PM

Plataforma de Visualização de Eventos Desportivos
Nova arquitetura e funcionalidades

PROJETO DE MESTRADO

João Pedro Vieira Belo
MESTRADO EM ENGENHARIA INFORMÁTICA

UNIVERSIDADE da MADEIRA
A Nossa Universidade
www.uma.pt

novembro | 2020

Plataforma de Visualização de Eventos Desportivos
Nova arquitetura e funcionalidades

PROJETO DE MESTRADO

João Pedro Vieira Belo
MESTRADO EM ENGENHARIA INFORMÁTICA

ORIENTAÇÃO
Eduardo Miguel Dias Marques

Plataforma de Visualização de Eventos Desportivos

Nova arquitetura e funcionalidades

Projeto de Mestrado em Engenharia Informática sob orientação do Professor Doutor Eduardo Miguel Dias Marques, apresentada a prova pública, na Universidade da Madeira, a 22 de janeiro de 2021, perante seguinte júri:

Professor Doutor Eduardo Miguel Dias Marques, Universidade da Madeira (Vogal);

Professora Doutora Frederica Margarida Camacho Gonçalves, Universidade da Madeira (Arguente);

Professora Doutora Karolina Baras, Universidade da Madeira (Vogal e Presidente de Júri).

Agradecimentos

Gostaria de agradecer a todos os que me ajudaram e apoiaram ao longo deste percurso, que agora culmina com este projeto de Mestrado.

Ao meu orientador, Professor Doutor Eduardo Miguel Dias Marques, pela sua disponibilidade, orientação e revisão de todo o trabalho.

À minha companheira Luciane por todo o seu apoio e paciência ao longo deste processo, estando sempre disponível para me apoiar nos momentos menos bons e que sempre me motivou a finalizar esta jornada.

À Sara e ao Carlos, meus colegas no Observatório Oceânico da Madeira, por todo o apoio prestado durante a escrita do documento. À Doutora Sónia Costa, investigadora do Observatório Oceânico da Madeira e do MARE, por me ter permitido conciliar o trabalho do dia a dia com este projeto.

Aos restantes colegas do Observatório Oceânico da Madeira e do *Interactive Technologies Institute*.

À minha família, pelo apoio dado ao longo da realização deste projeto.

Resumo

Atualmente, é essencial acompanhar e visualizar os resultados de um evento desportivo em tempo real. Longe vão os tempos em que era aceitável publicar uma lista em papel com os resultados de um evento. A realização e o acompanhamento de um evento desportivo envolvem uma grande quantidade de dados, bem como a interligação de diferentes sistemas. Os eventos de *trail running* têm características únicas, pois decorrem em zonas, muitas vezes, remotas, em percursos com elevado grau de complexidade e obstáculos naturais que podem pôr em risco a vida do atleta.

Apesar da existência de uma aplicação que permite acompanhar os eventos, a mesma apresenta alguns problemas como a falta de algumas funcionalidades na interação com o público, dificuldade na escalabilidade e desempenho. Existem ainda algumas limitações a nível arquitetural que dificultam a integração com outras aplicações.

Para atingir os objetivos propostos, nomeadamente, solucionar problemas de desempenho, interligar as diversas aplicações e aumentar as funcionalidades, optou-se por desenvolver uma nova arquitetura multicomponente, separando os componentes essenciais e facilitando a escalabilidade do sistema. Decidiu-se, ainda, desenvolver uma API de modo a facilitar a interligação em segurança das diferentes aplicações. Por fim, desenvolveu-se uma aplicação pública que permite ao público acompanhar a prova, através da apresentação de resultados e estatísticas em tempo real, e que presta, simultaneamente, apoio à organização, nomeadamente, no apuramento dos vencedores, deteção de irregularidades e verificação da chegada de todos os participantes em segurança.

Após o desenvolvimento das aplicações foram efetuados testes de desempenho utilizando uma ferramenta específica que permite analisar o comportamento das mesmas em diferentes cenários de utilização. Estes testes permitiram afinar a estrutura para os diferentes eventos. Por fim, foi solicitado aos utilizadores o preenchimento de um breve questionário sobre o grau de satisfação em relação à aplicação.

Analisando os resultados dos testes efetuados, assim como o *feedback* dos utilizadores (proveniente dos inquéritos), é possível concluir que a aplicação superou todas as expectativas e cumpre todos os objetivos inicialmente propostos.

Palavras chave: aplicação, API, arquitetura, desempenho, *framework*, *trail*

Abstract

Currently it is essential to monitor and visualize the results of a sporting event in real-time. Gone are the days when it was acceptable to publish a paper list of the results of an event. The realization and monitoring of a sporting event involves a large amount of data, as well as the interconnection of different systems. Trail running events have unique characteristics, as they take place in often remote areas, on routes with a high degree of complexity and natural obstacles that can endanger the athlete's life.

Despite the existence of an application that allows the following of the events, it presents some problems such as lack of some features in the interaction with the public, difficulty in scalability and performance. Some architectural limitations make integration with other applications difficult.

To achieve the proposed objectives, namely to solve performance problems, interconnect the various applications and increase the functionalities, it was decided to develop a new multi-component architecture, separating the essential components and facilitating the scalability of the system. It was also decided to develop an API to facilitate a secure interconnection of the different applications. Finally, a public application was developed that allows the public to follow the competition, through the presentation of results and statistics in real-time, and that simultaneously provides support to the organization, namely in determining the winners, detecting irregularities and verifying the arrival of all participants safely.

After the development of the applications, performance tests were carried out using a specific tool that allows to analyze their behavior in different usage scenarios. These tests made it possible to fine-tune the structure for the different events. Finally, users were asked to complete a short questionnaire concerning the degree of satisfaction with the application.

Analyzing the results of the tests carried out, as well as the users' feedback (from the surveys), it is possible to conclude that the application exceeded all expectations and fulfills all the objectives initially proposed.

Keywords: application, API, architecture, performance, framework, trail

Índice

Lista de abreviaturas e acrónimos

API	*Application Programming Interface*
CDN	*Content Delivery Network*
CPU	*Central Processing Unit*
CSS	*Cascading Style Sheets*
DDoS	*Distributed Denial-of-Service Attack*
DNS	*Domain Same System*
DOM	*Document Object Model*
DRY	*Don't Repeat Yourself*
FTP	*File Transfer Protocol*
GUI	*Graphical User Interface*
HDD	*Hard Disk Drive*
HTML	*HyperText Markup Language*
HTTP	*Hypertext Transfer Protocol*
HTTPS	*Hypertext Transfer Protocol Secure*
IMAP	*Internet Message Access Protocol*
IP	*Internet Protocol*
JSON	*JavaScript Object Notation*
LDAP	*Lightweight Directory Access Protocol*
MIUT	*Madeira Island Ultra Trail*
MVC	*Model View Controller*
MVT	*Model View Template*
NPM	*Node Package Manager*
ORM	*Object Mapping Layer*
PHP	*Hypertext Preprocessor*
POP	*Post Office Protocol*
PWA	*Progressive Web App*

RAM	*Random Access Memory*
RFID	*Radio Frequency Identification*
RGPD	Regulamento Geral sobre a Proteção de Dados
SMTP	*Simple Mail Transfer Protocol*
SMS	*Short Message Service*
SQL	*Structured Query Language*
SSD	*Solid-State Drive*
UI	*User Interface*
UML	*Unified Modeling Language*
URL	*Uniform Resource Locator*
WWW	*World Wide Web*

Índice de Figuras

Índice de Tabelas

Capítulo 1

Introdução

Este capítulo apresenta uma introdução ao projeto *timing system*, bem como ao desporto *trail running*, no geral, e os diversos problemas que levaram à criação deste projeto (1.1). Aborda ainda a motivação (1.2) e os principais objetivos (1.3). Por fim, a estrutura do documento (1.4) é apresentada.

1.1 *Contexto*

O trail running, vulgarmente conhecido por *trail*, é um desporto de corrida fora da pista, em trilhos de montanha ou em caminhos secundários. Durante o percurso, os atletas podem ter de ultrapassar subidas ou descidas íngremes bem como rios, riachos e outros obstáculos naturais. Este desporto diferencia-se das corridas ou caminhadas pois acontece em ambientes naturais e/ou fora de pistas e com provas onde a distância percorrida é superior a 100 Km e a duração superior a 24 horas [1].

Tendo em conta o facto de os eventos serem em ambientes remotos, como montanhas e florestas, uma grande preocupação é a segurança dos atletas. Um atleta pode facilmente sofrer uma lesão ou um esgotamento físico e necessitar de assistência médica ou então pode, simplesmente, perder-se e necessitar de assistência para encontrar o caminho. Sendo assim, o controlo dos atletas, por parte da organização, é fundamental.

No início da prova, o atleta recebe um kit de participação, que contém, além das ofertas de patrocinadores, um peitoral/dorsal único e que é composto pelo seu nome, número de inscrição e um identificador *Radio Frequency Identifier* (RFID), entre outros dados sobre a competição. Uma informação presente nos peitorais são os números de telefone/telemóvel de emergência, a serem utilizados em caso de necessidade [2].

As provas de *trail* iniciam-se, normalmente, de madrugada (por vezes às 00 horas) e são compostas por percursos de montanha e/ou floresta, existindo diversos postos intermédios nesse percurso (também designados *checkpoints* ou pontos de assistência), definidos antes da prova se iniciar. Quando um atleta passa por um *checkpoint*, além de poder-se abastecer e recuperar fisicamente, o seu tempo de passagem é registado através da leitura da tag RFID presente no peitoral [2] sendo, por vezes, feita uma vistoria ao material que o atleta leva, variando com os regulamentos de cada prova. Os atletas podem ainda receber comida, bebidas, aconselhamento ou apoio médico nestes *checkpoints*.

Durante a prova é imprescindível saber quais os atletas que já passaram em cada *checkpoint* e ainda, antecipar se algum atleta está com um atraso demasiado elevado (o contrário também se aplica, ou seja, se um atleta chegou ao *checkpoint* demasiado rápido). Para a organização isto é importante, pois permite saber se algum participante está em perigo, caso esteja demasiado atrasado, ou se algum atleta tentou burlar as regras da prova, caso esteja demasiado adiantado. No caso do público, é importante obter informações sobre da prova, nomeadamente saber como a prova está a decorrer para os seus atletas favoritos ou familiares, conhecendo os tempos de passagem pelos *checkpoints* e a classificação ao longo do tempo, bem como se existe algum problema com os mesmos. É igualmente relevante ser notificado quando um dos seus atletas favoritos efetua a passagem por um *checkpoint.*

O projeto *timing system* [3] é composto por diversas aplicações, desde a aplicação pública, que é utilizada pelo público no geral (para o acompanhamento da prova e consulta de resultados), passando pela aplicação responsável pelo lançamento dos tempos dos atletas (utilizada pelos voluntários), entre outras

aplicações voltadas para a organização que permitem controlar, corrigir e monitorizar as provas de *trail running*. Este projeto foca-se principalmente na aplicação pública descrita sucintamente na secção 4.1, esta aplicação é utilizada para o acompanhamento e consulta de resultados e na arquitetura de todo o sistema. Esta permite consultar os eventos que estão a decorrer e os eventos passados. Possibilita consultar o tempo dos atletas nos diversos pontos, apuramento de vencedores e ainda a consulta de algumas estatísticas. Esta aplicação, além de ser utilizada pelo público no geral, é manuseada pela organização do evento para gerar as listas de vencedores e monitorizar os atletas.[3] No entanto, esta aplicação sofre de diversos problemas, sendo um deles o desempenho, quem tem vindo a agravar-se ao longo dos anos. Um exemplo de utilização desta aplicação ocorre no evento *Madeira Island Ultra Trail* (MIUT) [4], que é uma prova de referência a nível mundial nesta modalidade, acontecendo todos os anos na Região Autónoma da Madeira. O MIUT contou, na sua última edição, com, aproximadamente, 2600 atletas de diversas nacionalidades, distribuídos por quatro provas. Este evento tem uma enorme procura, tendo inclusive esgotado as inscrições para o ano de 2019 em apenas 17 horas [5]. Este tipo de evento é composto por diversas provas de diferentes distancias, sendo que em uma das provas os atletas percorrem a Ilha da Madeira desde o Porto Moniz até Machico, conforme apresentado na Figura 1 [6]. Nestes casos o sistema tem de ser capaz de controlar os atletas nas diferentes provas com pontos de controlo partilhados mas com limites de tempos diferentes.

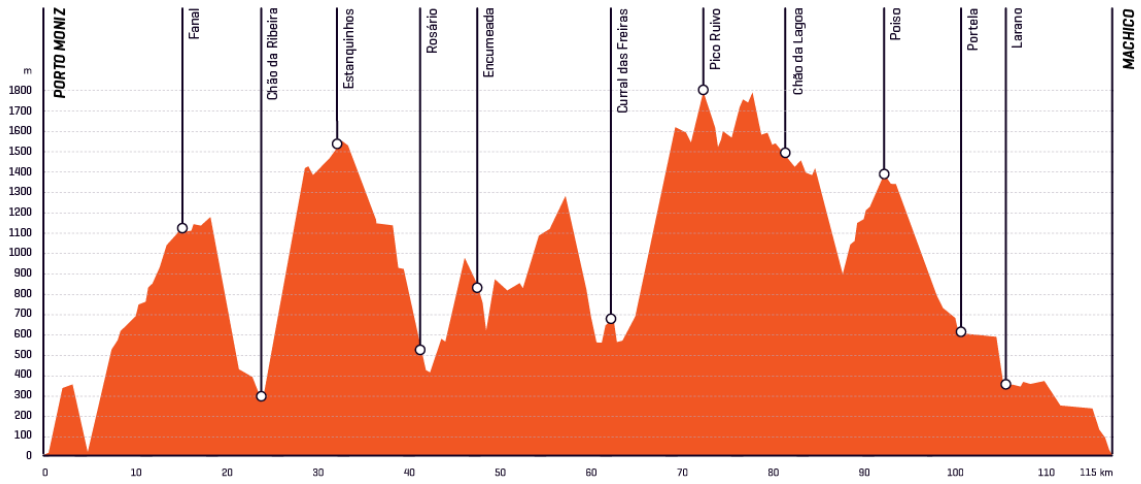


Figura 1 - Gráfico de elevação do percurso do MIUT de 2019. Percurso obtido no website do MIUT [4].

Numa prova com este nível de complexidade organizacional e elevado número de participantes, voluntários e público é necessário garantir inúmeras condições de segurança sendo, portanto, imperativo ter sistemas de informação que consigam suportar as diferentes necessidades em tempo real. Porém, tal não acontece, pois, como dito anteriormente, a aplicação sofre de problemas de desempenho, problemas estes que foram dados a conhecer por parte da organização em inúmeras reuniões. Em provas de grande dimensão, como esta, a aplicação simplesmente não consegue dar resposta ao número de visitantes, ficando com um fraco desempenho. Esta falta de desempenho causa inúmeros problemas, pois não é possível saber se os atletas já passaram pelos *checkpoints*, deixando os familiares

preocupados e, em casos extremos, atrasar o socorro a algum atleta necessitado. Outro problema que advém da inoperacionalidade da aplicação atual é a impossibilidade da organização saber os tempos dos atletas, o que impossibilita o apuramento de vencedores. De modo a evitar que a aplicação fique totalmente *offline* são utilizadas soluções de replicação. Ou seja, é efetuado uma cópia da aplicação, sendo essa cópia hospedada na nuvem.

Todavia, os problemas não se limitam apenas à aplicação utilizada pelo público. Existem problemas comuns a todas as aplicações, incluindo as que estão voltadas para a organização ou voluntários. Sendo estas aplicações extremamente importantes para o funcionamento da aplicação pública, um dos graves problemas é que cada uma destas aplicações utiliza sistemas diferentes de acesso à base de dados [7] sendo a mesma partilhada pelas diversas aplicações. Torna-se, assim, praticamente impossível monitorizar todos os diferentes sistemas de acesso de modo a garantir que os níveis de segurança exigidos estão a ser cumpridos.

1.2 *Motivação*

A motivação para este projeto começou pela vontade de solucionar as limitações e problemas existes na aplicação atual. Foram identificados problemas de desempenho e escalabilidade que obrigam a grandes alterações. Tendo em conta que os utilizadores já estavam habituados à aplicação atual, estas alterações podem originar alguma resistência à mudança [8], além das particularidades associadas a nível de segurança e resultados que uma prova de *trail* possui.

Outro grande desafio consiste em desenvolver uma arquitetura que suporte a integração das aplicações existentes, levando a um aumento da complexidade, quer ao nível da segurança (devido ao número de acessos a partir de pontos diferentes), quer das dificuldades associadas à criação de um sistema que funciona nos bastidores (que necessita de aceder ou manipular dados da base de dados).

1.3 *Objetivos*

Na sequência dos problemas encontrados e descritos na Introdução e detalhados no Capítulo 4, foram definidos objetivos para este projeto, e que estão detalhados de seguida:

- Solucionar os problemas de desempenho e as limitações de escalabilidade da atual aplicação utilizada pelas organizações dos eventos de *trails*;
- Alterar a arquitetura do sistema de forma a suportar a integração de outras aplicações de forma normalizada e segura;
- Aumentar as funcionalidades na visualização e interação com o público.

1.4 *Estrutura do documento*

O presente documento está dividido em sete capítulos, sendo o primeiro o capítulo introdutório, onde é descrito o contexto, os objetivos e a motivação.

O segundo capítulo consiste no estado da arte sobre a otimização e teste de aplicações *web*.

No terceiro capítulo é apresentado um estudo sobre as diversas *frameworks* para *front-end* e *back-end*, que foram necessários analisar para o desenvolvimento do projeto.

O quarto capítulo analisa, em maior detalhe, o problema, explicando os requisitos que foram obtidos, a nova arquitetura e os protótipos elaborados.

No quinto capítulo todo o processo de implementação e desenvolvimento são abordados, explicando-se as diversas escolhas feitas e os componentes desenvolvidos.

No sexto capítulo são apresentados os resultados obtidos nos testes efetuados à nova aplicação. É, ainda, apresentado o resultado do inquérito feito aos utilizadores da aplicação.

No sétimo capítulo são delineadas as conclusões obtidas com o desenvolvimento do projeto, bem como o possível trabalho futuro e melhorias. O oitavo capítulo agrupa as referências consultadas no decorrer do documento.

Capítulo 2

Otimização e Teste de Aplicações *Web*

Este capítulo apresenta um estudo sobre as técnicas de otimização das aplicações *web*, iniciando-se com uma breve introdução às mesmas (2.1). Expõe as diversas técnicas que podem ser utilizadas para a otimização (2.2), os algoritmos para gestão de carga (2.3) e encerra com diversos tipos de testes (2.4) utilizados para medir o desempenho das aplicações *web* e respetivas ferramentas e conclusões (2.5).

2.1 *Introdução*

A *World Wide Web* (WWW) fornece um acesso a grandes quantidades de dados geograficamente distribuídos. Muitas empresas utilizam aplicações baseadas na *web* levando os utilizadores, cada vez mais, a prestarem atenção à confiabilidade, desempenho e fiabilidade das mesmas. Claramente, a qualidade do serviço prestado por uma aplicação depende de uma grande variedade de fatores, alguns de um nível bastante complexo de quantificar. No entanto, afigura-se que os utilizadores avaliam o tempo de acesso aos dados numa aplicação *web* como um componente essencial para a qualidade, levando os mesmos a evitar aplicações *web*, em alguns casos, devido a diferenças na ordem dos milissegundos [9].

Os servidores web, que dão suporte às aplicações *web*, são componentes cruciais quando queremos garantir que uma aplicação *web* é executada de forma estável, influenciando diretamente o tempo de acesso aos dados [10]. Este tempo é composto por diversas variáveis, incluindo o tempo que a solicitação do cliente leva para chegar ao servidor, o tempo que o servidor demora para processar os dados, o tempo para a resposta do servidor chegar ao cliente e o tempo para o cliente processar a resposta. Todos estes tempos são afetados diretamente pelo *hardware*, pelas configurações, aplicações no servidor e no cliente e ainda a carga do servidor ou da rede [9].

Para evitar um atraso no acesso aos dados é importante ter recursos suficientes do lado do cliente, mas principalmente do lado do servidor, sendo esses recursos os processadores (CPU), o espaço de armazenamento e a memória *Random-Access Memory* (RAM) [11]. Assim, o *hardware* do servidor acaba por influenciar, de forma significativa, o tempo de acesso aos dados sendo por isso, necessário, em alguns casos, efetuar uma otimização do *hardware*. A otimização é normalmente atingida pelo investimento em equipamento de qualidade e desempenho superior e tem como objetivo, permitir que a aplicação seja executada em *hardware* mais avançado, melhorando o desempenho do sistema e os tempos de resposta. No entanto, este processo requer um grande investimento e, raramente, resolve os problemas de forma permanente, pois o *hardware* fica rapidamente obsoleto [10][12].

É importante ainda ter recursos de rede suficientes para suportar a carga prevista e controlar a quantidade de dados que são enviados para a rede. A qualidade do serviço de uma aplicação *web* é, assim, um equilíbrio delicado entre a rede e os recursos do servidor, sendo importante antecipar a afluência de pedidos e potenciais clientes [11].

As bases de dados são outro fator determinante no tempo de acesso à informação pois grande parte das aplicações *web* precisam de acesso a uma grande quantidade de informação armazenada numa base dados. Nestes casos, uma grande parte do tempo de carregamento é gasto no acesso aos dados. Deste modo, a otimização da base de dados é um dos aspetos mais importantes no desempenho de uma aplicação *web* [12].

A otimização da base de dados pode ser obtida de diversas formas, não sendo necessariamente um aspeto melhor que outro, mas idealmente um conjunto de aspetos deverá ser utilizado de forma a atingir

o melhor nível de desempenho. Uma técnica de otimização é criar uma matriz (contendo dados do índice e *logs*) armazenada em diferentes discos, estabelecendo um *cluster* do servidor da base de dados. Outra técnica consiste em otimizar as instruções *Structured Query Language* (SQL), isto é, melhorar a eficiência da consulta aos dados escrevendo instruções SQL mais eficientes. Estas instruções são utilizadas para consultar, editar, inserir e remover dados da base de dados, por isso é importante que estas sejam o mais eficientes possível. A utilização de *caches*, nas consultas mais frequentes, também contribui para reduzir os tempos de acesso aos dados [12].

Apesar do servidor, a rede e a base de dados serem fundamentais, ao longo deste capítulo serão explicadas ao detalhe diferentes abordagens que ajudam a otimizar o desempenho dos diferentes componentes de *software* que integram uma aplicação *web*, bem como diversos testes que podem ser efetuados de modo a garantir o correto e desejado funcionamento da aplicação *web*.

2.2 *Técnicas de otimização do front-end*

De acordo com a regra "*Performance Golden Rule*" de Steve Souders [13], em média, 80-90% do tempo de carregamento da página *web* é gasto no *front-end,* isto é, a parte da aplicação que fica visível para o cliente. Este tempo é gasto, principalmente, a descarregar recursos e a processar os mesmos. O tempo é aumentado pois a maioria dos navegadores pausa todos os outros processos durante o descarregamento e processamento dos *scripts*. O autor recomenda ainda aos programadores otimizarem partes do *front-end* dos sites e aplicações, antes de se focarem na otimização do *back-end* [13].

O entendimento do processo de carregamento das aplicações *web* é imperativo para implementar os métodos de otimização no *front-end,* pelo que, de seguida, se explica o processo de carregamento. O carregamento da aplicação *web* começa quando o utilizador digita o *Uniform Resource Locator* (URL) da aplicação na barra de endereços de um navegador. Em seguida, o navegador analisa o URL e determina o protocolo correto (normalmente *Hyper Text Transfer Protocol* (HTTP) ou *Hyper Text Transfer Protocol secure* (HTTPS)*,* o domínio e a porta. Assim que o navegador determina o domínio, faz uma pesquisa de *Domain Name System* (DNS) para o encontrar o endereço *Internet Protocol* (IP) do servidor que hospeda a aplicação. Logo de seguida, é estabelecida uma ligação de rede entre o servidor e o navegador do cliente. O navegador recebe a resposta, verifica se é um documento HTML e processa o mesmo.

Assim que o documento HTML é processado, é criado o *Document Object Model* (DOM) que consiste numa estrutura hierárquica a representar cada elemento HTML. Só após o DOM estar completamente processado é que o navegador começa a carregar os recursos adicionais, tais como os ficheiros *Cascading Style Sheets* (CSS), imagens e *scripts.*

Uma vez terminado o carregamento de cada ficheiro CSS, o navegador analisa e aplica o CSS a cada nó do DOM. Os CSS não modificam a estrutura do DOM permitindo assim ao navegador continuar a carregar

os restantes recursos em simultâneo. Assim que a estrutura do DOM é gerada a página começa a carregar, concluindo-se assim o processo de carregamento [11][12].

O principal objetivo da otimização de *front-end* é diminuir o tempo de carregamento da aplicação *web*, reduzindo a quantidade de recursos do cliente utilizados pela aplicação. Outro grande objetivo é permitir que o navegador carregue a página e os recursos de forma progressiva, gerando parcialmente a página o mais rápido possível, mas deixando o utilizador com a sensação que a página carregou rapidamente. Para atingir este objetivo existem diversas técnicas como a minimização do CSS/*JavaScript, cache-control-headers,* compressão Gzip, rede de entrega de conteúdo, posição do CSS e a remoção de *scripts duplicados,* que são explicadas, em detalhe, de seguida.

A minimização do CSS/*JavaScript* é o processo de remover todos os caracteres desnecessários do código-fonte das linguagens de programação, sem alterar a sua funcionalidade. Estes caracteres desnecessários geralmente incluem espaços em branco, caracteres de nova linha ou comentários. Esta minimização reduz assim o tamanho do código-fonte, tornando a sua transmissão numa rede, por exemplo, a Internet, mais eficiente [14]. No caso do *JavaScript*, o processo inclui uma abstração do código, que substitui os nomes das variáveis e dos métodos por outros mais curtos. Este processo também é conhecido como compressão, e pode combinar *scripts* ou CSS diferentes num único recurso. Desta forma consegue-se reduzir o tamanho dos ficheiros, diminuindo o número de recursos (levando a um número menor de ligações abertas por página carregada) [15], aumentando o desempenho.

Os *cache-control-headers* são utilizados para indicar aos navegadores se um recurso específico pode ,ou não, ser armazenado em *cache* do lado do cliente. O *cache-control-headers* foi introduzido no HTTP/1.1. De forma a ultrapassar algumas limitações, os *headers* expiram, sendo possível, e recomendado, definir a data em que devem expirar pois, o navegador não fará mais solicitações até que o recurso seja considerado expirado. Desta forma, uma vez que a página é carregada, os recursos são armazenados em *cache* e não devem ser solicitados novamente se a página for recarregada ou durante a navegação do site. Deve ter-se em atenção a periodicidade em que dados são utilizados e atualizados [11].

Outro método é a compressão Gzip. Este método é controverso porque, por um lado, reduz a largura de banda enquanto que, por outro, o processo de compressão adiciona uma carga adicional ao servidor. A combinação deste método com técnicas adequadas de armazenamento em *cache* levam a um número menor de solicitações. Este método acaba por ter um efeito positivo nos recursos baseados em texto, como, por exemplo, CSS e *scripts*, mas tem pouco efeito em imagens e outros documentos [16].

A rede de entrega de conteúdo, ou, em inglês, *Content Delivery Network* (CDN)*,* é outra técnica de otimização, pois as CDN são uma grande rede de servidores distribuídos geograficamente que permite disponibilizar conteúdo, aos utilizadores finais, com altas velocidades e grande disponibilidade. A distribuição é feita para chegar mais rapidamente aos pedidos dos utilizadores finais. Estas redes são utilizadas amplamente na *web*, para conteúdo estático e recursos para transferências, entre outros. Nas redes CDN, os servidores são divididos em dois grupos: servidores de proximidade - em inglês, *edge* - e

servidores de origem - em inglês, *origin*. O utilizador final comunica com os servidores proximidade mas, caso o servidor não contenha o recurso pretendido, o mesmo efetua um pedido ao servidor de origem. Assim que o servidor de proximidade recebe o conteúdo, envia o mesmo para o utilizador final e armazena uma cópia do recurso para solicitações futuras. Esta técnica permite que os recursos cheguem mais rapidamente ao utilizador final devido à proximidade do servidor. Utilizando as redes CDN, o servidor, onde está alojada a aplicação *web*, deve apenas lidar com as páginas dinâmicas, sendo os recursos estáticos controlados pela rede CDN [17].

Outra técnica é a utilização de *JavaScript* e CSS externos. A vantagem de utilizar recursos externos é que permitem ao utilizador final beneficiar dos recursos de *cache* do navegador pois o CSS e os *scripts* não serão detetados como parte do código HTML para cada página (não sendo descarregados novamente em cada solicitação) [11], permitindo assim reduzir o tempo de carregamento [18].

A posição do CSS na parte superior da página é importante pois, desta forma, o CSS está entre os primeiros recursos que o navegador começa por carregar, após o processamento do HTML. Sendo o CSS aplicado à arvore DOM antes do carregamento e execução de outros recursos, o navegador pode começar a desenhar a página, enquanto aguarda que outros recursos sejam carregados. Por este motivo é que devemos colocar as *tags* de *style* (CSS) o mais próximo possível do início da secção *head* do ficheiro HTML [11][19].

A remoção de *scripts* duplicados é outra parte essencial da otimização do *front-end*. A grande desvantagem dos *scripts* duplicados é o aumento do tempo de carregamento como resultado da análise e execução repetidas do *script*. [11].

Em resumo, todas as técnicas apresentadas anteriormente são uma forma simples e eficaz de reduzir os tempos de carregamento, sendo as *caches* uma das melhores opções. O ideal é combinar diversas técnicas de forma a melhorar a experiência dos utilizadores e reduzir a carga no servidor.

2.3 *Arquitetura e gestão de carga*

A arquitetura das aplicações *web* sofreu um processo evolutivo ao longos dos anos, deixando de ser um sistema único para passar a ser composta por dois, três ou mais sistemas interligados. Esta evolução, além de facilitar a organização da lógica de negócio do sistema, ajuda também a melhorar o desempenho do sistema. Uma arquitetura cliente-servidor simples é composta apenas por vários clientes e um servidor como os representados na Figura 2, onde os clientes efetuam pedidos ao servidor por meio de uma rede local ou através da Internet. Neste tipo de arquitetura, todos os dados são fornecidos pelo servidor *web*.

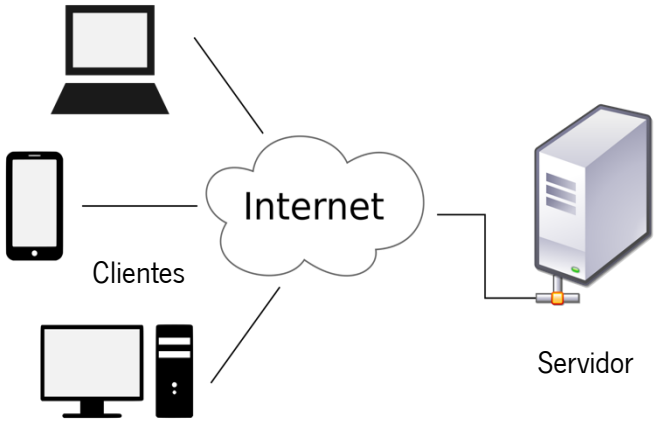


Figura 2 – Arquitetura cliente-servidor. Imagem adaptada de David Vignoni.

Numa arquitetura cliente-servidor, com o aumento do tráfego aumentam os pedidos levando a uma diminuição do desempenho, dado que no servidor *web* coexistem várias aplicações, como, por exemplo, o gestor de gestão de bases de dados que com o passar do tempo acaba por utilizar CPU, memória e outros recursos do sistema de forma intensiva, deixando as restantes aplicações a aguardar pela disponibilidade de recursos. Assim sendo, a melhor prática no sentido de otimizar a arquitetura é separar as diversas aplicações em servidores diferentes, com por exemplo um sistema para o servidor de bases de dados e outro para o servidor *web*, melhorando assim o desempenho, a capacidade de carga do sistema e facilitando uma futura expansão [20][12].

Quanto ao processo de gestão de carga, este consiste em distribuir um conjunto de tarefas pelos recursos disponíveis, com o objetivo de tornar o processamento mais eficiente/rápido. As técnicas de gestão de carga podem otimizar o tempo de resposta de cada tarefa, evitando uma sobrecarga desigual nas unidades de computação [21]. Na Figura 3 está representado a arquitetura de um sistema de gestão de carga.

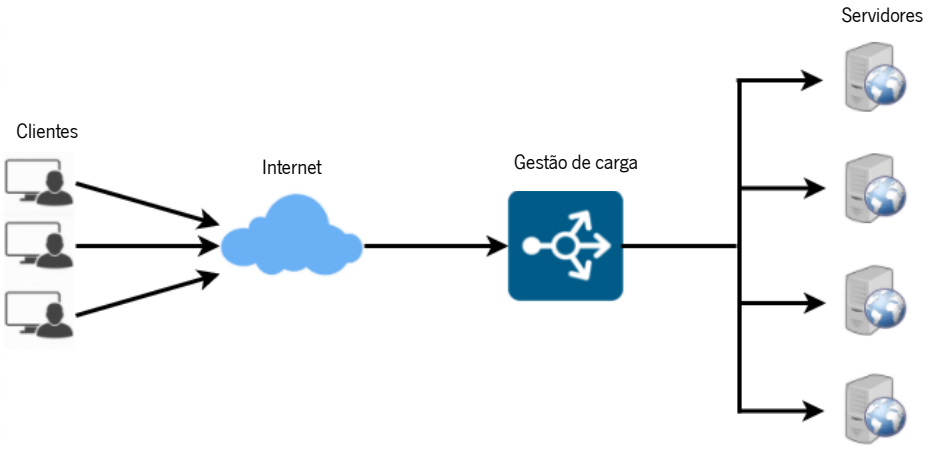


Figura 3 - Arquitetura de um sistema de gestão de carga. Imagem adaptada de educative.io

Neste tipo de arquitetura é utilizando um *load balancer,* ou, em português, gestor de carga, responsável por distribuir os pedidos que recebe pelos diversos servidores.

A distribuição dos pedidos é feita com base em algoritmos estáticos ou dinâmicos.

Os algoritmos estáticos não têm em consideração o estado atual das unidades de computação, mas necessitam de informação sobre o estado anterior do sistema, nomeadamente, memória e CPU, e são geralmente centrados em torno de um *router* que distribui as cargas. Estes algoritmos são fáceis de configurar e extremamente eficientes no processamento de solicitações HTTP comparativamente aos algoritmos dinâmicos [22]. No entanto, quando existe um elevado número de pedidos, pode existir uma sobrecarga de algumas unidades de computação e, consequentemente, um menor desempenho na gestão dos pedidos, levando a um aumento do tempo de resposta. Estes algoritmos funcionam bem em sistemas com pouca variação de pedidos [21][23].

Apresenta-se, de seguida, alguns exemplos de algoritmos estáticos de gestão de carga, também apresentados na Tabela I:

- ***Round robin:*** neste algoritmo, o tempo necessário para o processamento dos dados é dividido em "fatias". É dado, a cada unidade de computação, um intervalo de tempo para completar a tarefa; Se a tarefa não for terminada dentro desse intervalo, a mesma tem de esperar pela próxima vaga para finalizar ou continuar o processamento [24].
- **Min-Min:** neste algoritmo, primeiro é calculado o tempo mínimo para completar uma determinada tarefa em todas as unidades de computação. A tarefa é atribuída à unidade de computação que demora o menor tempo para a completar. Este processo é repetido até todas as tarefas estarem atribuídas [25].
- **Max-Min:** este algoritmo funciona na estratégia oposta ao algoritmo Min-Min, sendo a tarefa atribuída à unidade de computação que tem o maior tempo de computação. Neste algoritmo é calculado o tempo máximo para completar uma determinada tarefa em todas as unidas de computação, sendo a tarefa atribuída à unidade que demorar mais tempo [25].
- ***Opportunistic Load balancing:*** este algoritmo não calcula o tempo de execução de nenhuma tarefa. Simplesmente atribui as tarefas de forma aleatória às unidades de computação [26].

Os algoritmos dinâmicos, por seu lado, têm como principal objetivo melhorar o desempenho a um custo razoável. Os algoritmos dinâmicos tratam todos os pedidos de igual forma, independentemente da sua origem, são tolerantes a falhas e têm a capacidade de se expandir em conjunto com o sistema e manter a estabilidade em emergências (por exemplo, quando existe um aumento repentino de pedidos) [22].

Ao contrário dos algoritmos estáticos, os dinâmicos levam em consideração o estado de carga atual de cada unidade de computação no sistema, mas não têm em consideração o estado anterior do sistema.

Nesta abordagem, as tarefas podem ser movidas dinamicamente de um nó sobrecarregado para um nó menos sobrecarregado. Estes algoritmos são muito mais complicados de desenvolver, mas podem produzir excelentes resultados, principalmente quando o tempo de execução de uma tarefa varia comparativamente a outra tarefa que utilize algoritmos estáticos.

A arquitetura dos algoritmos dinâmicos é modular, pois estes não necessitam de uma unidade de computação dedicada à distribuição de tarefas, sendo estas redistribuídas permanentemente de acordo com o estado do sistema e a evolução do mesmo. É necessária muita comunicação entre as diversas unidades de computação para garantir a distribuição, o que pode atrasar a resolução do problema geral [21][22].

Apresenta-se, de seguida, alguns exemplos de algoritmos dinâmicos, também agrupados na Tabela I:

- ***Ant Colony optimization technique:*** este algoritmo é baseado na natureza, nomeadamente, das formigas, que formam uma rede à procura de comida. Este algoritmo, quando recebe um pedido, inicia o seu movimento visitando todas as unidades de computação, uma de cada vez, verificando se existe alguma unidade de computação sobrecarregada, guardando os dados recolhidos. Se encontrar alguma unidade de computação sobrecarregada, o mesmo partilha parte da carga com a unidade de computação anterior [25].
- ***Honey bee foraging:*** é outro algoritmo inspirado na natureza, descentralizado, e desenhado tendo por base o comportamento das abelhas produtoras de mel. Este algoritmo distribuí os pedidos de forma heterogénea por todas as unidades de computação, calculando a carga atual de uma unidade de computação e decidindo se a unidade está sobrecarregada ou não. As tarefas são atribuídas às unidades de computação com a menor carga [27].
- ***Biased random sampling*** **:** neste algoritmo, todas as unidades de computação são tratadas como elementos representados num grafo virtual em que, à medida que as tarefas são atribuídas, o sistema avança para o próximo elemento [26].
- ***Active clustering:*** este algoritmo é uma versão melhorada do algoritmo *Biased random sampling*, agrupando as unidades de computação semelhantes, tratando todas elas como um grupo [21].

Apresenta-se, na Tabela I, as vantagens e desvantagens, de forma resumida, de cada um dos algoritmos estáticos e dinâmicos referidos anteriormente:

Tabela I - Comparação dos algoritmos de gestão de carga. Informação compilada a partir de Deepa e Cheelu [28].

Algoritmo	Tipo	Vantagens	Desvantagens
Round Robin	Estático	Melhor desempenho em utilizações curtas de CPU.	Tarefas mais complexas demoram mais tempo a ser completas.
Max-Min	Estático	Funciona melhor quando os requisitos são conhecidos previamente.	Demora algum tempo para completar as tarefas.
Min-Min	Estático	Tempo mais curto para completar tarefas. Melhor resultados com tarefas pequenas.	Algumas tarefas podem ficar indeterminadamente à espera.
Opportunistic Load Balancing	Estático	Bom desempenho e boa utilização de recursos.	Leva mais tempo a completar tarefas.
Ant Colony Optimization	Dinâmico	Rápido a computar.	Procuras complexas levam mais tempo.
Honeybee Foraging	Dinâmico	Reduz o tempo de resposta e aumenta o *throughput.*	Tarefas com prioridade baixa demoram mais tempo.
Biased Random Sampling	Dinâmico	Bom desempenho e boa utilização de recursos.	Maior tempo de resposta.
Active clustering	Dinâmico	Unidades de computação semelhantes são agrupadas.	Unidades de computação muito diferentes levam a um mau desempenho do algoritmo.

Em resumo, a definição de uma arquitetura bem estruturada simplifica a manutenção e escalabilidade, sendo essencial para o desenvolvimento de uma boa aplicação. Os algoritmos de gestão de carga permitem distribuir os pedidos pelas diversas unidades de computação, permitindo melhores resultados

a nível de desempenho. A escolha do algoritmo é outro ponto importante, pois um algoritmo apropriado pode aumentar o desempenho de todo o sistema.

2.4 *Testes*

Os testes de desempenho de aplicações *web* são um processo de recolha de informação e da sua análise. Os dados recolhidos são utilizados para avaliar como uma determinada utilização irá afetar os recursos do sistema, permitindo assim criar estratégias para estes casos. Os testes permitem também estudar a resposta a determinados pedidos do utilizador em diferentes condições de utilização. O principal método dos testes de desempenho *web* é gerar tráfego simulado similar à carga real a que a aplicação poderá estar sujeita e identificar *bottlenecks* (pontos fracos), permitindo a sua correção, de forma a garantir o bom desempenho da aplicação [29][30]. Existem diversos tipos de testes, que se apresenta de seguida:

- **Testes de stress:** analisam a robustez do *software*. São efetuados aumentando, gradualmente, a carga do sistema, monitorizando as alterações no desempenho. Têm, como objetivo, encontrar os limites, levando o sistema a um ponto de elevada degradação de desempenho ou falha completa [31].
- **Testes de carga:** avaliam o comportamento do *software* perante uma enorme carga de pedidos em simultâneo, permitindo analisar a forma que o mesmo lida com uma grande quantidade de pedidos. Têm, como objetivo, provar que o sistema consegue lidar com uma determinada carga, com quase nenhuma degradação a nível de desempenho. Ao contrário dos testes de stress, os testes de carga não levam o sistema ao ponto de falha [32].
- **Testes de segurança:** ajudam a descobrir vulnerabilidades na aplicação, incluindo testes de penetração que tentam identificar as probabilidades de *hacking* ou *cracking* [33].
- **Teste *Smoke:*** é utilizado para verificar a estabilidade da aplicação, permitindo verificar o correto funcionamento das principais funções da aplicação. Por exemplo, se, num site de vendas, é feita uma grande atualização, um Teste *Smoke* seria verificar se o sistema de adicionar produtos ao carrinho está a funcionar corretamente [34].
- **Teste de unidade:** utilizado para testar cada componente ou módulo de uma aplicação, permitindo assim analisar o comportamento individual de todas as partes [35].
- **Testes de resistência:** são, no fundo, testes de stress, mas com um intervalo mais longo. De forma a ser considerado um teste de resistência, o mesmo deve durar várias horas ou dias, sendo que estes geram, normalmente, erros complexos [29].
- **Testes à interface gráfica do utilizador:** neste teste é verificado o funcionamento da interface gráfica com o utilizador, permitindo analisar a interação do utilizador com a aplicação [36].
- **Testes gorila:** neste tipo de teste, os módulos, ou componente de *software*, são testados repetidamente em cenários variados, com informações válidas e inválidas, verificando assim a

consistência e o bom funcionamento daquele módulo ou componente específico. Estes testes também são conhecidos por Testes Frustrantes [37].

- **Testes de desempenho:** envolvem a avaliação do *software* em aspetos multidimensionais, incluindo a velocidade, carga, tráfego, stress e vulnerabilidades [38].

Os testes anteriormente descritos são, geralmente, efetuados através de programas de testes automatizados. Estes simulam uma variedade de condições de utilização normais e anormais da aplicação. Tipicamente, as medidas de desempenho são: tempo de resposta, utilização dos recursos do sistema, número de utilizadores em simultâneo, transações HTTP (número de sessões e duração das mesmas) e estatísticas de rede, entre outros recursos [39]. Existe uma grande variedade de programas para efetuar este género de testes, destacando-se o *Locust*, *apache JMeter* e *HULK Automation*. Os mesmos são brevemente descritos abaixo [30]:

- **Locust:** é uma ferramenta desenvolvida em *Python*, utilizada para avaliar o comportamento das aplicações *web* com múltiplos e concorrentes utilizadores. É uma ferramenta de alto desempenho para testes de carga, gratuita e de código aberto [40].
- **JMeter:** ferramenta gratuita de código aberto que é utilizada para testar o desempenho e comportamento de aplicações *web*. Foi desenvolvida para realizar auditorias a aplicações *web*. Atualmente, é também utilizada para analisar componentes principais de rede. O JMeter pode testar diversos tipos de protocolos como o *File Transfer Protocol* (FTP), *Lightweight Directory Access Protocol* (LDAP), protocolos de email (*Simple Mail Transfer Protocol* - SMTP), *Post Office* Protocol (POP), *Internet Message Access Protocol* (IMAP)), protocolos *web* (HTTP, HTTPS) e ainda ligações à base de dados [41].
- **HULK Automation:** é uma ferramenta desenvolvida em *Python,* que simula um ataque *Distributed Denial-of-Service* (DDoS) [42] a um servidor, mas pode ser utilizada para o teste de aplicações *web*. Esta ferramenta permite verificar se a aplicação é capaz de lidar com grandes volumes de tráfego. Caso a mesma fique inoperacional, ou com demasiado atraso a carregar a informação, podem ser tomadas medidas preventivas pelos administradores [30].

As ferramentas de teste e os diferentes tipos de testes, acima mencionados, permitem simular diversas condições de utilização (mais específicas ou mais gerais), permitindo identificar pontos fracos do sistema, ou em módulos específicos e, por consequência, corrigir esses problemas, sendo importante testar diversos parâmetros antes de qualquer lançamento.

2.5 *Conclusão*

É importante ter em atenção que o número de utilizadores da Internet está constantemente a aumentar. Como tal, devemos aplicar diversas técnicas de otimização como as apresentadas anteriormente, pois estas permitem que as aplicações *web* obtenham mais utilizadores, sem modificar o *hardware* existente.

Uma das técnicas mais simples e eficazes a melhorar o desempenho de uma aplicação *web*, começando pelo *front-end*, é efetuar *cache*, do lado do cliente, de todos os recursos, apesar de ser raramente possível, pois muitos conteúdos são dinâmicos. A utilização de *cache* em alguns recursos do *site* permite aumentar a velocidade de carregamento ao navegar pelas páginas do *site*. Combinar as *caches* com o posicionamento correto do CSS e *JavaScript* permite que o navegador comece a carregar mais rapidamente a página, melhorando ainda a experiência do utilizador e reduzindo a carga do lado do servidor (devido ao menor número de ligações e menor tráfego gerado). A utilização de *cache* permite, pelos motivos já mencionados, evitar despesas em novo *hardware* e largura de banda.

A arquitetura do sistema também tem um papel importante pois uma arquitetura bem estruturada, escalável e com os diversos sistemas separados permite mais rapidamente a replicação. A gestão de carga tem também um papel importante para distribuir a mesma, de forma dinâmica e uniforme, pelos vários sistemas e unidades de computação, dado que a sobrecarga de uma unidade de computação de um sistema poderá levar a um desempenho não aceitável do mesmo e de outros dependentes. Por exemplo, a sobrecarga de uma unidade de computação do sistema, onde está instalada a *Application Programming Interface* (API), poderá levar a um desempenho inaceitável da mesma, levando a um mau desempenho dos diversos sistemas que dela dependem.

A utilização de um algoritmo de carga é também necessária, de forma a manter a utilização dos recursos de forma eficiente. Foram apresentados anteriormente diversos algoritmos de gestão de carga estáticos e dinâmicos com as respetivas vantagens e desvantagens. A escolha de um algoritmo é complexa, devido às inúmeras especificidades de cada sistema e, para uma escolha correta, devem ser efetuados diversos testes.

Outro aspeto a ter em conta é a complexidade das aplicações *web* que dificulta os testes, pois é importante testar diversos aspetos como a fiabilidade, segurança, escalabilidade e por fim, mas igualmente importante, o desempenho. Ao longo deste capítulo, foram apresentados diversos tipos de testes que ajudam a identificar problemas, permitindo assim a otimização do sistema. As ferramentas autónomas utilizadas para efetuar os testes são igualmente importantes pois, a escolha de uma ferramenta de qualidade inferior poderá não produzir um resultado fidedigno, levando a uma ideia errada e, consequentemente, problemas futuros.

Capítulo 3

Frameworks para o Desenvolvimento de Aplicações *Web*

Este capítulo apresenta um estudo sobre as *frameworks* mais atuais e utilizadas no desenvolvimento de aplicações *web*, tanto a nível de *front-end* (3.1) como de *back-end* (3.2). São comparadas as *frameworks* em diversos aspetos, terminando-se com as respetivas conclusões (3.3).

Antes de iniciar a implementação de qualquer sistema, é importante escolher a *framework* a ser utilizada. Apesar de ser difícil, ou até mesmo quase impossível encontrar uma *framework* ideal para o problema, é possível encontrar *frameworks* que ajudem significativamente no desenvolvimento do projeto.

De uma forma simples, as *frameworks* são um *software* desenvolvido e utilizado por programadores para construir aplicações. O principal objetivo de uma *framework* é resolver problemas recorrentes de uma forma genérica, permitindo aos programadores não começarem do zero quando necessitam de desenvolver um projeto [43][44].

A utilização de uma *framework* tem diversas vantagens, destacando-se o facto de que auxilia nas melhores práticas de programação e na aplicação de padrões de desenho, o código é mais seguro, evita código duplicado e, principalmente, reduz significativamente o tempo necessário para implementar uma aplicação, entre outras vantagens.

3.1 *Frameworks para Front-end*

As *frameworks* para *front-end* são, normalmente, desenvolvidos em *JavaScript* e funcionam em qualquer navegador moderno, sendo uma parte essencial do desenvolvimento do *front-end* da *web* moderna. Estes fornecem, aos programadores, ferramentas testadas e comprovadas para a construção das aplicações *web* interativas e escaláveis. Consequentemente, muitas empresas modernas utilizam *frameworks* como parte padrão das suas listas de ferramentas.

Após um estudo efetuado através de uma pesquisa extensiva sobre as *frameworks* mais populares dos últimos anos, selecionaram-se as seguintes três: *Angular* [45], *React.js* [46] e *Vue.js* [47]. Estas *frameworks* são as que apresentam uma maior utilização [48][49][50]. De seguida, cada uma das *frameworks* é explicada com mais detalhe, bem como as respetivas vantagens e desvantagens, e qual a escolhida para o presente projeto.

A *Angular* [45] é uma reescrita completa da *AngularJS* [51] pela mesma equipa da Google [52] que desenvolveu a *framework* original em 2010. A *Angular* foi lançada em setembro de 2016 como uma *framework* completamente nova, com novos recursos para resolver os desafios do desenvolvimento *web*.

É uma *framework* completa, em vez de um conjunto de bibliotecas, permitindo aos programadores focarem-se mais na realização de tarefas, sendo esta uma das suas principais vantagens, tendo sido criada a pensar no trabalho de grandes equipas, permitindo que cada parte da equipa possa trabalhar na sua parte do código [53][50].

De uma forma resumida, podem-se elencar as principais vantagens da *Angular* [50][49][54]:

- A arquitetura baseada em componentes da *Angular* permite criar *user interface* (UI) como partes individuais, ou seja, permite a reutilização desses componentes em várias partes da aplicação.

- *TypeScript* é a linguagem principal da *Angular*, é compilada em *JavaScript*, tornando o seu processo de codificação mais fácil.
- Bom desempenho, devido ao *Angular universal support*, que ajuda a carregar as aplicações, tendo a Google criado um conjunto de ferramentas especificas para esse fim.
- Atualizações constantes, pois a equipa da Google está, de forma continuada, a atualizar os componentes da *framework*.

Como todas as *frameworks*, a *Angular* também tem algumas desvantagens que são listadas de seguida [50][49][54]:

- A transferência de sistemas *AngularJS* para *Angular* é complicada pois a diferença entre a *AngularJS* e a *Angular* é enorme. Como tal, é necessário muito tempo e esforço para migrar os sistemas.
- A *Angular* é complexa, apesar da estrutura baseada na *web*, tornando-se complicada a gestão de todos os componentes. Por exemplo, os programadores necessitam de ter vários ficheiros para um único componente *Angular*.
- Dificuldade de aprendizagem, por ser uma *framework* complexa, contando com uma grande curva de aprendizagem inicial.
- A documentação é pouca e escrita de forma descuidada, o que obriga o programador a perder tempo à procura de mais documentação.

A *Vue*.js ou *Vue* [47] é uma *framework web* para a construção de interfaces de utilizador. É uma ferramenta independente e não requer extensões adicionais. A *Vue*.js foi criada por Evan You em fevereiro de 2014. Apesar de não existirem grandes empresas a utilizarem, esta *framework* é popular entre programadores e a sua popularidade tem aumentado consideravelmente. Pode ser integrada facilmente em diferentes bibliotecas e aplicada em projetos maiores. A sua equipa de desenvolvimento lançou algumas extensões para os navegadores, que ajudam no desenvolvimento ao permitir consultar a situação atual dos componentes, e manualmente alterar o estado dos mesmos. A sua documentação abrangente, que permite aos programadores pouparem tempo na aprendizagem, é também uma mais-valia [50][49].

As principais vantagens da *Vue* são as seguintes [50][49][54]:

- Clareza e simplicidade, sendo fácil de aprender permitindo começar o desenvolvimento em pouco tempo.
- Documentação detalhada, extensa e atualizada. Além disso, existe uma grande comunidade de programadores que compartilham os conhecimentos.
- A integração é simples, sendo possível reutilizar código, devido à sua estrutura lógica, permitindo aos programadores criar componentes flexíveis e reutilizá-los posteriormente, mesmo em outros projetos.

Não obstante, a *Vue* também tem algumas desvantagens, sendo apresentadas abaixo as principais [50][49][54]:

- Demasiada flexibilidade leva a irregularidades no código.
- Comunidade de programadores é mais pequena, quando comparada com *React* ou *Angular*.

A *React ou React.js* [46] é uma *framework* criada e desenvolvida pelo Facebook [55]. Foi concebida com o objetivo de obter um elevado desempenho, criando um UI eficaz. A *React* foi lançada como ferramenta *JavaScript* de código aberto em 2013. Em 2015 foi lançada a *React* Native [56] para Android [57] e IOS [58].

A estrutura foi pensada para criar aplicações centradas em componentes com compatibilidade com as versões anteriores, permitindo criar aplicações escaláveis. A *React* destaca-se ainda pelo seu DOM, que oferece uma funcionalidade excecional para as aplicações que antecipam um elevado volume de tráfego e precisam de uma base estável para lidar com todo esse volume [49][48].

Com o lançamento do novo algoritmo *core*, o "*React Fiber*", tornou-se a melhor *framework* para *front-end* devido às suas características distintas [50].

As principais vantagens da *React* são as seguintes [50][49][54]:

- O DOM virtual melhora a experiência do utilizador e facilita o trabalho do programador, permitindo propagar atualizações sem a interferência de outras partes através dos componentes isolados. Isto permite suavizar a experiência de todos os participantes.
- Permite poupar tempo ao reutilizar componentes em qualquer projeto, pois a *React* lida com os componentes de forma isolada.
- É uma biblioteca de código aberto com uma variedade de ferramentas e com uma enorme comunidade.

Tal como as restantes *frameworks*, a *React* também apresenta algumas desvantagens [50][49][54]:

- A curva de aprendizagem é um pouco longa, apesar de não ser tão longa como a da *Angular*.
- Devido ao elevado ritmo de desenvolvimento, a documentação acaba por ficar um pouco esquecida.

Analisando as transferências das *frameworks* apresentados anteriormente, efetuadas desde 2015 até agosto de 2020 (Figura 4), no (*Node Package Manager)* npm [59], através da aplicação npm *trends* [60], é possível concluir que as transferências da *Vue* têm vindo a aumentar ao longo dos anos, ultrapassando as da *Angular*. É, no entanto, a *React* quem tem o maior número de transferências, cerca de 8 milhões de transferências em 2020.

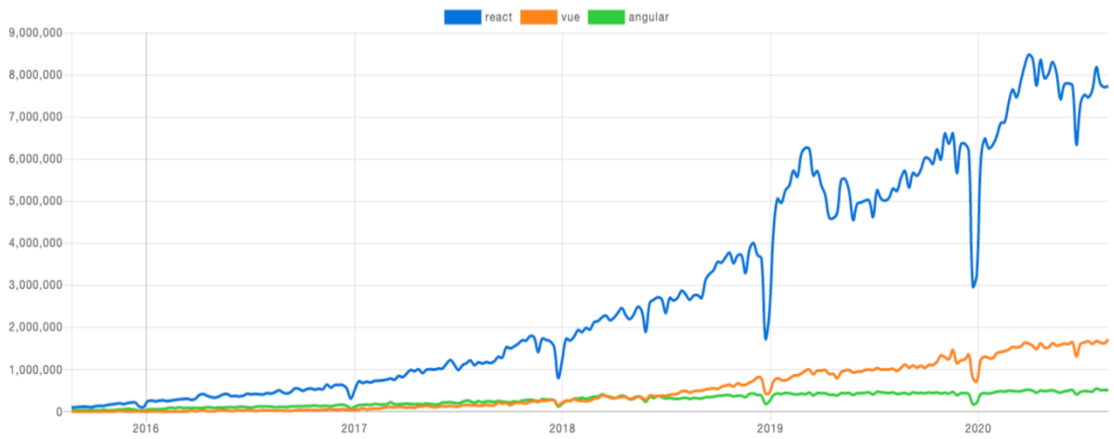


Figura 4 – Estatísticas obtidas no *npm trends* relativamente às *frameworks* de *front-end React, Vue e Angular*. Transferências para o período compreendido entre setembro de 2015 a setembro de 2020.

A Tabela II agrupa a informação disponível até setembro de 2020. De acordo com a mesma, é possível verificar que a *framework* com o menor tamanho é a *React*, sendo o tamanho importante pois influencia o tempo de carregamento da aplicação.

Tabela II - Estatísticas obtidas no npm *trends* relativamente às *frameworks* de *front-end React, Vue e Angular*, para o período compreendido entre setembro de 2015 a setembro de 2020.

Framework	**Estrelas no GitHub**	**Problemas no GitHub**	**Última atualização**	**Tamanho (KB)**
React	154606	562	22-08-2020	2,6
Vue.js	170513	532	20-08-2020	22,9
Angular	59510	466	22-08-2020	62,2

As atualizações também são importantes para a correção de problemas. Como se pode observar na Tabela II, neste ponto, todas as *frameworks* estão atualizadas [61]. O envolvimento da comunidade é outro fator importante pois, quanto maior a comunidade, mais simples é encontrar informação acerca de um determinado assunto ou até mesmo apoio para a resolução de problemas. Podemos observar na Tabela II que, no caso das Estrelas do GitHub (estrelas representam os favoritos), a *React* e a *Vue* têm muito mais estrelas que a *Angular*. Já a nível de questões no *Stack Overflow* [62], podemos observar na Figura 5 que a *framework React* é a que tem a maior percentagem de questões.

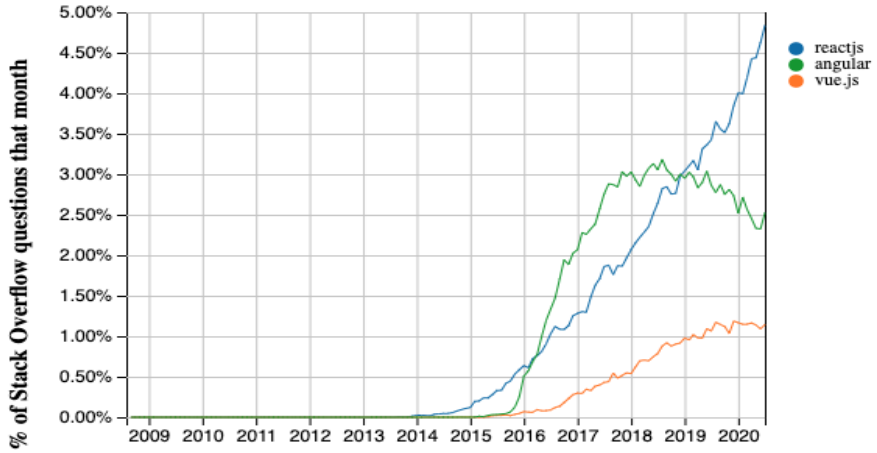


Figura 5 - Estatísticas obtidas no *Stack Overflow trends* relativamente às perguntas sobre as *frameworks* de *front-end React, Vue e Angular*, para o período compreendido entre 2009 a 2020.

A *React* tem apresentado uma tendência crescente nos últimos anos, tendo ultrapassando a *framework* Angular em 2019. A *framework Angular* tem registado uma tendência de descida, já a *framework Vue* mantem-se a alguma distância das outras e com um comportamento de subida mais lento que o da *React*.

3.2 *Frameworks para Back-end*

É necessário uma *framework back-end*, neste caso uma API, de modo a fornecer, ao *front-end* e a outras aplicações ou sistemas, o acesso aos dados necessários. As *frameworks* de *back-end* podem ser desenvolvidas em *Python*, *JavaScript*, PHP ou *java*, entre outras. A cada uma destas linguagens de programação estão associadas várias *frameworks*, sendo alguns exemplos apresentados na Tabela III [63].

Tabela III - Frameworks e respetivas linguagens de programação. Informação adaptada de Lets Nurture [61].

Linguagem de programação	***Framework***
Python	*Django, Flask, Tornado, web2py*
JavaScript	*Express.js, hapi.js, Node.js*
PHP	*Laravel, CodeIgniter, Symfony*
Java	*JHipster, Spring, JSF*

Foi efetuado um estudo com o objetivo de selecionar as *frameworks* de *back-end* mais utilizados. Assim, como API, foram selecionadas as seguintes: *Django*, *Express.js* e *Laravel* [63][64][65]. De seguida, passa-se a explicar, em mais detalhe, cada uma.

A *Django* foi lançado em 2005 e é uma *framework* de código aberto, bastante popular, desenvolvida em *Python* com uma arquitetura *Model View Template* (MVT). É, muitas vezes, referida como a *framework* para perfeccionistas com prazos, sendo baseado no princípio *Don't Repeat Yourself* (DRY), que tem como objetivo a reutilização do código existente, tornando o trabalho do programador mais rápido [64]. A *Django* é utilizada em algumas grandes empresas como o Pinterest, o Washington Post, Dropbox, Spotify, entre outras, contando com uma comunidade superior a 11 mil programadores espalhados por 166 países [66].

A *Express.js* ou *Express* foi lançada em 2010 e é uma *framework* de código aberto, desenvolvida em *JavaScript*. Fornece funcionalidades básicas, podendo ser complementada com inúmeros *plugins*. É rápida e escalável, de fácil aprendizagem e configuração [65]. A *Express* é utilizada por grandes empresas como o Fox Sports, PayPal, Uber, IBM, entre outras [67].

A *Laravel* foi lançada em 2011 e é uma *framework* de código aberto desenvolvida em *HyperText PreProcessor* (PHP) com uma arquitetura *Model View Controller* (MVC). A *Laravel* é a ferramenta ideal para o desenvolvimento do *back-end* de blogs, sites de notícias e aplicações de comércio eletrónico. Possui uma interface intuitiva, o que simplifica o desenvolvimento de aplicações *web* [65].

De modo a selecionar uma das *frameworks* apresentados anteriormente, é necessário analisar vários aspetos, começando pela comunidade. Analisando a Figura 6, podemos verificar que a *Django* é a *framework* com um maior número de questões por mês no *Stack Overflow* [62], seguido da *Laravel* e finalmente a *Express*, esta última com uma diferença considerável.

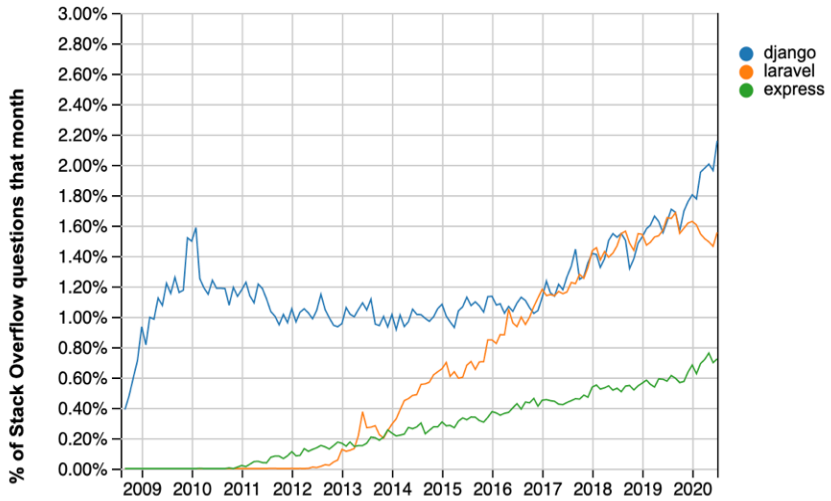


Figura 6 - Estatísticas obtidas no *Stack Overflow trends* relativamente às perguntas sobre as *frameworks* de *back-end Django, Laravel e Express*, para o período compreendido entre 2009 a 2020.

Outro ponto importante é a curva de aprendizagem. Por ser desenvolvido em *Python,* a *Django* tem uma sintaxe bastante simples, mas a arquitetura MVT pode ser um pouco complicada de entender. A *Express*, devido à sua natureza assíncrona, é um pouco mais complexa de aprender, quando comparada com o *Django*, principalmente para programadores novos ao *JavaScript*. No caso da *Laravel*, os programadores com conhecimentos básicos de PHP não deverão encontrar muitos problemas na sua utilização [63].

A nível de desempenho e escalabilidade, tanto a *Django* como a *Express* serão mais rápidas e escaláveis que a *Laravel*, pois o tempo de compilação do PHP é superior ao do *Python* ou do *JavaScript* [68][69][63]. A segurança também é um aspeto importante em qualquer projeto. A Django tem o seu próprio sistema de autenticação (que o torna mais simples e seguro), utilizando ainda um sistema de *QuerySet* [70], que abstrai as consultas, impedindo a utilização de injeções SQL [71]. A *Laravel* também tem os seus mecanismos de segurança, encriptando as *passwords* em *hash* e armazenando-as na base de dados.

O tempo necessário a dedicar pelas equipas para o desenvolvimento também é extremamente importante. Neste ponto, a *Django,* devido ao elevado número de *plugins* que suporta, acaba por reduzir o tempo de desenvolvimento.

3.3 *Conclusão*

Todos as frameworks, tanto de *front-end* como de *back-end,* apresentadas anteriormente, possuem uma posição sólida no mercado, ocupando as principais posições a nível de escolhas. No entanto, quando chega o momento de escolher a *framework* a utilizar no *front-end,* sem dúvida que a *React* será a melhor escolha pois, além de ser a *framework* mais utilizado no momento, a mesma é suportada por uma enorme comunidade de programadores e tem uma vasta escolha de módulos e bibliotecas. É ainda compatível com qualquer navegador moderno, permitindo ainda o desenvolvimento de aplicações para *Android* [57] e *iOS* [58] através da *React Native* [56].

A escolha para o *back-end* depende muito do tempo disponível para o desenvolvimento. A *Django* é a melhor escolha pois é rápida e segura, possui uma grande comunidade e conta com diversos *plugins* disponíveis, permitindo obter uma API minimamente funcional em pouco tempo, possibilitando que o programador se dedique exclusivamente ao desenvolvimento do sistema [63].

Capítulo 4

Análise e Especificação

Neste capítulo apresenta-se, em maior detalhe, os problemas que levaram ao desenvolvimento do presente projeto. São explicitados os requisitos, os casos de utilização, a arquitetura do sistema, os principais diagramas de atividade e, finalmente, os protótipos principais do *layout* da aplicação.

4.1 *Descrição do problema*

O controlo dos atletas em provas de *trails* reveste-se de uma dupla importância, não só por a prova ter um certo risco associado (e, consequentemente, ser importante monitorizar que participantes passaram em determinados pontos), como também por o controlo ser necessário para selecionar os vencedores. O controlo dos atletas processa-se do seguinte modo: 1) controlo no momento da partida, para saber quem realmente iniciou a prova; 2) controlo na passagem (através de uma aplicação móvel utilizada por voluntários), onde o tempo é enviado por *Short Message Service* (SMS) para o servidor. Juntamente com o tempo é enviada informação relativa ao *checkpoint* e o identificador do atleta, para ser possível identificá-lo. Quando os atletas terminam a prova são controlados através de um sistema ligeiramente diferente, sendo a leitura efetuada através de tapetes que detetam o RFID presente no dorsal do atleta [3].

Atualmente, existe uma aplicação *web* em funcionamento [72] onde é possível visualizar os resultados dos eventos *trail.* A aplicação existente [72] surgiu de uma dificuldade, por parte da organização, já que a mesma apenas conseguia obter o estado dos atletas de forma manual e desorganizada. Este tipo de controlo obrigava a um contacto constante, por parte da organização, com os pontos de controlo, de modo a apurar os atletas que já passaram, desistiram ou foram desqualificados [3].

Na Figura 7, está representado a lista de eventos, sendo esta a página principal da aplicação onde o utilizador tem de escolher o evento que deseja consultar de forma a obter acesso ao mesmo. Esta é uma página simples, sem nenhuma grande interação com o utilizador, não dando destaque à prova a decorrer.

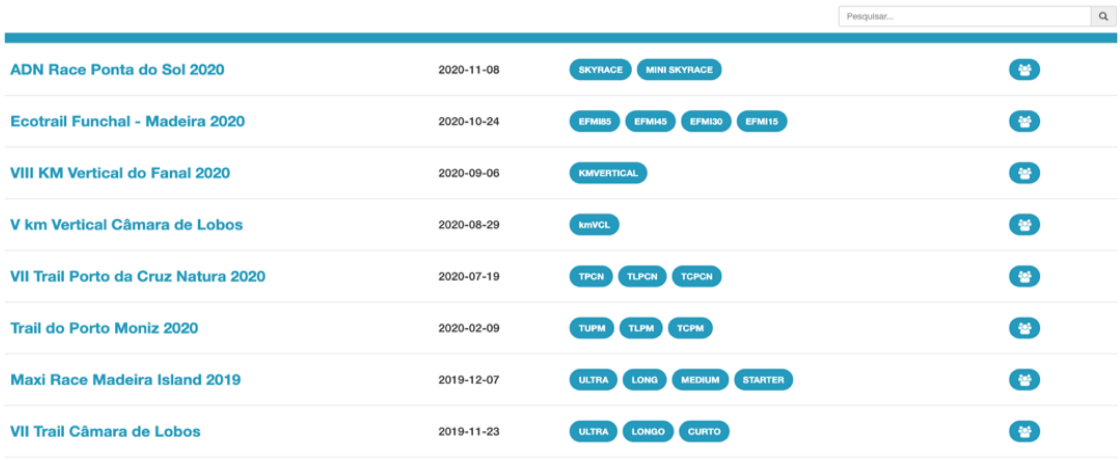


Figura 7 - Lista de eventos, obtida de apus.uma.pt

Na Figura 8, está representada a lista de passagens, onde o esquema de navegação escolhido, com os filtros ao lado esquerdo da tabela, dificulta, aos utilizadores menos experientes, percecionarem que *checkpoint* estão a visualizar, já que não existe nenhum destaque.

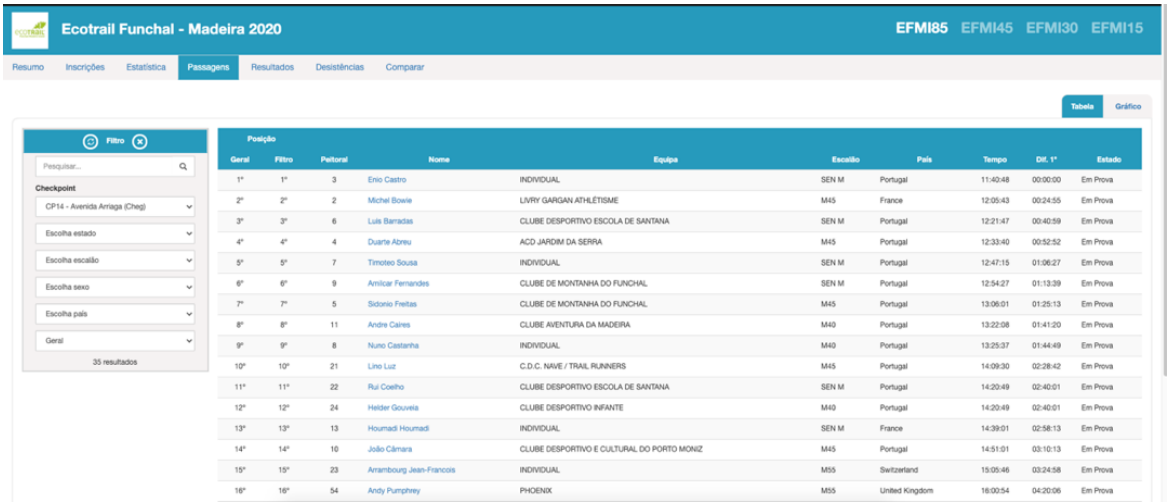


Figura 8 - Lista de passagens, obtida de apus.uma.pt

Por fim, na Figura 9, está representado o detalhe do atleta, podendo-se consultar o tempo, velocidade e a posição em cada *checkpoint*. É ainda possível identificar as posições gerais, juntamente com o estado do atleta. Algumas informações, como a velocidade min/km e alguns gráficos de comparação, encontram-se em falta.

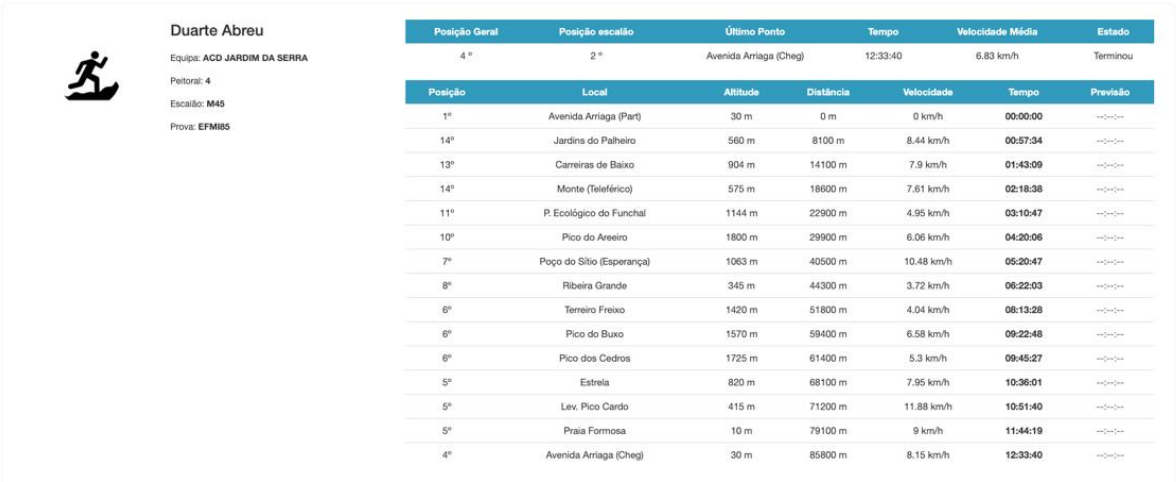


Figura 9 - Detalhe do atleta, obtido de apus.uma.pt

A aplicação anteriormente apresentada sofre de inúmeros problemas de desempenho e escalabilidade, tendo estes problemas sido apresentados, por parte da organização, em inúmeras reuniões. Estes acabam por afetar a organização, dificultando o acesso aos resultados ou ao estado da prova. Afetam

ainda os familiares ou amigos de um atleta, pois os mesmos não sabem o estado do atleta durante a prova. De forma a mitigar estes tipos de problemas, a organização tem utilizado redes CDN em eventos de maior dimensão, todavia este tipo de solução criou alguns desafios e problemas. Isto é, a CDN estava, constantemente, a pedir ao servidor a informação atualizada e, como o servidor não conseguia dar resposta aos pedidos da CDN, acabava por causar lentidão.

De seguida, apresenta-se o modelo entidade-relação, Figura 10, que contém as tabelas principais utilizadas pela aplicação pública, descritas de seguida.

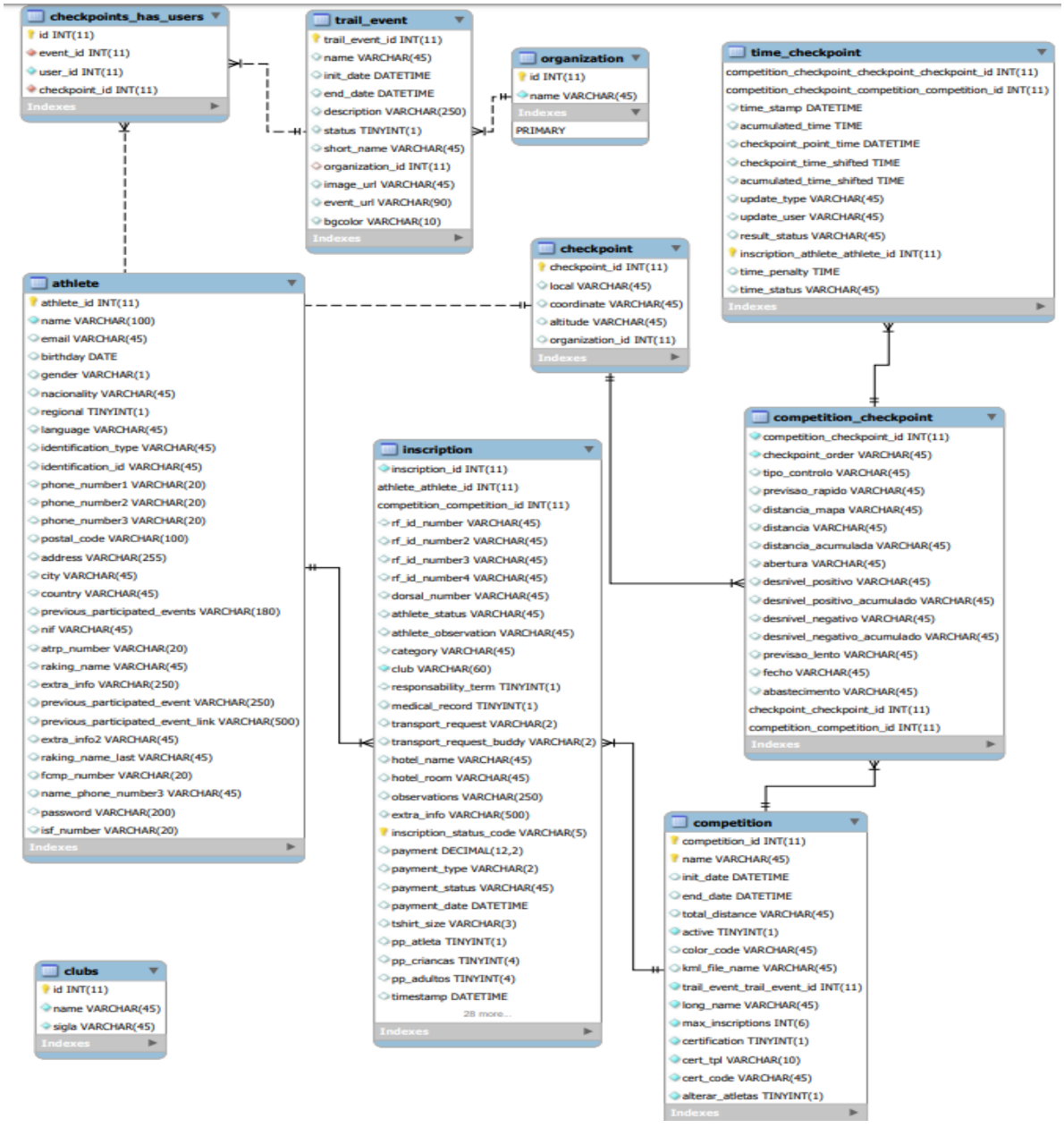


Figura 10 - Modelo entidade-relação da base de dados atual.

A tabela *athlete* é responsável por armazenar as informações pessoais de cada atleta, nomeadamente o seu nome, contacto, email, género, entre outras. A mais importante é o contacto pois, no caso do atleta estar perdido, torna-se necessário recorrer a essa informação para contactá-lo o mais rapidamente possível. É importante destacar que cada atleta tem armazenado vários contactos pois, no caso de o atleta estar incontactável, é necessário contactar algum familiar ou amigo.

A tabela *inscription* é responsável por armazenar as informações relativas à inscrição de um atleta, nomeadamente, a categoria e o clube entre outras, sendo o RFID a informação mais importante, pois é fundamental para identificar o atleta cada vez que ele passa por um *checkpoint*.

A tabela *trail_event* é responsável por armazenar os dados relativos a um evento *trail* (nome, data, organização).

A tabela *competition* armazena os dados das competições existentes de um determinado evento de *trail*.

A tabela *time_checkpoint* é responsável por armazenar os tempos dos atletas em cada *checkpoint* ao longo da competição.

A tabela *checkpoint* é responsável por armazenar os dados de cada *checkpoint* de cada prova.

Finalmente, a tabela *clubs* é responsável por armazenar os dados referentes aos clubes.

Após uma análise da base de dados, identificaram-se alguns problemas com a conceção da base de dados, problemas estes que acabam por dificultar *queries* simples e afetam, em última instância, o desempenho global do sistema. Por exemplo, a tabela *clubs* não têm nenhuma relação com outra tabela, não havendo assim a validação de um clube no ato da inscrição, existindo o mesmo clube escrito de forma diferentes ou com erros.

Não é possível efetuar grandes alterações à base de dados pois, devido à forma como todo o sistema está implementado, existe uma grande interligação entre a interface e a base de dados, ou seja, não é possível modificar a base de dados sem modificar a interface. Basicamente, todo o projeto está parado, pois este tipo de modelo dificulta a criação de novas funções ou, até mesmo, de novas aplicações como, por exemplo, uma aplicação móvel. Existem ainda preocupações de segurança por parte dos responsáveis. Estas preocupações não estão relacionadas apenas com a aplicação pública, mas sim com todas as aplicações que acedem à base de dados para consulta ou escrita. Dado que cada aplicação utiliza o seu próprio sistema de acesso e controlo à base dados (já que não existe um sistema unificado), qualquer mudança na base de dados é dificultada. É, assim, imperativo alterar a arquitetura da aplicação de modo a suportar a integração de outras aplicações de forma normalizada e segura, permitindo efetuar modificações sempre que se justifique.

4.2 *Requisitos*

Um requisito trata-se de uma necessidade que o cliente espera ser solucionada pelo sistema que será desenvolvido. Os requisitos são tipicamente utilizados como informações fundamentais para o desenvolvimento do projeto e especificam as propriedades e funções necessárias a serem consideradas no seu desenvolvimento [73].

Nesta fase, para uma melhor perceção das necessidades, foi feito um levantamento de requisitos, permitindo a identificação e organização de funcionalidades e requisitos, o que facilita a implementação e manutenção, assim como futuras modificações.

O levantamento foi realizado através de diversas reuniões com o cliente. As reuniões demostraram ser bastante produtivas, não só para a identificação de problemas críticos já existentes, como também para a identificação de novas funcionalidades pretendidas. Estes encontros possibilitaram ainda ganhar conhecimento sobre o funcionamento das provas de *trail* e os respetivos regulamentos.

Foram também obtidos requisitos através da análise de diversas aplicações de *trail* nacionais e internacionais, como a plataforma MYLAPS [3], *Ultra Trail Italy* [3], TransGranCanaria [3], Apus [74], Translantau [74] e Trailrunaustralia [74]. Após a conclusão deste processo, obtiveram-se requisitos globais, requisitos de configuração e requisitos não funcionais.

No que concerne aos requisitos globais (1), apresenta-se a lista obtida:

1.1. O sistema deverá apresentar os eventos;
1.2. O sistema deverá permitir apresentar as provas;
1.3. O sistema deverá apresentar os inscritos;
1.4. O sistema deverá apresentar os tempos de passagem;
1.5. O sistema deverá apresentar a lista dos desistentes;
    1.5.1. O sistema deverá apresentar onde desistiram;
1.6. O sistema deverá apresentar os participantes desqualificados;
    1.6.1. O sistema deverá informar onde e qual o motivo da desqualificação;
1.7. O sistema deverá apresentar os vencedores gerais, por género e escalão;
1.8. O sistema deverá apresentar as equipas vencedoras por género;
1.9. O sistema deverá permitir identificar atletas favoritos;
1.10. O sistema deverá notificar cada vez que um atleta favorito passa por um checkpoint;
1.11. O sistema deverá permitir comparar atletas ao longo da prova;
1.12. O sistema deverá permitir filtros sobre as listagens;
1.13. O sistema deverá apresentar estatísticas do evento, sob a forma de tabela e gráfico;
1.14. O sistema deverá ser multilingue.

Foi também criada uma lista de requisitos de configuração (2). Estes requisitos estão direcionados para as diversas aplicações existentes, não estando diretamente relacionados com a aplicação pública, mas a mesma precisa destas aplicações para funcionar:

2.1. Deverá ser possível criar, editar ou apagar um evento, apenas para utilizadores de um grupo específico;
2.2. Deverá ser possível criar, editar ou apagar uma prova, apenas para utilizadores de um grupo específico;
2.3. Deverá ser possível criar, editar ou apagar um participante, apenas para utilizadores de um grupo específico;
    2.3.1. Deverá apenas ser possível apagar um participante, se o mesmo não tiver tempos;
2.4. Deverá ser possível efetuar *check-ins* dos participantes, apenas para utilizadores de um grupo específico;
2.5. Deverá ser possível inserir ou editar um tempo de um participante, apenas para utilizadores de um grupo específico;
2.6. Deverá ser possível mudar o estado do atleta durante a competição, apenas para utilizadores de um grupo específico;
2.7. Os estados apenas podem ser I - inscrito, CI - inscrição confirmada, D - desistiu, EP - em prova, DNS - não começou, FIN - terminou, DNF - não terminou ou desistiu, DSQ – desqualificado;
2.8. Os estados têm de seguir a regra presente na Figura 11;
2.9. O sistema deverá disponibilizar uma API para outros sistemas se ligarem.

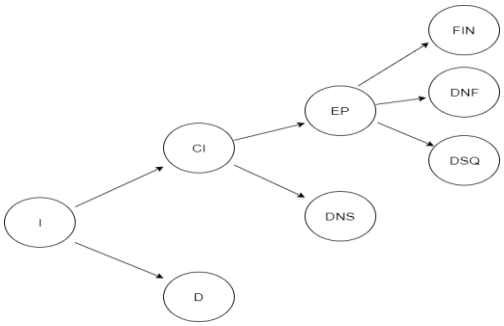


Figura 11 - Diagrama de estados dos atletas. I - inscrito, CI - inscrição confirmada, D - desistiu, EP - em prova, DNS - não começou, FIN - terminou, DNF - não terminou ou desistiu, DSQ – desqualificado.

Por fim, os requisitos não funcionais (3) são os requisitos relacionados com o uso da aplicação em termos de desempenho, usabilidade, confiabilidade, segurança, disponibilidade, manutenção e tecnologias envolvidas. De seguida, estão listados os requisitos não funcionais.

3.1. O sistema deverá ser rápido;
3.2. O sistema deverá ser seguro;
3.3. O sistema deverá ser consistente;
3.4. O sistema deverá estar disponível;
3.5. O sistema deverá ter flexibilidade;
3.6. O sistema deverá ter integridade;
3.7. O sistema deverá ser compatível;
3.8. O sistema deverá ser confiável;
3.9. O sistema deverá ser robusto;
3.10. O sistema deverá ser tolerante a falhas;
3.11. O sistema deverá ter a documentação necessária:
3.12. O sistema deverá cumprir os aspetos legais, principalmente tendo em atenção o Regulamento Geral sobre a Proteção de Dados (RGPD).

4.3 *Casos de utilização*

Casos de utilização são listas de ações, ou etapas de eventos, que geralmente definem as interações (funções) entre um ator e um sistema, para atingir um objetivo. O ator pode ser um humano ou outro sistema externo. Os casos de utilização são usados geralmente para representar objetivos ou requisitos. Neste caso, o diagrama de casos de utilização deriva dos requisitos funcionais [75]. No diagrama da Figura 12 estão representados dois tipos de utilizadores (os atores): o público (utilizadores que visitam a aplicação) e os administradores (não acedem diretamente ao site, mas sim à própria API que gera as listas *JavaScript Object Notation* (JSON's) que são enviadas para o site e outras aplicações).

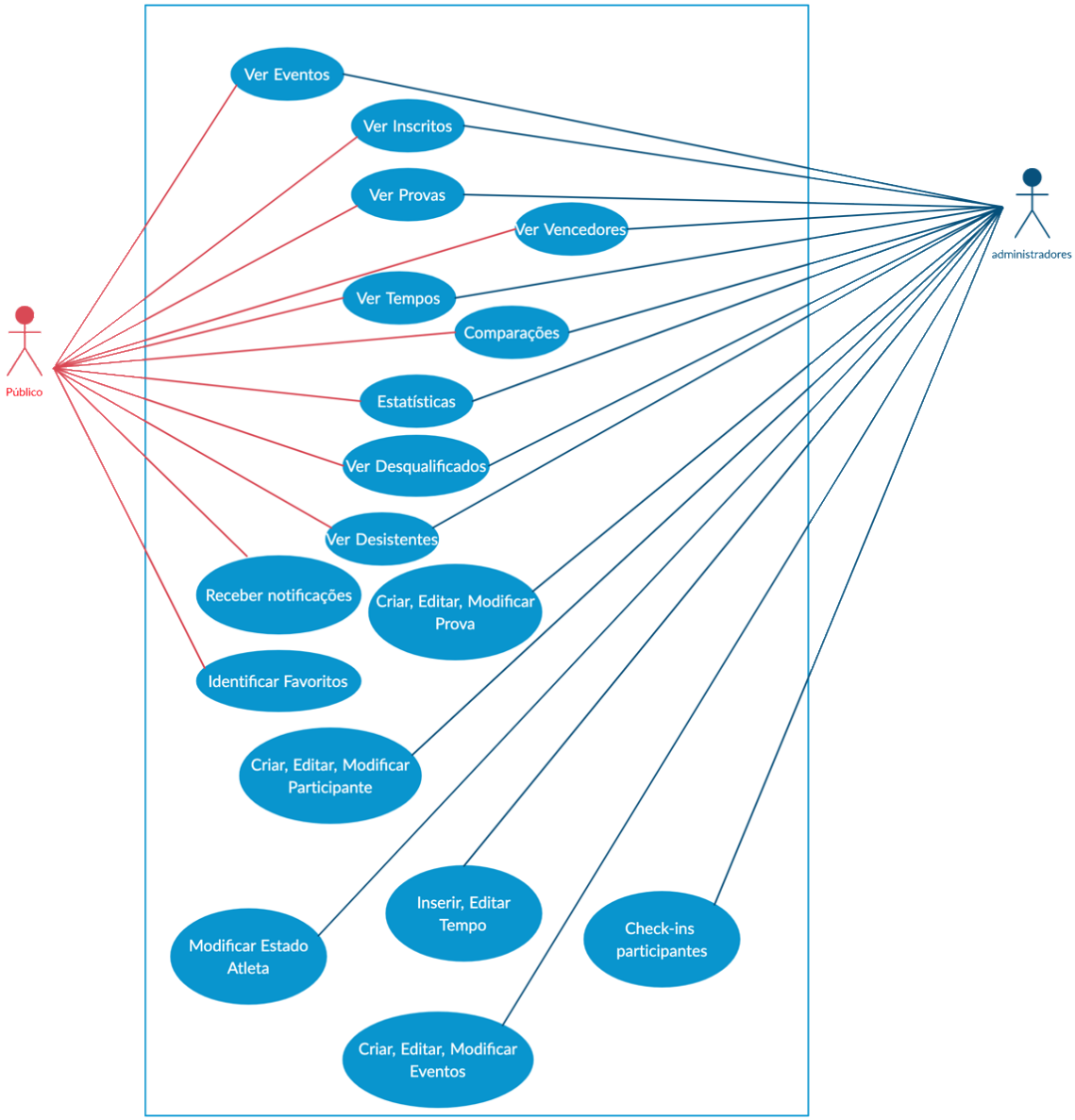


Figura 12 - Casos de utilização. Encontra-se representada a interação dinâmica entre o ator e o sistema.

Analisando a Figura 12, podemos concluir que o público consegue visualizar eventos e respetivas provas bem comos os vencedores, as inscrições nos eventos e nas provas, consultar os tempos dos atletas, desistências e desqualificações e, por fim, identificar favoritos (e, por conseguinte, receber notificações de quando um atleta favorito passa num determinado *checkpoint*). Já os administradores conseguem realizar as mesmas ações e efetuar modificações ou inserir novos dados.

4.4 *Diagrama de atividade*

O diagrama de atividade é um diagrama definido pela *Unified Modeling Language* (UML) e representa os fluxos de trabalho. É essencialmente um gráfico de fluxo, mostrando o fluxo de controlo de uma atividade para outra. Enquanto os diagramas de interação dão ênfase ao fluxo de controlo de um objeto para outro, os diagramas de atividades dão destaque ao fluxo de controlo de uma atividade para outra. Estes diagramas não são importantes somente para a modelação de aspetos dinâmicos de um sistema ou um fluxograma, mas também para a construção de sistemas informáticos. A cada caso de utilização está associado um conjunto de ações, quer por parte do cliente, quer por parte do sistema [76][77]. De seguida, são apresentados alguns diagramas de atividade mais importantes, os restantes encontram-se nos Anexo 1 e Anexo 2.

No diagrama de atividade da Figura 13 está representada a ação "Consultar os Eventos". Neste caso, o cliente solicita ao servidor os eventos. Para tal, é efetuado uma verificação de segurança para confirmar que o pedido vem de uma fonte autorizada e, caso seja autorizado, será devolvida a lista de eventos.

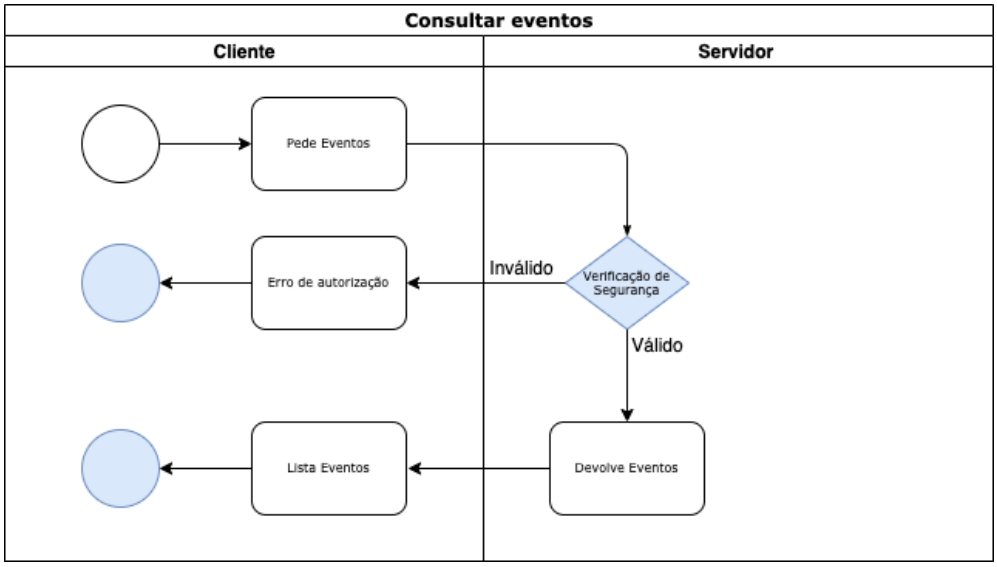


Figura 13 - Diagrama de atividade referente à ação "Consultar Eventos".

O diagrama de atividade da **Error! Reference source not found.** esquematiza a ação "Consultar as Inscrições" de um determinado evento. Após a seleção de um evento pelo utilizador, é efetuada uma verificação de segurança para confirmar que o pedido vem de uma fonte autorizada. Caso seja autorizado, será devolvida a lista de inscritos.

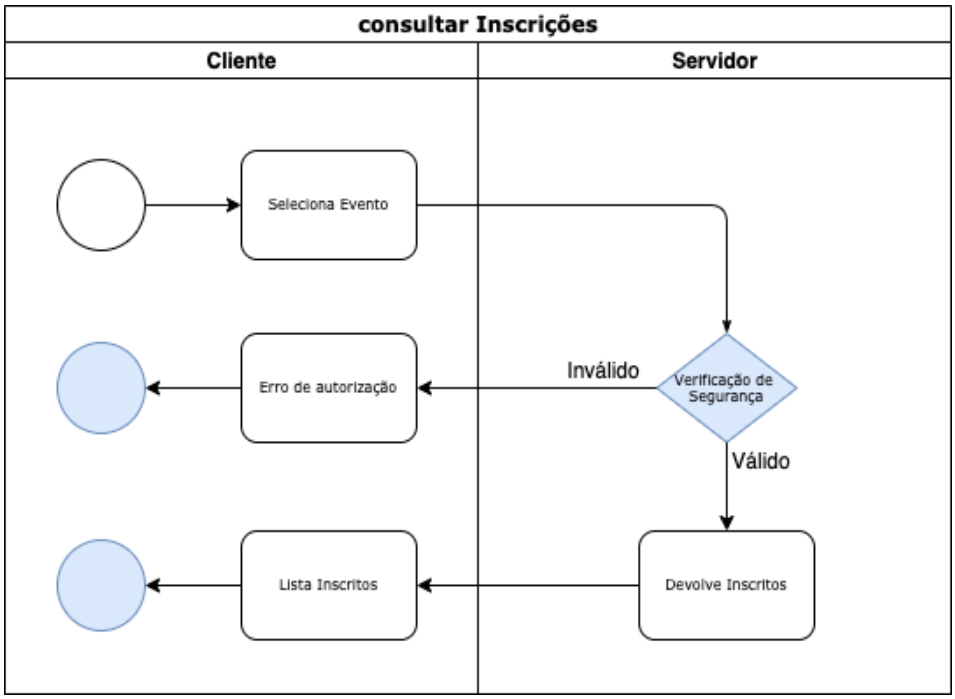


Figura 14 - Diagrama de atividade referente à ação "Consultar Inscrições".

No diagrama de atividade da Figura 15 está representada a ação "Adicionar Favoritos". Neste caso, o servidor recebe a informação que um utilizador (neste caso, o navegador) deseja adicionar um favorito e adiciona esse atleta à lista de atletas favoritos.

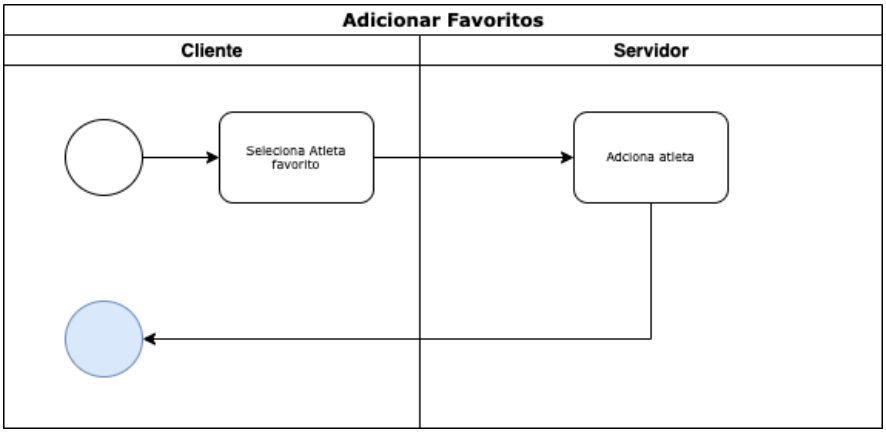


Figura 15 - Diagrama de atividade referente à ação "Adicionar Favoritos".

No diagrama de atividade da Figura 16 está representada a ação de "Gerar as Notificações". Neste caso, o sistema de tempos informa o servidor, que recebe a informação que tem de enviar uma notificação, verifica se essa informação é válida e verifica se o atleta pertence à lista de atletas favoritos. Caso pertença, o servidor seleciona os navegadores que assinalaram o atleta como favorito e envia as respetivas notificações.

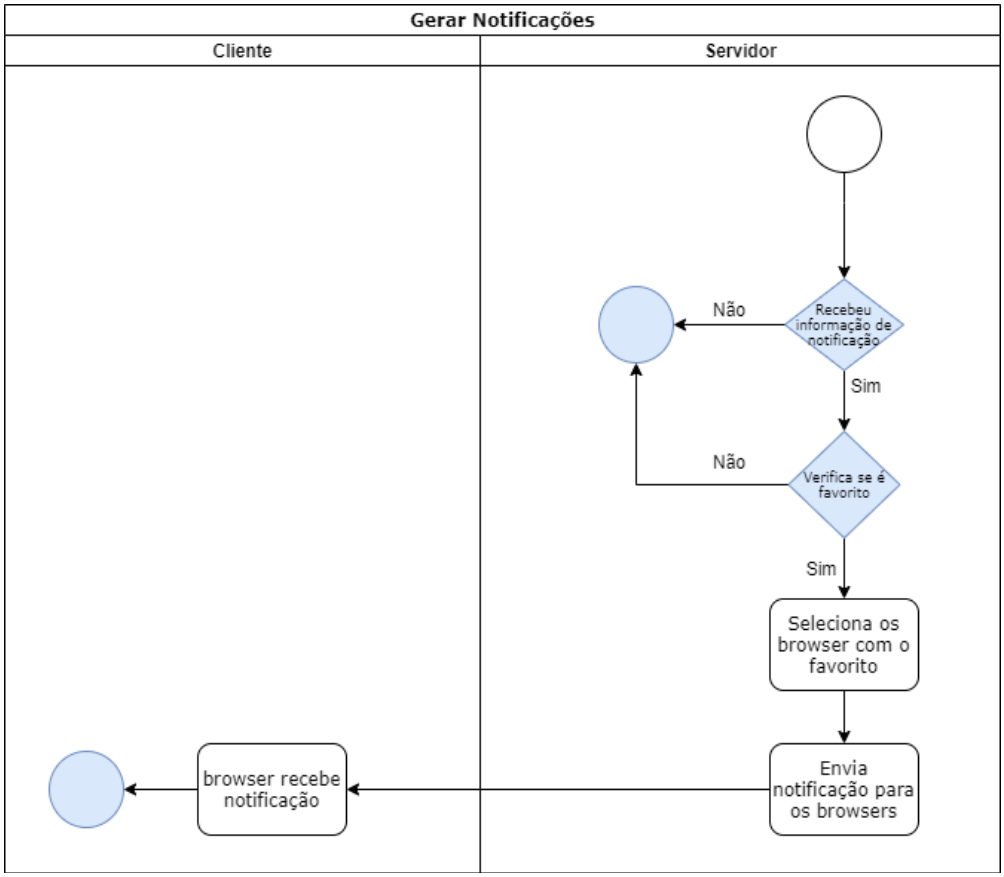


Figura 16 - Diagrama de atividade referente à ação "Gerar Notificações".

Em resumo, nos diagramas anteriormente apresentados está esquematizado o fluxo de trabalho dos principais componentes: consultar os eventos, consultar as inscrições, adicionar favoritos e gerar notificações. Apesar de existirem outros componentes também importantes, o seu fluxo de trabalho é semelhante ao diagrama de "Consultar Inscrições".

4.5 *Protótipos do layout*

Um protótipo é definido como sendo uma amostra ou modelo, ou seja, é uma versão inicial de um produto, criado para testar um conceito. Protótipo é um termo usado numa variedade de contextos, incluindo semântica, *design*, eletrónica e desenvolvimento de *software*. Um protótipo é geralmente usado para avaliar um novo *design* ou para aprimorar a precisão do projeto, por analistas de sistema e utilizadores. A prototipagem serve para fornecer especificações para um sistema de trabalho real, e não para um sistema teórico. Em alguns modelos de fluxo de trabalho de *design*, a criação de protótipos é o passo fundamental entre a formalização e a avaliação de uma ideia [78][79].

Durante este projeto, foram elaborados protótipos para as principais zonas da aplicação. O protótipo da Figura 17 diz respeito à lista de inscrições, sendo o esquema de *layout* o mesmo para todas as páginas da aplicação. Optou-se por adicionar filtros a cada coluna em vez de um filtro global com o objetivo de simplificar a utilização da aplicação.

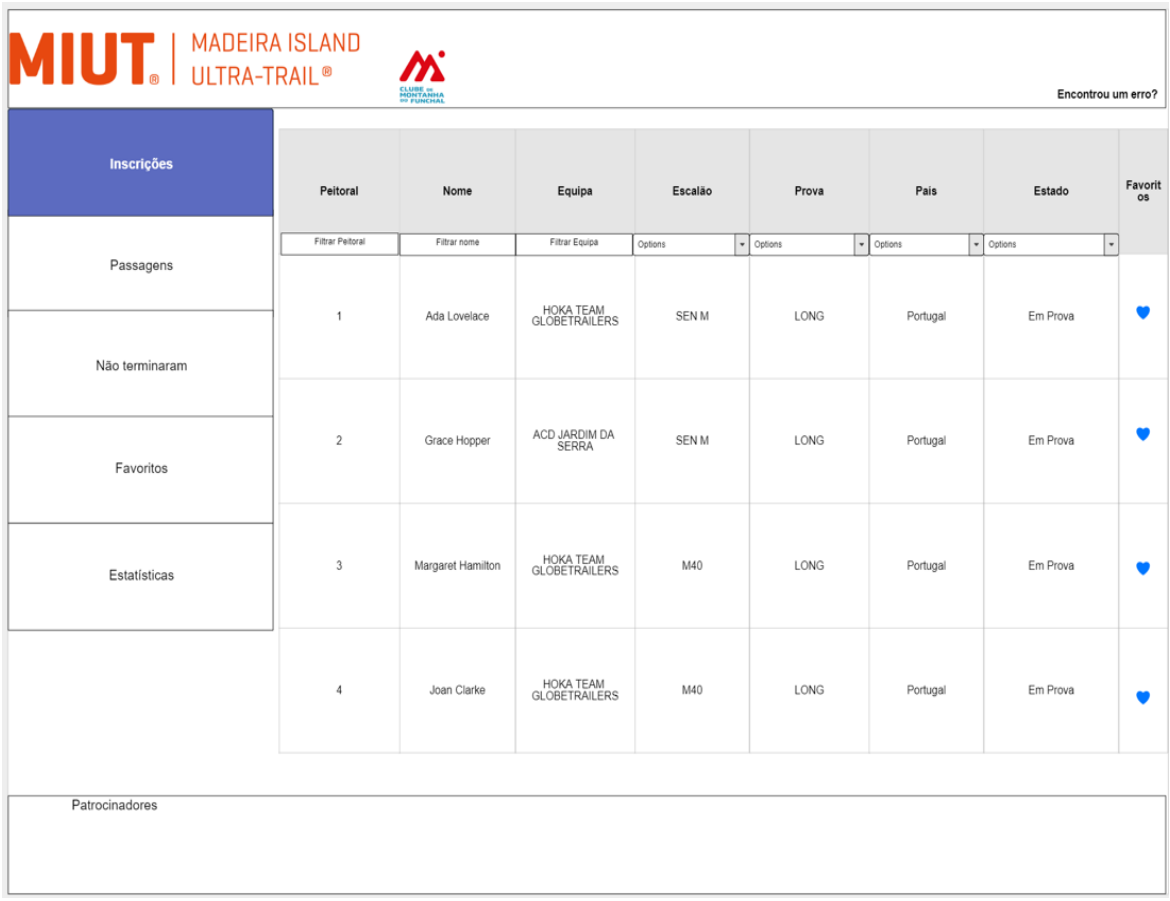


Figura 17 – Protótipo para a lista de inscritos.

O protótipo disponibilizado na Figura 18 diz respeito à lista de passagens. Sobre a tabela de passagens está a lista de *checkpoints,* em que o utilizador pode escolher qual o *checkpoint* que deseja consultar. Na parte superior é possível escolher a competição que o utilizador deseja consultar (MIUT, Ultra, Longa, Marathon e Curta).

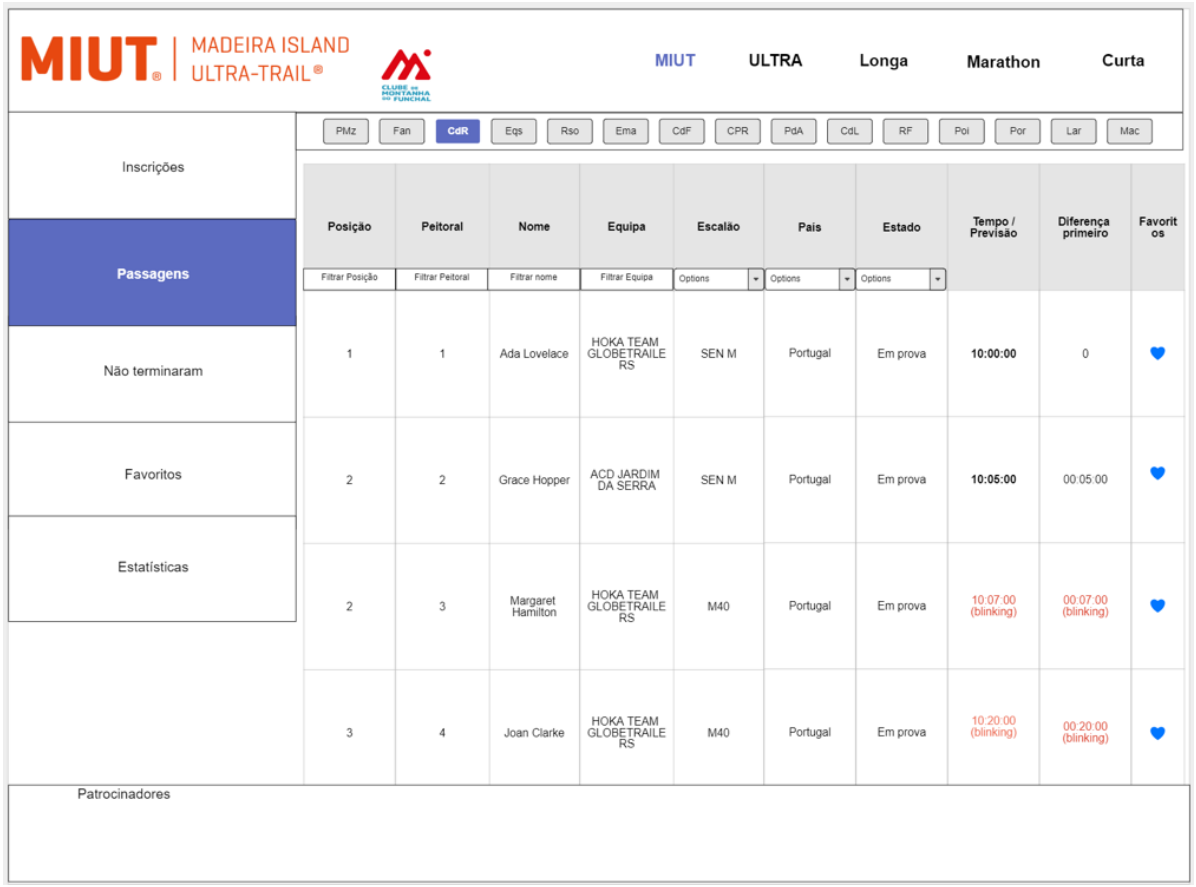


Figura 18 – Protótipo para a lista de passagens.

Nas restantes páginas, o *layout* será semelhante, à exceção da página do detalhe do atleta, exemplificada na Figura 19. Nesta página é mostrada a lista dos tempos do atleta em cada ponto bem como a previsão para o tempo no próximo ponto. A página contém ainda alguns gráficos estatísticos, permitindo comparar o atleta com outros.

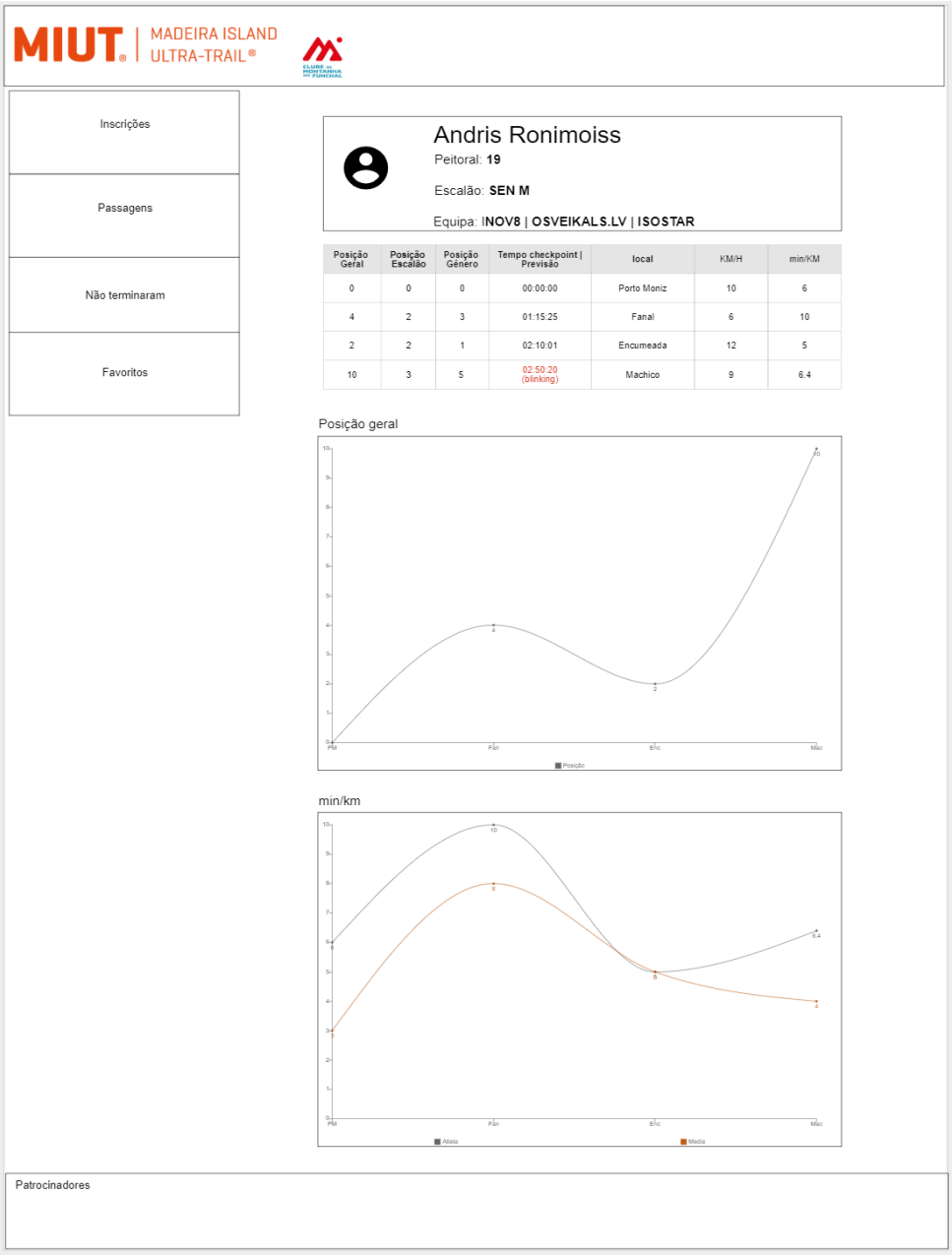


Figura 19 – Protótipo para o detalhe do atleta.

Em suma, foram apresentados anteriormente os principais protótipos, lista de inscritos, lista de passagens e detalhe do atleta. Apesar da existência de outros protótipos (Anexo 3 e Anexo 4), foram referidos os anteriores pois os restantes são semelhantes ao protótipo da lista de inscritos. Estes protótipos serão utilizados como guia na fase de implementação, podendo alguns ajustes serem efetuados caso se justifique.

4.6 *Arquitetura do conceptual*

A arquitetura conceptual refere-se às estruturas fundamentais dum sistema. É composta por diversos elementos, as relações e as propriedades de um sistema e seus componentes. Trata-se de fazer as escolhas fundamentais que são complicadas de alterar na fase de implementação [80]. A arquitetura conceptual deste projeto está representada na Figura 20.

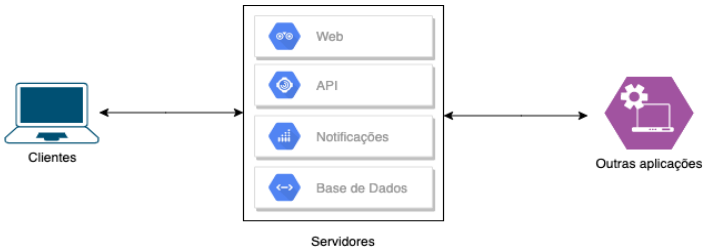


Figura 20 – Representação esquemática da arquitetura do sistema

A Figura 20 é composta pelos clientes que acedem à informação disponível nos servidores. Já os servidores estão separados pelos diversos sistemas necessários para um bom funcionamento, que se explica de seguida.

No servidor *web* está a aplicação pública que será utilizada pelo público em geral. O servidor da API é responsável pela interligação das diversas aplicações e pelo controlo dos acessos à base de dados. O servidor de notificações é responsável pela nova funcionalidade de enviar notificações aos utilizadores da aplicação pública. Por fim, o servidor de base de dados é onde está alojada a base de dados existente.

As outras aplicações (Figura 20) acedem aos servidores para obter, ou inserir, a informação necessária para o seu funcionamento. Esta arquitetura serve de base para a arquitetura que será implementada e que está descrita em maior detalhe na secção 5.1.

4.7 *Conclusão*

Este capítulo inicia-se com uma breve explicação da aplicação existente e respetivas limitações e problemas, nomeadamente a falta de desempenho e as dificuldades em efetuar modificações devido às limitações da arquitetura. Explicam-se as principais tabelas da base de dados e descrevem-se os requisitos da nova aplicação, os casos de utilização. São ainda explicados os principais diagramas de atividade e os protótipos do *design* da aplicação, tendo-se optado por um *layout* simples e moderno de forma a facilitar o acesso aos dados por parte do utilizador. Não tendo sido possível efetuar teste de usabilidade aos protótipos com os utilizadores. Por fim, é apresentada a arquitetura conceptual do sistema, explicando-se as funções dos diversos servidores.

Capítulo 5

Implementação

Este capítulo está organizado em três seções: a arquitetura do sistema (5.1), a API (5.2) e o *front-end* (5.3). Começa-se por explicar a arquitetura do sistema, seguindo-se para o que é uma API, quais as regras implementadas, neste projeto, a nível da nomenclatura, documentação, processos de autenticação e consulta de dados. De seguida, expõem-se as escolhas efetuadas no *front-end*, explicando-se as páginas que foram criadas, como foi feita a organização dos diferentes dados e as decisões tomadas para otimizar o desempenho da aplicação. Conclui-se apresentando a opinião pessoal acerca das *frameworks* utilizadas.

5.1 *Arquitetura do sistema*

A arquitetura de um sistema consiste na definição dos componentes do sistema, as suas propriedades externas e as suas relações com outros componentes. A documentação da arquitetura do sistema facilita a comunicação entre os *stakeholders*, regista as decisões e permite a reutilização do projeto ou dos seus componentes em outros projetos [81][82].

A arquitetura deste projeto está representada na Figura 21. É composta por cinco componentes, estando no servidor *web* a aplicação a que todos os visitantes terão acesso. Este servidor é uma máquina em *Linux*, utiliza o *Nginx* (servidor web, funciona também como *reverse proxy* e gestor de carga) e aloja a aplicação desenvolvida em *React*. Este servidor comunica ainda com o servidor da API de modo a obter os dados que necessita. O servidor da API é uma máquina em *Linux*, utiliza o *Nginx* e aloja a API desenvolvida em *Django*. Neste servidor são feitas as validações de segurança e, por sua vez, este comunica com o servidor de base de dados para obter, ou adicionar, os dados necessários de forma a dar resposta aos pedidos. O servidor de notificações é uma máquina em *Linux*, utiliza o *Node.js* e aloja o sistema de notificações. É responsável por enviar notificações aos clientes e obtém os dados através das outras aplicações, nomeadamente o sistema de tempos. As restantes aplicações são todas as que estão em funcionamento e estão relacionadas com um evento *trail* e que comunicam com o servidor da API de modo a obter ou inserir dados. Todas as comunicações são efetuadas através de HTTPS, sendo isto um requisito das diversas *frameworks* implementadas (à exceção da ligação da API com a base de dados, pois, como dito anteriormente, não é possível efetuar grandes alterações à base de dados nem à ligação, sob risco dos diversos sistemas deixarem de funcionar). A utilização do *Linux* e do *Nginx* foi sugerido por parte da administração de redes, pois já é amplamente utilizado noutros sistemas.

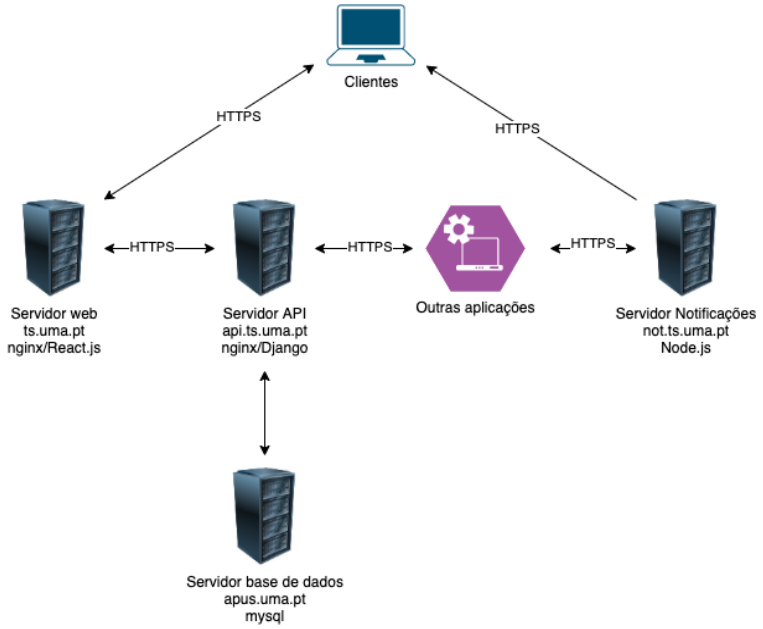


Figura 21 – Representação esquemática da arquitetura da implementação

Escolheu-se este modelo arquitetural ambicionando promover uma maior interligação entre as aplicações e um acesso aos dados controlados pela API. Esta solução também foi pensada para utilizar soluções de replicação sempre que se justifique, isto é, em provas de maior escala. Podem ser replicados apenas alguns servidores, evitando-se uma replicação completa do sistema com consequente poupança de recursos. Tendo em conta que o servidor da API é responsável por receber pedidos tanto por parte do servidor *web* como por parte das diversas aplicações, este é o mais previsível de necessitar de soluções de replicação pois está sujeito a uma maior carga de pedidos.

Em resumo, esta arquitetura foi pensada com o objetivo de ser adaptativa, escalável, promotora de uma interligação dos diferentes sistemas e um acesso mais controlado aos dados. A utilização de servidores em máquinas diferentes, ou até mesmo *containers* diferentes, permite manter em separado todos os subsistemas, facilitando a replicação de um subsistema, em vez do sistema completo.

5.2 *API - Application Programming Interface (back-end)*

Com intuito de alcançar o objetivo inicialmente proposto de interligar todas as aplicações e ajudar a melhorar o desempenho e escalabilidade do sistema, foi decidido desenvolver uma *Application Programming Interface* (API). Esta *interface* é um conjunto de padrões de programação que permitem a construção e utilização de aplicações que podem ser ligados a diversos sistemas e aplicações [83]. Neste projeto, a API foi criada com o propósito de ser a intermediária entre todas as comunicações e a base de dados. A API recebe um pedido de um cliente, consulta a base de dados e devolve uma resposta ao cliente, sendo ainda responsável pela autenticação dos clientes e aplicação de filtros.

A segurança de uma API é um fator importante pois, sem uma segurança adequada, estas estão vulneráveis a utilizadores mal-intencionados. Por exemplo, um utilizador pode expor publicamente dados médicos, financeiros e pessoais, ou alterar informações na base de dados, atos que poderão levar a inúmeras consequências legais. Contudo, nem todos os dados têm a mesma importância nem devem ser protegidos do mesmo modo. A abordagem de segurança de uma API a ser adotada dependerá do tipo de dados que são acedidos. Neste projeto, a autenticação será efetuada por *tokens* [84] (quando o acesso é efetuado através de uma aplicação) ou utilizador e password (quando o acesso é efetuado pelo navegador). Um *token* contém as credenciais de segurança para uma sessão, identificando o utilizador, os grupos do utilizador, os privilégios e em alguns casos, a aplicação específica.

Como este projeto contém alguns dados delicados, como os dados pessoais de cada atleta e informações vitais para a prova como o tempo de cada atleta, optou-se por dividir as consultas em dois tipos:

- Consultas públicas: estas consultas não contêm nenhuma informação pessoal ou confidencial. Assim sendo, qualquer utilizador ou aplicação com um *token* válido podem aceder.

- Consultas privadas: o utilizador ou aplicação necessita, além de um *token* válido, de pertencer a um grupo específico para efetuar a operação. Nas consultas privadas estão incluídas todas as operações que poderão alterar algum dado na base de dados.

Além da segurança, a organização da API também é importante, devendo estar separada por categorias para possibilitar aos utilizadores obter a informação que desejam mais facilmente. A presente API está divida nas seguintes seis categorias:

- *Auth*
- *Athlete*
- *Checkpoint*
- *Competion*
- *Results*
- *Trail*

Na categoria *Auth* estão todas as consultas relacionadas com a autenticação de utilizadores da API, bem como criação de *tokens*. Na categoria *trail* encontram-se todas as consultas relacionadas com as informações dos *trails* e consultas que permitem editar informação de um *trail*. Na categoria *competition* agrupam-se todas as consultas relacionadas com as informações das competições e consultas que permitem editar informação de uma competição. Na categoria *checkpoint* tem-se todas as consultas relacionadas com as informações dos pontos de controlo e consultas que possibilitam editar informação de um *checkpoint*. A categoria *athlete* engloba todas as consultas relacionadas com os atletas, nomeadamente consultas relacionadas com as informações do atleta e consultas para editar essas mesmas informações. Por fim, na categoria *results* temos todas as consultas relacionadas com os resultados, tanto dos atletas como das equipas.

Neste projeto foram definidas diversas regras, uma das quais: todas as consultas de uma categoria começam pelo nome da categoria seguido de uma "/" e o nome da consulta. No caso de existirem parâmetros, os mesmos deverão ter a abreviatura da categoria à qual pertencem, seguidos de um "_" e o nome do parâmetro.

No caso de uma consulta sem parâmetros, a mesma deverá ser efetuada como demostrado abaixo:

- < serveraddress >/ trail/ EventRecents/

No caso de uma consulta com parâmetros, deverá ser efetuada da seguinte forma:

- < serveraddress >/Athlete/AthName/ath_ID

No caso de existir mais que uma versão da mesma chamada, deverá ser efetuada de acordo com:

- <serveraddress>/V2/Athlete/AthList/

A API foi desenvolvida em *Django* pelas razões explicadas anteriormente. Recapitulando, a *Django* é baseada em *Python*, tem um bom desempenho e suporta inúmeros *plugins* reduzindo o tempo de desenvolvimento. Como podemos visualizar na Figura 22, um utilizador autorizado ao aceder à API, utilizando o navegador, terá acesso a todas as informações que precisa para começar a conhecer o sistema. Pode rapidamente aceder ou criar o *token* de acesso para a sua aplicação, aceder à área de administração, caso tenha permissões para isso ,ou simplesmente aceder à documentação [85].

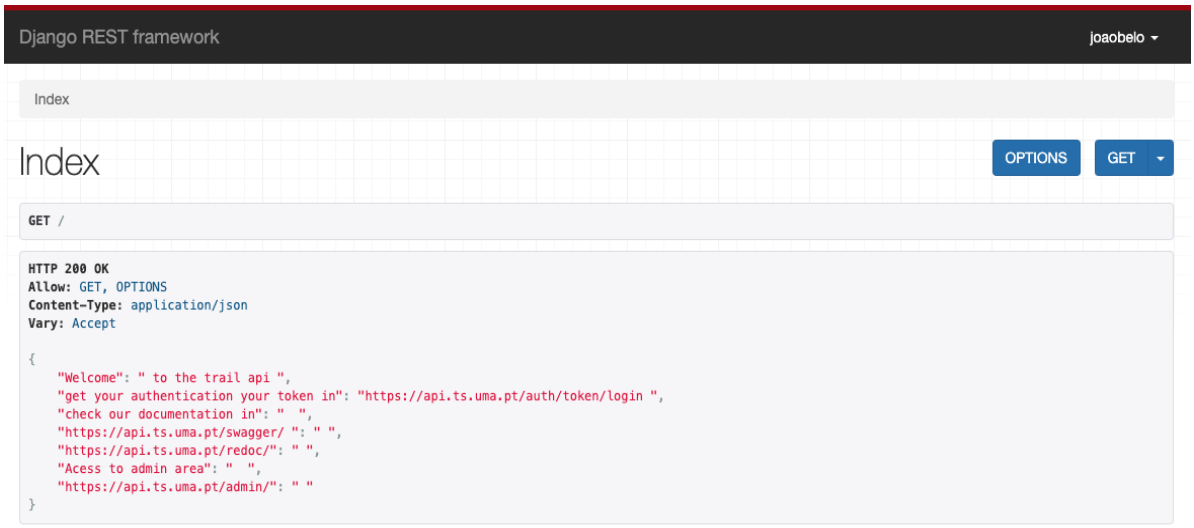


Figura 22 – Página principal da API desenvolvida.

A nível de documentação foi escolhida a utilização do *Redoc* [86] e do *Swagger* [87], devido à grande compatibilidade e integração com a *Django,* facilitando assim o processo de criação da documentação. Na Figura 23 está um exemplo de como o *Swagger* mostra os diferentes pedidos que podem ser efetuados.

Figura 23 – Documentação da API utilizando o *Swagger.*

O *Swagger* indica quais os campos que estão na resposta de um pedido com o respetivo tipo de dados (Figura 24 a). É ainda possível consultar o URL do pedido e efetuar um teste visualizando a resposta da API (Figura 24 b).

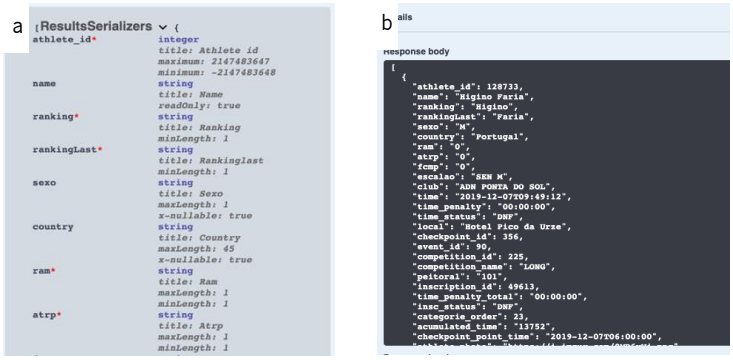


Figura 24 - Lista de campos com respetivos tipo de dados de um determinado pedido. (a) tipo de dados (b) resposta da API.

A *Django* permite também efetuar um pedido utilizando o navegador sem a necessidade de utilizar qualquer ferramenta externa. Caso um utilizador não autenticado tente aceder, o sistema irá pedir ao utilizador a sua autenticação (Figura 25 a), e o mesmo só terá acesso ao conteúdo do seu pedido depois de efetuar a autenticação (Figura 25 b).

Figura 25 - Resposta a um determinado pedido efetuado(a) falha na autenticação (b) autenticação válida.

Associado à *Django* está um *BackOffice* bastante completo que permite efetuar a gestão dos utilizadores e respetivos *tokens* (Figura 26 a), permitindo ainda consultar as ações mais recentes que ocorreram na parte da administração (Figura 26 b).

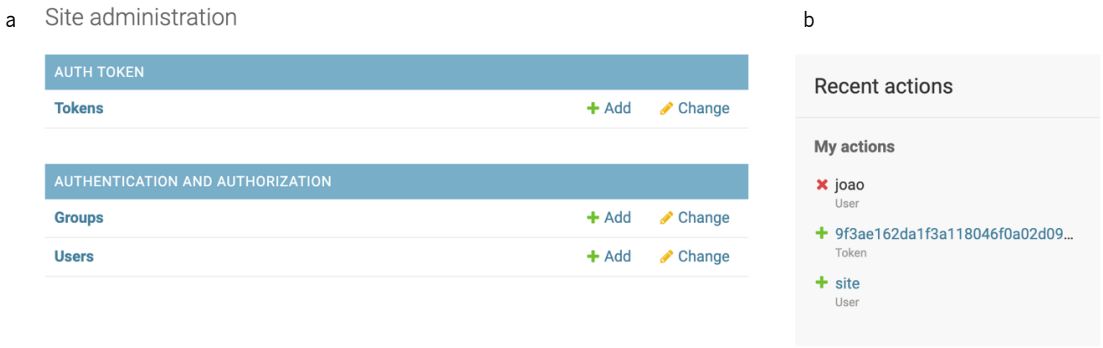


Figura 26 – Página de administração da *Django* (a) funções disponíveis (b) lista atividades recentes.

Observando com mais atenção a parte da administração, vemos que ainda é possível criar um grupo, possibilitando assim restringir o acesso às partes mais sensíveis da API apenas a alguns utilizadores ou aplicações. Outra vantagem do *BackOffice* é que permite facilmente bloquear o acesso de um utilizador, ou de uma aplicação, à API e, por conseguinte, à base de dados.

Por fim, caso um utilizador pretenda consultar ou criar o seu *token*, de uma forma simples e via navegador, sem a necessidade de utilizar nenhuma ferramenta externa, basta aceder ao link presente na página inicial da API e efetuar uma autenticação de segurança (Figura 27).

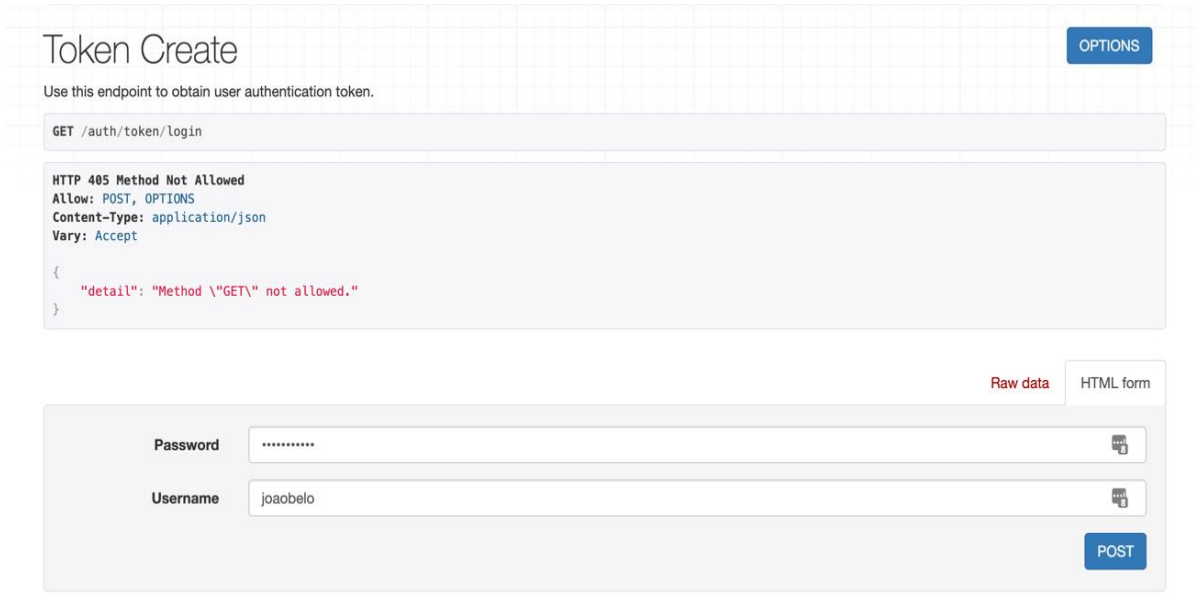


Figura 27 – Página de autenticação para obtenção do *token*

Após completar a autenticação de segurança, o utilizador é automaticamente redirecionado para uma página onde pode obter o seu *token* (Figura 28).

Figura 28 - *Token* de segurança, enviado após autenticação válida.

Em suma, a *Django* provou ser uma *framework* relativamente simples de utilizar e compatível com diversos *plugins*, adaptou-se bem à base de dados existente. Foram definidas regras iniciais, como a nomenclatura, que permite à API crescer ao longo do tempo. Foi escolhido o sistema de autenticação por *token,* mas pode ser facilmente alterado para outro.

5.3 *Front-end*

Atendendo a que um dos principais objetivos do presente projeto é solucionar os problemas de desempenho, escalabilidade e aumentar as funcionalidades na visualização e interação com o público, analisou-se exaustivamente a aplicação atual, chegando-se à conclusão que a mesma sofre de algumas limitações que dificultam a resolução destes problemas.

Começando pela *framework* escolhida, esta está bastante desatualizada e não é fácil atualizar pois imensas funções foram descontinuadas. A falta de documentação dificulta a compreensão do funcionamento de diversos métodos. Posto isto, decidiu-se desenvolver uma nova aplicação utilizando uma nova *framework React* pelas razões apresentadas anteriormente, ou seja, esta é a *framework* mais utilizada no momento, é suportada por uma enorme comunidade de programadores, tem uma vasta escolha de módulos e bibliotecas, além de apresentar um bom desempenho *out of the box*.

Pretendeu-se ainda desenvolver uma aplicação moderna, compatível com diversos dispositivos e com um bom desempenho, tendo-se optado por desenvolver uma *Progessive Web App* (PWA) [88]. As PWA têm a capacidade de serem executadas em várias plataformas, sendo totalmente independentes do sistema operativo. A aplicação não necessita de passar pelos processos de verificação para poder ser disponibilizada numa loja de aplicações, pois é acedida através de um URL. Estas permitem também o funcionamento *offline* pois utilizam *caches* e *service workers,* mas não têm a mesma experiência de utilização nos dispositivos móveis, como as aplicações nativas, pois não são desenvolvidas com o propósito de serem utilizadas naquele sistema operativo em específico. Basicamente, as PWA foram desenvolvidas para serem capazes, confiáveis e instaláveis [89].

Existem atualmente inúmeras PWA bastante conhecidas: Pinterest, Instagram, Google Maps, Twitter, entre outras [90]. Analisando o exemplo do Twitter, este teve um aumento de 65% por sessão, mais de 75% de *tweets* e uma diminuição de 20% na taxa de rejeição, conseguindo ainda reduzir o tamanho da aplicação em mais de 97% com a migração para PWA. Em suma, as PWA permitem que a aplicação seja instalada por qualquer pessoa, em qualquer lugar e em qualquer dispositivo, tudo com a mesma base de código [89].

Iniciou-se o desenvolvimento da nova aplicação pela página principal, tendo como ambição um *layout* simples e fácil de utilizar e, ao mesmo tempo, moderno. Na Figura 29 está apresentado o *layout* desenvolvido [91]: a versão para computador (Figura 29a); *layout* para telemóvel (Figura 29b). É possível visualizar a data e hora do evento, bem como o nome do mesmo. No caso do evento estar a decorrer, em vez da data e hora, será apresentado uma mensagem a informar que o mesmo está a decorrer, facilitando ao utilizador o acesso ao evento a decorrer.

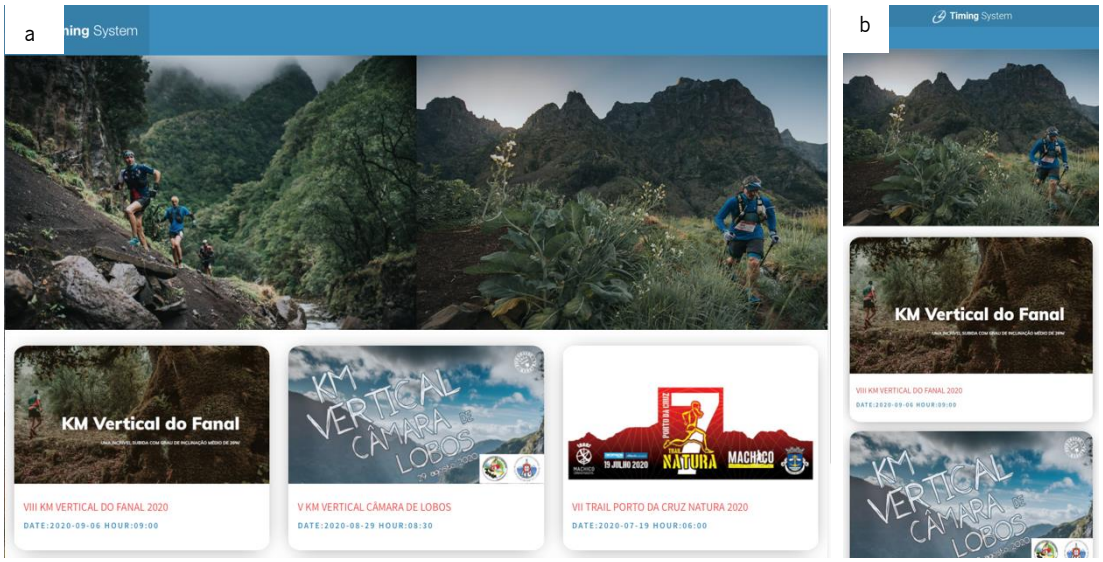


Figura 29 – *Layout* da página principal da aplicação desenvolvida para (a) computador e (b) telemóvel.

Após o utilizador selecionar o evento que deseja consultar, é encaminhado para a página de inscrições (Figura 30). Seguindo o modelo anterior, a versão para computador (Figura 30a) e o *layout* para telemóvel (Figura 30b), podendo-se observar que a página é composta por quatro breves estatísticas. Estas indicam o total de atletas, o total de equipas, o total de escalões ou categorias diferentes e o total de nacionalidades, permitindo assim que o utilizador tenha uma ideia global da dimensão da prova.

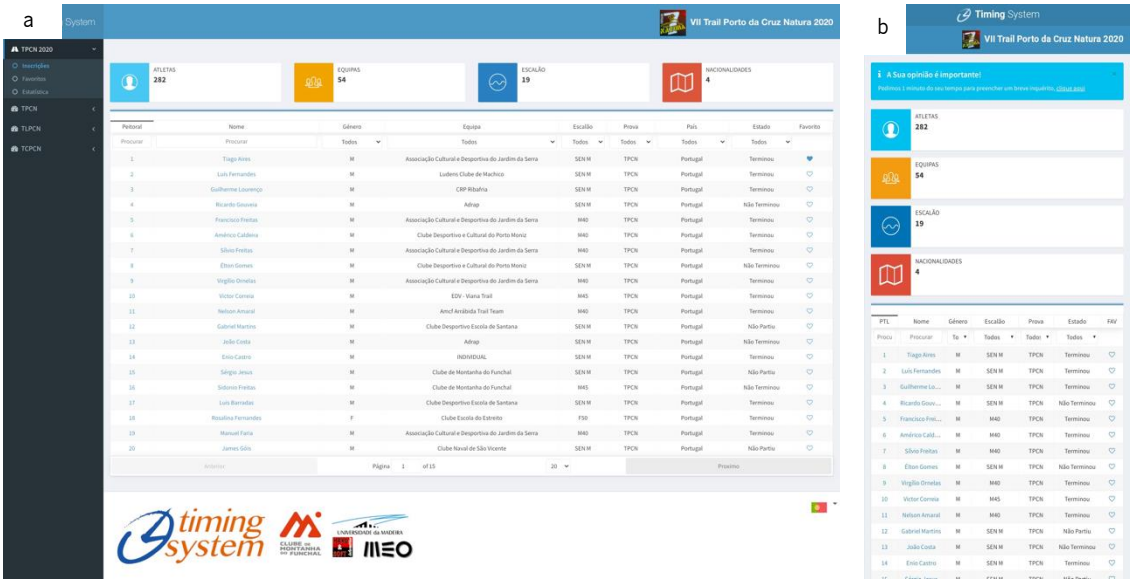


Figura 30 – *Layout* da página com a lista de inscritos da aplicação desenvolvida para (a) computador e (b) telemóvel.

Em seguida, temos a tabela com a lista dos atletas, que possibilita ao utilizador facilmente selecionar o atleta sobre o qual deseja obter mais informação com um simples clique no nome assinalado a azul. Para adicionar um atleta como favorito, o utilizador apenas necessita de clicar no ícone do coração, ficando o mesmo totalmente colorido, informando assim que o atleta favorito foi registado com sucesso.

Observando o menu podemos verificar que ocorreu uma alteração aos protótipos iniciais, mais precisamente, a lista de competições passou a ser parte integrante do menu, apresentando mais opções quando clicada.

Por fim, pode-se observar no rodapé os patrocinadores do evento bem como a opção para mudar o idioma do site, estando o mesmo acessível nos seguintes idiomas: Português, Inglês, Francês e Espanhol, podendo ser facilmente adicionados mais idiomas caso se verifique ser necessário.

Atendendo às suas funcionalidades, umas das páginas mais utilizada durante uma competição é a das passagens (Figura 31), pois esta permite ao utilizador ter uma ideia global de como está a decorrer a prova, incluído a posição dos atletas e os respetivos tempos.

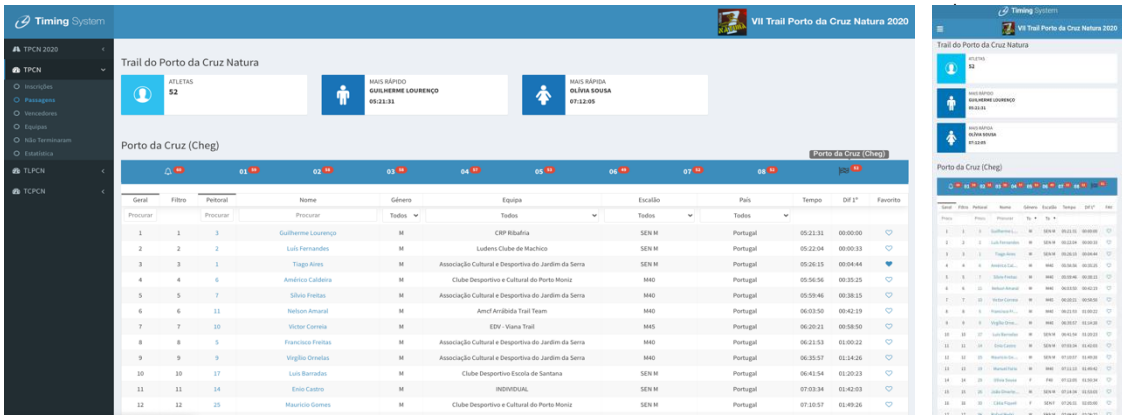


Figura 31 - *Layout* da página das passagens da aplicação desenvolvida para (a) computador e (b) telemóvel.

Nesta página, o utilizador pode selecionar o *checkpoint* que deseja consultar através da barra disponível por cima da tabela. Ao passar o rato por cima do número, o sistema mostra o nome do *checkpoint*. O círculo vermelho acima do número do *checkpoint* indica o total de atletas que já passaram no respetivo ponto, permitindo perceber quantos atletas já desistiram, ou foram desqualificados, desde o início da prova. A lista de participantes encontra-se ordenada, por defeito, pela posição e, no caso de vários atletas terem o mesmo tempo, o sistema ordena-os através do número do peitoral.

É importante referir que, no *layout* móvel, algumas colunas da tabela foram retiradas devido ao tamanho disponível no ecrã. Por fim, assinala-se a existência de algumas estatísticas na parte superior que, além do número de atletas, permite consultar o mais rápido dos masculinos e a mais rápida dos femininos.

Foi ainda decidido adicionar uma página dedicada exclusivamente aos vencedores (os atletas com direito a prémio), Figura 32, permitindo assim, aos utilizadores, uma consulta rápida à lista de vencedores de cada uma das diferentes categorias ou géneros.

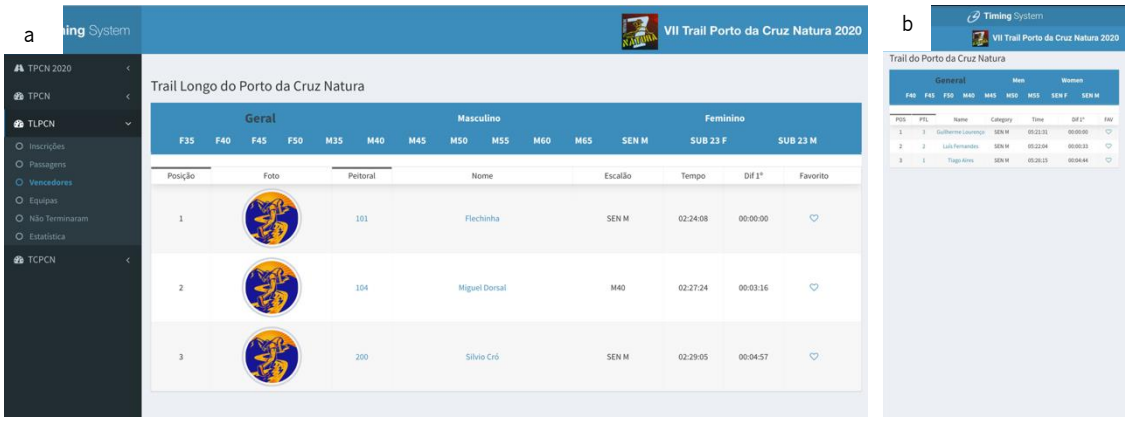


Figura 32 – *Layout* da página dos vencedores da aplicação desenvolvida para (a) computador e (b) telemóvel.

Analisando agora com mais detalhe a Figura 32, podemos verificar que o esquema de navegação é semelhante ao apresentado na Figura 31. Uma barra, na parte superior da tabela, divide-a em duas linhas: na primeira linha, os vencedores globais e os vencedores por género, e, na segunda linha, as diversas categorias. A disposição permite ao utilizador, tanto em dispositivos móveis como num *desktop*, aceder facilmente ao que deseja consultar. Esta página, apesar de parecer simples, tem algumas particularidades: apesar de raro, acontece algumas vezes vários atletas chegarem ao mesmo tempo à meta, ficando com um tempo idêntico, podendo, nestes casos, existir mais de três vencedores.

Na reformulação da aplicação foi ainda adicionada outra página dedicada apenas às equipas, permitindo que os utilizadores acedam à classificação das equipas, que resulta da soma dos tempos dos seus três atletas mais rápidos, separado por género, bem como às informações destes atletas.

Observando a Figura 33, que apresenta o *layout* das equipas, podemos visualizar uma barra superior semelhante à encontrada nas páginas anteriores e que é utilizada para selecionar o género. Em maior destaque, temos a posição e o nome da equipa, seguido do logo e respetivo tempo. A lista de atletas permite que o utilizador facilmente consulte, em detalhe, o atleta que deseja através de um simples clique.

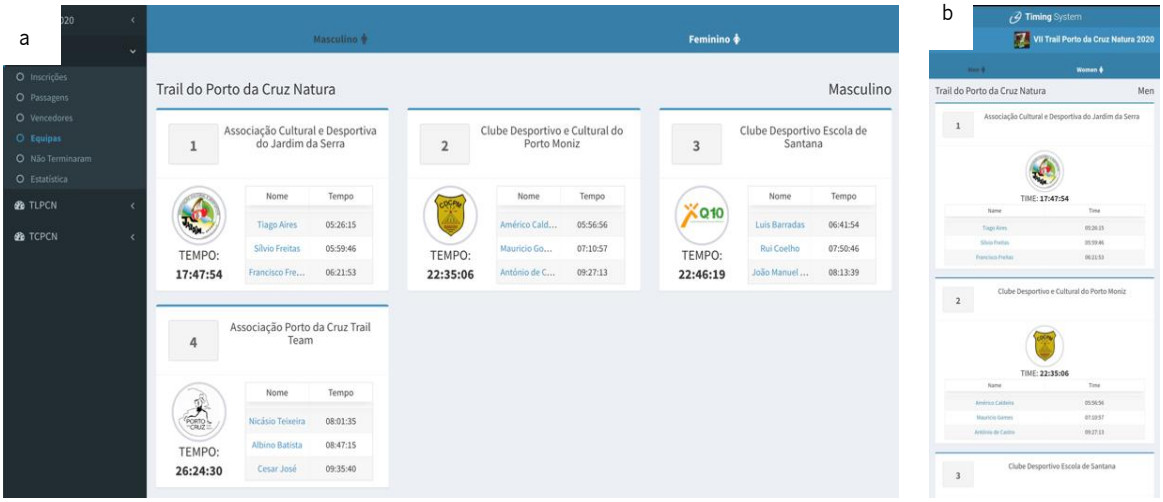


Figura 33 – *Layout* da página das equipas da aplicação desenvolvida para (a) computador e (b) telemóvel.

A página do detalhe do atleta foi pensada para conter toda a informação que seja relevante para o atleta, e para os seus apoiantes. Analisando a Figura 34 podemos verificar a existência de diversas informações. Começando pelas estatísticas iniciais, temos a posição do atleta nos três diferentes tipos (geral, escalão e género), o tempo na prova até ao momento, a velocidade média e o último *checkpoint* do atleta. Temos ainda as informações do atleta e a respetiva tabela com o tempo e velocidade em cada *checkpoint*.

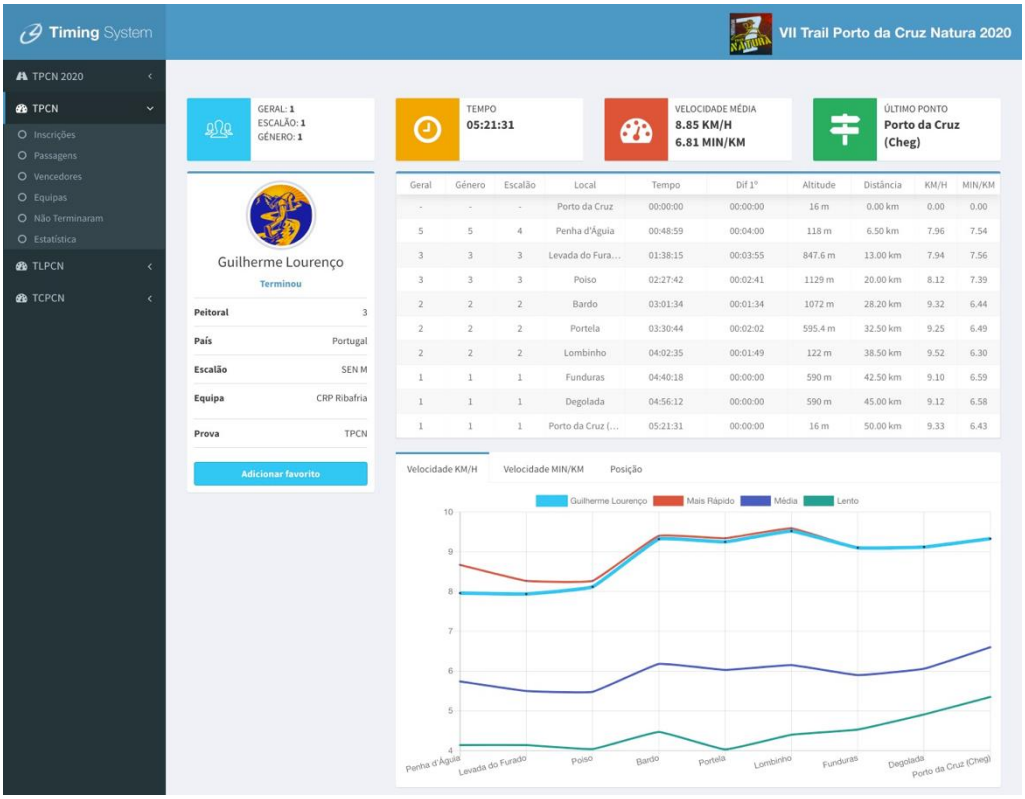


Figura 34 – *Layout* da página do detalhe do atleta da aplicação desenvolvida.

Esta tabela tem uma particularidade, pois em algumas provas existem previsões de tempo para um *checkpoint* ou para vários seguintes. Nesse caso, vão aparecer a azul, com uma informação a piscar na tabela, informando ao utilizador que é apenas uma previsão. As previsões são totalmente independentes desta aplicação. Quanto existem, são ativadas na base de dados num campo criado para o efeito e são apresentadas pela aplicação.

Foi decidido ainda adicionar gráficos de vários parâmetros, podendo o utilizador mudar de gráfico utilizando as abas existentes para o efeito. Nos gráficos de velocidade é possível comparar o atleta em questão com o atleta mais rápido, com o mais lento e com a média global dos atletas, permitindo assim ter uma ideia de como foi a prova do atleta e, sobretudo, identificar em que zonas o mesmo pode melhorar.

Ao longo deste capítulo abordou-se os "Favoritos", existindo a opção de adicionar ou remover atletas favoritos em praticamente todas as páginas. Além de ser uma forma rápida de o utilizador consultar os atletas que mais gosta, permite que o utilizador receba notificações no seu equipamento, caso permita, cada vez que um atleta passa num ponto. Na Figura 35, estão duas notificações recebidas num dispositivo móvel *Android*, referentes a dois atletas favoritos.

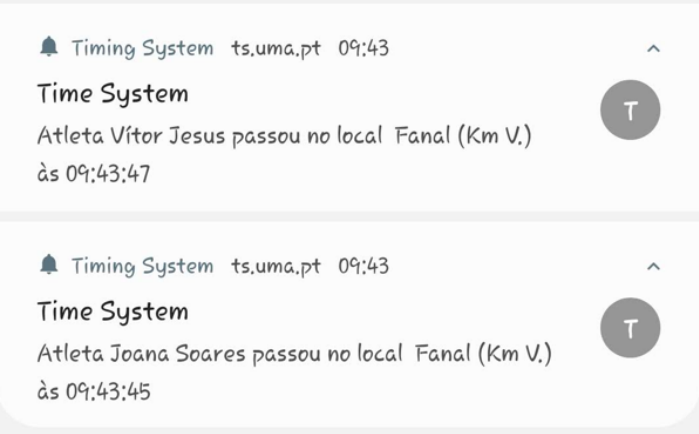


Figura 35 – Notificações da passagem de um atleta favorito num checkpoint.

As notificações funcionam via navegador, tendo compatibilidade com uma grande variedade de dispositivos móveis e *desktops*, sendo que praticamente todos os navegadores modernos suportam notificações. Como os favoritos utilizam o ID do atleta, as notificações são transversais a todas as provas, desde que o ID seja o mesmo. O utilizador pode gerir os seus favoritos através da Página de Favoritos (Figura 36) podendo facilmente remover um favorito, cancelando as notificações para aquele atleta.

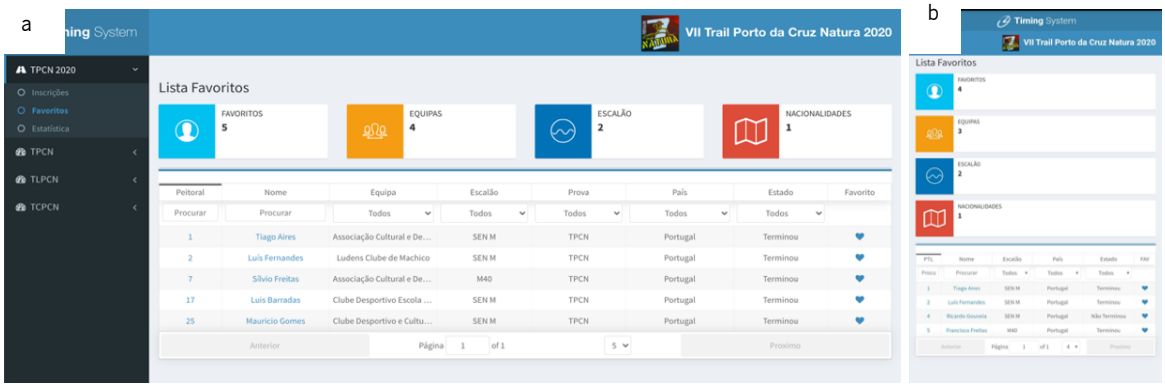


Figura 36 - *Layout* da página dos atletas favoritos da aplicação desenvolvida para (a) computador e (b) telemóvel.

Quanto às estatísticas, escolheu-se dividir em dois tipos: estatísticas relativas ao evento e estatísticas relativas às provas ou competições. Começando pelas estatísticas do evento (Figura 37), é possível consultar o total de atletas por escalão, género e por nacionalidade. Por fim, é possível consultar a quantidade de atletas nas respetivas competições.

Figura 37 – *Layout* da página das estatísticas relacionadas com um evento. (a) escalão dos atletas (b) género (c) nacionalidade (d) tipo de prova ou competição.

Para a estatística das competições (Figura 38) foi escolhido utilizar no escalão, género e nacionalidade um gráfico com 3 barras por categoria. As barras dizem respeito aos atletas inscritos, partidas e chegadas, e permitem ao utilizador ter uma ideia global de quantos desistiram ou foram desqualificados ao longo da prova. Tem-se ainda um gráfico com o total de atletas por *checkpoint* e, por fim, o gráfico da altitude de cada ponto, que permite ao utilizador perceber o percurso que o atleta tem de fazer.

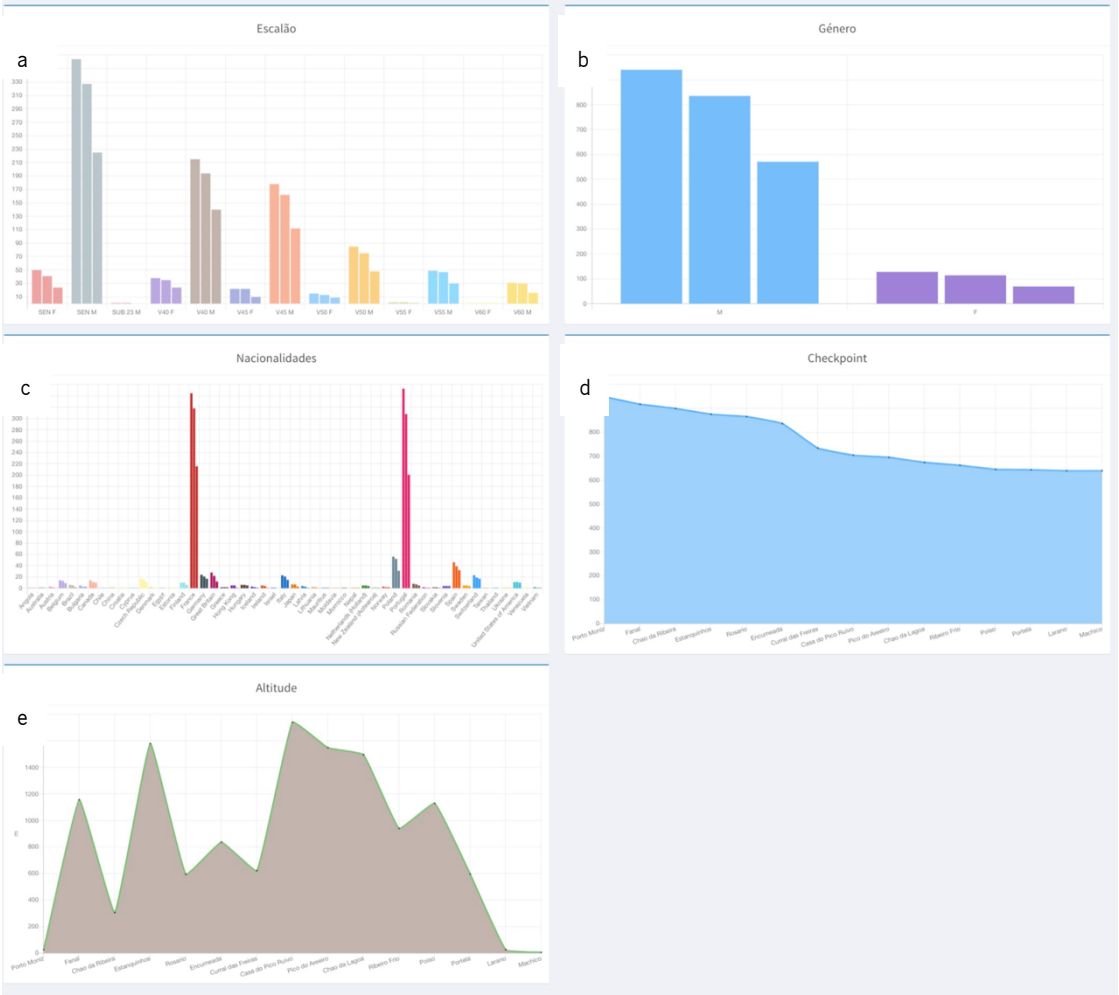


Figura 38 - *Layout* da página das estatísticas relacionadas com uma prova ou competição (a) escalão dos atletas (b) género (c) nacionalidade (d) número de atletas por *checkpoint* (e) relevo do terreno.

Em suma, a escolha inicial de desenvolver uma PWA, em vez de um *website*, provou, do ponto de vista pessoal, ser a escolha acertada pois permite que o utilizador instale o *site* como uma aplicação numa grande variedade de dispositivos. Para instalar como aplicação, o utilizador apenas tem de aceder uma vez ao URL e clicar na respetiva opção do navegador (esta opção varia de navegador para navegador, aparecendo no *Chrome* como uma mensagem e como um botão no *Firefox*). Depois de instalada, o utilizador acede e utiliza a PWA, como se se tratasse de uma aplicação normal.

No início do desenvolvimento do projeto, o foco estava sobretudo nos *desktops* ou *laptops*. Porém, após acompanhar algumas provas, chegou-se à conclusão que tinham de ser incluídos os dispositivos móveis (*smartphones* e/ou *tablets*) pois a maior parte do público, bem como a organização, utiliza estes dispositivos. Foram, portanto, feitas diversas modificações de modo a que fosse possível acompanhar a prova nos dispositivos móveis. Ao longo deste capítulo, apresentou-se tanto o *layout* para *desktop* como para telemóveis, devendo-se salientar que é possível efetuar *zoom* através do gesto para o efeito (*pinch*) em qualquer parte da aplicação, o que facilita a visualização dos dados por parte do utilizador.

A escolha da *React,* com todos os seus componentes e compatibilidades, facilitou o processo de implementação, permitindo criar uma aplicação simples e moderna compatível com os diversos sistemas atuais.

5.4 *Conclusão*

A escolha da *Django* revelou-se uma escolha acertada pois simplificou o desenvolvimento da API e permitiu poupar tempo. As ferramentas de administração da *Django* simplificaram imenso todo o processo de criação, gestão de utilizadores e autorizações. A grande compatibilidade com o *Swagger* facilitou todo o processo de documentação. Sentiram-se algumas dificuldades na utilização das *querysets* (*queries* SQL efetuadas na *Django*), principalmente nas mais complexas que envolviam várias tabelas da base de dados e alguns filtros. Existiram também algumas limitações da parte da *Django* com alguns tipos de dados, que dificultaram um pouco o processo de desenvolvimento, nomeadamente, a existência na base de dados de campos do tipo *Time* no formato (HH:MM: SS), sendo que para a *Django* as horas não podem ser superiores a 24h (mas podem, no entanto, existir provas com duração total de mais de um dia). Como não é possível alterar a base de dados, devido às diversas aplicações que existem em funcionamento, foi necessário criar algumas *views* com o tempo convertido em segundos, de forma a ultrapassar esta limitação.

Quanto à escolha da *framework React*, considera-se que foi uma boa escolha, apesar da curva de aprendizagem inicial. A grande variedade de componentes compatíveis com a *React* facilitou o desenvolvimento, permitindo poupar algum tempo apesar de se terem encontrado algumas limitações.

Outra grande dificuldade sentida, devido à falta de documentação por parte da *React*, foi o funcionamento das *caches* e notificações.

O desempenho foi o principal foco deste projeto, pois foram utilizadas várias técnicas para obter um desempenho satisfatório, sendo a principal a utilização de *caches* em várias páginas da aplicação. Após aceder a uma página pela primeira vez, a informação é armazenada numa *cache*, neste caso no armazenamento do navegador e, de cada vez que o utilizador aceder de novo a essa página, o sistema mostra automaticamente a informação guardada na *cache* (não existindo praticamente nenhum tempo de carregamento). Considerando que a informação fica obsoleta ao longo do tempo, principalmente numa aplicação deste tipo, as *caches* necessitam de ser constantemente atualizadas, podendo ser apresentada informação antiga por breves segundos, enquanto a *cache* é atualizada.

Capítulo 6

Testes e Resultados

Este capítulo está organizado em duas secções: testes de desempenho (6.1) e inquérito aos utilizadores (6.2) e respetivas conclusões (6.3). Começa-se por explicar os testes de desempenho efetuados a ambas as aplicações, e os respetivos resultados, passando-se, de seguida, para o inquérito a que os utilizadores da aplicação responderam, e os respetivos resultados.

6.1 *Testes*

Foi efetuada uma análise de desempenho por testes de carga e *stress*, permitindo perceber a capacidade de resposta e estabilidade de um sistema sob uma determinada carga. Esta permite igualmente investigar e validar determinados atributos do sistema. No presente projeto foram testadas ambas as aplicações: a aplicação pública desenvolvida no decorrer deste projeto e a aplicação pública atual, permitindo comparar o desempenho e estabilidade das mesmas.

Foi elaborado um plano de testes, tendo-se selecionado o evento *Madeira Island Ultra Trail 2019*, pois foi o maior evento do ano 2019 (contou com 2724 atletas). Selecionaram-se as páginas de inscritos, passagens e vencedores pois estas, além de conterem informação importante, são transversais a ambas as aplicações. Os testes foram efetuados numa máquina dentro da rede da Universidade da Madeira, eliminando assim diversos problemas da rede com o exterior, reconhecidos pelos administradores de rede. Foi utilizada a ferramenta *jmeter*, desenvolvida em *java,* que permite medir diversas variáveis e que tem como principal foco aplicações *web.* Esta ferramenta efetua testes de carga e de *stress*. O *jmeter* permite configurar diversos parâmetros, sendo que, para os testes em questão, foram configurados os seguintes:

- Tarefas: utilizadas para simular um utilizador, em que quantas mais tarefas, mais utilizadores.
- *Ramp-up*: tempo em segundos necessário para o sistema criar todas as tarefas.
- *Loop Count*: tempo em segundos de execução de cada tarefa.

Tendo em conta que o desempenho é um ponto importante deste projeto escolheu-se efetuar testes de *stress*. Estes testes de *stress* podem ser vistos individualmente como testes de carga pois permitem analisar se a aplicação consegue dar, ou não, resposta a uma determina carga. Após pesquisa e análise da literatura [29][39][92][93], escolheu-se utilizar as configurações para os testes presentes na Tabela IV. Estas condições foram idênticas para ambas as aplicações e permitem simular diversas condições de utilização.

Tabela IV – Configurações do programa utilizado (jmeter) nos testes a ambas as aplicações.

Nº Teste	Tarefas	*Ramp-up*	*Loop Count*
1	500	20	15
2	1000	15	5
3	2000	20	60
4	4000	20	60
5	6000	20	60
6	10000	30	60

Por exemplo, no caso do teste número 3 (Tabela IV), serão criadas 2000 tarefas num período de 20 segundos, o que significa que serão criados 100 utilizadores por segundo. Neste exemplo, o *Loop Count* é de 60 segundos para cada tarefa, ou seja, cada tarefa irá ficar ativa durante 60 segundos (simula a consulta do utilizador a uma página durante 60 segundos). Caso seja desejado intensificar mais a carga ao servidor pode-se aumentar o número de tarefas, bem como reduzir o *Ramp-up*, simulando a chegada de utilizadores mais rapidamente ao *site*. Na Figura 39 está representada a arquitetura dos testes. Como mencionado anteriormente, os testes serão efetuados na rede interna, utilizando uma máquina específica.

Figura 39 - Arquitetura dos testes à aplicação.

Após a realização dos testes, o *jmeter* devolve diversos parâmetros para comparação entre aplicações, tendo sido escolhidos os seguintes:

- Média: tempo médio gasto por todas as tarefas em milissegundos [94].
- 90% *line*: valor abaixo do qual 90% das tarefas ficam; é uma medida estatística padrão em milissegundos [94].
- Min: é o menor tempo de resposta em milissegundos [94].
- Max: é o maior tempo de resposta em milissegundos [94].
- % Erro: é a percentagem de tarefas que falharam [94].
- *Throughput*: solicitações por segundo com que o servidor consegue lidar em segundos [94].

Note-se que o *hardware* da atual aplicação é diferente do *hardware* da nova aplicação. Além disso, a nova aplicação está em *containers* e a atual não, mas a base de dados é partilhada entre ambas as aplicações.

Isto dificulta a comparação, no entanto, devido à dificuldade em migrar o sistema atual, não foi possível comparar as aplicações em máquinas idênticas. Mantendo-se as aplicações nos sistemas reais em que são utilizados, é possível obter uma visão real do desempenho de cada uma.

Na Tabela V estão os resultados obtidos no teste à aplicação atual e, na Tabela VI, os resultados para a aplicação desenvolvida no projeto. Comparando os resultados verifica-se que os tempos médios gastos pelas tarefas são inferiores em todos os testes na nova aplicação. Refira-se, no entanto, que apesar das médias serem mais baixas na nova aplicação, os tempos são aceitáveis em ambas.

Começa-se a notar diferenças consideráveis quando se analisa o tempo máximo de espera em todos os testes, com os tempos da aplicação atual a serem consideravelmente superiores aos tempos da nova aplicação. A nível da percentagem dos erros, a aplicação atual começa a apresentar erros a partir do teste 4, subindo consideravelmente a percentagem até ao teste 6 (a percentagem de erro é superior a 50% no teste 6, o que não é de todo aceitável). No caso da nova aplicação, esta só começa a apresentar erros no teste 6, sendo a percentagem de erro inferior a 2%. Por fim, o *throughput* da nova aplicação é superior em todos os testes, sendo que, em alguns testes, consegue ser mais de cinco vezes superior. Isto significa que a nova aplicação consegue responder a uma quantidade superior de solicitações por segundo em comparação à atual.

Tabela V - Testes à aplicação atual, usando o programa *jmeter*. Foram avaliados os parâmetros média, 90%line, mínimo, máximo, percentagem de erro e o *throughput*.

Nº Teste	Média	90% *line*	Min	Max	% Erro	*Throughput*
Página de inscritos						
1	4	1	0	921	0	164
2	21	13	0	1501	0	192
3	25	17	0	3533	0	88
4	44	23	0	8378	2,3	193
5	244	27	0	33802	24,1	303
6	4953	325	0	111421	55,2	12
Página das passagens						
1	8	2	0	1253	0	101
2	22	5	0	1700	0	145
3	24	9	0	2630	0	72
4	40	15	0	9563	0	225
5	352	25	0	34782	5,1	228
6	7500	510	0	165242	42,2	25
Página dos vencedores						
1	2	1	0	921	0	235
2	18	2	0	1501	0	165
3	19	37	0	5533	0	423
4	35	39	0	7252	0	325
5	120	45	0	18522	5,6	235
6	3265	95	0	238511	25,2	45

Tabela VI - Testes à nova aplicação, usando o programa *jmeter*. Foram avaliados os parâmetros média, 90%line, mínimo, máximo, percentagem de erro e o *throughput*.

Nº Teste	Média	90% *line*	Min	Max	Error %	*Throughput*
Página de inscritos						
1	0	1	0	39	0	378
2	0	1	0	77	0	206
3	16	14	0	1485	0	403
4	40	20	0	2587	0	1343
5	91	21	0	11140	0	428
6	3459	242	0	23128	1,7	174
Página das passagens						
1	0	1	0	65	0	380
2	1	1	0	86	0	470
3	13	10	0	1506	0	329
4	100	34	0	3405	0	1125
5	215	36	0	16572	0	267
6	4763	423	0	13344	2	145
Página dos vencedores						
1	0	1	0	46	0	379
2	0	1	0	101	0	240
3	4	23	0	4391	0	999
4	4	23	0	4391	0	772
5	4	24	0	14078	0	499
6	12	70	0	18682	1,1	104

Considerando os resultados anteriormente apresentados, podemos concluir que a aplicação desenvolvida durante este trabalho consegue suportar um maior número de utilizadores em simultâneo, conseguindo processar um número superior de pedidos por segundo, e respondendo assim aos pedidos mais rapidamente.

6.2 *Inquéritos aos utilizadores*

A opinião e satisfação dos utilizadores é uma das principais preocupações pois a aplicação foi desenvolvida a pensar nos mesmos. Foi colocada uma mensagem na aplicação a pedir a todos os utilizadores que respondessem a um breve inquérito. Este pedido permaneceu *online* durante três eventos: "*Trail* do Porto da Cruz Natura 2020", o "V KM Vertical de Câmara de Lobos" e o "VIII KM Vertical do Fanal 2020". Estes três eventos, não obstante, normalmente já serem de uma dimensão reduzida, devido à situação pandémica COVID-19 viram a sua dimensão ser ainda menor. Apesar das limitações, foram obtidas 93 respostas, podendo-se consultar os resultados em bruto no Anexo 5.

As perguntas colocadas aos utilizadores da aplicação foram as seguintes:

1. Qual a sua faixa etária?
   R: Menos de 18, 19-25, 26-30, 31-40, 41-50, 50+
2. O que achou da aparência e funcionalidades gerais?
   R: Escala de 1 a 5
3. O que achou da velocidade em que os dados são apresentados?
   R: Escala de 1 a 5
4. Foi fácil de aceder aos dados que desejava?
   R: Escala de 1 a 5
5. Achou útil a opção de adicionar favoritos?
   R: Escala de 1 a 5
6. Recomendaria a aplicação a um familiar ou colega?
   R: Escala de 1 a 5

O questionário pretendeu ser o mais simples e rápido possível, apenas questionando o utilizador em pontos importantes do sistema para posterior análise. A escala utilizada nas respostas foi a *Escala de Likert* [95]. De seguida apresentam-se as respostas e respetiva análise dos resultados.

Questão 1: Qual a sua idade?

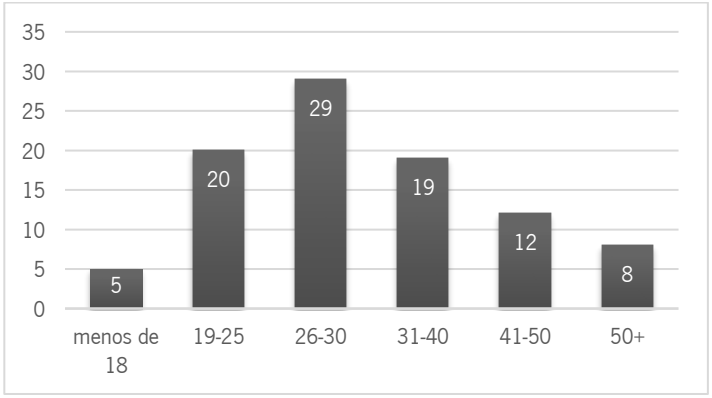


Figura 40 – Resultados à questão 1 do inquérito: "Qual a sua idade?". As opções de resposta eram "menos de 18", "19-25", "26-30", "31-40", "41-50" e "50+".

Analisando as respostas à questão "Qual a sua idade", presentes na Figura 40, podemos verificar que a aplicação foi utilizada por pessoas de todas as faixas etárias. A maioria dos respondentes encontrava-se entre os 19-40 anos. Este tipo de pergunta permite perceber a faixa etária dos principais utilizadores, adequando assim a aplicação aos mesmos, e ainda comparar com os participantes do evento, percebendo assim se os atletas utilizam a aplicação depois da prova. Analisando a amostra, podemos verificar que o maior número de utilizadores se encontra entre os 19 e os 40 anos, sendo igualmente estes os escalões com mais participantes.

Questão 2: O que achou da aparência e funcionalidades gerais?

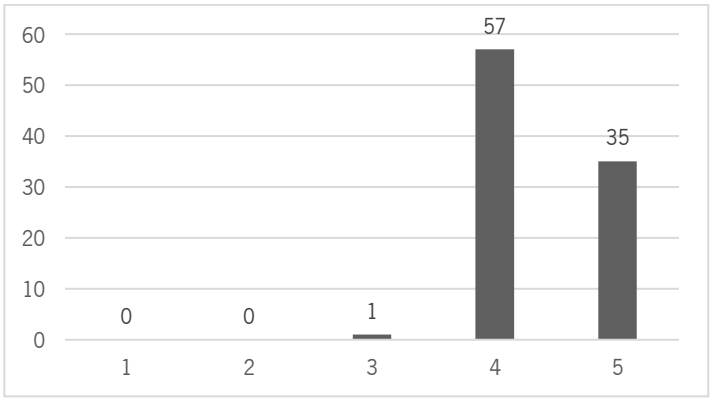


Figura 41 - Resultados à questão 2 do inquérito: "O que achou da aparência e funcionalidades gerais?". As opções de resposta eram "1", "2", "3", "4" e "5".

Para a questão "O que achou da aparência e funcionalidades gerais?", e analisando os resultados presentes na Figura 41, verifica-se que a maioria atribuiu uma pontuação de 4 valores. Apenas um utilizador classificou com 3 valores e 35 utilizadores atribuíram uma pontuação de 5. Estas avaliações são bastante positivas, tendo em conta a grande mudança introduzida no esquema de navegação.

Questão 3: O que achou da velocidade em que os dados são apresentados?

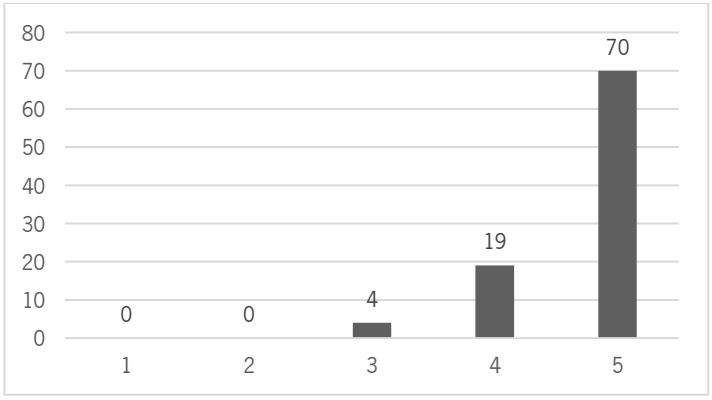


Figura 42 - Resultados à questão 3 do inquérito: "O que achou da velocidade em que os dados são apresentados?". As opções de resposta eram "1", "2", "3", "4" e "5".

A Figura 42 agrupa as respostas à pergunta "O que achou da velocidade em que os dados são apresentados?", onde a maioria atribuiu uma pontuação de 5 valores à velocidade a que o sistema apresenta os dados. Destaca-se o facto de apenas quatro utilizadores apresentaram uma classificação de 3 pontos. Os utilizadores em questão deram a sua classificação durante o início do primeiro evento, sendo que o sistema estava bastante instável devido a algumas alterações que estavam a ser efetuadas. Considerando que a velocidade é um dos pontos principais, os resultados obtidos são extremamente satisfatórios, pois mais de 95% dos inquiridos avaliou numa pontuação igual ou superior a 4 a velocidade a que os dados são apresentados.

Questão 4: Foi fácil de aceder aos dados que desejava?

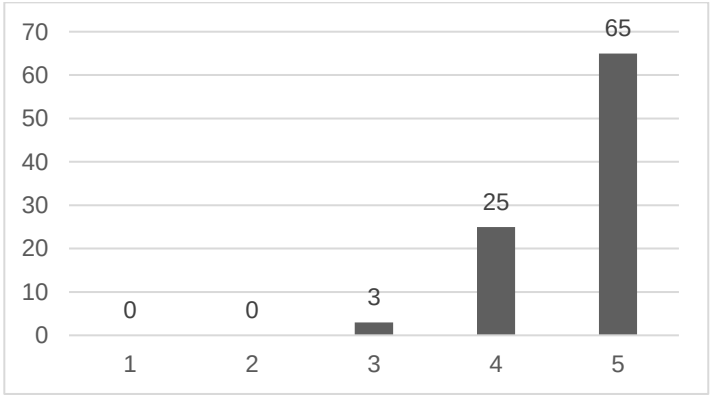


Figura 43- Resultados à questão 4 do inquérito: "Foi fácil de aceder aos dados que desejava?". As opções de resposta eram "1", "2", "3", "4" e "5".

Quando questionados acerca da facilidade a aceder aos dados pretendidos, Figura 43, verifica-se que a maioria dos inquiridos atribuiu uma pontuação de 5 valores, indicando que o esquema de navegação desenvolvido foi aceite e considerado simples de utilizar, o que significa que um dos objetivos do projeto foi alcançado com sucesso.

Questão 5: Achou útil a opção de adicionar favoritos?

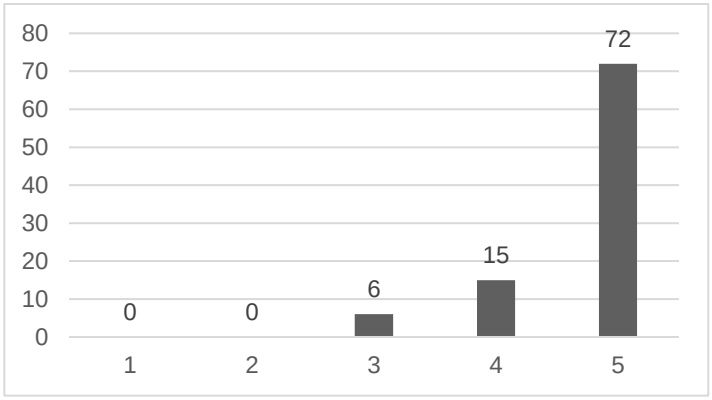


Figura 44 - Resultados à questão 5 do inquérito: "Achou útil a opção de adicionar favoritos?". As opções de resposta eram "1", "2", "3", "4" e "5".

Analisando os resultados referentes à funcionalidade de adicionar Favoritos demonstrados na Figura 44, constata-se que a maioria dos utilizadores achou útil a opção de adicionar atletas favoritos. Atendendo a que esta foi uma funcionalidade introduzida durante este projeto, a opinião dos utilizadores reveste-se de uma importância fulcral para continuar a desenvolver e aperfeiçoar esta funcionalidade.

Questão 6: Recomendaria a aplicação a um familiar ou colega?

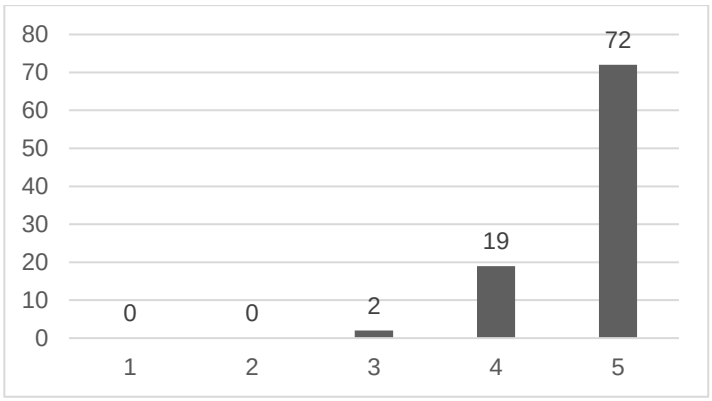


Figura 45 - Resultados à questão 6 do inquérito: "Recomendaria a aplicação a um familiar ou colega?" As opções de resposta eram "1", "2", "3", "4" e "5".

A última questão pretendia avaliar qual a possibilidade do utilizador recomendar a aplicação a familiares e colegas. As respostas encontram-se agrupadas na Figura 45, sendo claro que os utilizadores não hesitariam em sugerir a aplicação a conhecidos (97% dos utilizadores recomendaria a aplicação, considerando-se as respostas pontuadas em 4 e 5). Esta é uma pergunta importante pois permite obter a opinião geral do sistema.

Pode-se concluir que, apesar da situação de contingência que se vivenciou e que afetou o número de eventos e consequentemente o número de utilizadores da aplicação, obtiveram-se 93 respostas válidas que suportam e validam a nova aplicação. Apesar da existência de algumas respostas com classificação 3, a maior parte dos utilizadores gosta da aplicação, das novas funcionalidades e acha que os dados são apresentados de uma forma rápida, corroborando a implementação da mesma.

6.3 *Conclusão*

Neste capítulo começa-se por explicar os testes de *stress* e de carga efetuados, utilizando a ferramenta *jmeter*, bem como as configurações utilizadas permitindo que qualquer pessoa a qualquer altura possa executar os testes e comparar resultados. Analisando os resultados obtidos, podemos concluir que a aplicação desenvolvida lida com uma carga superior relativamente à aplicação antiga tendo suportado todas as combinações sem praticamente erros, apenas apresentando uma pequena percentagem de erros na combinação 6.

Comparando os resultados obtidos nos testes acima apresentados com os requisitos não funcionais apresentados na secção 4.2 podemos concluir que o sistema compre os seguintes requisitos não funcionais:

- O sistema deverá ser rápido;
- O sistema deverá ser consistente;
- O sistema deverá estar disponível;
- O sistema deverá ter flexibilidade;
- O sistema deverá ser confiável;
- O sistema deverá ser robusto;
- O sistema deverá ser tolerante a falhas;

Relativamente aos resultados dos inquéritos efetuados aos utilizadores, obtiveram-se 93 respostas. Tendo em conta a situação atual e o número de participantes nos eventos, pode-se concluir que se obteve uma participação satisfatória nos inquéritos. A maior parte das respostas situa-se entre os 4 e 5 valores, apesar da existência de algumas com classificação 3 (embora sejam são residuais). Podemos assim concluir que a maior parte dos utilizadores gostou da aplicação.

Capítulo 7

Conclusões e Trabalho Futuro

Este capítulo está organizado em duas secções, as conclusões (7.1) e o trabalho futuro (7.2). Na primeira, apresentam-se as principais conclusões do trabalho realizado, e na segunda, exploram-se as perspetivas e trabalhos futuros que podem ser desenvolvidos.

7.1 *Conclusões*

O desenvolvimento deste projeto acabou por ser mais complexo do que inicialmente se esperava, devido às particularidades e à complexidade em desenvolver, simultaneamente, duas aplicações (aplicação pública e API), em linguagens diferentes (*Python JavaScript*) e com fins diferentes, sem poder efetuar grandes modificações na base de dados previamente existente.

Inicialmente, quanto ao objetivo de modificar a arquitetura, de forma a interligar as diversas aplicações, concluiu-se (após várias reuniões) que era imperativo o desenvolvimento de uma API. Sem esta, não era humanamente possível garantir a segurança e a integridade dos diversos sistemas e aplicações previamente existentes.

A API é responsável por fornecer mecanismos de autenticação que permitem identificar qual a aplicação ou utilizador que está a aceder aos dados e, rapidamente, bloquear acessos indevidos. Esta providencia ainda um mecanismo uniforme de acesso aos dados, permitindo ao utilizador aceder rapidamente ao que deseja. Por outro lado, a aplicação pública permite que qualquer utilizador, sem nenhum tipo de autenticação, mas com interesse no evento, consiga acompanhar a prova, em tempo real, bem como consultar eventos passados. A aplicação foi ainda desenvolvida a pensar no desempenho e escalabilidade, sendo este um dos principais objetivos conforme referido anteriormente.

Quanto ao objetivo de adicionar funcionalidades de interação com o público, foram implementadas novas funcionalidades, como os "Favoritos" e, por consequência, as notificações. Esta opção permite que o utilizador seja informado sempre que um dos seus atletas favoritos passa num *checkpoint*. Estas notificações também provaram ser significativamente úteis quando se pretende passar uma informação rapidamente para o público em geral. O formato geral de apresentação dos dados na aplicação foi pensado para que o público consiga rapidamente aceder à informação que deseja. No entanto, salienta-se ainda que o lado do atleta não foi esquecido, na medida em que a aplicação permite que o mesmo tenha noção de como foi a sua prova, comparativamente aos restantes.

O levantamento dos requisitos e a elaboração dos diagramas antes do desenvolvimento das aplicações acabaram por revelar-se extremamente úteis durante a implementação das mesmas, servindo como um guia detalhado sobre como a mesma deve funcionar. Decorrente do trabalho de pesquisa, identificaram-se as *frameworks* mais atuais e adequadas, que foram posteriormente utilizadas no desenvolvimento das aplicações. Foram ainda utilizadas diversas técnicas relacionadas com o desempenho, como as *caches*, o posicionamento dos css e *scripts* conforme descrito na secção 2.2.

Após a finalização do processo de desenvolvimento, as aplicações foram testadas num evento de *trail* realizado no Porto da Cruz, tendo sido efetuadas algumas alterações com base no *feedback* dos utilizadores (como por exemplo, a adição do filtro "Género"). Para validar o correto funcionamento da aplicação, foi ainda efetuado um conjunto de testes de desempenho, simulando diferentes números de

utilizadores em simultâneo. Os resultados demostraram que a aplicação lida eficazmente com diferentes números de utilizadores atingindo assim o objetivo de melhorar o desempenho.

Por fim, solicitou-se aos utilizadores que respondessem a um breve questionário, onde 97% dos mesmos indicaram que recomendariam a utilização da aplicação a familiares, amigos e colegas. Dos inquiridos, 95% consideraram a aplicação rápida a apresentar os dados e 93% avaliaram como útil a funcionalidade de adicionar atletas aos favoritos.

Analisando globalmente os resultados, é possível concluir que a aplicação cumpre com os objetivos inicialmente propostos, superando não só as expectativas a nível de desempenho, pois não só sendo rápida, mas sendo confiável e tolerante a falhas, lindado eficazmente com um grande número de pedidos. Como também a nível da usabilidade, interação e aceitação por parte do utilizador final.

Analisado os dados provenientes do *google analytics* [96], face ao bom número de utilizadores (+900), sessões (+2500) e páginas visitadas (+17000) e o bom desempenho demostrado pela aplicação, bem como uma excelente taxa de satisfação, reforça-se que a aplicação atingiu todos os objetivos.

7.2 *Trabalho futuro*

Algumas perspetivas futuras podem ser traçadas após a conclusão do projeto. Relativamente à API, dever-se-á adicionar *caches* a todas as consultas, evitando que a API esteja a aceder constantemente à base de dados. Deve-se ainda ter em atenção o tempo de validade das *caches*, pois, alguns dados são atualizados mais rapidamente que outros.

Já na aplicação pública deverão ser adicionadas *caches* nas restantes consultas, sendo útil ser acrescentada uma opção que possibilite ao atleta adicionar a sua foto (ou, alternativamente, contemplar essa opção nas inscrições). Para as situações em que um atleta é desqualificado deverá ser adicionado o motivo da desqualificação, por exemplo, o atleta cortou caminho num determinado ponto. Seria igualmente oportuno adicionar algumas funcionalidades mais avançadas, como um mapa com a função de acompanhamento em tempo real do atleta bem como a transmissão de vídeo em tempo real de câmaras em cada *checkpoint*.

Além dos princípios de desenho gráfico já aplicados deve-se reforçar e rever alguns métodos utilizados como o alinhamento do texto, cor, tipográfica e escrita de mensagens. Bem como aplicar um inquérito *System Usability Scale* [97] de forma a comparar a usabilidade do sistema interativo. Outro ponto que deve ser revisto é a base de dados atual, já que a mesma sofre de alguns problemas que devem ser corrigidos, juntamente com a migração das aplicações existentes para a API desenvolvida.

Por fim, salienta-se a preparação de um artigo científico, contendo todo o trabalho aqui apresentado e que será submetido a uma revista da especialidade.

Referências

[1] «Ranking ITRA - carreras de ultrafondo». https://www.ultratrails.com/ultras-internacionales/ranking-itra/ (acedido Jun. 08, 2020).

[2] «Automatic Identification and Data Collection (AIDC)». https://www.mhi.org/fundamentals/automatic-identification (acedido Jun. 08, 2020).

[3] V. Camacho e E. Marques, «Análise e Visualização de Resultados de Eventos Desportivos», Universidade da Madeira, 2016.

[4] «Página principal MIUT». https://www.miutmadeira.com/pt/ (acedido Jun. 08, 2020).

[5] «Atletas de 52 nacionalidades presentes no MIUT 2019 - Desporto - RTP Madeira - RTP». https://www.rtp.pt/madeira/desporto/atletas-de-52-nacionalidades-presentes-no-miut-2019_23358 (acedido Out. 15, 2020).

[6] «Recorde de inscrições para o MIUT de 2019». https://www.jm-madeira.pt/desporto/ver/46088/Recorde_de_inscricoes_para_o_MIUT_de_2019 (acedido Jun. 08, 2020).

[7] L. Liu, *Encyclopedia of Database Systems*. New York: Springer, 2016, doi:10.1007/978-1-4899-7993-3.

[8] R. Doshi, «I Think, Therefore I Am?», *Bluesci*, pp. 16–17, Jan. 2010, ISSN:1748-6920.

[9] J. C. Bolot e P. Hoschka, «Performance engineering of the World Wide Web: Application to dimensioning and cache design», *Comput. Networks ISDN Syst.*, vol. 28, n. 7–11, pp. 1397–1405, 1996, doi: 10.1016/0169-7552(96)00073-6.

[10] N. H. Nik Zulkipli e N. Idris, «An empirical study on performance evaluation of parallel architecture for web application services», em *IEEE Symposium on Computers and Informatics, ISCI 2013*, 2013, pp. 216–221, doi: 10.1109/ISCI.2013.6612406.

[11] I. Iliev e G. P. Dimitrov, «Front end optimization methods and their effect», em *2014 37th International Convention on Information and Communication Technology, Electronics and Microelectronics, MIPRO 2014 - Proceedings*, 2014, pp. 467–473, doi: 10.1109/MIPRO.2014.6859613.

[12] Z. Zhu, F. Xiao, e G. Yang, «Research on performance optimization for the web-based university educational management information system», em *Proceedings - 2011 International Conference on Intelligence Science and Information Engineering, ISIE 2011*, 2011, pp. 261–264, doi: 10.1109/ISIE.2011.53.

[13] «The Golden Rule of Performance / Sudo Null IT News». https://sudonull.com/post/140065-The-Golden-Rule-of-Performance (acedido Jun. 23, 2020).

[14] A. Ciminian e C. Dobre, «DIGUA: MINIFIER AND OBFUSCATOR FOR WEB RESOURCES», Darmstadt. doi: 10.1145/3308558.3313752.

[15] P. Skolka, C.-A. Staicu, M. Pradel, S. Francisco, J. B. Sartor, e T. D. ' Hondt, «Anything to Hide? Studying Minified and Obfuscated Code in the Web», doi: 10.1145/3308558.3313752.

[16] «How To Optimize Your Site With GZIP Compression – BetterExplained».

https://betterexplained.com/articles/how-to-optimize-your-site-with-gzip-compression/ (acedido Jun. 23, 2020).

[17] K. Xu, X. Li, S. K. Bose, e G. Shen, «Joint Replica Server Placement, Content Caching, and Request Load Assignment in Content Delivery Networks», *IEEE Access*, vol. 6, pp. 17968–17981, Mar. 2018, doi: 10.1109/ACCESS.2018.2817646.

[18] «Make JavaScript and CSS External | Web Site Performance Improvement | DML Research». http://www.digital-media-lab.com/research_web_design_make_javascript_and_css_external.html (acedido Jun. 24, 2020).

[19] J. Lowery, «Dreamweaver MX Bible», 2002. https://books.google.pt/books?id=fMn6ltLm6RsC&pg=PA271&lpg=PA271&dq=placing+the+styles+at+the+top+of+the+page+advantages&source=bl&ots=zFZG9JAMSK&sig=ACfU3U3su8TUfErwhSUGnQ8h4ePWFBhAPQ&hl=pt-PT&sa=X&ved=2ahUKEwjk9rXGo5rqAhVIyxoKHRX6CXoQ6AEwAHoECAYQAQ#v=onepage&q=placing the styles at the top of the page advantages&f=false (acedido Jun. 24, 2020), ISBN: 9780764549311.

[20] M. Wang e Z. Qi, «Research and practice of web server optimization», em *2nd International Symposium on Electronic Commerce and Security, ISECS 2009*, 2009, vol. 2, pp. 432–436, doi: 10.1109/ISECS.2009.153.

[21] A. M. Alakeel e S. Arabia, «A Guide to Dynamic Load Balancing in Distributed Computer Systems», *Int. J. Comput. Sci. Inf. Secur.*, vol. 10, n. 6, pp. 153–160, 2010, Acedido: Jun. 30, 2020. [Em linha]. Disponível em: https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.456.5175&rep=rep1&type=pdf.

[22] S. Zhang, H. Wu, W. Wang, B. Yang, P. Liu, e A. V Vasilakos, «Distributed workload and response time management for web applications», em *2011 7th International Conference on Network and Service Management, CNSM 2011*, 2011, pp. 1–9, doi: 10.5555/2147671.2147702.

[23] M. Rahman, S. Iqbal, e J. Gao, «Load balancer as a service in cloud computing», em *Proceedings - IEEE 8th International Symposium on Service Oriented System Engineering, SOSE 2014*, 2014, pp. 204–211, doi: 10.1109/SOSE.2014.31.

[24] S. Tekale, J. Gladson, M. Britto, e A. S. Gousia Banu, «Load Balancing in Cloud Computing», *Int. J. Eng. Adv. Technol.*, pp. 2249–8958, 2019, doi: 10.35940/ijeat.F1418.0986S319.

[25] K. Nishant *et al.*, «Load balancing of nodes in cloud using ant colony optimization», em *Proceedings - 2012 14th International Conference on Modelling and Simulation, UKSim 2012*, Mar. 2012, pp. 3–8, doi: 10.1109/UKSim.2012.11.

[26] T. Sharma e V. K. Banga, «Efficient and Enhanced Algorithm in Cloud Computing», *Int. J. Soft Comput. Eng.*, 2013, ISSN: 2231-2307.

[27] Jayant Adhikari; Prof. Sulabha Patil, «Load Balancing The Essential Factor In Cloud Computing», *Int. J. Eng. Res. Technol.*, vol. 1, n. 10, pp. 1–5, 2012, doi: 10.5555/579099.

[28] T. Deepa e D. Cheelu, «A comparative study of static and dynamic load balancing algorithms in cloud computing», em *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing, ICECDS 2017*, Jun. 2018, pp. 3375–3378, doi:

10.1109/ICECDS.2017.8390086.

[29] K. Zhu, J. Fu, e Y. Li, «Research the performance testing and performance improvement strategy in web application», em *ICETC 2010 - 2010 2nd International Conference on Education Technology and Computer*, 2010, vol. 2, doi: 10.1109/ICETC.2010.5529374.

[30] S. Pradeep e Y. K. Sharma, «A Pragmatic Evaluation of Stress and Performance Testing Technologies for Web Based Applications», em *Proceedings - 2019 Amity International Conference on Artificial Intelligence, AICAI 2019*, Abr. 2019, pp. 399–403, doi: 10.1109/AICAI.2019.8701327.

[31] W. Jun e F. Meng, «Software testing based on cloud computing», em *Proceedings - 2011 International Conference on Internet Computing and Information Services, ICICIS 2011*, 2011, pp. 176–178, doi: 10.1109/ICICIS.2011.51.

[32] R. Abbas, Z. Sultan, e S. N. Bhatti , «Comparative Study of Load Testing Tools: Apache JMeter, HP LoadRunner, Microsoft Visual Studio (TFS), Siege», *Sukkur IBA J. Comput. Math. Sci.*, vol. 1, n. 2, p. 102, Dez. 2017, doi: 10.30537/sjcms.v1i2.24.

[33] B. Potter e G. McGraw, «Software security testing», *IEEE Security and Privacy*, vol. 2, n. 5. pp. 81–85, Set. 2004, doi: 10.1109/MSP.2004.84.

[34] Steve MConnell (construx software builders), «Daily build and Smoke Test», 1996.

[35] E. Dustin, *Effective Software Testing: 50 Ways to Improve Your Software Testing*. Addison-Wesley Longman Publishing Co., Inc.75 Arlington Street, Suite 300 Boston, MAUnited States, 2002.

[36] A. M. Memon, «Gui testing: Pitfalls and process», *Computer (Long. Beach. Calif).*, vol. 35, n. 8, pp. 87–88, 2002, doi: 10.1109/MC.2002.1023795.

[37] D. D. Phan, «Software Quality and Management: How the World's Most Powerful Software Makers do it», *Inf. Syst. Manag.*, vol. 18, n. 1, pp. 56–67, 2001, doi: 10.1201/1078/43194.18.1.20010101/31265.7.

[38] G. Denaro, A. Polini, e W. Emmerich, «Early performance testing of distributed software applications», em *Proceedings of the fourth international workshop on Software and performance - WOSP '04*, 2004, p. 94, doi: 10.1145/974044.974059.

[39] R. K. Lenka, S. Mamgain, S. Kumar, e R. K. Barik, «Performance Analysis of Automated Testing Tools: JMeter and TestComplete», em *Proceedings - IEEE 2018 International Conference on Advances in Computing, Communication Control and Networking, ICACCCN 2018*, Out. 2018, pp. 399–407, doi: 10.1109/ICACCCN.2018.8748521.

[40] T. F. Dullmann, R. Heinrich, A. Van Hoorn, T. Pitakrat, J. Walter, e F. Willnecker, «CASPA: A platform for comparability of architecture-based software performance engineering approaches», em *Proceedings - 2017 IEEE International Conference on Software Architecture Workshops, ICSAW 2017: Side Track Proceedings*, Jun. 2017, pp. 294–297, doi: 10.1109/ICSAW.2017.26.

[41] E. H. Halili, *Apache JMeter A practical beginner's guide to automated testing and performance measurement for your websites*. 2008, ISBN: 9781847192950.

[42] K. Huang, L. X. Yang, X. Yang, Y. Xiang, e Y. Y. Tang, «A Low-Cost Distributed Denial-of-Service Attack Architecture», *IEEE Access*, vol. 8, pp. 42111–42119, 2020, doi: 10.1109/ACCESS.2020.2977112.

[43] «Para iniciantes». https://tableless.github.io/iniciantes/manual/js/o-que-framework.html

(acedido Ago. 24, 2020).

[44] «What is Frameworks? [Definition] Types of Frameworks». https://hackr.io/blog/what-is-frameworks (acedido Ago. 24, 2020).

[45] «Angular». https://angular.io/ (acedido Ago. 24, 2020).

[46] «React – A JavaScript library for building user interfaces». https://reactjs.org/ (acedido Ago. 24, 2020).

[47] «Vue.js». https://vuejs.org/ (acedido Ago. 24, 2020).

[48] «What are the most in-demand frontend frameworks in 2020? - LogRocket Blog». https://blog.logrocket.com/state-of-javascript-what-are-the-most-in-demand-frontend-frameworks-in-2020/ (acedido Ago. 24, 2020).

[49] «Best Frontend Frameworks of 2020 for Web Development». https://www.simform.com/best-frontend-frameworks/ (acedido Ago. 24, 2020).

[50] «Top Front-End Frameworks in 2020 | Existek Blog». https://existek.com/blog/top-front-end-frameworks-2020/ (acedido Ago. 24, 2020).

[51] «AngularJS — Superheroic JavaScript MVW Framework». https://angularjs.org/ (acedido Ago. 24, 2020).

[52] «Google - About Google, Our Culture & Company News». https://about.google/?utm_source=google-PT&utm_medium=referral&utm_campaign=hp-footer&fg=1 (acedido Ago. 25, 2020).

[53] «List of Top JavaScript Frameworks 2020 For Front-End Developers». https://www.freecodecamp.org/news/complete-guide-for-front-end-developers-javascript-frameworks-2019/ (acedido Ago. 24, 2020).

[54] «Social Network for Programmers and Developers». https://morioh.com/p/7687cbec6e12 (acedido Ago. 24, 2020).

[55] «(3) Facebook». https://www.facebook.com/ (acedido Ago. 25, 2020).

[56] «React Native · A framework for building native apps using React». https://reactnative.dev/ (acedido Ago. 25, 2020).

[57] «Android». https://www.android.com/intl/pt_pt/ (acedido Ago. 25, 2020).

[58] «iOS 13 - Apple (PT)». https://www.apple.com/pt/ios/ios-13/ (acedido Ago. 25, 2020).

[59] «npm | build amazing things». https://www.npmjs.com/ (acedido Ago. 25, 2020).

[60] «npm trends: Compare NPM package downloads». https://www.npmtrends.com/ (acedido Ago. 25, 2020).

[61] «react vs vue vs angular | npm trends». https://www.npmtrends.com/react-vs-vue-vs-angular (acedido Ago. 25, 2020).

[62] «Stack Overflow - Where Developers Learn, Share, & Build Careers». https://stackoverflow.com/ (acedido Ago. 25, 2020).

[63] «Please come back... :)». https://www.letsnurture.com/blog/django-vs-laravel-vs-node-js.html

(acedido Ago. 25, 2020).

[64] «Top 7 Backend Frameworks for Web Development in 2020». https://www.kelltontech.com/kellton-tech-blog/7-most-popular-backend-web-development-frameworks-2020 (acedido Ago. 25, 2020).

[65] «Top 10 Backend Frameworks for Web Development in 2020». https://www.decipherzone.com/blog-detail/top-10-best-backend-frameworks-for-web-development-in-2020 (acedido Ago. 25, 2020).

[66] «Why Django is the Best Web Framework for Your Project». https://steelkiwi.com/blog/why-django-best-web-framework-your-project/ (acedido Ago. 25, 2020).

[67] «Companies using Express». https://expressjs.com/en/resources/companies-using-express.html (acedido Ago. 25, 2020).

[68] «When Python scores over PHP?». https://www.softkraft.co/python-vs-php/#php-vs-python-performance-comparison (acedido Ago. 26, 2020).

[69] «Node.js vs PHP: Which is better for web development? | Hacker Noon». https://hackernoon.com/nodejs-vs-php-which-is-better-for-your-web-development-he7oa24wp (acedido Ago. 26, 2020).

[70] «QuerySet API reference | Django documentation | Django». https://docs.djangoproject.com/en/3.1/ref/models/querysets/ (acedido Ago. 26, 2020).

[71] «SQL Injection | OWASP». https://owasp.org/www-community/attacks/SQL_Injection (acedido Ago. 26, 2020).

[72] «Timing System». https://apus.uma.pt/trail/#/events (acedido Nov. 02, 2020).

[73] Ieee, *IEEE Standard Glossary of Software Engineering Terminology*, vol. 121990, n. 1. 1990, DOI: 10.1109/IEEESTD.1990.101064.

[74] H. Sousa e E. Marques, «Visualização de um evento trail running», Universidade da Madeira, 2019, .

[75] I. Jacobson, I. Spence, e K. Bittner, «USE-CASE 2.0», 2011, ISBN:978-0201544350.

[76] G. Booch, I. Jacobson, e J. Rumbaugh, «Unified Modeling Language for Object-Oriented Development Documentation Set Version 0.91Addendum UML Update», 1996. Acedido: Set. 14, 2020. [Em linha]. Disponível em: http://www.rational.com.

[77] R. Khosla e Q. Li, «Unified problem modeling language for knowledge engineering of complex systems», *Soft Comput.*, vol. 8, n. 7, pp. 491–511, 2004, doi: 10.1007/s00500-003-0307-x.

[78] Prototype." *UXL Encyclopedia of Science*, edited by Amy Hackney Blackwell and Elizabeth Manar, 3rd ed., UXL, 2015. *Gale In Context: Science*, link.gale.com/apps/doc/ENKDZQ347975681/SCIC?u=dclib_main&sid=SCIC&xid=7944326d.

[79] V. Berzins e R. T. Yeh, «Calhoun: The NPS Institutional Archive DSpace Repository A Prototyping Language for Real-Time Software A Prototyping Language for Real-Time Software», 1988. Acedido: Set. 14, 2020. Disponível em: http://hdl.handle.net/10945/39162.

[80] P. Clements, D. Garlan, R. Little, R. Nord, e J. Stafford, «Documenting software architectures:

Views and beyond», *Proc. - Int. Conf. Softw. Eng.*, pp. 740–741, 2003, doi: 10.1109/icse.2003.1201264.

[81] «A Survey of Architecture Description Languages». doi: 10.5555/857204.858261.

[82] N. Medvidovic e R. N. Taylor, «A classification and comparison framework for software architecture description languages», *IEEE Trans. Softw. Eng.*, vol. 26, n. 1, pp. 70–93, 2000, doi: 10.1109/32.825767.

[83] Apigee, «Web API Design: The Missing Link», pp. 1–50, 2016.

[84] «Access Tokens». https://auth0.com/docs/tokens/access-tokens (acedido Set. 04, 2020).

[85] «Index – Django REST framework». https://api.ts.uma.pt/ (acedido Nov. 10, 2020).

[86] «REDOC – AN OPENAPI-POWERED DOCUMENTATION UI». https://swagger.io/blog/api-development/redoc-openapi-powered-documentation/ (acedido Set. 04, 2020).

[87] «API Documentation & Design Tools for Teams | Swagger». https://swagger.io/ (acedido Set. 04, 2020).

[88] «web.dev». https://web.dev/progressive-web-apps/ (acedido Nov. 01, 2020).

[89] «What are Progressive Web Apps?» https://web.dev/what-are-pwas/ (acedido Set. 09, 2020).

[90] «Progressive Web App (PWA): what they are, pros and cons and the main examples on the market | by IQUII | IQUII | Medium». https://medium.com/iquii/progressive-web-app-pwa-what-they-are-pros-and-cons-and-the-main-examples-on-the-market-318f4538c670 (acedido Set. 09, 2020).

[91] «Timing System». https://ts.uma.pt/ (acedido Nov. 10, 2020).

[92] S. Kiran, A. Mohapatra, e R. Swamy, «Experiences in performance testing of web applications with Unified Authentication platform using Jmeter», *2nd Int. Symp. Technol. Manag. Emerg. Technol. ISTMET 2015 - Proceeding*, pp. 74–78, 2015, doi: 10.1109/ISTMET.2015.7359004.

[93] Q. Wu e Y. Wang, «Performance testing and optimization of J2EE-based web applications», *2nd Int. Work. Educ. Technol. Comput. Sci. ETCS 2010*, vol. 2, pp. 681–683, 2010, doi: 10.1109/ETCS.2010.583.

[94] «Apache JMeter - User's Manual: Glossary». https://jmeter.apache.org/usermanual/glossary.html (acedido Set. 23, 2020).

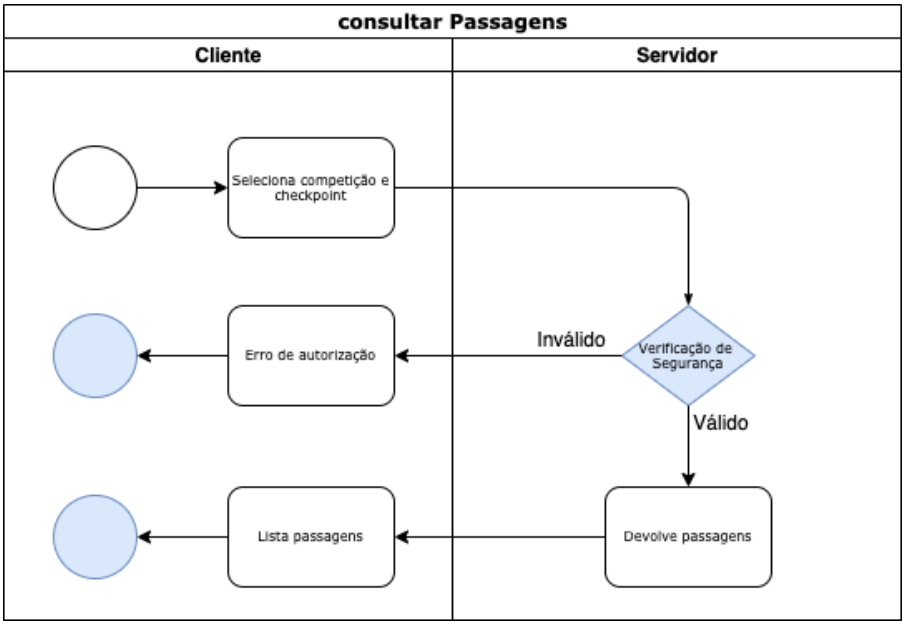
[95] «5-Point Likert Scale», em *Handbook of Disease Burdens and Quality of Life Measures*, Springer New York, 2010, pp. 4288–4288.

[96] «Analytics». https://analytics.google.com/analytics/web/#/a175463588w243269505p226635460/admin (acedido Nov. 10, 2020).

[97] J. R. Lewis, «The System Usability Scale: Past, Present, and Future», *Int. J. Hum. Comput. Interact.*, vol. 34, n. 7, pp. 577–590, Jul. 2018, doi: 10.1080/10447318.2018.1455307.

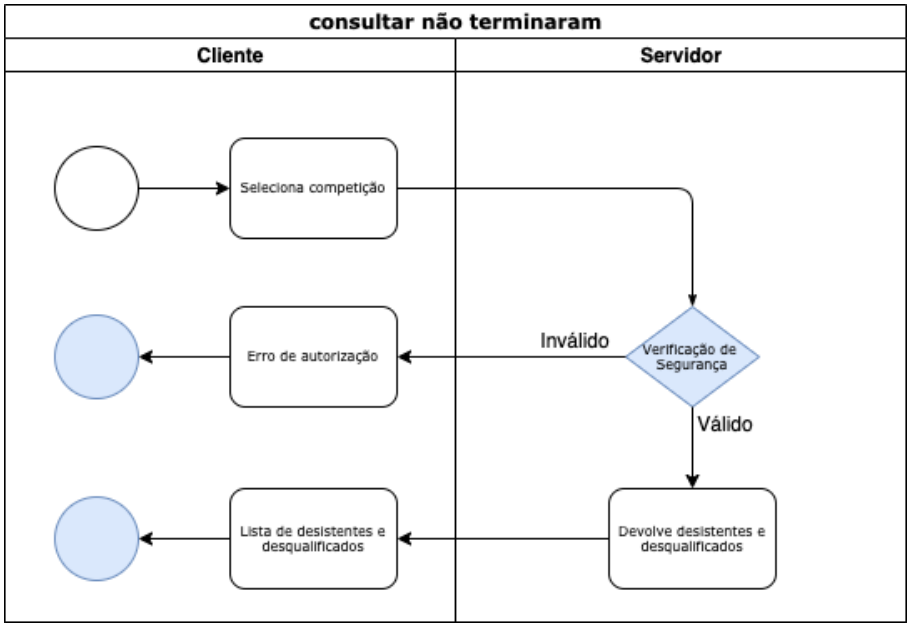
________________________ Anexos __________________________

Anexo 1 – Diagrama de atividade referente à ação “Consultar Passagens”



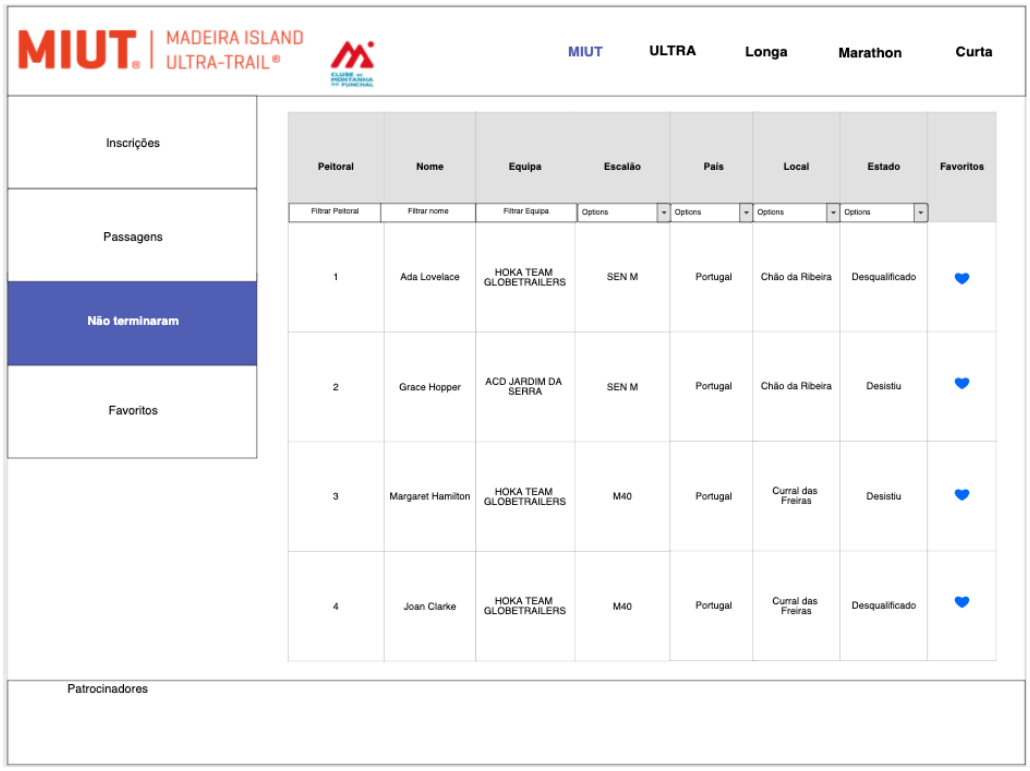
Anexo 1 - Diagrama de atividade referente à ação “Consultar Passagens”.

Anexo 2 – Diagrama de atividade referente à ação “consultar não terminaram”



Anexo 2 - Diagrama de atividade referente à ação “Consultar não terminaram”.

Anexo 3 – Protótipo para a lista de atletas que não terminaram



Anexo 3 - Protótipo para a lista de atletas que não terminaram.

Anexo 4 – Protótipo para a lista de favoritos do utilizador

Anexo 4 - Protótipo para a lista de favoritos do utilizador.

Anexo 5 - Dados em bruto os inquéritos

Qual a sua idade?	O que achou da aparência e funcionalidades gerais?	O que achou da velocidade em que os dados são apresentados?	Foi fácil de aceder aos dados que desejava?	Achou útil a opção de adicionar favoritos?	Recomendaria a plataforma a um familiar ou colega?
31-40	5	5	5	5	5
41-50	5	5	5	5	5
26-30	4	5	4	5	4
51+	4	4	5	5	5
19-25	5	5	4	5	5
19-25	5	5	5	5	4
31-40	4	5	5	5	5
41-50	4	3	4	5	4
31-40	4	3	4	3	4
41-50	3	3	3	3	3
51+	5	5	4	4	5
41-50	4	4	5	4	4
26-30	5	4	5	5	5
41-50	4	4	5	5	5
31-40	5	5	4	5	5
41-50	4	4	5	5	4
51+	5	5	4	5	5
26-30	5	5	4	5	5
26-30	5	4	5	5	5
51+	4	5	5	4	4
41-50	5	5	5	4	4
19-25	5	5	5	5	5

menos de 18	4	4	3	5	5
menos de 18	5	5	4	5	5
41-50	5	5	4	5	5
26-30	5	4	5	5	5
19-25	5	5	5	5	5
19-25	4	5	5	5	5
19-25	4	5	5	5	5
51+	4	4	5	5	4
51+	4	5	4	5	5
19-25	5	5	4	5	5
31-40	4	5	5	4	5
31-40	5	4	5	5	4
31-40	5	5	5	4	5
26-30	5	4	5	5	5
19-25	5	5	3	4	5
19-25	5	5	4	5	5
26-30	5	5	5	5	5
41-50	5	5	4	5	5
31-40	5	5	5	5	5
51+	5	4	5	5	4
26-30	5	5	4	5	5
31-40	5	5	4	5	5
41-50	5	5	5	4	5
19-25	5	5	4	5	5
31-40	5	5	4	5	4
menos de 18	5	5	4	5	5
41-50	5	4	5	5	5
19-25	5	5	5	4	5

26-30	5	5	5	4	4
19-25	4	5	4	4	3
31-40	4	4	4	3	4
41-50	4	4	4	3	4
menos de 18	4	5	5	5	5
26-30	4	5	5	3	5
31-40	4	5	5	5	5
19-25	4	5	5	5	5
26-30	4	5	5	5	5
31-40	4	5	4	4	5
26-30	4	5	5	5	5
19-25	4	5	4	5	5
31-40	4	5	5	5	5
26-30	4	5	5	5	5
26-30	4	5	5	5	5
51+	4	5	5	5	5
26-30	4	4	5	5	5
19-25	4	5	5	5	5
26-30	4	3	5	5	5
19-25	4	5	5	5	4
26-30	4	5	5	4	5
19-25	4	5	5	4	5
26-30	4	5	5	5	5
26-30	4	4	5	5	4
menos de 18	4	5	5	5	5
26-30	4	5	5	5	5
19-25	4	5	5	5	5
26-30	4	5	5	5	5
26-30	4	5	5	5	5

31-40	4	5	5	3	5
19-25	4	5	5	5	5
26-30	4	5	5	5	5
31-40	4	5	5	5	5
31-40	4	4	5	5	5
26-30	4	5	4	5	5
26-30	4	5	5	5	5
26-30	4	5	5	5	5
26-30	4	5	5	5	4
31-40	4	4	5	4	5
26-30	4	5	5	5	5
19-25	4	5	5	5	5
31-40	4	5	5	5	4
26-30	4	5	5	5	5

A Nossa Universidade

Colégio dos Jesuítas
Rua dos Ferreiros - 9000-082, Funchal

Tel: +351 291 209400
Fax: +351 291 209410
Email: gabinetedareitoria@uma.pt