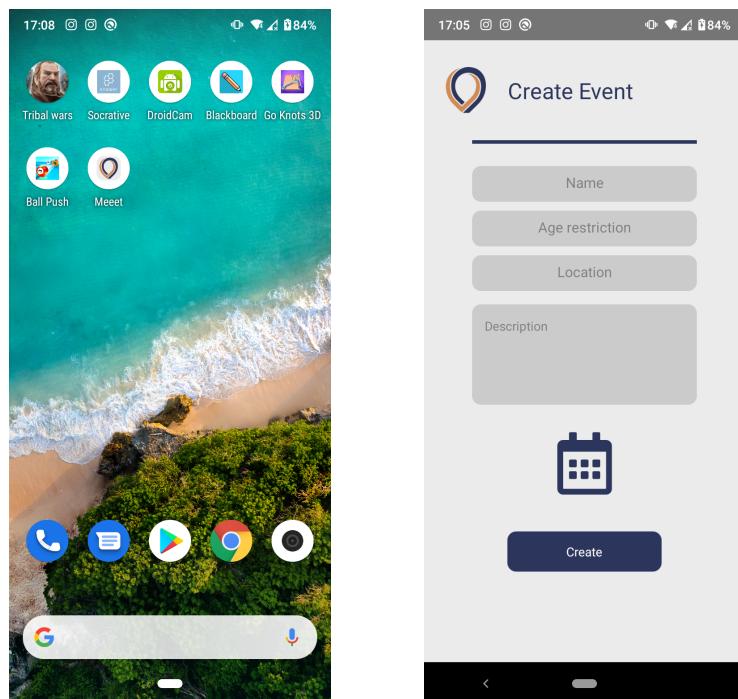
Uma das funcionalidades que o grupo definiu inicialmente como sendo importantes para o destaque da aplicação de outras com serviços semelhantes seria o da opção de partilha de localização pessoal para o evento, inicialmente o grupo procurou implementar a partilha em tempo real, no entanto, quando este serviço se revelou demasiado complexo para o foco do projeto o grupo optou por manter a funcionalidade usando no entanto a localização instantânea guardando-a no user no momento da partilha.

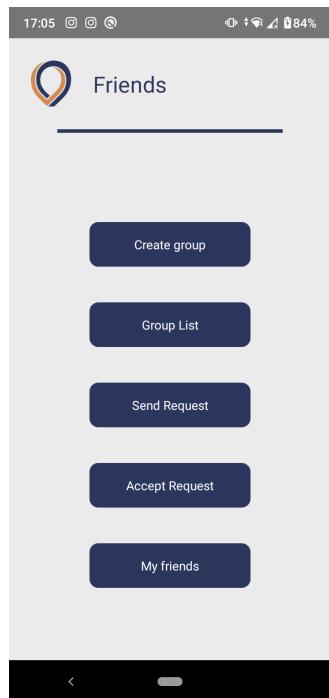
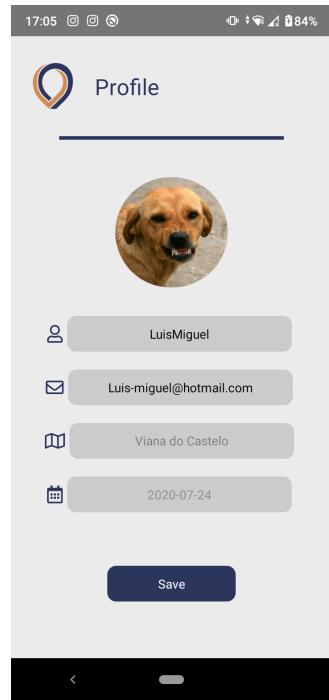
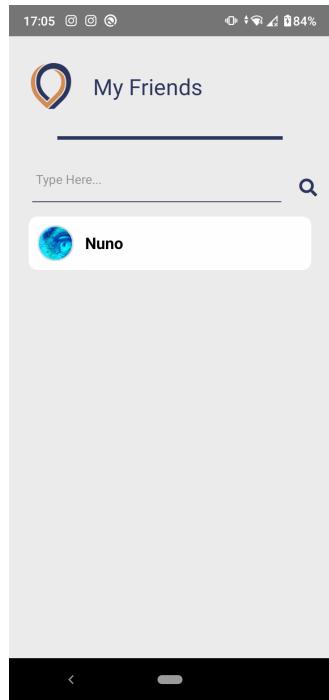
## 8.4 Funcionalidades por Implementar

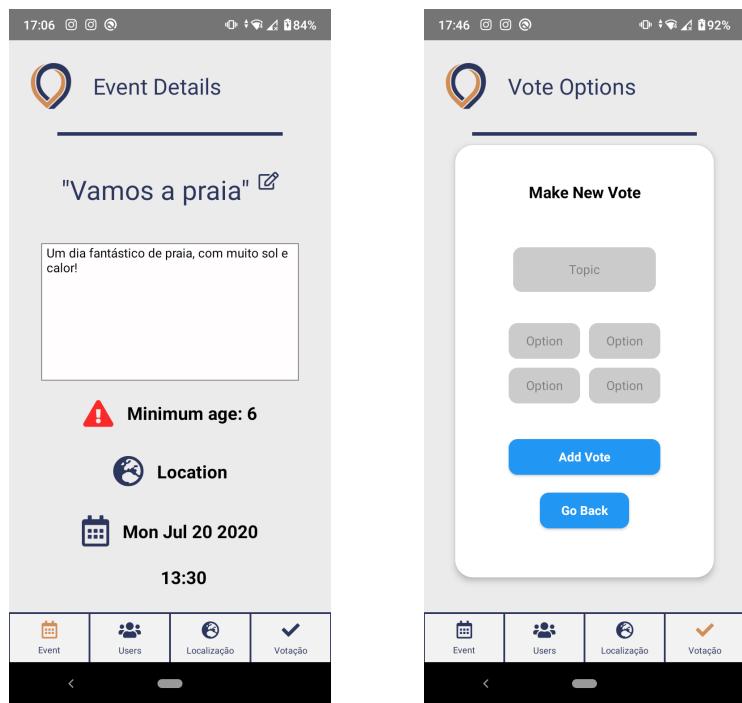
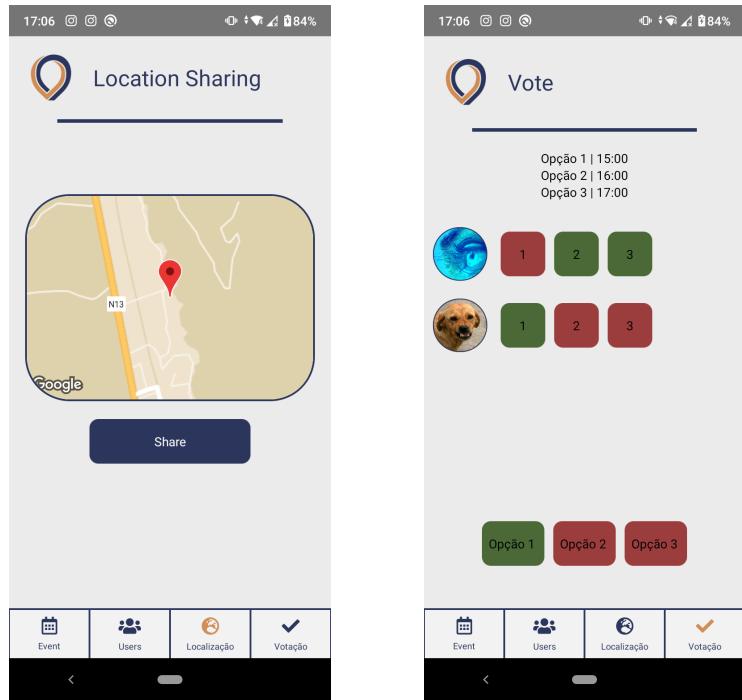
Devido aos motivos descritos acima o grupo viu-se forçado a excluir funcionalidades pensadas inicialmente como por exemplo a consulta meteorológica para o dia do evento e as notificações, estas seriam no entanto prioridades para uma fase seguinte do projeto.

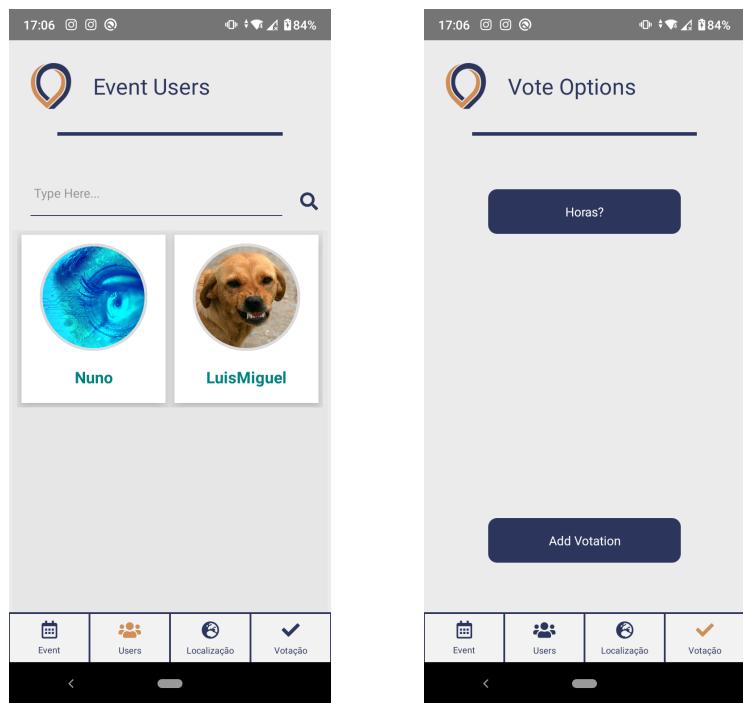
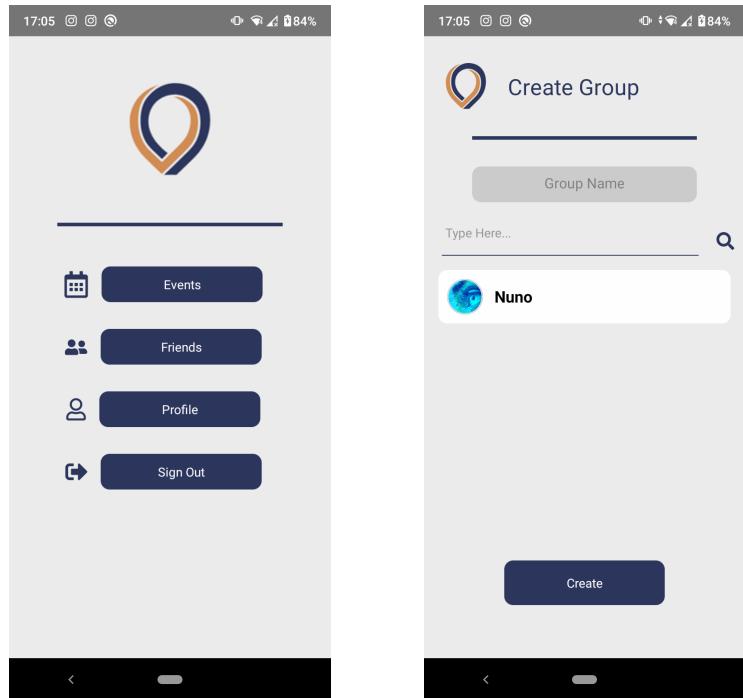
# 9 Resultado Final

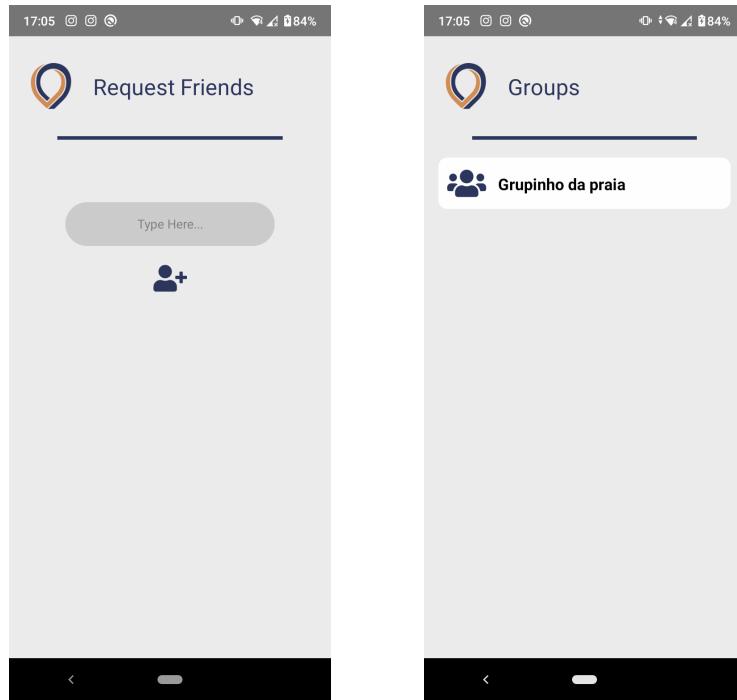
## 9.1 FrontEnd











## 9.2 BackOffice

MeeetWebsite / Users: 8 / Events: 1

[Home](#) [Users](#) [Events](#) [Groups](#) [Friends](#)

### Bem vindo ao backoffice da aplicacao Meeet!

Aqui pode facilmente dar manage em dados presentes na base de dados como:

- Os users, atraves do Users
- Os eventos, atraves do Events
- Os grupos, atraves do Groups
- Os amigos, atraves do Friends

Dentro destes pode facilmente

- **Ver que dados estao presentes na BD** pois estes estao presentes numa *table* de facil interpretacao.
- **Adicionar novos dados e relacoes** usando os botoes de *Add*
- **Remover por completo o dado que esta a dar manage (e todas as suas relacoes!)** usando os botoes de *Remove*

## All Users

This component demonstrates fetching data from the server.

| <b>Id</b> | <b>Username</b> | <b>Email</b>            | <b>Password</b> | <b>Latitude</b> | <b>Longitude</b> | <b>Morada</b>    |
|-----------|-----------------|-------------------------|-----------------|-----------------|------------------|------------------|
| 1         | Klezio          | legamerk@email.fixe     | amoAmbox        | 41.62541        | -8.77518         | Minha casa       |
| 2         | Ramos           | bro@que.queres          | ok              | 0               | 0                | tuga moradia     |
| 3         | luis pedro      | luis@pedro.pt           | luispedro       | 0               | 0                | luis pedro       |
| 4         | Nuno            | Nuno@nuno.com           | nuno            | 37.421997       | -122.084         | Nuno city        |
| 5         | LuisRamos       | Luis@gmail.com          | lulia           | 41.73433        | -8.856936        | Braga            |
| 6         | Azevedo         | Azevedo@azevedo.com     | azevedo         | 0               | 0                | Azevedo city     |
| 7         | Alvez           | Alves@alves.com         | alves           | 41.643623       | -8.777719        | Alves city 2     |
| 9         | LuisMiguel      | Luis-miguel@hotmail.com | luis            | 0               | 0                | Viana do Castelo |

Username:  Email:  Password:  Morada:

Id:

## All Events

This component demonstrates fetching data from the server.

| <b>Id</b> | <b>Nome</b>      | <b>Data Hora</b>    | <b>Longitude</b> | <b>Latitude</b> | <b>Id Admin</b> |
|-----------|------------------|---------------------|------------------|-----------------|-----------------|
| 21        | Vamos a praia    | 2020-07-20T13:30:00 | -8.8214035       | 41.698658       | 9               |
| 22        | Shops            | 2020-07-31T18:14:00 | 0                | 0               | 9               |
| 23        | Sessao de Estudo | 2020-07-05T16:55:00 | 0                | 0               | 4               |
| 24        | Caminhada        | 2020-07-08T15:23:00 | 0                | 0               | 4               |
| 25        | Concerto         | 2020-07-09T15:23:00 | 0                | 0               | 4               |
| 26        | Limpar a casa    | 2020-07-05T15:23:00 | 0                | 0               | 4               |

Nome:  Latitude:  Longitude:  Id de Admin:

Id de Evento:  Id de User:   Id:

## All Users participating in Event

Waiting...

## All Groups

This component demonstrates fetching data from the server.

| Id | Nome              |
|----|-------------------|
| 5  | Grupinho da praia |

Nome:   Id de Grupo:  Id de User:   
 Id:

## All Friends

This component demonstrates fetching data from the server.

*Waiting...*

Id de user:

## **10 Conclusão**

Em suma, cremos que neste projeto alcançamos os objetivos que tínhamos delineado, fazendo um balanço positivo do desenvolvimento do projeto, pelo que sentimos que foi desafiante e motivante e que aprendemos com o desenvolvimento deste. Ao longo da implementação deste projeto conseguimos também ganhar algum conhecimento em algumas linguagens e softwares com os quais nunca tinhamos lidado, tais como C#, React Native, React JS e o Azure. Podemos com este trabalho, trabalhar sobre uma aplicação segundo uma metodologia realista e que envolveu vários conceitos de várias UCs que tivemos, num só projeto, e também envolveu outros conceitos que não aprendemos na universidade, envolvendo assim todo um trabalho de pesquisa e aprendizagem que, sem dúvida, contribuiu para a nossa preparação para o mundo empresarial e para a potencialização da nossa maturidade como futuros engenheiros.

# Apêndice A

## Use Cases

### UC 1: Criação de conta

**Descrição:** Um utilizador insere os seus dados e cria uma conta no sistema.

**Pré-condição:**

**Pós-condição:** Novo utilizador é adicionado ao sistema.

**Fluxo normal:**

1. Utilizador seleciona opção ‘Sign Up’.
2. Utilizador insere o seu email, username e password.
3. Sistema valida dados inseridos.
4. Conta é criada com sucesso e um novo utilizador é adicionado ao sistema.

**Fluxo alternativo:** [Username já está registado no sistema](Passo 4)

**4.1** Sistema avisa que já há uma conta com aquele username registado.

**4.2** Salta para passo(2).

**Fluxo de exceção:** [Email já está registado no sistema](Passo 4)

**4.1** Sistema avisa que já há uma conta criada com o email dado pelo utilizador.

**4.2** A ação da criação de conta é cancelada.

## **UC 2: Iniciar sessão**

**Descrição:** Utilizador inicia sessão na sua conta.

**Pré-condição:**

**Pós-condição:** Utilizador tem sessão iniciada.

**Fluxo normal:**

- 1.** Utilizador seleciona opção ‘LogIn’.
- 2.** Utilizador insere o seu email e password.
- 3.** Sistema valida dados inseridos.
- 4.** Utilizador tem sessão iniciada.

**Fluxo de exceção 1:** [Email não registado no sistema](Passo 4)

**4.1** Sistema indica que o username inserido não existe.

**4.2** A ação é cancelada.

**Fluxo de exceção 2:** [Password inserida está incorreta](Passo 4)

**4.1** Sistema indica que a password inserida não está correta.

**4.2** A ação é cancelada.

## **UC 3: Terminar sessão**

**Descrição:** Utilizador termina sessão na sua conta.

**Pré-condição:** Utilizador tem sessão iniciada.

**Pós-condição:** Utilizador não tem sessão iniciada.

**Fluxo normal:**

- 1.** Utilizador seleciona opção ‘LogOut’.
- 2.** Sistema termina a sessão.

## **UC 4: Enviar pedido de amizade**

**Descrição:** Um utilizador envia um pedido de amizade a outro utilizador.

**Pré-condição:** Utilizador tem sessão iniciada.

**Pós-condição:** Novo pedido de amizade é efetuado.

### **Fluxo normal:**

- 1.** Utilizador seleciona opção ‘Enviar pedido de amizade’.
- 2.** Utilizador insere o username do utilizador a quem quer enviar o pedido.
- 3.** Sistema procura o utilizador e verifica se já são amigos.
- 4.** Sistema apresenta o utilizador no ecrã.
- 5.** Sistema inquere ao utilizador se quer proceder com a ação.
- 6.** Utilizador afirma que quer continuar.
- 7.** Sistema processa o pedido e este é enviado ao outro utilizador.

**Fluxo de exceção 1:** [O username não existe]](Passo 4)

**4.1** Sistema avisa que o username não existe.

**4.2** A ação é cancelada.

**Fluxo de exceção 2:** [Utilizador já tem adicionado o outro utilizador na sua lista de amigos](Passo 4)

**4.1** Sistema avisa que já são amigos.

**4.2** A ação é cancelada.

**Fluxo de exceção 3:** [Utilizador não quer continuar](Passo 5)

**5.1** A ação é cancelada.

## **UC 5: Gerir pedidos de amizade recebidos**

**Descrição:** Um utilizador consulta a sua lista de pedidos de amizade recebidos e decide aceitar ou rejeitá-los.

**Pré-condição:** Utilizador tem sessão iniciada.

**Pós-condição:** Utilizador aceitou ou rejeitou pedidos recebidos.

**Fluxo normal:**

1. Utilizador seleciona opção ‘Gerir pedidos de amizade’.
2. Sistema apresenta a lista de pedidos de amizade.
3. Utilizador seleciona determinado pedido.
4. Utilizador escolhe aceitar esse pedido.
5. Sistema adiciona o username na lista de amigos do utilizador.
6. Sistema retira pedido da lista.

**Fluxo alternativo:** [Utilizador escolhe rejeitar esse pedido](Passo 4)

**4.1** Salta para passo(6).

**Fluxo de exceção:** [A lista de pedidos recebidos está vazia](Passo 2)

**2.1** Sistema avisa que lista está vazia.

**2.2** A ação é cancelada.

## **UC 6: Criar um evento**

**Descrição:** Utilizador cria o evento de uma festa e os seus amigos convidados são notificados.

**Pré-condição:** Utilizador tem sessão iniciada.

**Pós-condição:** Um novo evento é criado.

### **Fluxo normal:**

- 1.** Utilizador seleciona opção ‘Criar evento’.
- 2.** Utilizador insere os dados necessários.
- 3.** Sistema valida os dados.
- 4.** Sistema apresenta meteorologia para determinado data e hora.
- 5.** Utilizador escolhe continuar.
- 6.** Sistema apresenta lista de amigos do utilizador.
- 7.** Utilizador seleciona os amigos que pretende convidar.
- 8.** Sistema cria o evento e notifica todos os convidados.

### **Fluxo alternativo 1: [Utilizador escolhe alterar data e hora](Passo 5)**

**5.1** Utilizador insere nova data e hora.

**5.2** Salta para passo(3).

### **Fluxo alternativo 2: [Utilizador escolhe convidar um grupo pré-definido para o evento](Passo 7)**

**7.1** Utilizador seleciona o grupo que pretende convidar.

**7.2** Salta para passo(8).

### **Fluxo de exceção: [Utilizador escolhe cancelar](Passo 5)**

**5.1** A acção é cancelada.

## **UC 7: Criar um grupo de amigos**

**Descrição:** Utilizador cria um grupo com utilizadores da sua lista de amigos.

**Pré-condição:** Utilizador tem sessão iniciada.

**Pós-condição:** Utilizador tem um novo grupo de contactos.

### **Fluxo normal:**

- 1.** Utilizador seleciona opção ‘Criar grupo’.
- 2.** Utilizador insere nome do grupo.
- 3.** Sistema apresenta ao utilizador a sua lista de amigos.
- 4.** Utilizador seleciona os amigos que quer adicionar ao grupo.
- 5.** Sistema cria o grupo.

**Fluxo alternativo:** [Utilizador não seleciona nenhum amigo](Passo 5)

**5.1** Sistema avisa que não foi selecionada nenhuma amiga.

**5.2** Salta para passo(4).

**Fluxo de exceção:** [Utilizador não tem amigos na sua lista de amigos](Passo 3)

**3.1** Sistema informa que não tem amigos.

**3.2** A acção é cancelada.

## **UC 8: Pedir partilha de localização**

**Descrição:** Um utilizador pede a localização de outro para que seja mais fácil encontrarem-se.

**Pré-condição:** Ter sessão iniciada e estar a participar num evento.

**Pós-condição:** Recebe uma resposta ao pedido de partilha de localização.

### **Fluxo normal:**

- 1.** Utilizador seleciona opção ‘Pedir partilha de localização’.
- 2.** Sistema apresenta lista de pessoas presentes no evento.
- 3.** Utilizador seleciona pessoa a quem quer pedir a partilha.
- 4.** Sistema envia pedido de partilha ao outro utilizador.
- 5.** Sistema obtem resposta e recebe as coordenadas.
- 6.** Sistema apresenta localização pedida.

**Fluxo alternativo 1:** [Sistema não recebe as coordenadas](Passo 4)

- 4.1** Sistema avisa que não foi aceite o pedido.  
**4.2** Salta para passo(2).

**Fluxo alternitavo 2:** [Sistema não obtem resposta passados cinco minutos](Passo 4)

- 4.1** Sistema avisa que não obteve resposta e retira pedido de partilha.  
**4.2** Salta para passo(2).

## **UC 9: Gerir pedidos de partilha de localização**

**Descrição:** O utilizador partilha a sua localização com outro.

**Pré-condição:** Ter sessão iniciada e estar a participar num evento.

**Pós-condição:** Envia uma resposta ao pedido de partilha de localização.

### **Fluxo normal:**

- 1.** Utilizador seleciona opção ‘Gerir pedidos de partilha de localização’.
- 2.** Sistema apresenta lista de pedidos de partilha de localização.
- 3.** Utilizador seleciona pessoa a quem aceita partilha.
- 4.** Sistema envia localização ao outro utilizador.
- 5.** Sistema remove pedido da lista.

**Fluxo alternativo:** [Utilizador não escolhe ninguém](Passo 4)

**4.1** Sistema avisa que não foi escolhido ninguém.

**4.2** Salta para passo(2).

**Fluxo de exceção 1:** [Lista de pedidos vazia](Passo 2)

**2.1** A ação é cancelada.

## **UC 10: Gerir convites para eventos.**

**Descrição:** Um utilizador indica que irá comparecer a um ou mais eventos.

**Pré-condição:** Ter sessão iniciada.

**Pós-condição:** Respondeu a convite para evento.

### **Fluxo normal:**

- 1.** Utilizador seleciona opção ‘Gerir convites para eventos’.
- 2.** Sistema apresenta lista de convites para eventos.
- 3.** Utilizador seleciona eventos que pretende participar.
- 4.** Sistema retira pedidos de eventos da lista e adiciona eventos na pagina do utilizador.

**Fluxo alternativo:** [Utilizador não escolhe ninguém](Passo 4)

**4.1** Sistema avisa que não foi escolhido ninguém.

**4.2** Salta para passo(2).

**Fluxo de exceção 1:** [Lista de pedidos vazia](Passo 2)

**2.1** A ação é cancelada.