

Universidade do Minho
Mestrado Integrado em Engenharia Informática
Departamento de Informática

LABORATÓRIO EM ENGENHARIA INFORMÁTICA

Aplicação para a recolha de estatísticas em eventos desportivos

Daniel Fernandes Veiga Maia A77531
Maria de La Salete Dias Teixeira A75281
Vitor Emanuel Carvalho Peixoto A79175

Ano Letivo de 2018/2019

Conteúdo

1	Introdução	3
2	Planeamento	4
3	Recursos Necessários	5
4	Requisitos	6
4.1	Objetivos do Projeto	6
4.2	Requisitos Funcionais	6
4.2.1	Funcionalidades genéricas	6
4.2.2	Registo de eventos	6
4.2.3	Utilizadores e permissões	8
4.3	Requisitos de Usabilidade	9
4.4	Requisitos de Desempenho	9
4.5	Requisitos de Disponibilidade	10
5	Modelação	11
5.1	Modelo de Domínio	11
5.2	Diagrama de Casos de Uso	13
5.2.1	Matriz de Funcionalidades	14
5.3	Diagrama de Classes	16
6	Prototipagem	18
7	Implementação	24
7.1	Arquitetura	24
7.2	Base de Dados	24
7.3	<i>Back end</i>	25
7.4	<i>Front end</i>	25
7.5	Comunicação	25
7.6	API	26
7.6.1	Adicionar atleta	26
7.6.2	Adicionar clube	26
7.6.3	Adicionar formação	27
7.6.4	Adicionar jogo	27
7.6.5	Adicionar tipo de evento	28
7.6.6	Alterar grelhas de um técnico	29

7.6.7	Alterar grelhas de um jogo	30
7.6.8	Alterar informações de um evento	30
7.6.9	Assinalar que já foram escolhidos os convocados para um jogo	31
7.6.10	Associar sugestão de tipo de evento a técnico	32
7.6.11	Definir atletas convocados para um jogo e 5 iniciais . .	32
7.6.12	Mudar atleta de formação	33
7.6.13	Obter atletas em campo	34
7.6.14	Obter atletas suplentes	35
7.6.15	Obter informações de um clube a partir de uma formação	35
7.6.16	Obter informações sobre um evento	36
7.6.17	Obter informações sobre um jogo	37
7.6.18	Obter informações sobre um tipo de evento	38
7.6.19	Obter informações sobre um utilizador	39
7.6.20	Obter sugestões de tipos de eventos para um técnico .	40
7.6.21	Obter todas as formações de um clube	41
7.6.22	Obter todos os atletas de uma formação	41
7.6.23	Obter todos os clubes	42
7.6.24	Obter todos os eventos de um jogo	42
7.6.25	Obter todos os jogos de um clube	44
7.6.26	Obter todos os tipos de eventos	45
7.6.27	Registar evento	45
7.6.28	Registar gestor	46
7.6.29	Registar técnico	47
7.6.30	Remover evento	47
7.6.31	Remover sugestão de tipo de evento de um técnico . .	48
7.6.32	Sinalizar ou remover sinalização de evento	48
7.6.33	Terminar jogo	48
8	Configuração	49
9	Resultados	50
9.1	Interface final	50
9.2	Funcionalidades	62
10	Conclusão	64

1 Introdução

Com a crescente acessibilidade de ferramentas de análise informáticas, tem vindo a ocorrer a difusão de aplicações analistas para uma variedade de modalidades desportivas. Ainda assim, a modalidade de hóquei em patins encontra-se carente de tais métodos desenhados para simplificar e agilizar o processo de análise de estatísticas desportivas.

É daqui que emerge o projeto de uma aplicação para a recolha de estatísticas em eventos desportivos, proposto pela Unidade Curricular de Laboratório de Engenharia Informática. Esta trata-se de uma aplicação que visa a permitir um utilizador introduzir dados relativos a jogos, em particular da modalidade de hóquei, e receber um conjunto de estatísticas relevantes relativos aos mesmos.

Ao longo deste relatório será explicitado o planeamento previsto da modelação e implementação da aplicação e os recursos necessários para cada estágio do projeto. Para além destes, serão levantados requisitos do sistema a implementar e destes será efetuada a modelação do mesmo. De seguida, serão elaborados um conjunto de *mockups* indicativos da interface visionada para a aplicação. Por fim, será efetuado um balanço do trabalho efetuado, dos resultados obtidos e do trabalho futuro na implementação do sistema.

2 Planeamento

O planeamento de um projeto é uma técnica muito importante e cada vez mais útil, sendo essencial em projetos de grandes dimensões. Com um bom planeamento a comunicação e organização entre a equipa é facilitada e o cliente tem uma melhor percepção dos passos necessários para chegar ao resultado final. Para além disso, ao criar o planeamento, fica-se com uma ideia mais clara de que etapas se devem tomar primeiro, quais as funcionalidades que dependem umas das outras e torna-se também possível avaliar o desempenho da equipa ao longo do desenvolvimento, sendo fácil identificar atrasos preocupantes.

Assim, com o intuito de otimizar o desenvolvimento do projeto, tentando maximizar o aproveitamento do tempo disponível, recorreu-se a um diagrama de *Gantt* com as etapas do projeto, as datas de início e a duração prevista para cada tarefa.

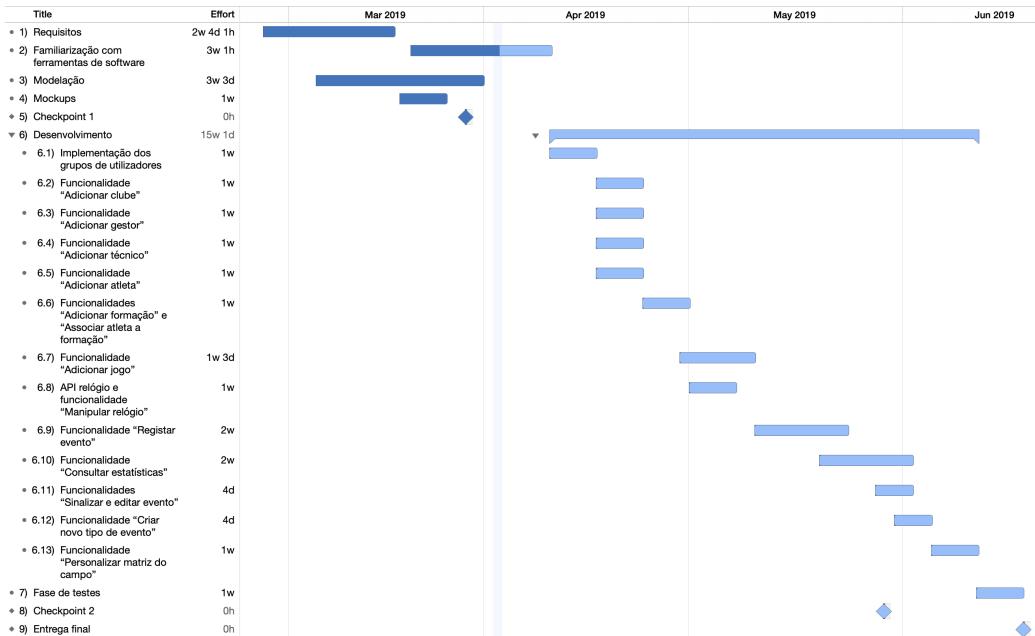


Figura 1: Diagrama de Gantt do projeto.

3 Recursos Necessários

Para o desenvolvimento desta aplicação foi necessário recorrer a várias ferramentas, tanto para o seu planeamento como para a implementação. Assim, para o planeamento utilizou-se o *Trello*, uma aplicação para organização de projetos, o *Lucidchart*, que permite desenvolver a modelação, e o *Inkscape*, uma ferramenta de criação de *mockups*.

Relativamente à implementação do projeto, esta teve que ser uma decisão mais ponderada, tendo-se tentado optar por ferramentas práticas, com uma boa curva de aprendizagem e ao mesmo tempo eficientes para os resultados pretendidas. Relativamente à base de dados, escolheu-se o *PostgreSQL* devido à sua notoriedade na comunidade e facilidade de utilização. Passando para a linguagem de programação, optou-se por codificar a aplicação em *Python*, pois esta é uma linguagem que engloba todas as características enunciadas, sendo bastante prática e intuitiva, ideal para um projeto desta dimensão e com uma boa curva de aprendizagem e uma biblioteca de funções diversa. Para o *back end*, optou-se pela *framework Django*, onde é possível aplicar a lógica de negócio e que lida automaticamente com a camada de dados, inclusive a criação das tabelas necessárias na base de dados. Por último, para o desenvolvimento do *front end* tem-se como características fundamentais uma interface rápida e responsiva, que seja capaz de se adaptar aos diferentes tamanhos de ecrã (computador, tablet e smartphone). Para tal, optou-se por utilizar a *framework* para *JavaScript Vue.js*, complementando esta com o *Bootstrap*, sendo este último o responsável pelo ajuste do *layout*.

É também importante referir que, numa fase mais avançada do projeto, será necessário eleger ferramentas adequadas para disponibilizar a aplicação ao público, nomeadamente um servidor online centralizado e o tipo de servidores locais para o registo de eventos no decorrer de um jogo.

4 Requisitos

4.1 Objetivos do Projeto

O projeto tem como objetivo desenvolver uma aplicação que facilite a análise e deteção de problemas que permitam auxiliar no treino e desenvolvimento de uma equipa de hóquei em patins.

O uso de uma aplicação num dispositivo móvel que permita o registo de eventos ocorridos numa partida, facilita a análise estatística de todos os aspetos táticos de uma equipa. Deste modo a equipa técnica pode consultar os pontos fortes e fracos da sua equipa e trabalhar neles, de modo a melhorar os resultados desportivos da sua equipa e a acelerar a evolução individual dos seus jogadores.

Este tipo de aplicações são bastante comuns no universo desportivo em Portugal, porém a modalidade de hóquei em patins encontra-se tecnologicamente atrasada. Assim, esta aplicação é vista como uma novidade que permitirá desenvolver a qualidade de jogo praticado em Portugal.

4.2 Requisitos Funcionais

4.2.1 Funcionalidades genéricas

1. A aplicação deve permitir o registo de ocorrências/eventos durante uma partida de hóquei em patins.
2. A aplicação deve gerar e permitir consultar as estatísticas durante e após uma partida, sendo que as estatísticas geradas devem ser apresentadas graficamente num campo do jogo.
3. A aplicação deve permitir a visualização das estatísticas relativas a um específico jogador, tipo de evento ou momento do jogo.
4. A aplicação deve apresentar um relógio semelhante ao utilizado na modalidade, permitindo que este seja alterado e pausado quando necessário.

4.2.2 Registo de eventos

5. O registo de um evento deve incluir a seguinte informação: tipo de evento; atleta e equipa; instante de tempo atual; parte do jogo; instante

de tempo atualizado; zona de campo; zona da baliza e *timestamp* do horário de registo do evento.

6. Dependendo do tipo de evento registado, alguns dos detalhes do evento são dispensáveis, como equipa, atleta, instante de tempo atualizado, zona do campo ou zona da baliza.
7. Alguns tipos de evento existentes devem estar predefinidos na aplicação, podendo o utilizador registar novos tipos de eventos.
8. Os tipos de evento que a aplicação permite registar devem ser pelo menos os seguintes: ataque organizado; ataque rápido; contra-ataque; erro defensivo; remate à baliza; remate fora; remate intercetado; perda de bola; roubo de bola; recuperação de bola; golo; falta; falta de equipa; livre direto; penálti; bloqueio; golpe duplo; 5 segundos; 10 segundos; 45 segundos; cartão azul; cartão vermelho; substituição; *timeout*; *powerplay*; fim do *powerplay* e atualização do relógio.
9. O registo dos eventos deve ser efetuado através da introdução de códigos simples num teclado numérico.
10. Aquando o registo de um evento, o instante de tempo deverá ser obtido automaticamente, a partir de uma API.
11. No registo de eventos, para selecionar a zona do campo onde este ocorreu, deve ser utilizada uma matriz para dividir o terreno de jogo, sendo cada zona representada por um número. O utilizador deverá selecionar a zona do campo utilizando novamente o teclado numérico.
12. A matriz utilizada para dividir o campo deve ser, por omissão, 8x4 e deve poder ser personalizada pelo utilizador.
13. A matriz utilizada para dividir a baliza deve ser, por omissão, 3x3 e deve poder ser personalizada pelo utilizador.
14. A seleção do atleta responsável por um evento na partida deve ser feita pelo teclado numérico também, digitando o número da sua camisola.
15. A sinalização, remoção e alteração de eventos incorretamente registados deve ser facultada através de uma interface simplificada quando acedida durante o decorrer do jogo.

16. A sinalização, remoção alteração de eventos incorretamente registados deverá ser facultada através de uma interface que inclui todas as opções de personalização do evento quando acedida após o término do respetivo jogo.

4.2.3 Utilizadores e permissões

17. A aplicação pode ser utilizada por um administrador principal, gestores de clubes e utilizadores comuns (técnicos de equipa).
18. O registo e *login* de qualquer tipo de utilizador deve incluir o email e password.
19. O administrador principal deve poder registar clubes e respetivo gestor.
20. Os gestores de clubes podem registrar outros gestores, atletas, técnicos, formações (equipas) do clube e respetivos jogadores e jogos a realizar. Podem também consultar as estatísticas das partidas de todas as formações do clube.
21. Os técnicos de equipa podem criar novos tipos de eventos, registar, sinalizar, remover e editar os eventos de uma partida, consultar as estatísticas da sua formação relativamente a eventos anteriores e selecionar tipos de eventos para aparecerem nas sugestões aquando o registo.
22. Os atletas devem ser identificados pelo seu número de licença.
23. Os atletas devem ter associado um número de camisola.
24. Os técnicos podem definir a numeração da equipa na secção de definições ou alterar a numeração antes de cada partida, mas nunca durante.
25. A criação de um jogo deve incluir a equipa adversária, os atletas inscritos por cada equipa, o dia, hora e local do jogo (casa ou fora) e o seu caráter (oficial ou amigável).

4.3 Requisitos de Usabilidade

1. Um técnico com conhecimento da modalidade de hóquei em patins deverá conseguir utilizar 70% das funcionalidades da aplicação após 2 horas de utilização e 100% das funcionalidades básicas, como o registo de eventos.
2. A aplicação deverá ser construída para que o registo de um evento ocorrido durante a partida não dure mais que 5 segundos.
3. O registo de eventos através de códigos deve ser preciso, de modo a razão de erro no registo de eventos seja inferior a 5%.
4. A interface deverá ser intuitiva, de modo a que um utilizador com periodicidade de utilização semanal, seja capaz de manusear a aplicação com a mesma facilidade na semana seguinte.
5. A aplicação deve suportar a correção de eventos incorretamente registrados em menos de 10 segundos.
6. O acerto no tempo do relógio deverá demorar, no máximo, 3 segundos.

4.4 Requisitos de Desempenho

1. A aplicação deve conseguir apresentar estatísticas num tempo de resposta máximo de 2 segundos após a inserção de um evento.
2. O registo de uma nova ocorrência deverá ser processado de imediato, de modo a que seja possível inserir um novo evento imediatamente a seguir.
3. A submissão das estatísticas de um jogo, de uma base de dados local para a base de dados geral da aplicação deve ser efetuada com uma latência nunca superior a 10 segundos.

4.5 Requisitos de Disponibilidade

1. O produto deverá estar disponível para utilização um mínimo de 360 dias por ano.
2. O produto deverá estar disponível durante um período mínimo 240 minutos ininterruptos de cada vez.
3. Os dados e estatísticas relativos a um evento a decorrer devem estar disponíveis de imediato (armazenadas localmente), mesmo sem ligação ao servidor central.
4. Os dados e estatísticas de eventos passados deverão estar disponíveis para consulta e edição mediante uma conexão ao servidor central da aplicação.

5 Modelação

5.1 Modelo de Domínio

Após uma análise dos requisitos levantados, elaborou-se um modelo das entidades envolvidas no funcionamento do sistema e as respetivas interações entre as mesmas.

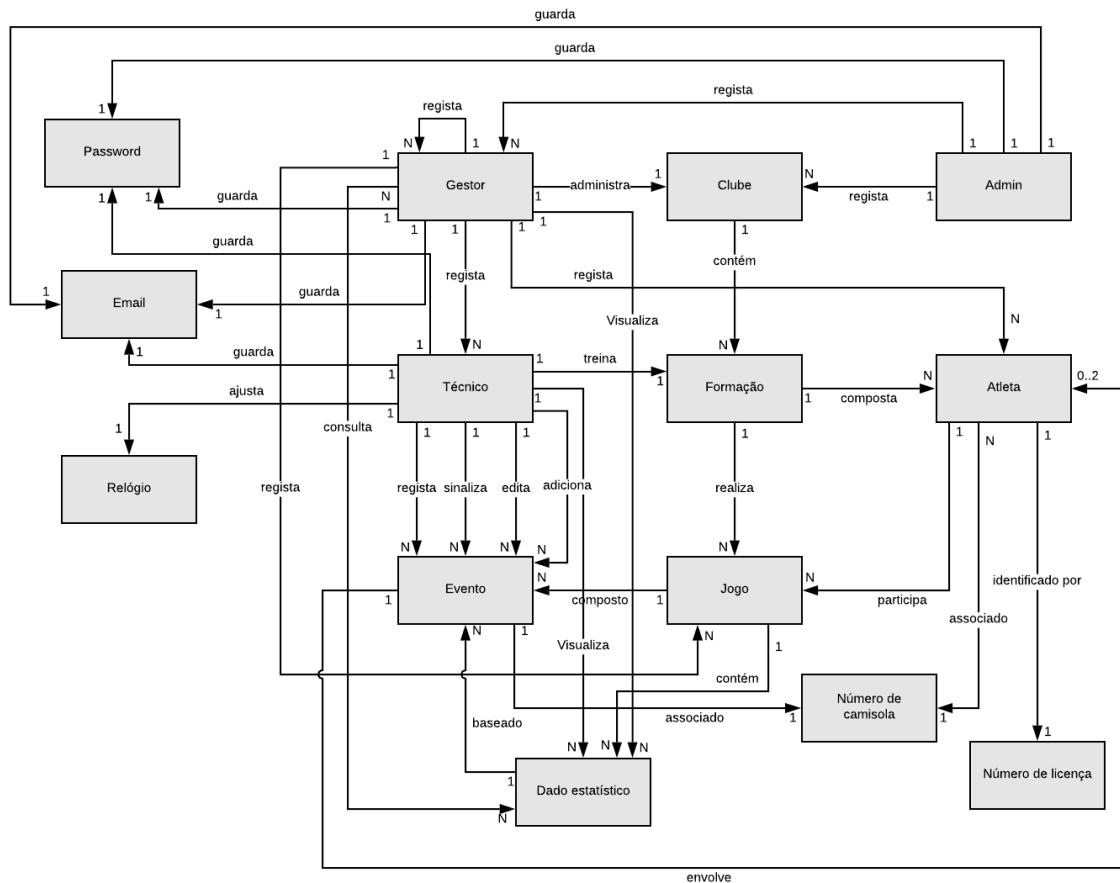


Figura 2: Modelo de Domínio

Neste sistema, existe um administrador responsável pelo registo de clubes e dos seus respetivos gestores. Estes gestores poderão registar outros gestores do mesmo clube, bem como atletas e técnicos de equipa. O conjunto de

administradores, gestores e técnicos trata-se dos utilizadores do sistema, e, como tal, têm associados um email único e uma palavra passe com os quais acedem ao sistema.

Para além disto, cada técnico é responsável por treinar uma formação de atletas do clube e gerir os eventos e relógio de quaisquer jogos em que a sua formação participe. Estes jogos são utilizados para gerar dados estatísticos para análise posterior.

Por fim, cada atleta tem um número de licença único e imutável, bem como um número de camisola variável entre jogos. Este número de camisola pode ser alterado pelo técnico da formação respetivo do atleta.

5.2 Diagrama de Casos de Uso

A análise dos requisitos apresentados levou à criação de um diagrama de casos de uso, onde se definiu as funcionalidades para as três entidades referidas nos requisitos: administrador, gestor de clube e técnico.



Figura 3: Diagrama de Casos de Uso

O Administrador pode também realizar as ações de adicionar clube e,

consequentemente, adicionar um gestor associado ao mesmo clube.

O Gestor de Clube partilha com o Administrador a ação de adicionar gestor de clube, sendo que este apenas o pode fazer para o seu próprio clube. Este tem também associado a si os *use cases* de adicionar um novo jogo, adicionar um técnico, adicionar uma nova formação, adicionar um atleta, associar um atleta a uma formação e consultar estatísticas referentes ao seu clube.

Por último, o Técnico partilha com o Gestor de Clube o caso de uso de consultar estatísticas, podendo também realizar todo o tipo de ações relacionadas com a criação de eventos para os jogos que desejar. Estas ações passam por criar um novo tipo de evento, personalizar a matriz do campo, alterar camisola de um atleta, selecionar os jogadores convocados e os 5 iniciais, manipular relógio, ou seja, pausar, incrementar ou decrementar o relógio em segundos, registar, sinalizar e editar um evento.

5.2.1 Matriz de Funcionalidades

A partir do Diagrama de Casos de Uso, desenvolveu-se uma matriz de funcionalidades, que permite uma melhor visualização das funcionalidades por tipo de utilizador. Assim, torna-se mais fácil identificar quais os utilizadores que partilham funcionalidades e a melhor ordem para o desenvolvimento das mesmas, conseguindo assim identificar que ações dependem de outras.

	Administrador	Gestor	Técnico
Adicionar clube			
Adicionar gestor			
Adicionar técnico			
Adicionar formação			
Adicionar atleta			
Associar atleta a formação			
Adicionar jogo			
Consultar estatísticas			
Alterar camisola de atleta			
Selecionar convocados e 5 iniciais			
Personalizar matriz do campo			
Manipular relógio			
Criar novo tipo de evento			
Selecionar tipo de evento			
Registar evento			
Sinalizar ou remover evento			
Editar evento			

Tabela 1: Matriz de Funcionalidades

Com os requisitos especificados e as funcionalidades identificadas, é possível associar quais os requisitos que cada funcionalidade deve respeitar.

- **Adicionar clube:** deve estar de acordo com o requisito 19;
- **Adicionar gestor:** deve estar de acordo com os requisitos 19 e 20;
- **Adicionar técnico:** deve estar de acordo com o requisito 20;
- **Adicionar formação:** deve estar de acordo com o requisito 20;
- **Adicionar atleta:** deve estar de acordo com os requisitos 20 e 21;
- **Associar atleta a formação:** deve estar de acordo com os requisitos 20;
- **Adicionar jogo:** deve estar de acordo com os requisitos 20 e 25;
- **Consultar estatísticas:** deve estar de acordo com os requisitos 2, 3, 20 e 21;

- **Alterar camisola de atleta:** deve estar de acordo com os requisitos 23 e 24;
- **Selecionar convocados e 5 iniciais:** deve estar de acordo com o requisito 25;
- **Personalizar matriz do campo:** deve estar de acordo com o requisito 12;
- **Manipular relógio:** deve estar de acordo com o requisito 4;
- **Criar novo tipo de evento:** deve estar de acordo com os requisitos 5, 6 e 7;
- **Selecionar tipo de evento:** deve estar de acordo com o requisito 21;
- **Registar evento:** deve estar de acordo com os requisitos 1, 5, 6, 7, 8, 9, 10, 11, 14 e 21;
- **Sinalizar ou remover evento:** deve estar de acordo com os requisitos 15, 16 e 21;
- **Editar evento:** deve estar de acordo com os requisitos 15, 16 e 21.

5.3 Diagrama de Classes

O diagrama de classes é uma das partes mais importantes da modelação, sendo o que fornece mais informações e o que mais se aproxima da implementação do projeto. Neste definem-se as classes, atributos e tipos de dados que se deve ter presente na aplicação.

Para o desenvolvimento deste diagrama, teve-se em conta a *framework* utilizada para o *back end* desta aplicação, nomeadamente o *Django*. Depois de uma breve pesquisa, o grupo aprendeu como gerar automaticamente o diagrama de classes a partir dos *models* do projeto. Assim, começou-se por definir os *models* necessários e só depois se criou o diagrama, que se apresenta de seguida.

Tendo em conta que, durante o desenvolvimento do projeto, o grupo se foi deparando com a necessidade de adicionar mais atributos aos modelos para o desenvolvimento das funcionalidades, o diagrama de classes encontra-se mais completo e com mais pormenores do que o modelo de domínio apresentado.

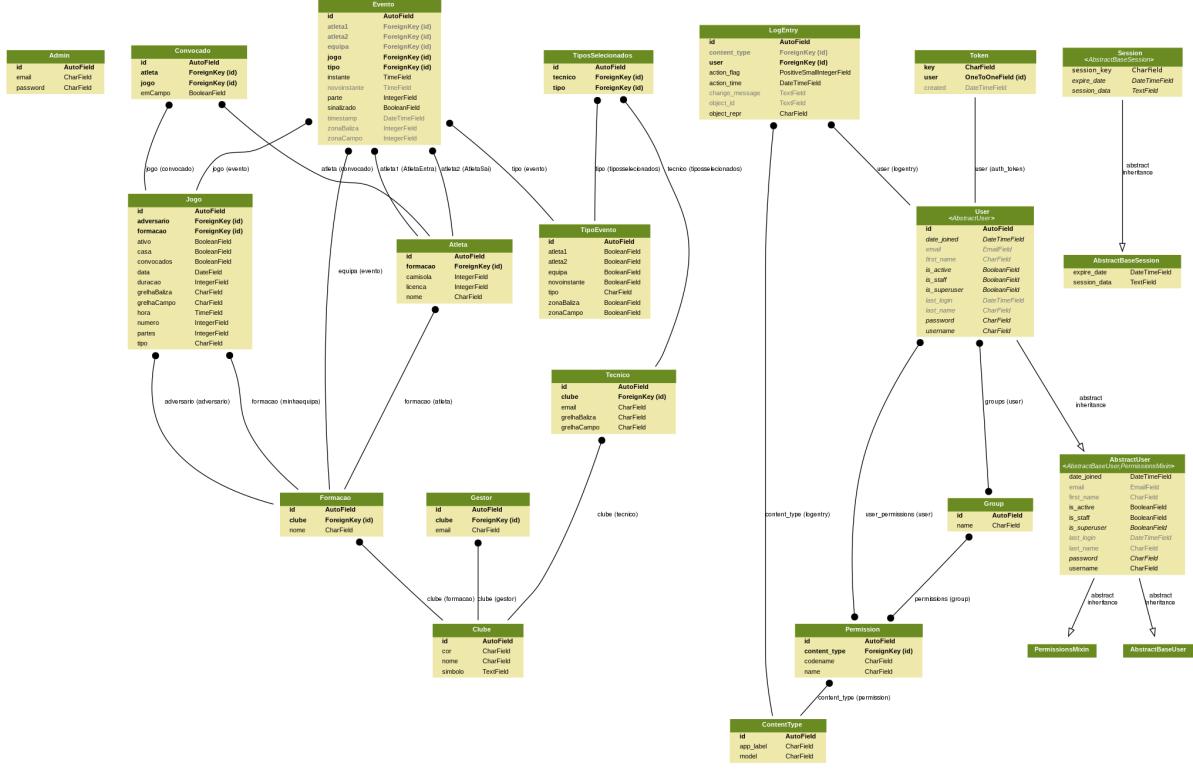


Figura 4: Diagrama de Classes

6 Prototipagem

Tendo em conta o objetivo principal desta aplicação, o registo de eventos, rapidamente se percebe que a interface é uma das partes mais importantes do projeto. Esta deve ser bastante intuitiva, simples e prática, permitindo que o registo de um evento seja uma ação rápida de concluir.

Assim, para se concluir a aplicação com uma interface que seja capaz de cumprir todos os pontos necessários, optou-se por desenvolver protótipos para a mesma, tendo estes sido validados pelo cliente. De seguida, apresentam-se os *mockups* desenvolvidos.



Figura 5: Página inicial com a listagem completa de jogos.



Figura 6: Página representativa do resumo de um jogo.

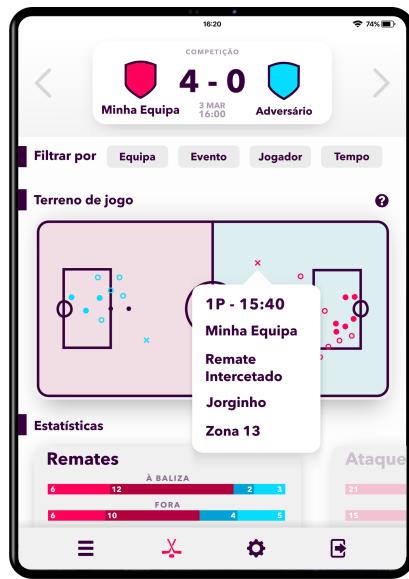


Figura 7: Exemplo da representação detalhada de um determinado evento de um jogo.



Figura 8: Exemplo de seleção de eventos que serão monitorizados no decorrer de um determinado jogo.



Figura 9: Página de um jogo a realizar e ligação para encontros anteriores.



Figura 10: Página de seleção dos convocados de uma formação para um determinado jogo.

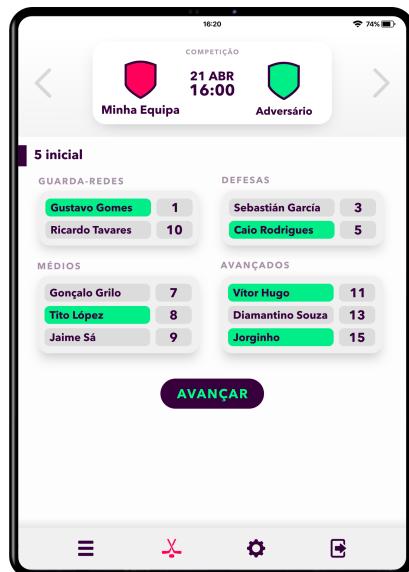


Figura 11: Página de seleção do 5 inicial de uma formação para um determinado jogo.



Figura 12: Exemplo demonstrativo do ecrã de marcação de eventos.



Figura 13: Exemplo demonstrativo do mapa do campo durante a marcação de eventos.



Figura 14: Exemplo demonstrativo da grelha da baliza durante a marcação de eventos.

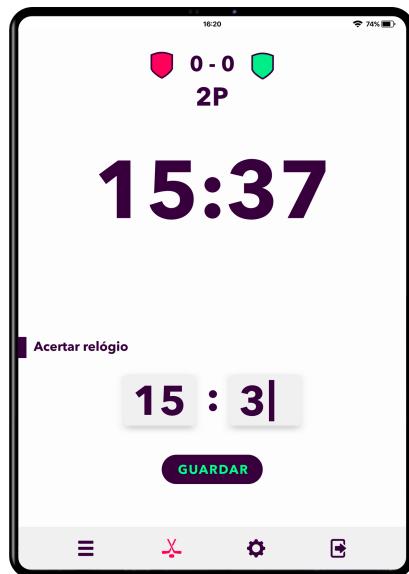


Figura 15: Página de visualização e alteração do relógio do jogo.

7 Implementação

7.1 Arquitetura

A aplicação web, à qual se denominou *HoqueiStas*, tem como principais componentes o *front end*, *back end* e a base de dados. O *front end* disponibiliza a interface para os clientes, neste caso técnicos, gestores ou administrador, poderem realizar ações. Dependendo das ações dos clientes, o *front end* envia pedidos HTTP para o *back end* e recebe de volta respostas HTTP. Para processar os pedidos e realizar as respostas, o *back end* acede e armazena/atualiza/remove dados da base de dados.

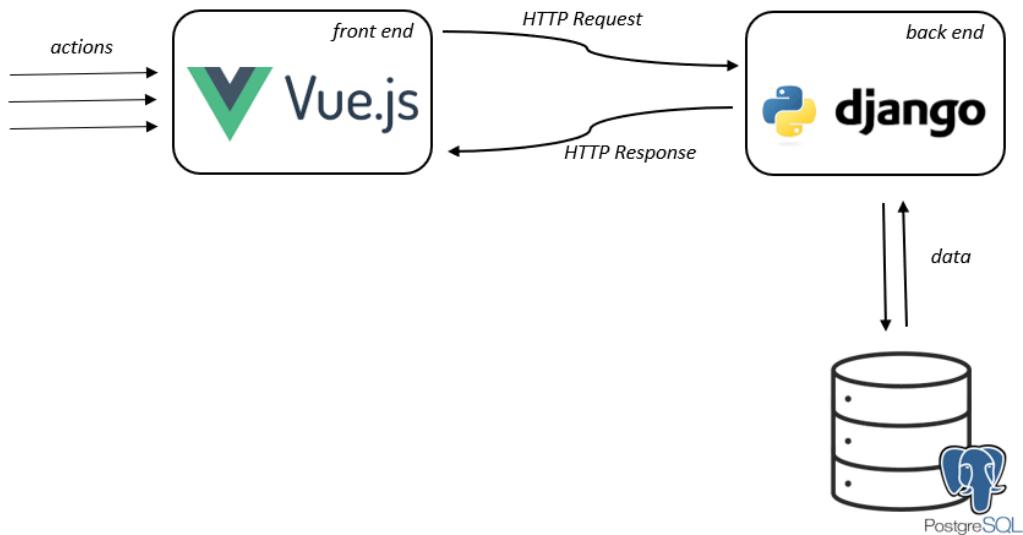


Figura 16: Arquitetura da aplicação.

7.2 Base de Dados

Tendo-se usado como ferramenta para o *back end* o *Django*, não houve a necessidade de construir a base de dados do projeto manualmente. Isto é, foi apenas criada uma base de dados *PostgreSQL* e conectou-se a mesmo ao projeto *Django*. Depois, a partir dos modelos (*models*) desenvolvidos no mesmo, migrou-se os mesmos para a base de dados, gerando automaticamente as tabelas com os atributos necessários.

As tabelas e atributos gerados na base de dados correspondem aos mesmos apresentados no diagrama de classes (figura 4), também criado a partir do *back end*.

7.3 *Back end*

O *back end* é responsável por toda a lógica de negócio e pelo tratamento dos dados, sendo este que tem acesso à base de dados. Tal como já foi referido anteriormente, o *back end* da aplicação foi desenvolvido em *Python*, recorrendo ao *Django*. Assim, seguiram-se as normas gerais desta framework, tendo-se desenvolvido vistas (*views*), modelos (*models*) e urls.

Duma maneira geral, quando o *back end* recebe um pedido http, tenta fazer *match* com um dos urls disponibilizados, encaminhando o pedido para a vista associada ao url correto, caso este exista. Nas vistas, são processadas as ações definidas, sendo os dados necessários carregados e/ou guardados/atualizados na base de dados a partir dos modelos correspondentes.

7.4 *Front end*

O *front end* é o que permite ter uma aplicação web com interface dinâmica. Para tal, desenvolveram-se páginas em HTML, que se tornaram dinâmicas com *JavaScript*. Para melhorar a estética e design das páginas, recorreu-se ao CSS e a *packages* do *Vue.js*, como o *Vuetify*. Para além disso, tirou-se também partido do *Vue.js* para facilitar a utilização do *JavaScript* na composição das páginas.

7.5 Comunicação

Para a comunicação entre o *front end* e o *back end* recorreu-se a uma abordagem *Restfull*, em que é feita a troca de pedidos e respostas sob a forma de pedidos HTTP GET e POST. Para tal, no *front end* utilizou-se o *package* *axios* e no *back end* utilizou-se o *package* de http do *Django*.

Esta abordagem tem como principal vantagem a sua simplicidade de implementação e como principal desvantagem o facto de se ter que ter perfeito conhecimento sobre os urls definidos no *back end* para se poder realizar os pedidos.

7.6 API

Nesta secção estão representadas todas as ações que se pode realizar pelo *back end*, apresentando-se o tipo de pedido HTTP, o url e os dados necessários para cada uma, bem como a resposta obtida.

7.6.1 Adicionar atleta

Para adicionar um atleta à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘licenca’, ‘camisola’, ‘nome’ e ‘formacao’. A ‘formacao’ corresponde ao ‘id’ da formação ao qual se deve associar o atleta.

url: ‘<http://localhost:8000/server/atleta/>’

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "licenca": "12345",  
    "camisola": 12,  
    "nome": "Atleta 12",  
    "formacao": 1  
}
```

7.6.2 Adicionar clube

Para adicionar um clube à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: nome, cor e simbolo.

url: ‘<http://localhost:8000/server/clube/>’

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST

esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "nome": "SL Benfica",  
    "cor": "#EF2F22",  
    "simbolo": "https://www.zerozero.pt/img/logos/equipas/4_imgbank.png"  
}
```

7.6.3 Adicionar formação

Para adicionar uma formação à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘nome’ e ‘clube’. O ‘clube’ corresponde ao ‘id’ do clube ao qual se deve associar a formação.

```
url: 'http://localhost:8000/server/formacao/'
```

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "nome": "Juvenis",  
    "clube": 1  
}
```

7.6.4 Adicionar jogo

Para adicionar um jogo à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘numero’, ‘casa’, ‘data’, ‘hora’, ‘tipo’, ‘duracao’, ‘partes’, ‘formacao’ e ‘formacaoAdv’. O ‘numero’ corresponde ao

número de jogo, ‘casa’ indica se o jogo se realiza em casa ou fora, ‘tipo’ indica se é competição ou amigável, ‘formacao’ corresponde ao ‘id’ da formação do clube que vai jogar e ‘formacaoAdv’ corresponde ao ‘id’ da formação adversária. É importante ter em atenção que no campo casa deve ser enviado um boolean, sendo que o true corresponde ao jogo ser realizado em casa e o false a ser realizado fora. Para além disso, data deve ser enviada no formato ”AAAA-MM-DD”, a hora no formato ”HH:MM:SS” e a duração, que está em minutos, necessita apenas de um inteiro.

url: 'http://localhost:8000/server/jogo/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "numero": 4,  
    "casa": false,  
    "data": "2019-06-25",  
    "hora": "17:30:00",  
    "tipo": "Competição",  
    "duracao": 10,  
    "partes": 4,  
    "formacao": 1,  
    "formacaoAdv": 5  
}
```

7.6.5 Adicionar tipo de evento

Para adicionar um tipo de evento à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘tipo’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘zonaC’, ‘zonaB’, ‘novoinst’. O ‘tipo’ é a denominação do evento, o ‘atleta2’ corresponde ao atleta suplente (por exemplo o atleta que vai entrar em campo), a ‘zonaC’ corresponde à zona de campo, a ‘zonaB’ corresponde à zona da

baliza e o ‘novoinst’ corresponde ao novo instante. À parte do ‘tipo’, todos os restantes campos devem receber um boolean, indicando se são ou não obrigatórios no registo de eventos deste tipo.

url: 'http://localhost:8000/server/tipo_evento/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "tipo": "Golo",  
    "equipa": true,  
    "atleta1": true,  
    "atleta2": false,  
    "zonaC": true,  
    "zonaB": true,  
    "novoinst": false  
}
```

7.6.6 Alterar grelhas de um técnico

Para alterar as grelhas de campo e baliza consideradas por um técnico deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘email’, ‘grelhaC’ e ‘grelhaB’. A ‘grelhaC’ corresponde à grelha do campo e a ‘grelhaB’ à grelha da Baliza, sendo que o formato da *string* das grelhas deve ser “mxn” correspondendo o ‘m’ ao número de linhas e o ‘n’ ao número de colunas num campo vertical e numa baliza.

url: 'http://localhost:8000/server/change_tecnico/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o

código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "email": "tecnico1@gmail.com",  
    "grelhaC": "8x4",  
    "grelhaB": "3x3"  
}
```

7.6.7 Alterar grelhas de um jogo

Para alterar as grelhas de campo e baliza utilizadas num determinado jogo deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘id’, ‘grelhaC’ e ‘grelhaB’. O ‘id’ corresponde ao ‘id’ do jogo em questão, a ‘grelhaC’ à grelha do campo e a ‘grelhaB’ à grelha da baliza. O formato da *string* das grelhas deve ser “mxn” correspondendo o ‘m’ ao número de linhas e o ‘n’ ao número de colunas num campo vertical.

url: 'http://localhost:8000/server/change_jogo/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ’ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "id": 5,  
    "grelhaC": "8x4",  
    "grelhaB": "3x3"  
}
```

7.6.8 Alterar informações de um evento

Para alterar os dados de um evento deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário

JSON com os campos: ‘id’, ‘instante’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘zona-Campo’, ‘zonaBaliza’, ‘novoinstante’, ‘parte’ e ‘jogo’. O tipo de evento não pode ser alterado, quando tal é necessário aconselha-se a remover o evento e registar um novo. Os campos que não são obrigatórios devem ser enviados a null. Tanto o ‘instante’ como o ‘novoinstante’ devem ser enviados com o formato “HH:MM:SS”. Mesmo que se queira alterar apenas um dado do evento, todos os campos devem ser enviados, sendo que os campos que se quer manter devem ser enviados com as informações anteriores.

url: ‘http://localhost:8000/server/change_evento/’

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "id": 7,  
    "instante": "00:12:12",  
    "equipa": 1,  
    "atleta1": 0,  
    "atleta2": null,  
    "zonaCampo": 15,  
    "zonaBaliza": null,  
    "novoinstante": null,  
    "parte": 1,  
    "jogo": 5  
}
```

7.6.9 Assinalar que já foram escolhidos os convocados para um jogo

Para indicar que já foram escolhidos os atletas convocados para um jogo deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do jogo em questão.

url: 'http://localhost:8000/server/confirm_convocados/<int:id>/'

No caso do ‘id’ ser válido, quando o *back end* recebe este pedido, altera o campo convocados do jogo em questão para False e envia um ‘ok’ num HTTP *response* com código 200. Caso o ‘id’ não seja válido, não é efetuada nenhuma ação.

7.6.10 Associar sugestão de tipo de evento a técnico

Para selecionar um tipo de evento como sugestão para um determinado técnico deve ser enviado um GET para o url apresentado de seguida. Devem ser enviados como parâmetros do GET o ‘id’ do tipo de evento e o email do técnico em questão.

url: 'http://localhost:8000/server/tipo_selecionado/<int:tipo>/<str:email>/'

No caso dos parâmetros serem válidos, quando o *back end* recebe este pedido, cria o novo tipo selecionado para o tipo de evento e técnico considerados e envia um ‘ok’ num HTTP *response* com código 200. Caso o ‘id’ do tipo de evento ou o email não sejam válidos, é enviado um HTTP *response* com o código 404 (Not Found).

7.6.11 Definir atletas convocados para um jogo e 5 iniciais

Para definir os atletas convocados para um jogo e os 5 atletas iniciais em campo deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘jogo’, ‘atletas’ e ‘inicial’. O jogo corresponde ao ‘id’ do jogo em questão. O campo atletas corresponde a um array que contém dicionários JSON, sendo que cada dicionário contém os campos ‘id’ e ‘camisola’. Assim, no array atletas são enviados os atletas convocados para o jogo. O campo inicial corresponde a um array que contém dicionários JSON, sendo que estes devem corresponder a entradas existentes no array de atletas. No array ‘inicial’ são enviados os 5 atletas que entram em campo no início do jogo. O campo ‘camisola’ foi considerado para se poder alterar a camisola dos atletas no início de cada jogo.

url: 'http://localhost:8000/server/convocados/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "jogo": 5,  
    "atletas": [  
        {  
            "id": 1,  
            "camisola": 12  
        },  
        {  
            "id": 5,  
            "camisola": 4  
        }  
    ],  
    "inicial": [  
        {  
            "id": 1,  
            "camisola": 12  
        }  
    ]  
}
```

7.6.12 Mudar atleta de formação

Para alterar a formação de um atleta deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘licenca’ e ‘formacao’. A ‘formacao’ corresponde ao ‘id’ da nova ‘formação’ ao qual se deve associar o atleta.

```
url: 'http://localhost:8000/server/change_atleta/'
```

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado

pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "licenca": "12345",  
    "formacao": 2  
}
```

7.6.13 Obter atletas em campo

Para obter as informações sobre os atletas de uma formação que estão em campo num determinado jogo deve ser enviado um GET para o url apresentado de seguida. Devem ser enviados como parâmetros do GET os ‘id’s da formação e do jogo em questão.

url: 'http://localhost:8000/server/get_atletas_campo/<int:formacao>/<int:jogo>/'

No caso de os ‘id’s serem válidos, a este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionários contém os campos: ‘id’, ‘camisola’ e ‘nome’. Caso algum dos ‘id’s não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações dos atletas da formação com **id=1** que estão em campo no jogo com **id=5**, deve ser enviado um GET

'http://localhost:8000/server/get_atletas_campo/1/5/' , ao qual o *back end* responde com o JSON que se segue.

```
[  
    {  
        "licenca": 0,  
        "id": 0,  
        "nome": "Carles Grau",  
        "camisola": 1  
    }  
]
```

7.6.14 Obter atletas suplentes

Para obter as informações sobre os atletas de uma formação que estão no banco num determinado jogo deve ser enviado um GET para o url apresentado de seguida. Devem ser enviados como parâmetros do GET os ‘id’s da formação e do jogo em questão.

url: 'http://localhost:8000/server/get_atletas_suplentes/<int:formacao>/<int:jogo>/'

No caso de os ‘id’s serem válidos, a este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionário contém os campos: ‘id’, ‘camisola’ e ‘nome’. Caso algum dos ‘id’s não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações dos atletas da formação com **id=1** que estão no banco no jogo com **id=5**, deve ser enviado um GET

'http://localhost:8000/server/get_atletas_suplentes/1/5/' , ao qual o *back end* responde com o JSON que se segue.

```
[  
  {  
    "licenca": 6,  
    "id": 6,  
    "nome": "Andrés Castaño",  
    "camisola": 4  
  }  
]
```

7.6.15 Obter informações de um clube a partir de uma formação

Para obter as informações sobre um clube a partir de uma das suas formações deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ da formação em questão.

url: 'http://localhost:8000/server/adversario_nome/<int:form_id>/'

No caso de a formação ser válida, a este pedido, o *back end* responde com um dicionário JSON com os campos: ‘id’, ‘nome’, ‘cor’ e ‘simbolo’. Caso o

‘id’ da formação não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações do clube da formação com `id=1`, deve ser enviado um GET `'http://localhost:8000/server/adversario_nome/1/'`, ao qual o *back end* responde com o JSON que se segue.

```
{  
    "id": 1,  
    "nome": "FC Porto",  
    "cor": "#0F579E",  
    "simbolo": "FC_Porto.svg/1200px-FC_Porto.svg.png"  
}
```

7.6.16 Obter informações sobre um evento

Para obter as informações sobre um evento deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do evento em questão.

`url: 'http://localhost:8000/server/get_evento/<int:id>/'`

No caso de o ‘id’ ser válido, a este pedido, o *back end* responde com um dicionário JSON com os campos: ‘id’, ‘jogo’, ‘tipo’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘novoinstante’, ‘instante’, ‘timestamp’, ‘parte’, ‘sinalizado’, ‘zonaCampo’ e ‘zonaBaliza’. Caso o ‘id’ não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações de um evento com `id=7`, deve ser enviado um GET `'http://localhost:8000/server/get_evento/7/'`, ao qual o *back end* responde com o JSON que se segue.

```
{  
    "id": 7,  
    "jogo": 5,  
    "tipo": 2,
```

```

    "equipa": 1,
    "atleta1": 0,
    "atleta2": null,
    "novoinstante": null,
    "instante": "00:17:12",
    "timestamp": "2019-05-25T18:32:23.347Z",
    "parte": 1,
    "sinalizado": false,
    "zonaCampo": 6,
    "zonaBaliza": null
}

```

7.6.17 Obter informações sobre um jogo

Para obter as informações sobre um jogo deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do jogo em questão.

url: ’http://localhost:8000/server/get_jogo/<int:id>/’

No caso de o ‘id’ ser válido, a este pedido, o *back end* responde com um dicionário JSON com os campos: ‘id’, ‘numero’, ‘tipo’, ‘c’, ‘casa’, ‘data’, ‘hora’, ‘resultado’, ‘ativo’, ‘grelhaCampo’, ‘grelhaBaliza’, ‘adversario’, ‘formacao’, ‘adv_nome’, ‘form_nome’, ‘clube_nome’, ‘logoMe’, ‘logoAdv’, ‘clube_cor’, ‘adv_cor’, ‘duracao’ e ‘partes’. O ‘tipo’ indica se o jogo é de competição ou amigável, os campos ‘c’ e ‘casa’ indicam se o jogo é em casa ou fora e o campo ‘ativo’ indica se o jogo já se realizou ou não. Caso o ‘id’ não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações de um jogo com `id=5`, deve ser enviado um GET `’http://localhost:8000/server/get_jogo/5/’`, ao qual o *back end* responde com o JSON que se segue.

```
{
    "id": 5,
    "numero": 5,
    "tipo": "Competição",
    ...
}
```

```

    "c": "C",
    "casa": "CASA",
    "data": "2019-05-25",
    "hora": "18:00",
    "resultado": "0-0",
    "ativo": true,
    "grelhaCampo": "8x4",
    "grelhaBaliza": "3x3",
    "adversario": 9,
    "formacao": 1,
    "adv_nome": "HC Turquel",
    "form_nome": "Séniores",
    "clube_nome": "FC Porto",
    "logoMe": "FC_Porto.svg/1200px-FC_Porto.svg.png",
    "logoAdv": "https://www.zerozero.pt/img/logos/equipas/209898_imgbank.png",
    "clube_cor": "#0F579E",
    "adv_cor": "#111111",
    "duracao": 10,
    "partes": 4
}

```

7.6.18 Obter informações sobre um tipo de evento

Para obter as informações sobre um tipo de evento, isto é, sobre os campos que são obrigatórios para esse tipo, para além dos campos comuns a todos, deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do tipo de evento em questão.

url: ‘http://localhost:8000/server/get_tipo_evento/<int:id>’

No caso de o ‘id’ ser válido, a este pedido, o *back end* responde com um dicionário JSON com os campos: ‘id’, ‘tipo’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘zonaCampo’, ‘zonaBaliza’, ‘novoinstante’. Caso o ‘id’ não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações de um tipo de evento com `id=0`, deve ser enviado um GET ‘http://localhost:8000/server/get_tipo_evento/0/’ , ao qual o

back end responde com o JSON que se segue.

```
{  
  "id": 0,  
  "tipo": "Golo",  
  "equipa": true,  
  "atleta1": true,  
  "atleta2": false,  
  "zonaCampo": true,  
  "zonaBaliza": true,  
  "novoinstante": false  
}
```

7.6.19 Obter informações sobre um utilizador

Para obter as informações sobre um utilizador deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o email do utilizador em questão.

url: 'http://localhost:8000/server/info_user/<str:email>/'

No caso de o email ser válido, a este pedido, o *back end* responde com um dicionário JSON com os campos: ‘email’, ‘nome’, ‘pass’, ‘tipo’, ‘clube’ e ‘clube_logo’. O ‘tipo’ corresponde ao tipo de utilizador, Técnico ou Gestor, e, no caso de o utilizador ser um Técnico, são enviados também os campos: ‘grelhaCampo’ e ‘grelhaBaliza’. Caso o email não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações de um Gestor com email ‘gestor1@gmail.com’, deve ser enviado um GET ’http://localhost:8000/server/info_user/gestor1@gmail.com/’ , ao qual o *back end* responde com o JSON que se segue.

```
{  
  "email": "gestor1@gmail.com",  
  "nome": "Gestor 1",  
  "pass": "gestor1",  
  "tipo": "gestor",
```

```

    "clube": 1,
    "clube_logo": "FC_Porto.svg/1200px-FC_Porto.svg.png"
}

```

7.6.20 Obter sugestões de tipos de eventos para um técnico

Para obter as sugestões de tipos de eventos de um técnico deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o email do técnico em questão.

url: 'http://localhost:8000/server/get_tipos_selecionados/<str:email>/'

No caso de o email ser válido, a este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionários contém os campos: ‘id’, ‘idEvento’ e ‘tipo’. O ‘idEvento’ corresponde ao ‘id’ do tipo de evento e o ‘tipo’ à denominação do evento. Caso o email não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as sugestões de um técnico com email ‘tecnico1@gmail.com’ , deve ser enviado um GET

‘http://localhost:8000/server/get_tipos_selecionados/tecnico1@gmail.com/’ , ao qual o *back end* responde com o JSON que se segue.

```
[
{
  "id": 0,
  "idEvento": 0,
  "tipo": "Golo"
},
{
  "id": 1,
  "idEvento": 1,
  "tipo": "Remate à baliza"
},
{
  "id": 3,
  "idEvento": 15,
}
```

```

        "tipo": "Time-out"
    }
]
```

7.6.21 Obter todas as formações de um clube

Para obter as informações sobre todas as formações de um clube deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do clube em questão.

url: 'http://localhost:8000/server/get_formacoes/<int:clube>/'

No caso de o id ser válido, a este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionário contém os campos: id, nome e clube_id. Caso o id do clube não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações das formações do clube com **id=1**, deve ser enviado um GET 'http://localhost:8000/server/get_formacoes/1/' , ao qual o *back end* responde com o JSON que se segue.

```
[
{
    "id": 1,
    "nome": "Séniores",
    "clube_id": 1
}]
```

7.6.22 Obter todos os atletas de uma formação

Para obter as informações sobre todos os atletas de uma formação deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ da formação em questão.

url: 'http://localhost:8000/server/get_atletas/<int:formacao>/'

No caso de o id ser válido, a este pedido, o *back end* responde com um

array de dicionários JSON, em que cada dicionário contém os campos: ‘licença’, ‘id’, ‘nome’ e ‘camisola’. Caso o ‘id’ da formação não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações dos atletas da formação com *id=1*, deve ser enviado um GET `'http://localhost:8000/server/get_atletas/1/'`, ao qual o *back end* responde com o JSON que se segue.

```
[  
  {  
    "licenca": 0,  
    "id": 0,  
    "nome": "Carles Grau",  
    "camisola": 1  
  },  
  {  
    "licenca": 6,  
    "id": 6,  
    "nome": "Andrés Castaño",  
    "camisola": 4  
  }  
]
```

7.6.23 Obter todos os clubes

Para obter as informações sobre todos os clubes existentes na base de dados deve ser enviado um GET para o url apresentado de seguida.

url: `'http://localhost:8000/server/get_clubes/'`

A este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionário contém os campos: ‘id’, ‘nome’, ‘simbolo’ e ‘cor’.

7.6.24 Obter todos os eventos de um jogo

Para obter as informações sobre todos os eventos de um jogo deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do jogo em questão.

```
url: 'http://localhost:8000/server/get_eventos/<int:idJogo>/'
```

No caso de o ‘id’ ser válido, a este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionário contém os campos: ‘id’, ‘jogo’, ‘tipo’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘instante’, ‘novoinstante’, ‘zonaBaliza’, ‘timestamp’, ‘parte’, ‘sinalizado’, ‘gcy’, ‘gcx’, ‘size’. O ‘gcy’ corresponde à coordenada ‘y’ da zona do campo, o ‘gcx’ corresponde à coordenada ‘x’ e o ‘size’ é um atributo necessário para a representação gráfica dos eventos no campo. Caso o ‘id’ do jogo não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

Exemplo de aplicação

Para obter as informações dos eventos do jogo com `id=5`, deve ser enviado um GET `'http://localhost:8000/server/get_eventos/5/'`, ao qual o *back end* responde com o JSON que se segue.

```
[  
  {  
    "id": 7,  
    "jogo": 5,  
    "tipo": "Remate intercetado",  
    "equipa": "FC Porto",  
    "atleta1": "Carles Grau",  
    "atleta2": " - ",  
    "instante": "00:22:12",  
    "novoinstante": " - ",  
    "zonaBaliza": " - ",  
    "timestamp": "2019-05-25T18:32:23.347Z",  
    "parte": 1,  
    "sinalizado": false,  
    "gcy": 38,  
    "gcx": 27,  
    "size": 1  
  }]  
]
```

7.6.25 Obter todos os jogos de um clube

Para obter as informações sobre todos os jogos de um clube deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do clube em questão.

url: 'http://localhost:8000/server/get_jogos/<int:clube>/'

No caso de o ‘id’ ser válido, a este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionário contém os campos: ‘numero’, ‘id’, ‘tipo’, ‘c’, ‘casa’, ‘ativo’, ‘data’, ‘hora’, ‘resultado’, ‘grelhaCampo’, ‘grelhaBaliza’, ‘adversario’, ‘formacao’, ‘adv_nome’, ‘form_nome’, ‘duracao’, ‘partes’ e ‘convocados’. O ‘tipo’ indica se o jogo é de competição ou amigável, os campos ‘c’ e ‘casa’ indicam se o jogo é em casa ou fora, o campo ‘ativo’ indica se o jogo já se realizou ou não e o campo ‘convocados’ indica se já foram ou não escolhidos os atletas convocados para esse jogo. Caso o ‘id’ do clube não seja válido, é enviado um HTTP *response* com o código 404 (Not Founds).

Exemplo de aplicação

Para obter as informações dos jogos do clube com **id=1**, deve ser enviado um GET 'http://localhost:8000/server/get_jogos/1/' , ao qual o *back end* responde com o JSON que se segue.

```
[  
 {  
     "numero": 6,  
     "id": 6,  
     "tipo": "Competição",  
     "c": "C",  
     "casa": "CASA",  
     "ativo": false,  
     "data": "2019-03-16",  
     "hora": "17:00:00",  
     "resultado": "3-1",  
     "grelhaCampo": "8x4",  
     "grelhaBaliza": "3x3",  
     "adversario": 3,  
     "formacao": 1,  
     "adv_nome": "Sporting CP",
```

```

    "form_nome": "Séniores",
    "duracao": 20,
    "partes": 2,
    "convocados": true
}
]

```

7.6.26 Obter todos os tipos de eventos

Para obter as informações sobre todos os tipos de eventos existentes na base de dados, isto é, os campos obrigatórios em cada tipo de evento existente, para além dos campos comuns a todos, deve ser enviado um GET para o url apresentado de seguida.

url: 'http://localhost:8000/server/get_tipos_eventos/'

A este pedido, o *back end* responde com um array de dicionários JSON, em que cada dicionário contém os campos: ‘id’, ‘tipo’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘zonaCampo’, ‘zonaBaliza’, ‘novoinstante’.

7.6.27 Registar evento

Para adicionar um evento à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘tipo’, ‘jogo’, ‘instante’, ‘parte’, ‘equipa’, ‘atleta1’, ‘atleta2’, ‘zonaC’, ‘zonaB’, ‘novoinst’. O ‘tipo’ corresponde ao ‘id’ do tipo de evento, o ‘jogo’ corresponde ao ‘id’ do jogo onde ocorreu o evento, o ‘instante’ corresponde ao instante do relógio do jogo, a parte corresponde à parte do jogo em que ocorreu o evento, a ‘zonaC’ corresponde à zona do campo, a ‘zonaB’ corresponde à zona da baliza e o ‘novoinst’ corresponde ao novo instante. No caso de a equipa, ‘atleta1’, ‘atleta2’, ‘zonaC’, ‘zonaB’ e/ou ‘novoinst’ não serem necessários, deve ser enviado o valor null nesses campos. Tanto o instante como o novoinst devem ser enviados com o formato “HH:MM:SS”.

url: 'http://localhost:8000/server/evento/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado

pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "tipo": 0,  
    "jogo": 6,  
    "instante": "00:08:50",  
    "parte": 1,  
    "equipa": 1,  
    "atleta1": 1,  
    "atleta2": null,  
    "zonaC": 11,  
    "zonaB": 9,  
    "novoinst": null  
}
```

7.6.28 Registar gestor

Para adicionar um gestor à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘email’, ‘password’, ‘nome’ e ‘clube’. O ‘clube’ corresponde ao ‘id’ do clube a que se deve associar o gestor.

```
url: 'http://localhost:8000/server/gestor/'
```

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{  
    "email": "gestor1@gmail.com",  
    "password": "gestor1",  
    "name": "Gestor 1",  
    "club": 1  
}
```

```
        "nome": "Gestor 1",
        "clube": 1
    }
```

7.6.29 Registar técnico

Para adicionar um técnico à base de dados deve ser enviado um POST para o url apresentado de seguida. No corpo do POST deve ser enviado um dicionário JSON com os campos: ‘email’, ‘password’, ‘nome’ e ‘clube’. O ‘clube’ corresponde ao ‘id’ do clube a que se deve associar o técnico.

url: 'http://localhost:8000/server/tecnico/'

No caso de o POST ser válido e a inserção ser bem sucedida, é enviado pelo *back end* um ‘ok’ num HTTP *response* com código 200. Caso o POST esteja incorreto ou alguma ação falhe, é enviado um HTTP *response* com o código 400 (Bad Request) ou com o código 500 (Internal Server Error).

Exemplo de um JSON a ser enviado no POST

```
{
    "email": "tecnico1@gmail.com",
    "password": "tecnico1",
    "nome": "Tecnico 1",
    "clube": 1
}
```

7.6.30 Remover evento

Para eliminar o evento de um jogo deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do evento em questão.

url: 'http://localhost:8000/server/del_evento/<int:id>/'

No caso do ‘id’ ser válido, quando o *back end* recebe este pedido, elimina o evento em questão e envia um ‘ok’ num HTTP *response* com código 200. Caso o ‘id’ do evento não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

7.6.31 Remover sugestão de tipo de evento de um técnico

Para remover um tipo de evento como sugestão para um determinado técnico deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do tipo selecionado.

url: ‘http://localhost:8000/server/del_tipo_selecionado/<int:id>’

No caso do ‘id’ ser válido, quando o *back end* recebe este pedido, remove o tipo selecionado considerado e envia um ‘ok’ num HTTP *response* com código 200. Caso o ‘id’ do tipo selecionado não seja válido, é enviado um HTTP *response* com o código 404 (Not Found).

7.6.32 Sinalizar ou remover sinalização de evento

Para alterar a sinalização do evento de um jogo deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do evento em questão.

url: ‘http://localhost:8000/server/sinalizar_evento/<int:id>’

No caso do ‘id’ ser válido, quando o *back end* recebe este pedido, altera o campo sinalizado do evento em questão e envia um ‘ok’ num HTTP com código 200. Caso o ‘id’ do evento não seja válido, é enviado um HTTP com o código 404 (NOT FOUND). Assim, se o evento tiver o sinalizado a True, este passa a False e vice-versa.

7.6.33 Terminar jogo

Para indicar que um jogo já se realizou, ou seja, terminar um jogo, deve ser enviado um GET para o url apresentado de seguida. Deve ser enviado como parâmetro do GET o ‘id’ do jogo em questão.

url: ‘http://localhost:8000/server/end_jogo/<int:id>’

No caso do ‘id’ ser válido, quando o *back end* recebe este pedido, altera o campo ativo do jogo em questão para False e envia um ‘ok’ num HTTP *response* com código 200. Caso o ‘id’ não seja válido, não é efetuada nenhuma ação.

8 Configuração

Para tirar partido da aplicação desenvolvida é necessário correr uma base de dados *PostgreSQL*, o projeto *Django* e o projeto *Vue.js*. Neste momento, o *back end* está conectado a uma base de dados com as definições apresentadas de seguida. Deste modo, deve ser criada uma base de dados equivalente ou devem ser alteradas as configurações no *Django*.

```
DATABASES = {  
    'default': {  
        'ENGINE': 'django.db.backends.postgresql',  
        'NAME': 'hoqueistats',  
        'USER': 'hoqueistats',  
        'PASSWORD': 'hoqueistats',  
        'HOST': '127.0.0.1',  
        'PORT': '5432',  
    }  
}
```

Para iniciar o *back end*, para além de ter o *Django* e outros *packages* necessários instaladas, deve-se correr no terminal os seguintes comandos, sendo os dois primeiros utilizados para migrar os modelos para a base de dados (necessários apenas a primeira vez) e o último utilizado para correr o projeto.

```
$ python3 manage.py makemigrations  
$ python3 manage.py migrate  
$ python3 runserver
```

Por último, para iniciar o *front end*, para além de ter o *Vue.js*, *npm* e outros *packages* instalados, deve-se correr o seguinte comando.

```
$ npm run dev
```

Numa fase posterior, pode ser feito o *deployment* da aplicação, devendo a base de dados ser criada num servidor remoto central.

Para mais informações sobre o funcionamento da aplicação, nomeadamente como tirar partido das sua funcionalidades, deve ser consultado o Manual de Utilização.

9 Resultados

9.1 Interface final

O resultado final da aplicação encontra-se representado de seguida, onde se apresentam as páginas principais e funcionalidades. As páginas foram desenvolvidas tendo em consideração a prototipagem elaborada numa fase anterior pelo grupo. Assim, consideramos que de uma maneira geral, a interface final vai ao encontro dos *mockups* idealizados.

The screenshot shows the main dashboard of the 'Técnico' application. At the top right, there is a search bar labeled 'Pesquisar por adversário...'. Below it is a table listing seven matches. The columns are: # (Match Number), Local (Home/Away), Adversário (Opponent), Resultado (Score), and Data (Date). The table rows are:

#	Local	Adversário	Resultado	Data
6	C	Sporting CP	3-1	2019-03-16
5	C	HC Turquel	0-0	2019-05-25
4	F	AD Valongo	1-0	2019-05-19
3	C	Riba D'Ave HC	0-0	2019-05-15
2	F	FC Barcelona	0-0	2019-05-11
1	F	Sporting CP	0-0	2019-05-04
0	F	OC Barcelos		2019-04-27

Below the table, there are two buttons: 'ESTATÍSTICAS' and 'ESTATÍSTICAS AO VIVO'. At the bottom of the screen, there is a dark navigation bar with icons for menu, home, settings, and user profile.

Figura 17: Página inicial do Técnico com a listagem completa de jogos.

Pesquisar por adversário.

#	Local	Adversário	Resultado	
6	C	Sporting CP	3-1	201 03-1
5	C	HC Turquel	0-0	201 05-2
4	F	AD Valongo	1-0	201 05-1
3	C	Riba D'Ave HC	0-0	201 05-1
2	F	FC Barcelona	0-0	201 05-1
1	F	Sporting CP		201 05-0
0	F	OC Barcelos		201 04-2

ADICIONAR JOGO

≡ ⌂ ⚙ ⏴

Figura 18: Página inicial do Gestor com a listagem completa de jogos, display de telemóvel.

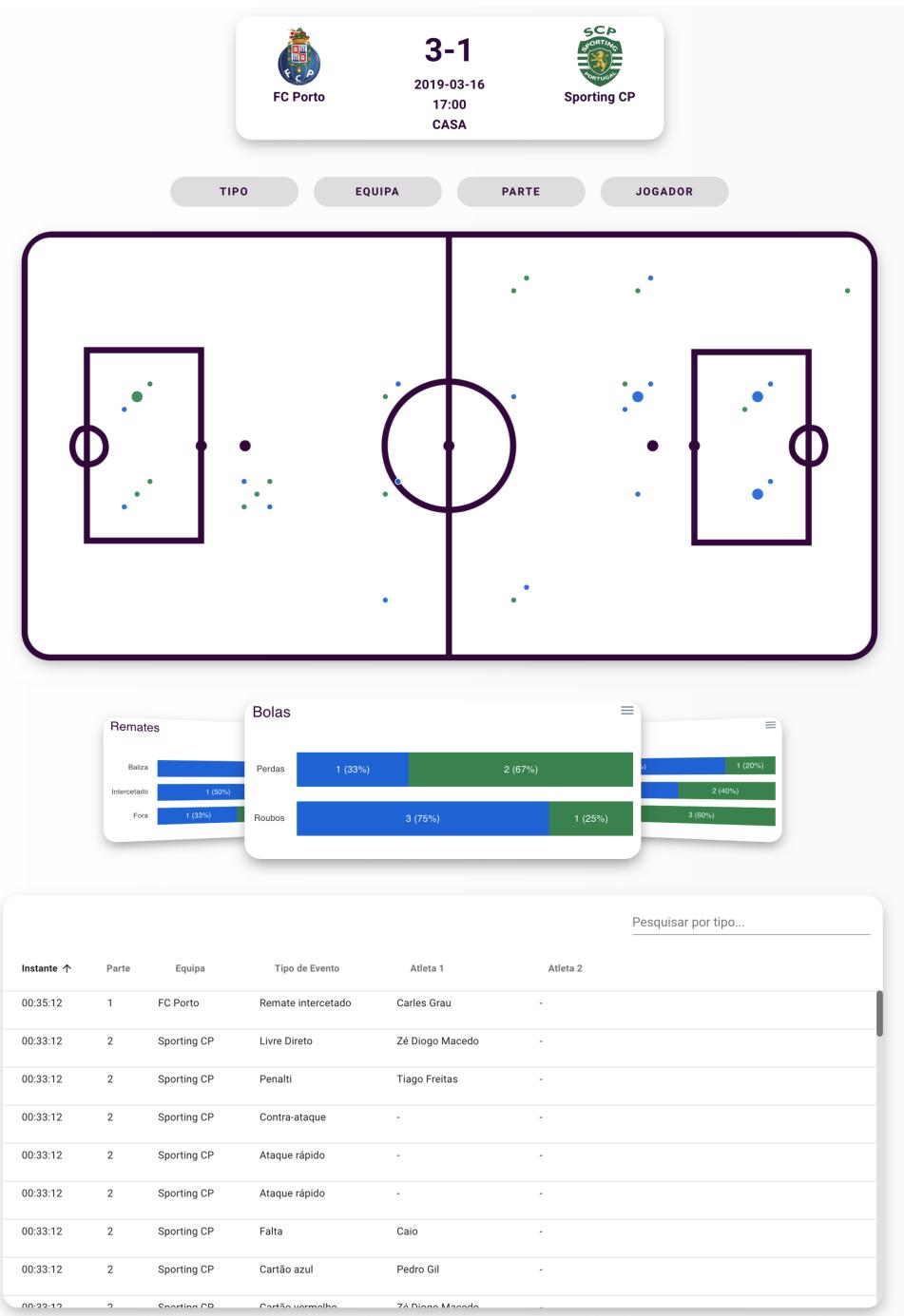


Figura 19: Página representativa do resumo de um jogo, display de computador.



Figura 20: Página representativa do resumo de um jogo, display de telemóvel.



Figura 21: Exemplo da representação detalhada de um determinado evento de um jogo.



Figura 22: Exemplo de seleção de eventos que serão monitorizados no decorrer de um determinado jogo.

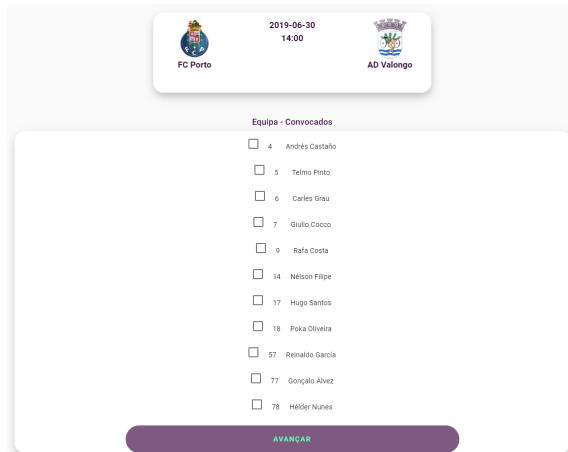


Figura 23: Página de seleção dos convocados de uma formação para um determinado jogo.

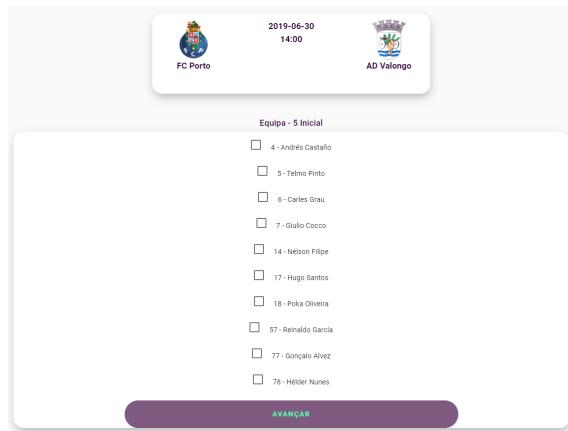


Figura 24: Página de seleção do 5 inicial de uma formação para um determinado jogo.

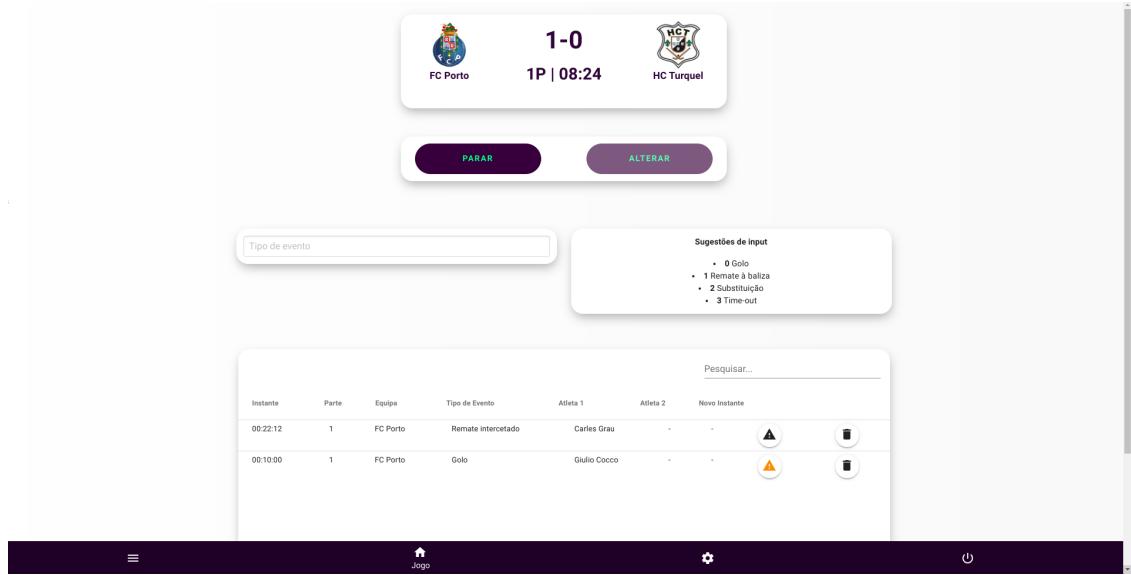


Figura 25: Exemplo demonstrativo do ecrã de marcação de eventos.

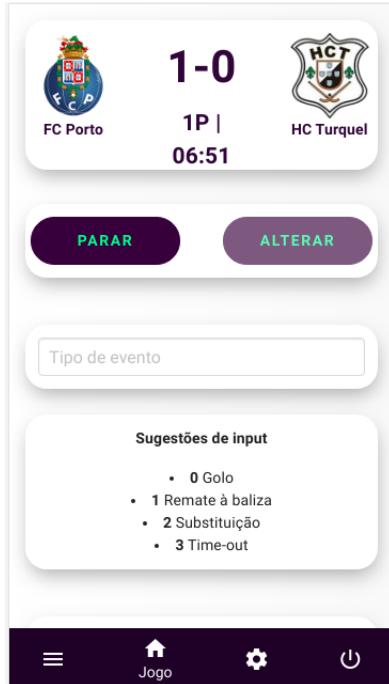


Figura 26: Exemplo demonstrativo do ecrã de marcação de eventos, display de telemóvel.

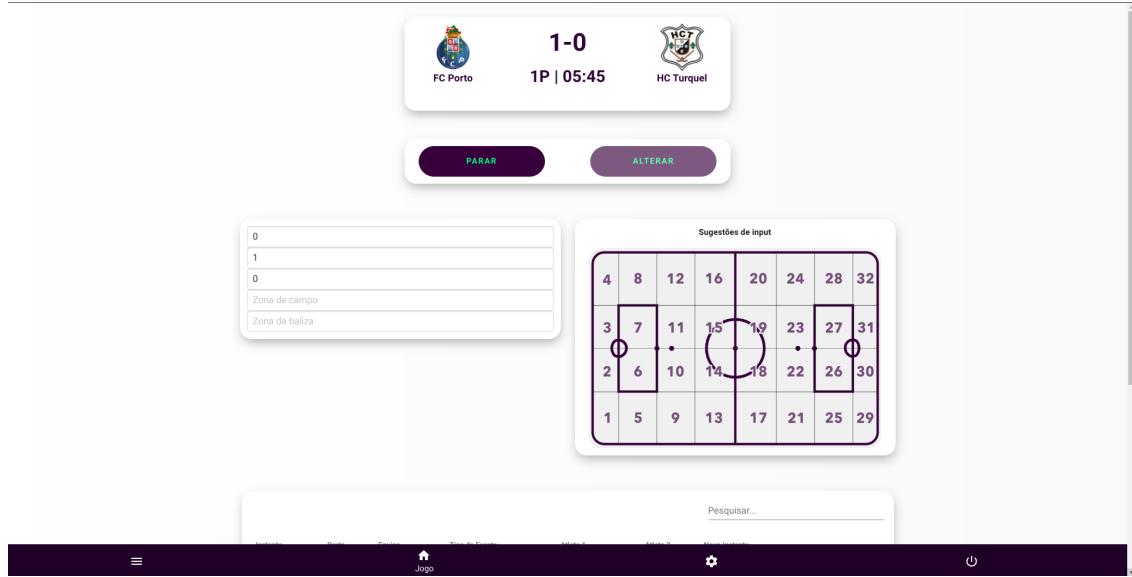


Figura 27: Exemplo demonstrativo do mapa do campo durante a marcação de eventos.



Figura 28: Exemplo demonstrativo do mapa do campo durante a marcação de eventos, display de telemóvel.

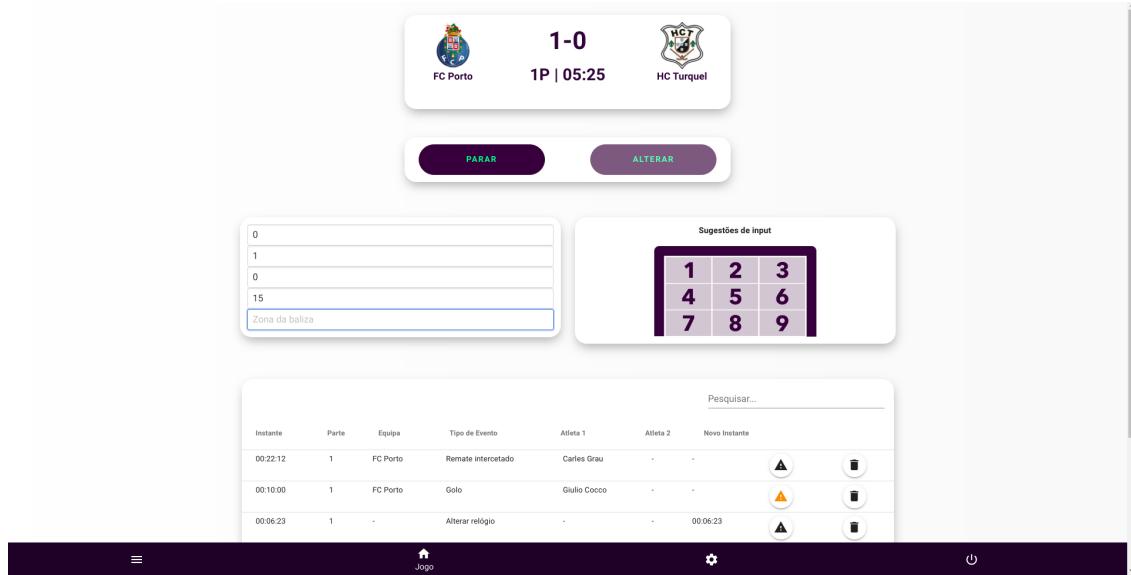


Figura 29: Exemplo demonstrativo da grelha da baliza durante a marcação de eventos.

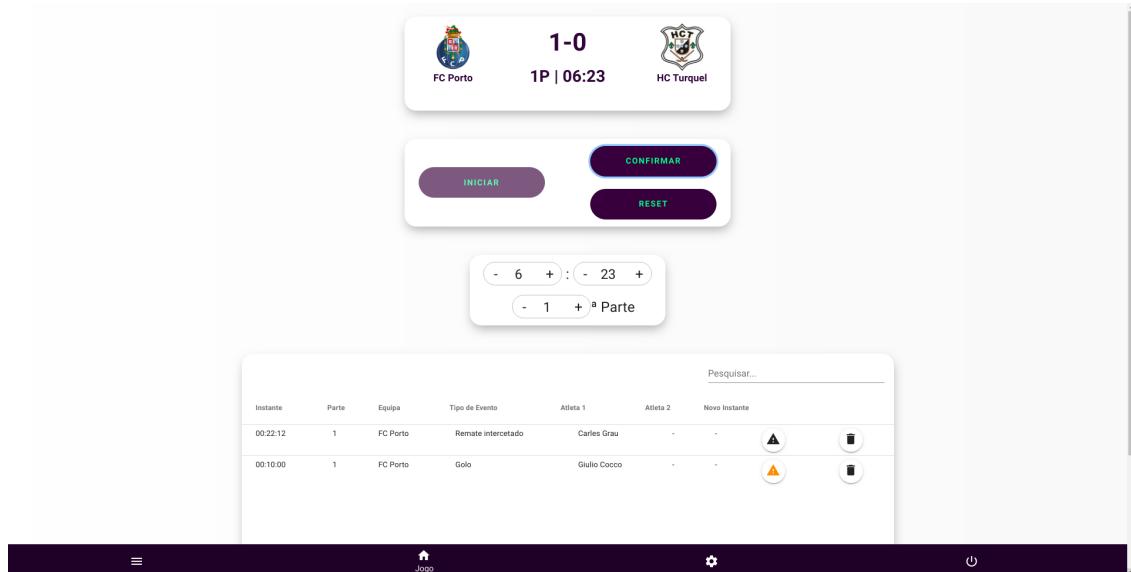


Figura 30: Alteração do relógio do jogo.

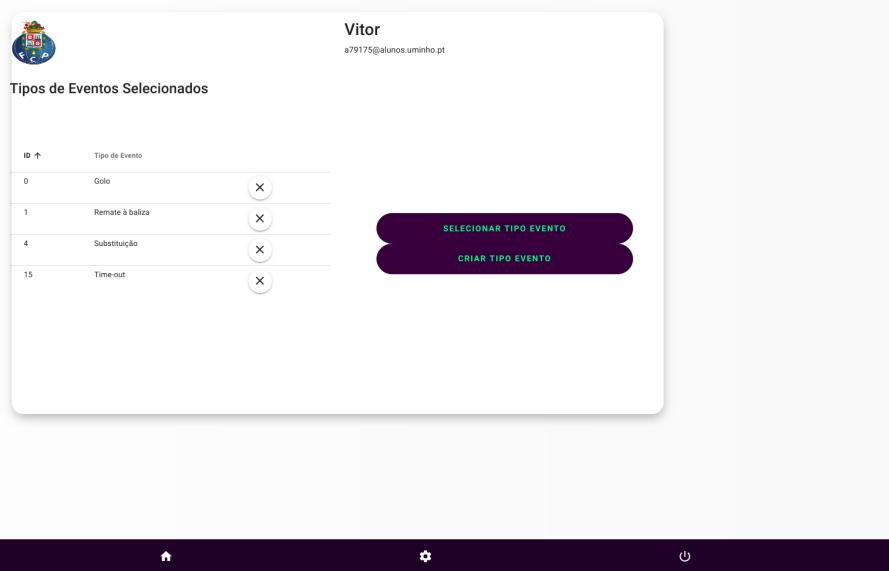


Figura 31: Página de definições do Técnico.

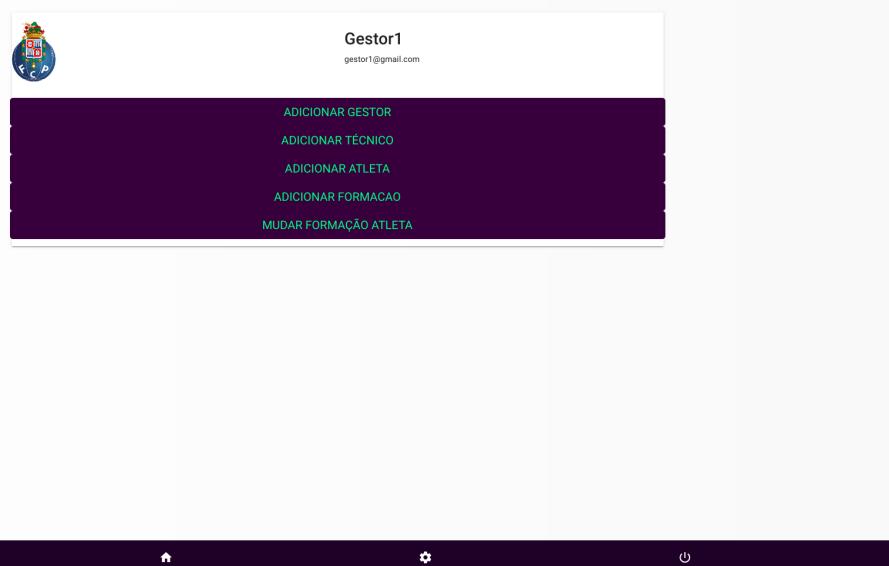


Figura 32: Página de definições do Gestor.

9.2 Funcionalidades

Para uma melhor percepção do trabalho desenvolvido, optou-se por criar uma matriz onde se apresentam as funcionalidades implementadas, as funcionalidades que se encontram incompletas e as funcionalidades que o grupo não conseguiu implementar.

Para além disso, considerou-se interessante avaliar as diferentes funcionalidades por um nível de relevância de 0 a 10, sendo 10 uma funcionalidade essencial para a aplicação web e 0 uma funcionalidade prescindível.

Ao contrário do planeado inicialmente, o grupo começou por implementar as funcionalidades imprescindíveis para cumprir o objetivo da aplicação, sendo estas o registo de eventos e consulta de estatísticas. Estando estas implementadas, seguiu-se com o desenvolvimento das restantes funcionalidades do Técnico, sendo este o tipo de utilizador mais recorrente na aplicação e por último as funcionalidades do Gestor.

A funcionalidade do Administrador ”Adicionar clube” e a funcionalidade do Técnico ”Personalizar matriz do campo” não foram implementadas, no entanto o *back end* está preparado para as processar, faltando apenas o seu desenvolvimento no *front end*. Para além disso, considerou-se a funcionalidade ”Adicionar gestor” incompleta, pois esta está disponível nas definições do Gestor, mas não está disponíveis para o Administrador, sendo que não foi desenvolvido a interface do Administrador.

	Relevância	Por Fazer	Incompleto	Completo
Adicionar clube	7			
Adicionar gestor	6			
Adicionar técnico	7			
Adicionar formação	7			
Adicionar atleta	7			
Associar atleta a formação	7			
Adicionar jogo	7			
Consultar estatísticas	10			
Alterar camisola de atleta	5			
Selecionar convocados e 5 iniciais	4			
Personalizar matriz do campo	4			
Manipular relógio	8			
Criar novo tipo de evento	6			
Selecionar tipo de evento	3			
Registar evento	10			
Sinalizar ou remover evento	8			
Editar evento	8			

Tabela 2: Implementação das Funcionalidades

10 Conclusão

Do trabalho efetuado resultou uma aplicação *web* capaz de registar e gerir os acontecimentos de jogos de hóquei de uma determinada equipa. O levantamento inicial dos requisitos funcionais e de qualidade, em conjunto com uma análise dos métodos tradicionais de registo de eventos permitiu a formação de uma noção geral dos dados que seriam tratados pelo sistema, bem como um conjunto de vistas imprescindíveis ao seu funcionamento, traduzidas posteriormente *mockups*.

Com a modelação completa, procedeu-se à implementação do sistema, começando pela sua lógica de negócio e pela API que permitiria o acesso por parte do *front end*. A utilização de *Python* através da sua *framework Django* permitiu um desenvolvimento intuitivo do *back end*, completa com modelos de dados e métodos de os gerir, bem como processos através dos quais o cliente os possa aceder.

O cliente do programa provou ser um maior desafio, nomeadamente na construção de representações visuais de estatísticas e na recolha dos dados necessários ao registo de eventos. A necessidade de aprendizagem de utilização das ferramentas providenciadas pelo **Vue.js**, em conjunto com a compatibilidade requerida tornou a criação desta página particularmente desafiante. Ainda assim, foi possível criar uma interface eficiente que permite a funcionalidade base do sistema, bem como a produção de estatísticas úteis, como é o caso do gráfico de eventos por campo, ou o rápido e eficiente registo de eventos durante a partida.

Apesar da implementação dos requisitos mais imprescindíveis, houve ainda a incapacidade de desenvolver por completo um conjunto das funcionalidades adicionais modeladas. Principalmente, a elaboração de um tipo de utilizador Administrador não foi completa, tendo sido apenas criada a lógica de *back end* para a mesma. Foi também posta de lado a capacidade de personalizar a matriz do campo para registo de eventos.

De um modo geral, podemos então concluir a realização deste projeto positivamente, com a perspetiva de futuramente implementar as restantes funcionalidades desejadas e possivelmente expandir o *software* desenvolvido para outras áreas ou desportos, dada a sua versatilidade e escalabilidade.