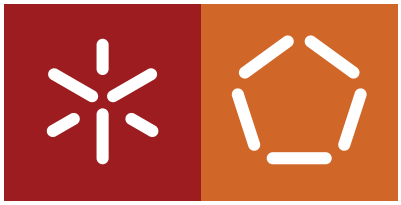


Universidade do Minho

Mestrado Integrado em Engenharia Informática

Aplicação Móvel Para Gestão De Uma Equipa Desportiva

Laboratórios de Engenharia Informática

Orientação de: *António Luís Sousa*

João Vieira (A78468)
Hugo Oliveira (A78565)
Raphael Oliveira (A78848)

25 de Junho de 2019

Resumo

Este documento descreve o processo de desenvolvimento do projeto prático da Unidade Curricular de Laboratórios de Engenharia Informática. O objetivo do projeto passa pelo estudo e implementação de uma aplicação *mobile* que sirva de apoio a um clube desportivo, neste caso, o Hóquei Clube de Penafiel, permitindo a gestão de eventos, atletas e outras informações.

Conteúdo

Lista de Figuras

Lista de Tabelas

1 Introdução

Este projeto surge com a intenção de ser desenvolvida uma aplicação *mobile* que permita ajudar no processo gestão da equipa *Hóquei Clube Penafiel*. A aplicação *web* atual, desenvolvida com o *Odoo*, carece de alguma funcionalidades e de alguns problemas de responsividade de ecrãs, nomeadamente para dispositivos móveis.

O nosso objetivo passa por selecionar um conjunto de requisitos e funcionalidades que são oferecidas através da aplicação *web* e que consideramos importantes manter na versão *mobile*, tendo em conta os vários perfis de pessoas que compõe o clube em causa.

Para que se tire proveito das funcionalidades que os dispositivos móveis oferecem, pretendemos implementar notificações para que os utilizadores da aplicação consigam ter informações relevantes o mais rápido possível, desde que estes estejam conectados à *Internet*.

Para o objetivo ser alcançado, é necessário adicionar novas funcionalidades ao *backend* da aplicação, de modo a que a nossa aplicação *mobile* consiga fornecer todas as funcionalidades previstas.

2 Contexto da aplicação móvel

Hoje em dia, é muito fácil fornecermos um serviço em dispositivos móveis. Além disso, desde muito cedo que os mais jovens começam a ter contacto com estas novas tecnologias. Deste modo, torna-se mais intuitiva para os jovens a manipulação de uma aplicação móvel do que uma aplicação *web*.

Tendo por base o estudo do domínio do problema apresentado pelo grupo responsável pelo desenvolvimento da aplicação *web*, podemos constatar que no contexto da aplicação *mobile*, devemos fornecer suporte de gestão aos seguintes **tipos de utilizadores**:

- **Atleta:** participante ativo em evento desportivos (treinos e jogos) e com alguns dados associados a si e geridos pelo clube;
- **Treinador:** responsável pela definição dos exercícios de treino, pelas convocatórias de atletas e seccionistas, etc;
- **Seccionista:** responsável por auxiliar os Treinadores;
- **Pai:** responsável por um ou mais Atletas.

Para além deste utilizadores, é necessário ter em conta que os utilizadores que podem fazer parte de vários grupos. Por exemplo, a mesma pessoa pode ser pai e treinador e, com isso, ser necessário saber o que este utilizador pode aceder na nossa aplicação.

O facto de existirem vários tipos de pessoas e intervalos de idades que utilizarão a aplicação, torna-se também um desafio desenvolver uma aplicação que seja de fácil usabilidade e interface amigável para todos os utilizadores da mesma.

O nosso foco, uma vez mais, consiste em fornecer uma aplicação que seja de fácil interação, que permita uma rápida e eficiente gestão de eventos desportivos pelos responsáveis, como por exemplo, a criação de eventos ou registar presenças e atrasos num determinado evento. Quanto ao lado dos atletas (e pais), terá de permitir uma fácil perceção da sua participação nos eventos, podendo registar a sua disponibilidade.

Por fim, para uma maior eficiência nesta gestão, o contexto de notificações para os diversos utilizadores é importante, sendo que estes podem conseguir reagir mais rápido a um determinado evento. Aliando um *chat* a este conjunto de funcionalidades, temos a certeza que todo o processo de gestão será melhorado.

3 Requisitos funcionais

O principal objetivo deste projeto, **consiste no desenvolvimento de uma aplicação móvel com o objetivo de facilitar a gestão de uma equipa desportiva**, tornando toda a gestão mais **intuitiva e facilitada** que, conforme os contextos mencionados anteriormente, seja possível proporcionarmos uma maior adesão dos utilizadores à plataforma já existente, que de momento, limita-se a um acesso via interface *web*.

Neste sentido, e dada a grande complexidade da aplicação, apresentamos em baixo, os requisitos funcionais da aplicação tendo em conta os utilizadores existentes.

3.1 Requisitos gerais

Efetuar *login* e *logout* na aplicação

Como Utilizador da aplicação, **quero** efetuar *login* e *logout*, **para** aceder e sair da aplicação.

Consultar dados pessoais

Como Utilizador da aplicação, **quero** consultar os meus dados pessoais, como o nome, o contacto telefónico, o *e-mail* e a data de nascimento, **para** poder validar os mesmo.

Redefinir a palavra-passe

Como Utilizador da aplicação, **quero** redefinir a palavra-passe de acesso à aplicação, **para** poder proteger o meu acesso à mesma.

Redefinir a imagem de utilizador

Como Utilizador da aplicação, **quero** redefinir a imagem de utilizador, **para** poder atualizar a mesma.

Consultar eventos gerais do clube

Como Utilizador da aplicação, **quero** consultar os dados sobre eventos gerais do clube, **para** poder ficar a par dos mesmos.

Aceder a um *chat*

Como Utilizador da aplicação, **quero** aceder a um *chat*, **para** poder comunicar com outros utilizadores.

3.2 Atleta

Consultar dados de atleta

Como Atleta, **quero** consultar os meus dados de atleta, como o número da camisola, o escalão e a posição em campo, **para** poder validar os mesmo.

Receber notificações sobre as minhas convocatórias

Como Atleta, **quero** receber uma notificação quando tiver sido convocado para um evento desportivo, **para** poder ficar a par do mesmo.

Consultar dados sobre as minhas convocatórias

Como Atleta, **quero** consultar os dados sobre uma dada convocatória em que esteja presente, como os treinadores, os atletas, o escalão, a hora de início e a sua localização, **para** poder ficar a par dos mesmos.

Registar indisponibilidade numa convocatória

Como Atleta, **quero** poder registar a indisponibilidade para uma convocatória, **para** indicar que não posso comparecer a um treino ou a um jogo.

3.3 Pai

Consultar dados pessoais dos filhos

Como Pai, **quero** consultar os dados pessoais dos meus filhos, **para** poder validar os mesmos.

Consultar dados sobre as convocatórias dos filhos

Como Pai, **quero** consultar os dados sobre as convocatórias dos meus filhos, como os treinadores, a hora de início e a sua localização, **para** ficar a par dos mesmos.

Registar indisponibilidade de um filho numa convocatória

Como Pai, **quero** poder registar a indisponibilidade do meu filho para uma convocatória, **para** indicar que o meu filho não pode comparecer a um treino ou a um jogo.

Receber notificações sobre as convocatórias dos filhos

Como Pai, **quero** receber uma notificação quando um filho meu for convocado para um evento desportivo, **para** ficar a par do mesmo.

Receber notificações sobre as lesões dos filhos

Como Pai, **quero** receber uma notificação quando um filho meu ficar lesionado num evento desportivo, **para** ficar a par da lesão.

3.4 Treinador

Criar um evento desportivo

Como Treinador, **quero** criar um novo evento desportivo, **para** registar a existência de um treino ou de um jogo.

Consultar dados de um evento desportivo

Como Treinador, **quero** consultar os dados de um novo evento desportivo, como a hora de início, os treinadores, os seccionistas, os atletas convocados e a sua localização, **para** validar os dados em caso de necessidade.

Editar dados de um evento desportivo

Como Treinador, **quero** editar os dados de um novo evento desportivo, como a hora de início, os treinadores, os seccionista, a sua localização e adicionar e remover atletas, **para** alterar os dados em caso de necessidade.

Editar a disponibilidade de um atleta num evento desportivo

Como Treinador, **quero** editar a disponibilidade de um atleta num evento desportivo, **para** alterar o estado do evento desportivo.

Receber notificações de indisponibilidade de um atleta

Como Treinador, **quero** receber uma notificação quando um atleta está indisponível, **para** ficar a par da situação.

Fechar período de convocatórias

Como Treinador, **quero** fechar o período de convocatórias de um evento desportivo, **para** poder avançar para a próxima fase de um evento.

Fechar evento desportivo

Como Treinador, **quero** fechar um evento desportivo, **para** poder dar como concluído o evento.

Consultar dados dos atletas

Como Treinador, **quero** consultar os dados dos atletas registados na aplicação, como o seu nome, o seu escalão, o seu número de camisola e as suas lesões **para** poder validar os mesmos.

Registar lesão de um atleta

Como Treinador, **quero** registar uma lesão de um atleta num evento desportivo, **para** ficar no seu registo.

3.5 Seccionista

Consultar dados de um evento desportivo

Como Seccionista, **quero** consultar os dados de um novo evento desportivo, como a hora de início, os treinadores, os seccionistas, os atletas convocados e a sua localização, **para** validar os dados em caso de necessidade.

Editar a disponibilidade de um atleta num evento desportivo

Como Seccionista, **quero** editar a disponibilidade de um atleta num evento desportivo, **para** alterar o estado do evento desportivo.

Receber notificações de indisponibilidade de um atleta

Como Seccionista, **quero** receber uma notificação quando um atleta está indisponível, **para** ficar a par da situação.

Fechar período de convocatórias

Como Seccionista, **quero** fechar o período de convocatórias de um evento desportivo, **para** poder avançar para a próxima fase de um evento.

Fechar evento desportivo

Como Seccionista, **quero** fechar um evento desportivo, **para** poder dar como concluído o evento.

Consultar dados dos atletas

Como Seccionista, **quero** consultar os dados dos atletas registados na aplicação, como o seu nome, o seu escalão, o seu número de camisola e as suas lesões **para** poder validar os mesmos.

Registar lesão de um atleta

Como Seccionista, **quero** registar uma lesão de um atleta num evento desportivo, **para** ficar no seu registo.

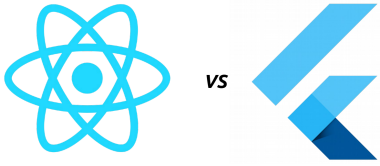
4 *Frameworks* utilizadas

Para o desenvolvimento de aplicações *mobile* existem, atualmente, várias opções de ferramentas e *frameworks* que se podem utilizar, sendo por isso necessário analisar as várias possibilidades e comparar as mesmas para perceber qual ou quais as que melhor se adequam ao objetivo pretendido.

A primeira restrição por nós introduzida na escolha de ferramentas, está relacionada ao facto de querermos desenvolver uma aplicação para a plataforma `Android` e `iOS`. Desta forma, e contrariando o que acontecia há uns anos atrás, não era de todo uma possibilidade para nós desenvolver código diferenciado para cada plataforma, pelo que, a utilização de uma *framework cross-platform* era essencial e, só aqui, um grande conjunto de *frameworks* foi posto de parte.

Existem *frameworks*, como é o caso do **Ionic**, que permite o desenvolvimento de uma aplicação *mobile cross-platform* mas que no entanto, toda a aplicação, está a correr num ambiente baseado em HTML e CSS, deste modo muito limitada em termos de *performance*. Em contra partida, permite a realização de uma interface mais apelativa, uma vez que não tem por base os componentes específicos da plataforma, como botões, *headers*, etc. Assim, excluímos logo todas as *frameworks* que não convertam o código de desenvolvimento para código nativo, de forma a fornecermos uma aplicação com segurança, *performance* e escalabilidade.

Tendo isto em conta, as mais fortes possibilidades para o desenvolvimento da aplicação passaram por utilizar **React Native** ou **Flutter**, *frameworks cross-platform* populares que permitem ter uma fonte de código única para `Android` e `iOS`, convertendo todo o desenvolvimento para código nativo.



Analisando as caraterísticas de uma e outra *framework*, percebemos que `React Native` utiliza `JavaScript` ou `Typescript` como linguagem de programação, ao passo que `Flutter` utiliza a linguagem de programação `Dart`, um pouco como uma mistura entre `JavaScript` e `Java`. Neste ponto, a nossa preferência cai para o `React Native` pelo conhecimento que os membros do grupo já tinham de `JavaScript` e não existir a necessidade de nos adaptarmos a uma nova linguagem. Deste modo é possível aplicarmos o nosso tempo a melhorias na aplicação.

Quanto ao ecossistema das duas *frameworks* em questão, percebemos que o `React Native` já se encontra no mercado há mais tempo e, como tal, tem um ecossistema mais rico que o `Flutter`, sendo que adicionalmente, o uso de `JavaScript` em `React Native` permite a utilização de várias (mas não todas) livrarias que a linguagem oferece. Para além disto, `React Native` é também bastante utilizado por grandes companhias (como o próprio *Facebook* que é o criador da mesma), garantindo desde já, uma confiança na nossa escolha.

Em jeito de conclusão neste ponto, podemos referir que `React Native` é uma *framework* mais madura e com uma maior comunidade, o que faz com que possua uma larga documentação, assim como a oferta de vários componentes prontos para

(re)utilização que poupam imenso trabalho aos *developers* de aplicações *mobile*. O facto de existirem componentes no *React Native*, permite que no futuro consigamo aplicar a mesma interface em vários ecrãs, de modo que qualquer alteração seja apenas necessária num ficheiro de código. Por todas estas razões, sentimos que a escolha certa para o nosso projeto seria utilizar o `React Native`.

Para além disto, é importante ainda referir a utilização do `Redux`, uma biblioteca *open-source* `JavaScript` utilizada para a gestão do estado da aplicação e muitas vezes utilizada em conjunto com as tecnologias `React`. Para além das caraterísticas que isto traz à aplicação, permite também aos membros do grupo ganhar um maior conforto e experiência com este tipo de técnicas cada vez mais utilizadas no mercado.

5 Desafios encontrados

Durante o processo de estudo e implementação da aplicação *mobile*, foram surgindo alguns desafios cuja resolução era crucial para a concretização da aplicação, de modo alcançarmos os nossos objetivos.

Um dos desafios passava por realizar a distinção dos tipos de utilizadores através de um menu de navegação, tendo em conta os quatro perfis apresentados anteriormente e as várias possibilidades que daí surgem, atendendo a que um utilizador pode ter mais do que um perfil (ou cargo). Esta distinção tem como objetivo selecionar qual o conteúdo e as funcionalidades oferecidas a cada tipo de perfil , de modo a termos um controlo de navegação e autorização na aplicação.

Outro dos desafios encontrados, também passou pelo estudo da inclusão das notificações na aplicação e quais as modificações que seriam necessárias de efetuar no *backend* do aplicação *web*.

Para além disto, e face ao desafio que nos foi lançado de incluirmos um *chat* totalmente funcional na aplicação, surgiu também daqui a dificuldade de perceber como iríamos interagir com o `Odoo` de forma a tornar o *chat* possível de implementar.

Por fim, fomos ainda desafiados a incluir na nossa aplicação um sistema de navegação que permita que os utilizadores se dirijam aos locais dos eventos desportivos no passo de um clique, obtendo assim, a rota de navegação.

Sendo assim, nas próximas secções é explicada a forma como cada um destes desafios foi ultrapassado.

5.1 Divisão do menu de navegação por perfis

Como foi referido anteriormente, um dos desafios encontrados foi a divisão do menu de navegação pelos variados perfis de utilizadores.

De acordo com os requisitos descritos, decidiu-se criar um `Painel de Navegação`, que pode ser acedido a qualquer momento em qualquer ecrã. Em baixo, apresentamos os diferentes itens presentes no menu de navegação com base nos perfis de utilizadores;

Utilizador Geral

- **Início:** ecrã onde o utilizador pode visualizar os seus seis eventos futuros mais recentes e, no caso de não existirem eventos associados ao utilizador, são visualizados os seis eventos futuros mais recentes em que não participe, caso existam;
- **Calendário:** ecrã onde o utilizador pode consultar um calendário, permitindo a consulta de uma forma eficiente todos os eventos em que participe e, caso queira, consultar todos os eventos gerais do clube, semelhante ao realizado no ecrã início;
- **Perfil:** ecrã onde o utilizador pode consultar o seu perfil, em que estão presentes informações como a sua data de nascimento, o seu *e-mail* e o seu nome, também tendo a possibilidade de redefinir a sua *password* e a sua imagem de perfil;

  - **Atleta:** no caso de o utilizador ser atleta, pode consultar as suas lesões e ainda estão presentes informações como o número da camisola e o escalão em que está inserido;

- **Chat:** ecrã onde o utilizador visualiza um *chat*, constituído pelas suas mensagens diretas com outros utilizadores e em canais públicos em que esteja inserido;

Treinador ou Seccionista

Além dos separadores descritos anteriormente, este tipo de utilizador tem acesso ainda aos separadores:

- **Gestão:** ecrã onde o utilizador pode realizar a gestão dos eventos (treinos e jogos). Nesta gestão fazem parte:
  - criação dos eventos, com todos os dados necessários (só existe esta possibilidade se o utilizador for pelo menos **Treinador**);
  - consulta das convocatórias abertas, podendo fechar essas mesmas convocatórias, de acordo com os requisitos determinados, isto é, registar indisponibilidade de atletas ou editar os dados do evento (no caso de ser pelo menos Treinador);
  - consulta das convocatórias fechadas, podendo fechar esses eventos, isto é, registar as presenças e atrasos dos atletas e registar lesões caso ocorram. Por fim, existe a possibilidade de inserir um sumário, no caso do evento ser um treino;
  - consulta dos treinos fechados.
- **Atletas:** ecrã onde o utilizador pode visualizar todas as informações dos atletas presentes na equipa, divididos primeiramente por escalão.

Atleta

Além dos separadores descritos anteriormente para o Utilizador Geral, este tipo de utilizador tem acesso ainda ao separador:

- **Convocatórias:** ecrã onde o utilizador pode consultar as suas convocatórias. Em cada convocatória pode consultar todos os dados da mesma, como por exemplo, o local e a sua respetiva rota (utilizando o *Google Maps* para o efeito) e, caso esta convocatória não esteja fechada ou o evento não se tenha já realizado, o atleta pode registar a sua (in)disponibilidade.

Pai

Além dos separadores descritos anteriormente para o Utilizador Geral, este tipo de utilizador tem acesso ainda ao separador:

- **Filhos:** ecrã onde o utilizador pode consultar todas as informações dos seus filhos, como as suas respetivas convocatórias, lesões e ainda outras informações pessoais.

O painel de navegação para cada tipo de utilizador pode ser verificado na seguinte figura:

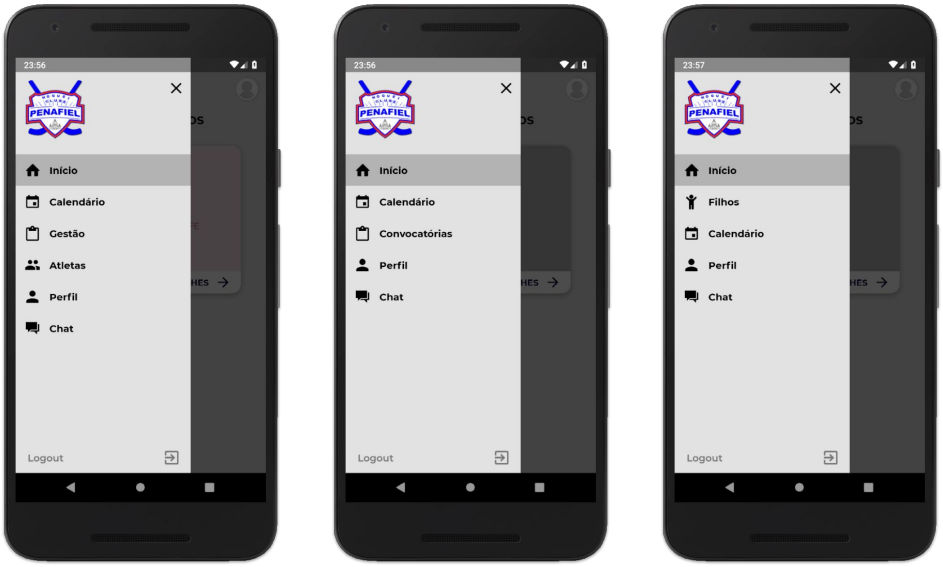


Figura 1: Exemplo dos menus de navegação dos utilizadores Treinador, Atleta e Pai respetivamente

Foi verificado ainda que um utilizador pode ter vários papéis ao mesmo tempo. Sendo assim, decidiu-se agregar os separadores de cada tipo de utilizador, para estes casos:

- `Atleta e (Treinador ou Seccionista)`: O painel de navegação é constituído pelos separadores do tipo de utilizador Atleta e pelos separadores do tipo utilizador Treinador ou Seccionista;
- `Atleta e Pai`: O painel de navegação é constituído pelos separadores do tipo de utilizador Atleta e pelos separadores do tipo utilizador Pai;
- `Pai e (Treinador ou Seccionista)`: O painel de navegação é constituído pelos separadores do tipo de utilizador Pai e pelos separadores do tipo utilizador Treinador ou Seccionista;

Um exemplo destes tipos de utilizador pode ser verificado na seguinte figura:

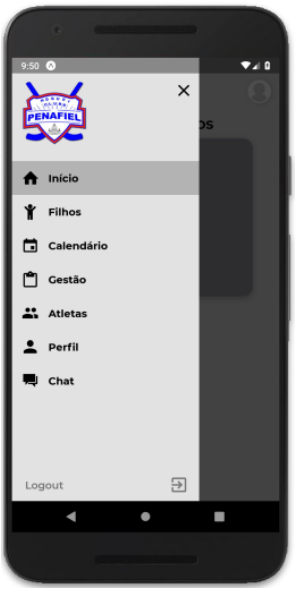


Figura 2: Exemplo dos menus de navegação de um utilizador `Pai e Seccionista`

Realizadas estas decisões, a implementação deste painel de navegação foi realizada com base na pesquisa dos grupos de cada utilizador (Treinador, Seccionista, Atleta, Pai), logo após o utilizador efetuar o *login*. Depois de encontrados os grupos do utilizador, decide-se qual o painel de navegação a mostrar, de acordo com os critérios definidos anteriormente. Isto requer que todos os ecrãs estejam *harded code*, no entanto, para podemos otimizar o código e para aumentarmos a legibilidade do mesmo, temos uma única estrutura `json` (conjuntos de objetos `javascript`) com todos os ecrãs existentes. Todas as possibilidades de combinações utilizam estes mesmos dados, permitindo a eficiência no código.

5.2 Notificações da aplicação

Para conseguirmos implementar notificações na aplicação desenvolvida, foi necessário criar um novo módulo `Odoo` que servisse de extensão ao módulo inicial, visto ser necessário modificar o *backend* do sistema para implementar tal funcionalidade.

Sendo assim, neste novo módulo foi criado um novo modelo `ges.token` que contem um `device` e o `token` desse mesmo *device*, *token* esse utilizado para enviar as notificações. É ainda definido que um utilizador (`ges.user`) estabelece uma relação de um para muitos com `ges.token`.

A partir daqui, é definido que sempre que um utilizador se autentica na aplicação *mobile*, envia para o *backend* o seu *token* utilizado para *push notifications*, assim como o nome do *device* a ser utilizado. Desta forma, no novo módulo `Odoo` é verificado se o utilizador em causa já tem um *token* registado com aquele dispositivo (modificando o mesmo caso seja necessário), caso contrário é adicionado um novo `ges.token` à lista de *tokens* do utilizador com o respetivo dispositivo e *token* recebido. Quando o

utilizador faz *logout*, é enviado o nome do *device* para o *backend* que trata de desassociar o *token* desse dispositivo ao utilizador em causa. Um *token* de notificações `Expo` segue o seguinte formato:

```
ExponentPushToken[xxxxxxxxxxxxxxxxxxxxx]
```

Para serem depois enviadas as notificações, são extendidos métodos de certos modelos definidos anteriormente, como por exemplo métodos de criação de treinos e jogos, lesões, de alteração de disponibilicade de um atleta num dado evento desportivo, etc.

O código utilizado para extender estes métodos trata de selecionar os utilizadores que pretendemos notificar, construir a mensagem de notificação a ser enviada a cada um deles, recolher os *tokens* de cada um e assim por fim enviar as notificações através de um `POST` para os servidores do `Expo`.

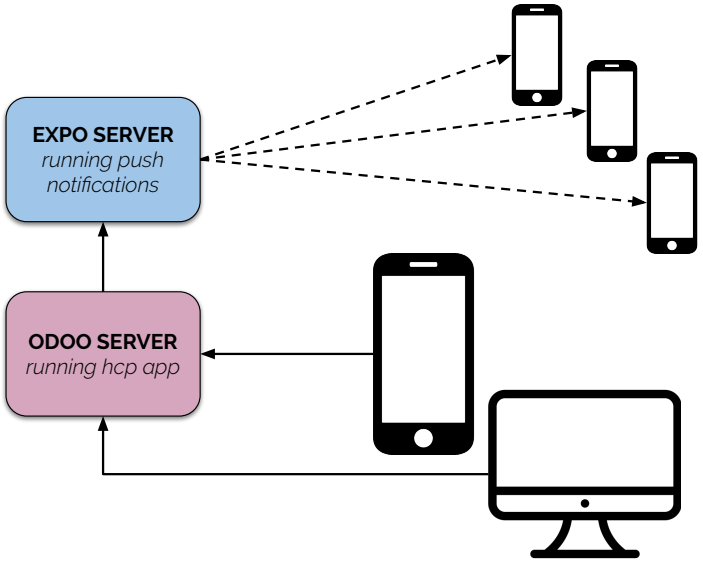


Figura 3: Processo de envio de notificações

Tivemos especial atenção para o facto de ser possível enviar mais do que uma notificação de cada vez e, assim, evitar fazer um `POST` para cada notificação que se pretende enviar. Desta forma, e sendo que o servidor `Expo` suporta o envio de até 100 notificações por cada `POST`, a lista de notificações a enviar é assim dividida caso seja necessário em partes de 100. Um exemplo de `body` para um `POST` com mais do que uma notificação no caso do registo de uma lesão de um atleta e por consequente o envio de notificação ao mesmo e aos seus pais pode ser:

```
[{
  "to": "ExponentPushToken[xxxxxxxxxxxxxxxxxxxxx]",
  "title": "Lesão registada",
  "body": "Foi adicionada uma nova lesão relacionada contigo."
}, {
  "to": "ExponentPushToken[yyyyyyyyyyyyyyyyyyyyy]",
  "title": "Lesão registada",
  "body": "Foi adicionada uma nova lesão relacionada com Nome 29."
}]
```

Figura 4: Exemplo de uma notificação de indisponibilidade

5.3 *Chat* da aplicação

Fomos desafiados a implementar um *chat* na aplicação, para permitir um fácil diálogo entre os diferentes utilizadores envolvidos no clube de hóquei de Penafiel.

Definimos como indispensável as seguintes funcionalidades, relativamente à implementação do *chat*:

- página inicial com os canais onde o utilizador está inserido, juntamente com indicação da última mensagem de cada canal;
- permitir que um utilizador consiga entrar num canal de grupo que seja público;

- permitir que um utilizador envie mensagens diretas (ou privadas) a um utilizador;
- permitir que um utilizador consiga sair de um canal de grupo ou desmarcar uma conversa (mensagens diretas com um utilizador);
- ser possível enviar e receber mensagens numa determinada conversa;
- ser possível aceder aos detalhes de um canal;

O primeiro desafio para conseguirmos implementar o *chat* foi perceber como o mesmo funcionava através do *browser*, visto o mesmo ser um *addon* do `Odoo` e não algo implementado no projeto a que temos acesso. Desta forma, através da consola do terminal que corre o processo do `Odoo` conseguimos perceber que os pedidos efetuados através da interatividade com o *browser* eram realizados ao modelo `mail.channel`, tendo nós a partir deste ponto procurado a documentação do `Odoo` que nos desse informação sobre quais os métodos disponíveis que este modelo disponibiliza, de forma a conseguirmos utilizar os mesmos na nossa aplicação *mobile*.

Para mostrar quais os canais em que um utilizador está inserido é feita a utilização do método `channel_fetch_slot` para obter os *id's* dos canais, sendo a resposta devolvida com os 3 tipos de canais: canais públicos, canais privados e canais de mensagens diretas. A partir daqui é utilizada a lista dos *id's* para pesquisar no modelo `mail.channel` informações como o nome, descrição ou imagem de cada canal. É ainda utilizado o método `channel_fetch_preview` que retorna a última mensagem de um dado canal (para recolher dados como autor, conteúdo ou data da mensagem). Nesta página é ainda definido um *update* a cada 2.5 segundos de forma a manter a página o mais atualizada possível.

Quando o utilizador se encontra num determinado canal é utilizado o método `channel_fetch_message` para obter as mensagens do canal, tanto as que se pretende mostrar inicialmente como as que se pretendem buscar caso o utilizador procure mensagens anteriores. Para mostrar em *tempo real* as mensagens novas do canal é definida uma atualização procurando novas mensagens a cada 1.5 segundos. Parte da lógica mais importante para conseguir um correto funcionamento da conversa passa pela gestão dos *id's* das mensagens já obtidas, sendo necessário essa gestão para conseguir mostrar tanto mensagens mais recentes como as anteriores. Para o utilizador enviar uma nova mensagem para a conversa é utilizado o método `message_post`.

No caso do utilizador pretender sair dum canal de grupo é utlizado o método `action_unfollow` ao qual é passado o *id* do canal em questão. Já para desmarcar uma conversa de mensagens diretas é utilizado o método `channel_pin`.

Com o objetivo de mostrar os detalhes de um canal, é feito um pedido de leitura ao modelo `mail.channel` com o *id* do canal em causa com a intenção de obter dados como o *id* dos participantes, a imagem do canal, a descrição ou a data de criação do mesmo. Com os *id's* dos participantes é então depois obtida a informação sobre os mesmos.

No que toca ao requisito do utilizador se poder juntar a um canal de grupo público é utilizado o método `channel_search_to_join` para mostrar os canais de grupo disponíveis e, caso o mesmo pretenda ser inserido num deles é utilizado o método `channel_join_and_get_info`.

Por último, para permitir o envio de mensagem direta a um utilizador à escolha é primeiramente apresentada a lista dos utilizadores e, caso o utilizador escolha uma pessoa para enviar mensagem é obtido o canal para os dois através da chamada do método `channel_get`, que trata de criar um canal novo caso necessário.

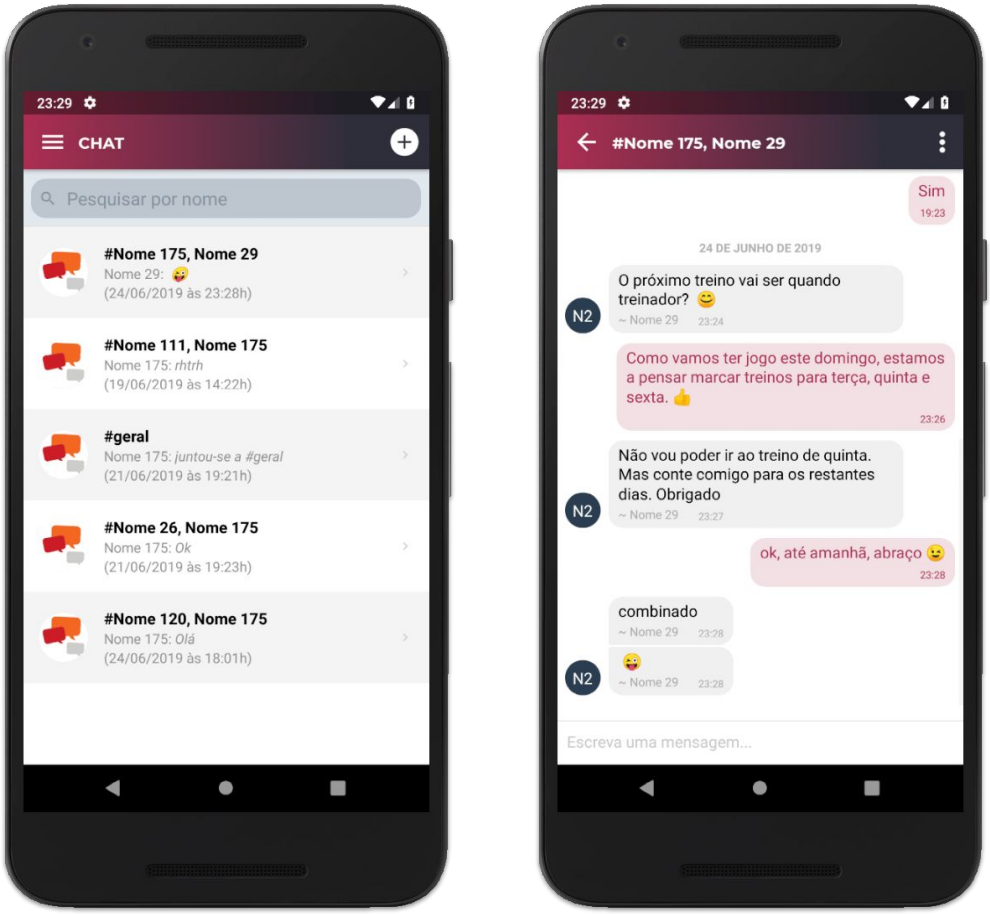


Figura 5: Canais disponíveis e exemplo de uma conversa

5.4 Percurso para os locais dos eventos

Fomos também desafiados a integrar um sistema de navegação para um dado evento desportivo, ou seja, permitir que o utilizador consiga facilmente obter a rota de navegação caso o destino tenha coordenadas definidas no sistema.

Integrar um sistema de navegação na própria aplicação nativa, requer a utilização *APIs* que geralmente são pagas ou estão limitadas no número de utilizações.

A nossa solução passou por, ao invés de integrarmos nativamente um sistema de navegação, utilizarmos um serviço nativo presente em quase todos os telemóveis *Android*, o ***Google Maps***. Deste modo, não temos custos monetários na utilização da *API* da *Google Maps* e conseguimos fornecer a funcionalidade de uma forma muito intuitiva, como podemos observa na figura em baixo.

Assim, a nossa solução passou por obtermos as coordenadas do local associado a um determinado evento. Caso o local tenha a sua localização definida na base de dados, os valores da latitude e da longitude do lugar em questão são obtidos. Juntamente com estes valores, são obtidas, se possível, as coordenadas do utilizador, para que posteriormente, se possa definir a rota de navegação com a ajuda do *Google Maps*.

Esta funcionalidade não está apenas limitada aos dispositivos *Android*, uma vez que nos dispositivos *iOS* a aplicação *Google Maps* também pode ser instalada.

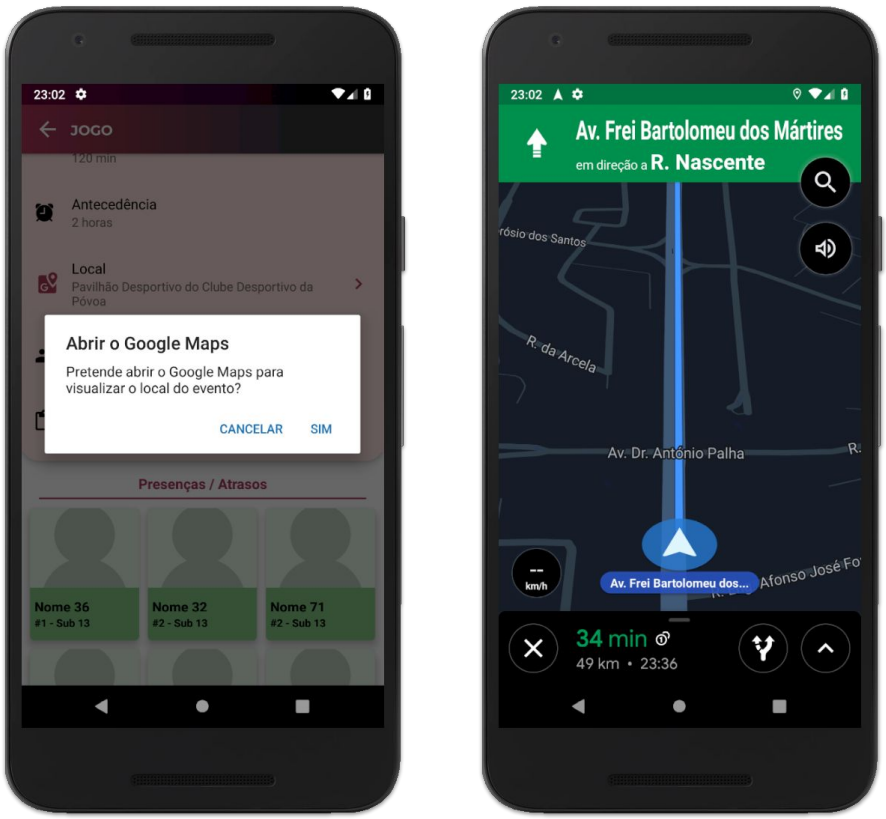


Figura 6: Obtenção da rota para o evento desportivo

6 Algumas considerações com o *OdooRPC*

Ao longo do desenvolvimento da aplicação móvel e da interação com o *OdooRPC*, fomos percebendo que eram necessários alguns cuidados na utilização dos métodos fornecidos pelo `OdooRPC`.

Apesar das inúmeras funcionalidades `CRUD` fornecidas pelo textttOdooRPC e ainda pela utilização de algumas funcionalidades de um sistema de base de dados (como *limit*, *order* ...) foram encontrados alguns problemas durante a implementação da aplicação.

Em baixo apresentamos alguns dos problemas que consideramos importantes.

6.1 Inserção de novos valores com relações

Consideremos a criação de evento desportivo. Neste processo, é necessário fornecermos os `ids` dos intervenientes no evento, como o dos atletas, treinadores e secretários. Através do código disponibilizado relativo ao *backend* da aplicação, percebemos que um evento e, por exemplo, um alteta, têm uma relação de *muitos para muitos*.

Durante a implementação, percebemos que era impossível **criar** um novo evento desportivo, dado que era necessário fornecer determinados valores, que são obrigatórios quando estamos perante uma relação de *um para muitos* e de *muitos para muitos*. Estes valores podem ser encontrados na `ORM API`[1] do `Odoo`.

No contexto apresentado em cima, a solução passou por utilizarmos os seguintes argumentos:

```
1 {
2     start: data e hora de início...,
3     stop:  data e hora de início...,
4     escalao: id do escalão...,
5     local: id do local,
6     treinador: [[6,0, array com os ids dos treinadores]],
7     seccionistas: [[6,0, array com os ids dos seccionista]],
8     atletas: [[6,0, array com os ids dos atletas]],
9 }
```

Como podemos observar no excerto de código em cima, nas linhas 6,7 e 8, estamos a utilizar os valores $(6, \_, ids)$, que, tal como é explicado na *API*, permite que possamos redefinir (neste caso definir) os intervenientes de um evento.

[1] `https://www.odoo.com/documentation/11.0/reference/orm.html#odoo.models.Model.write`

6.2 Operadores lógicos

Consideremos a pesquisa de eventos desportivos futuros. Neste processo, é importante que cada utilizador consiga visualizar os eventos a ele associado. No caso de não existirem, são exibidos os eventos gerais do clube, ao qual o utilizador deverá ter acesso de leitura (definido nos privilégios do `Odoo`).

Como já foi referido neste documento, um utilizador pode exercer vários cargos na aplicação: ser um atleta, um treinador, um seccionista ou um pai e ainda, ser uma combinação destes.

Deste modo, é necessário definirmos um domínio (*domain* no `OdooRPC`). Este domínio aceita uma lista de condições que, por omissão, operador $\&$ é utilizado entre elementos da lista. Portanto, caso se pretenda realizar conjunções ou interceções entre condições, é necessário utilizarmos uma anotação muito específica. Esta anotação denomina-se **Notação Polaca**, muito frequente em Álgebra e Lógica e permite a eliminação de parênteses ou outros delimitadores para indicar os cálculos que devem ser realizados primeiramente.

Em baixo, apresentamos uma tabela com algumas conversões de condições para a Notação Polaca.

Notação Normal	**Notação Polaca**	***Odoo***
A	A	$[\ A\ ]$
$A \wedge B$	$\&\ A\ B$	$[\ '\&',\ A,\ B\ ]$
$A \vee B$	$\mid A\ B$	$[\ '\mid',\ \ A,\ B\ ]$
$(A \vee B) \wedge C$	$\&\mid A\ B\ C$	$[\ '\&',\ '\mid',\ A,\ B,\ C\ ]$
$(A \wedge B) \vee (C \wedge D)$	$\mid \&\ A\ B\ \&\ C\ D$	$[\ '\mid',\ \ '\&',\ A,\ B,\ '\&',\ C,\ D\ ]$

Tabela 1: Conversão da notação normal para notação polaca.

Reparamos, ao longo do desenvolvimento da aplicação, que muitos dos utilizadores tinham certos problemas ao fazerem pedidos ao servidor *backend Odoo*. Isto é, existem alguns problemas de restrições de acesso aos dados nomeadamente utilizadores que sejam simultaneamente **treinadores** e **atletas**. Estes conflitos podem ser manifestados ao tentar criar um evento desportivo ou ao consultar determinados dados sobre os eventos, especialmente os eventos do tipo treino. Outra restrição frequente e facilmente manifestável,

Um exemplo em concreto deste cenário passa pele criação de um treino no *browser* através da conta de um utilizador que é simultaneamente treinador e atleta, processo durante o qual não conseguimos escolher os treinadores a convocar para o treino. Mesmo ignorando esta etapa e optando por guardar o evento, temos a mensagem de erro que nos diz que a operação solicitada não pode ser concluída face a restrições de segurança.

Com base nestes comportamentos, **toda a nossa implementação teve especial cuidado** em analisar estes erros e a exportá-los, em forma de "alerta", para o utilizador. Desta forma, mantemos a estabilidade da aplicação minimizando o número de *bugs* possíveis.

7 Construção do ficheiro APK

Apesar do *React Native* permitir o desenvolvimento de aplicações para ***iOS*** e para ***Android***, o grupo focou-se apenas no desenvolvimento para *Android*, uma vez que, para o desenvolvimento de uma aplicação *iOS*, é necessário um dispositivo *Apple* ou um computador com o sistema operativo *Mac OS*. Como nenhum elemento do grupo possui estes requisitos, ficamos limitados ao desenvolvimento para *Android*. No entanto, acreditamos que toda a aplicação no *iOS* esteja praticamente funcional.

Ao longo desta secção, abordaremos não só as informações dos ficheiros de configuração, bem como todos os passos necessários para a construção do ficheiro APK da aplicação.

Com a utilização do *Expo* é muito simples configurarmos as características da aplicação, como é o caso do nome, *icon* da aplicação e da notificação, orientação da aplicação e as permissões que serão requeridas ao utilizador no processo de instalação. Para alterarmos esta configurações basta editarmos o ficheiro `app.json` na diretoria do projeto. As configurações disponíveis podem ser consultadas no *site* da *Expo*[2].

7.1 Geração de uma chave privada para a aplicação

Uma aplicação *Android* requer que exista no ficheiro APK um certificado (uma chave privada única) que permite validar a autenticidade da aplicação. **Caso queiramos** fazer o *upload* da aplicação para um serviço como a ***Google Play***, é estritamente necessário que tenhamos estes dados, pois, sem eles, estamos impossibilitados de fazer atualizações à aplicação.

Estes dados podem ser armazenados num ***keystore***. Para isso podemos utilizar um computador pessoal, os servidores da *Google* ou os servidores da *Expo*. Uma vez que existem formas mais seguras do que armazenar estas chaves num computador pessoal, excluímos esta mesma possibilidade.

Como referido anteriormente, a *Google* disponibiliza um serviço de *keystore* denominado *App Signing*, que armazena a chave privada da aplicação. Depois destes dados estarem seguros nos serviços da *Google*, é apenas necessário gerarmos uma chave de *upload*, que será utilizada pela *Google* para validar a nossa autenticidade. Depois de construirmos o APK com esta chave de *upload* e fazermos o *upload* do mesmo, a *Google* remove a chave e adiciona ao APK a chave privada previamente adicionada à *keystore*. Com a utilização deste serviço, podemos redefinir a chave de *upload* sempre que esta for perdida ou comprometida, permitindo que sejam realizadas alterações na aplicação. Em baixo apresentamos um esquema com este processo.

[2] `https://docs.expo.io/versions/latest/workflow/configuration/`

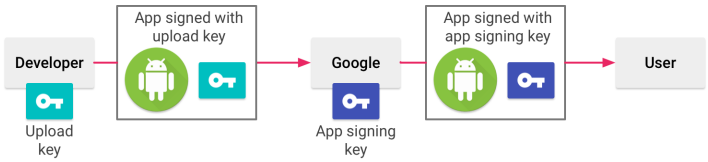


Figura 7: Serviço *App Signing* da *Google*

O serviço *Expo* fornece também um serviço de *keystore* e, para isso, devemos ter uma conta registada no *web site* deles.

O serviço de *keystore* fornecido permite que seja gerada uma chave privada e que esta seja armazenada nos servidores deles, para que, sempre que o APK for gerado, o *Expo* aplica automaticamente a chave na aplicação. O *Expo* recomenda que guardemos localmente na nossa máquina a *keystore* (backup da *keystore* presente nas máquinas da *Expo*). Para isso devemos efetuar o seguinte comando:

```
expo fetch:android:keystore
```

Caso não queiramos ter do nosso lado a *keystore*, podemos requisitar o certificado da nossa aplicação, uma vez que é necessário no processo de *upload* da aplicação na *Google Play*. Para obtermos o certificado público devemos:

```
expo fetch:android:upload-cert
```

7.2 Processo de construção do APK

Para construirmos o ficheiro APK é necessário efetuarmos uma sequência de passos simples. A construção do ficheiro APK não será efetuada na máquina local, mas sim numa máquina remota fornecida pelo *Expo*, pelo que todo o projeto será carregado para os serviços deles. Depois deste processo estar terminado, é possível distribuirmos o APK para os clientes ou para uma plataforma como o *Google Play*. Em baixo apresentamos os passos necessários:

- `expo login`
  - Tal como referido anteriormente, os ficheiros irão ser transferidos para os servidores do *Expo*, pelo que é necessário que exista uma conta e que se faça a respetiva autenticação;
- `expo start`
  - É necessário que o *Expo packager* esteja a correr para podermos efetuar todo o processo;
- `expo build:android`
  - noutro terminal, iniciar o processo de construção da aplicação;
  - neste comando irá ser questionado se se pretende utilizar a chave privada presente na *keystore* da *Expo* ou se se pretende utilizar uma *keystore* local;

Ao fim do *upload* estar concluído e do ficheiro APK estar gerado, podemos proceder à instalação da aplicação num dispositivo *Android*.

8 Conclusão

Como foi mostrado ao longo deste documento, todo o nosso esforço e dedicação centrou-se, sobretudo, no desenvolvimento de uma aplicação apelativa, intuitiva e cheia de funcionalidades capazes de aumentarem a produtividade no processo de gestão do Hóquei Clube de Penafiel. De notar no entanto que, toda a aplicação pode ser facilmente adaptada para um outro contexto desportivo, dado que a lógica da aplicação apresenta muitas semelhanças, por exemplo, num contexto futebolístico. Assim, podemos concluir que seria interessante alterar a aplicação atual de forma a que esta possa ser implementada num contexto geral e que permita a utilização da mesma por uma séria de equipas de várias modalidades desportivas.

A escolha e utilização do *React Native* só vem fortalecer o que mencionamos anteriormente. A vantagem que esta *framework* tem em nos fornecer o desenvolvimento de componentes reutilizáveis, permite que consigamos construir uma aplicação para uma equipa de hóquei ou de outro desporto muito facilmente. Podemos reutilizar, por exemplo, os componentes de consulta de informações dos atletas, da consulta e criação dos eventos desportivos, do registo das presenças e atrasos, do registo das convocatórias, do calendário, do *chat*, entre outros.

Tendo por base a aplicação *web* existente e o resultado final da aplicação *mobile*, conseguimos perceber que muitos dos passos de criação, alteração e consulta de dados podem agora ser facilmente executados. Para além dos requisitos definidos na fase inicial do projeto, acreditamos que, após a disponibilização da aplicação para produção, surjam novas funcionalidades que permitam (ainda mais) uma melhor gestão da equipa. Deste modo, o grupo mostra-se disponível e fazer alterações na aplicação, dependendo da disponibilidade dos elementos e da dificuldade das alterações.