

Universidade da Beira Interior

Departamento de Informática



**Departamento de
Informática**

**Nº 1 - 2023: *Desenho e Desenvolvimento de
Plataforma e Website para o Centro Académico
Clínico das Beiras (CACB)***

Elaborado por:

Marta Godinho 45762

Orientador:

Mestre João Sequeiros

Co-orientadores:

Professor Doutor Vítor Figueiredo

Professora Doutora Anabela Almeida

1 de julho de 2023

Agradecimentos

Para começar, quero agradecer ao meu orientador, Mestre Bernardo Sequeiros e aos meus coorientadores Professora Doutora Anabela Almeida e Professor Doutor Vítor Figueiredo por me confiarem a tarefa de realizar este projeto e por me ajudarem sempre com a maior prontidão.

Agradeço à minha família, a minha mãe Anabela e os meus avós Manuel e Conceição por todo o apoio dado ao longo destes três anos e por todos os conselhos e promessas de puxões de orelhas. Também quero agradecer aos meus primos, Flávia, Luís e Kyara por me ajudarem e darem forças para continuar e por prontamente ajudarem-me com o que precisava.

Quero também agradecer às minhas amigas Bruna, Carolina e Laura e à minha namorada, Teresa por me distraírem com as suas brincadeiras e conversas que me ajudaram a superar os momentos mais difíceis e por terem sempre uma palavra de encorajamento. Em especial, quero agradecer à Bruna por me apoiar e ajudar em todos os aspetos, pela sua amizade muito especial que me ajudou a superar todas as dificuldades e obstáculos ao longo do meu percurso académico.

Por fim, quero agradecer aos meus restantes amigos e família.

Conteúdo

Conteúdo	iii
Lista de Figuras	vii
Lista de Tabelas	ix
1 Introdução	1
1.1 Enquadramento e Motivação	1
1.2 Objetivos	1
1.3 Organização do Documento	2
1.4 Algumas Dicas – [RETIRAR DA VERSÃO FINAL]	2
2 Estado da Arte	5
2.1 Introdução	5
2.2 Plataformas Semelhantes	5
2.2.1 2CA Braga Centro Académico Clínico	5
2.2.2 ABC - Algarve <i>Biomedical Center</i>	6
2.3 Ferramentas Semelhantes	7
2.3.1 <i>Wordpress</i>	7
2.3.2 <i>Django</i>	7
2.4 Conclusões	7
3 Tecnologias Utilizadas e Engenharia de Software	9
3.1 Introdução	9
3.2 Tecnologias Utilizadas	9
3.2.1 HTML	9
3.2.2 CSS	10
3.2.3 EJS	10
3.2.4 Heidi SQL	10
3.2.5 SQL	10
3.2.6 JS	11
3.2.7 <i>Express.js</i>	11
3.2.8 <i>bcrypt</i>	11

3.2.9	<i>body-parser</i>	11
3.2.10	<i>multer</i>	11
3.2.11	<i>mysql</i>	12
3.3	Engenharia de <i>Software</i>	12
3.3.1	Análise de Requisitos	12
3.3.1.1	Requisitos Funcionais	12
3.3.1.2	Requisitos Não Funcionais	13
3.3.2	Diagramas UML	13
3.3.2.1	Diagramas de Caso de Uso	14
3.3.2.2	Diagramas de Atividade	16
3.3.3	Modelação de Dados	16
3.3.3.1	Modelo Conceptual	16
3.3.3.2	Modelo Físico	17
3.4	Conclusões	18
4	Funcionalidades e Implementação	19
4.1	Introdução	19
4.2	Funcionalidades da Aplicação	19
4.3	Implementação	26
4.3.1	<i>Home Page</i>	26
4.3.2	Página Eventos	27
4.3.3	Criar Conta	28
4.3.4	<i>Login</i>	30
4.3.5	Página Pessoal	31
4.3.6	Criação de um Evento	32
4.3.7	Ver Eventos Criados	33
4.3.8	Editar um Evento	33
4.3.9	Editar Perfil	34
4.3.10	Contas Pendentes	35
4.3.11	Contas Aceites	37
4.4	Conclusões	37
5	Manual de Instalação e Testes	39
5.1	Introdução	39
5.2	Manual de Instalação	39
5.3	Testes	40
5.4	Manual de Utilização	40
5.5	Conclusões	40
6	Conclusões e Trabalho Futuro	41
6.1	Conclusões Principais	41

6.2 Trabalho Futuro	41
A Excerto de código para mostrar a rota da <i>home page</i>	43
B Excerto de código para apresentar a organização dos <i>posts</i> de <i>blog</i> e testemunhos	45
C Excerto de código para apresentar a organização conteúdo da página Eventos	49
D Excerto de código para mostrar a verificação de privilégio de um utilizador	51
E Excerto de código para apresentar como se cria um evento	53
F Excerto de código para a alteração do email	55
G Excerto de código para criar as tabelas da base de dados	57
Bibliografia	63

Listas de Figuras

2.1	Captura de ecrã da <i>Home Page</i> do Centro Académico Clínico de Braga.	6
2.2	Captura de ecrã da <i>Home Page</i> do ABC - Algarve <i>Biomedical Center</i>	6
3.1	Diagrama de caso de uso de um utilizador sem privilégios.	14
3.2	Diagrama de caso de uso de um utilizador com privilégios.	15
3.3	Diagrama de caso de uso de um "super administrador".	15
3.4	Diagrama de caso de atividade.	16
3.5	Modelo conceptual da aplicação.	17
3.6	Modelo físico da aplicação.	17
4.1	Captura de ecrã demonstrativa da primeira parte da <i>home page</i> da aplicação.	20
4.2	Captura de ecrã demonstrativa da segunda parte da <i>home page</i> da aplicação.	20
4.3	Captura de ecrã demonstrativa da página Evento da aplicação.	21
4.4	Captura de ecrã demonstrativa da página Criar Conta da aplicação.	21
4.5	Captura de ecrã demonstrativa da página <i>Login</i> da aplicação.	22
4.6	Captura de ecrã demonstrativa da página Página Pessoal da aplicação.	23
4.7	Captura de ecrã demonstrativa da página Criar Evento da aplicação.	24
4.8	Captura de ecrã demonstrativa da página Ver Eventos criados da aplicação.	24
4.9	Captura de ecrã demonstrativa da página Editar Perfil da aplicação.	25
4.10	Captura de ecrã demonstrativa da página Contas Aceites da aplicação.	25
4.11	Captura de ecrã demonstrativa da página Contas Pendentes da aplicação.	26

Listas de Tabelas

3.1	Requisitos funcionais da plataforma.	12
3.2	Continuação da tabela requisitos funcionais da plataforma.	13
3.3	Requisitos não funcionais da plataforma.	13

Listas de Excertos de Código

4.1	Rota para apresentar a página Eventos.	27
4.2	Paginação para aparecerem apenas dez publicações por página.	28
4.3	Rota post para criar uma conta e definição das variáveis para definir o privilégio.	29
4.4	Uso da biblioteca <i>bcrypt</i> para gerar o hash da palavra-passe.	29
4.5	Apresentar mensagem de sucesso ao criar a conta.	30
4.6	Apresentar mensagem de erro ao criar a conta.	30
4.7	Rota /login para efetuar o login e pesquisa na base de dados para encontrar o utilizador.	30
4.8	Rota /pagina_pessoal para apresentar a página pessoal.	31
4.9	Condição para mostrar ou não a "caixa" dos eventos.	32
4.10	Verificação da existência da variável <i>priv_utilizador</i>	32
4.11	Rota ver_eventos para ir buscar a informação dos eventos da base de dados.	33
4.12	Verificação da edição dos campos do formulário.	33
4.13	Atualização da base de dados.	34
4.14	Alteração da palavra-passe.	35
4.15	Consulta à base de dados para ver as contas pendentes.	35
4.16	Atualização do estado da conta para aceite.	36
4.17	Atualização do estado da conta para rejeitado.	36
4.18	Consulta à base de dados para ver as contas que já foram aceites.	37
4.19	Remoção de uma conta da base de dados.	37
5.1	Comando para instalar as bibliotecas necessárias.	39
5.2	Comando para inicializar o projeto.	40
5.3	Outra opção de comando para inicializar o projeto.	40
5.4	Comandos para criar um novo utilizador e para lhe dar os privilégios.	40
A.1	Rota para apresentar a <i>home page</i>	43
B.1	Organização das publicações de <i>blog</i> e testemunhos na <i>home page</i>	45
C.1	Organização do conteúdo página Eventos.	49
D.1	Verificação do privilégio de um utilizador.	51
E.1	Rota novo_evento para inserir um novo evento na base de dados.	53

F.1 Rota alterar_email para o utilizador alterar o email.	55
G.1 Criar as tabelas da base de dados.	57

Acrónimos

- CACB** Centro Académico Clínico das Beirasl
CSS *Cascading Style Sheets*
EJS *Embedded JavaScript*
HTML *HyperText Markup Language*
HTTP *HyperText Transfer Protocol*
JS *JavaScript*
PDF Portable Document Format
SGBD Sistema de Gestão de Base de Dados
SQL *Structured Query Language*
UBI Universidade da Beira Interior
UML Unified Modeling Language

Capítulo

1

Introdução

Este relatório foi realizado no contexto da unidade curricular de Projeto, que pertence à Licenciatura Informática Web da Universidade da Beira Interior (UBI). O primeiro capítulo aborda o Enquadramento e Motivação (1.1), os Objetivos (1.2) e a Organização do Documento (1.3).

1.1 Enquadramento e Motivação

Os Centros Académicos Clínicos são uma iniciativa nacional que procura melhorar a qualidade dos cuidados de saúde, as práticas de pesquisa, o ensino e a formação de profissionais na área da saúde. O Centro Académico Clínico das Beirasl (CACB) é uma das instituições que integra esta iniciativa e reúne diversas instituições de ensino, pesquisa e saúde dos distritos de Castelo Branco, Guarda e Viseu. A sua sede está localizada na Faculdade de Ciências da Saúde da Universidade da Beira Interior e uma das principais missões é a pesquisa e formação de profissionais, com o objetivo final de melhorar os cuidados de saúde e os indicadores de saúde da região. É de extrema importância, portanto, ter uma plataforma para divulgar, comunicar, compartilhar informações e permitir o contacto entre os profissionais, as instituições e o Centro, permitindo que a comunidade de profissionais de saúde assim como o público em geral possam aceder às informações que lhes possam interessar.

1.2 Objetivos

O objetivo principal deste projeto é reformular e criar um *website*, bem como criar uma plataforma (*backend* e *backoffice*) preparada e capaz de suportar

futuras expansões, onde seja possível divulgar notícias, formações e outras informações relevantes do CACB. Outros objetivos são:

- Incluir uma parte que funcione como *blog* onde novas informações possam ser partilhadas e subscritas pelos utilizadores;
- Registar inscrições de participantes nos vários eventos organizados pelo CACB;
- Garantir a segurança de todos os dados na plataforma e que esta seja robusta no seu funcionamento geral.

1.3 Organização do Documento

Este documento encontra-se organizado da seguinte forma:

1. O primeiro capítulo – **Introdução** – apresenta o projeto, o seu enquadramento e motivação, quais os objetivos e como o documento se encontra organizado;
2. O segundo capítulo – **Estado da Arte** – descreve a existência de outras plataformas como a que foi desenvolvida, assim como, ferramentas semelhantes às usadas;
3. O terceiro capítulo – **Tecnologias Utilizadas** – apresenta quais as tecnologias mais importantes utilizadas para realizar o projeto.
4. O quarto capítulo – **Implementação e Testes** –
5. O quinto capítulo – **Conclusões e Trabalho Futuro** – apresenta as conclusões do trabalho realizado e o que pode vir a ser melhorado no futuro.

1.4 Algumas Dicas – [RETIRAR DA VERSÃO FINAL]

Os relatórios de projeto são individuais e preparados em L^AT_EX, seguindo o formato disponível na página da unidade curricular. Deve ser prestada especial atenção aos seguintes pontos:

1. O relatório deve ter um capítulo Introdução e Conclusões e Trabalho Futuro (ou só Conclusões);

2. A última secção do primeiro capítulo deve descrever sucintamente a organização do documento;
3. O relatório pode ser escrito em Língua Portuguesa ou Inglesa;
4. Todas as imagens ou tabelas devem ter legendas e ser referidas no texto (usando comando \ref{}).

Capítulo

2

Estado da Arte

2.1 Introdução

Neste capítulo são apresentadas algumas plataformas como a que foi desenvolvida e ferramentas que já existem que também ajudam na criação de *websites*. Na secção Plataformas Semelhantes (2.2) são apresentadas plataformas que são semelhantes a esta, estando dividida em duas subsecções 2.2.1 e 2.2.2, que referem dois *websites* de Centros Académicos Clínicos que já existem. Na secção seguinte Ferramentas Semelhantes (2.3), são explicitadas quais as ferramentas que são semelhantes às tecnologias utilizadas para realizar este projeto e está dividida em duas subsecções, 2.3.1 e 2.3.2, onde são apresentadas duas ferramentas semelhantes às usadas.

2.2 Plataformas Semelhantes

Os Centros Académicos Clínicos estão a apostar na criação de *websites* próprios. Nesta secção serão apresentados os dois *websites* que já existem e quais são alguns pontos positivos e negativos dos mesmos.

2.2.1 2CA Braga Centro Académico Clínico

O 2CA Braga Centro Académico Clínico [1] apresenta um *design* muito apelativo como se pode observar na figura 2.1. Na sua *home page* apresenta algumas notícias sobre o centro e que formações está a proporcionar. O *website* também apresenta quais são os serviços que o centro oferece, quais são os seus projetos, recursos humanos, etc. A navegação é fácil e bastante intuitiva, sem grande dificuldade para encontrar o que se pretende. No *website* não se

encontram os eventos relacionados com o centro, o que seria importante para promover ainda mais o centro.



Figura 2.1: Captura de ecrã da *Home Page* do Centro Académico Clínico de Braga.

2.2.2 ABC - Algarve *Biomedical Center*

O ABC - Algarve *Biomedical Center*, Centro Académico Clínico de Investigação e Formação Biomédica do Algarve [2] tem um *design* mais antiquado mas também apresenta uma boa organização. A primeira página é onde se esconde qual a linguagem em que se quer ver o *website* e só depois se pode navegar pelo mesmo, o que torna a experiência menos apelativa visto que podia ser uma opção que estaria no conteúdo das páginas.



Figura 2.2: Captura de ecrã da *Home Page* do ABC - Algarve *Biomedical Center*.

2.3 Ferramentas Semelhantes

Já existem plataformas próprias para fazer *websites* e pode-se aceder ao *bac-koffice* de maneira intuitiva e fácil, assim como outras ferramentas de desenvolvimento de aplicações *web*.

2.3.1 Wordpress

Uma destas ferramentas é o *Wordpress* [3] que tem várias ferramentas como *design* personalizável, é otimizado para SEO (*Search Engine Optimization*), permite uma gestão em qualquer lugar, etc. No entanto, existem algumas falhas como não existir muita documentação, ser um serviço de código aberto, mas não é grátis, tem um tempo de resposta lento e como é uma ferramenta de código aberto, como já foi mencionado pode ter falhas de segurança e quando existe a atualização de *plugins* torna-se muito mais difícil de fazer a manutenção do *site* [4].

2.3.2 Django

O *Django* é uma *framework* criada para desenvolver aplicações *web* de maneira rápida, que já inclui muitas bibliotecas e outros recursos que ajudam no desenvolvimento das aplicações. Também possui bastante documentação, tutoriais que facilitam o seu uso e também possui soluções de segurança já embutidas que ajudam a uma plataforma mais segura [5]. No entanto, sendo uma plataforma que não foi abordada nas unidades curriculares seria uma desvantagem utilizar, pois teria de ser aprendida antes de se desenvolver este projeto podendo atrasar o seu desenvolvimento.

2.4 Conclusões

Tendo acesso às plataformas semelhantes que já existem foi possível perceber quais eram os assuntos que ambas tratavam e foi tentado fazer algo semelhante, atendendo às necessidades do CACB. Também foram abordadas duas plataformas que poderiam ter sido utilizadas para desenvolver o projeto e as razões pelas quais não foram escolhidas.

Capítulo

3

Tecnologias Utilizadas e Engenharia de Software

3.1 Introdução

Neste capítulo irão ser apresentadas as tecnologias e ferramentas utilizadas, bem como a engenharia de *software*. Está dividido pelas secções Tecnologias Utilizadas (3.2) onde são referidas as tecnologias utilizadas bem como as bibliotecas necessárias. A segunda secção 3.3 que aborda temas como requisitos funcionais, não funcionais, diagramas Unified Modeling Language (UML) e a modelação de dados.

3.2 Tecnologias Utilizadas

Para desenvolver este projeto foram utilizadas várias tecnologias, que são *HyperText Markup Language* (HTML), *Cascading Style Sheets* (CSS), *Embedded JavaScript* (EJS), *HeidiSQL*, *Structured Query Language* (SQL), *JavaScript* (JS) e também as bibliotecas *Express.js*, *bcrypt*, *body-parser* e *multer* que vão ser explicadas nas secções a seguir e para o que foram utilizadas no contexto do projeto.

3.2.1 HTML

O HTML é uma linguagem de marcação que é, normalmente, utilizada para criar páginas e aplicações *web* e permite organizar a estrutura do texto, ou outros elementos em páginas *web* sendo um documento que tem várias *tags* e cada uma delas identifica um elemento diferente [6]. Foi utilizado, neste

trabalho, para criar as páginas *web* estáticas que não precisavam de estar a receber nova informação.

3.2.2 CSS

O CSS é uma linguagem de estilos que é usada para estilizar a aparência e formatação de uma linguagem de marcação e é geralmente utilizado com HTML e JS para criar aplicações e páginas *web* [7]. Foi este o propósito de usar CSS neste projeto, pois foi utilizado para fazer o *design* de todas as páginas, quer do *frontend* como do *backoffice*.

3.2.3 EJS

O EJS é um *template engine* que permite fazer páginas HTML que contêm JS para que as páginas sejam dinâmicas [8]. Neste projeto foi utilizado para que as páginas dinâmicas pudessem ir sendo atualizadas de acordo com os dados inseridos na plataforma de *backoffice* para que o *website* se encontre sempre atualizado com os dados que o CACB pretenda publicar.

3.2.4 Heidi SQL

Para a criação e manutenção da base de dados foi utilizado o *software* HeidiSQL que foi utilizado com o Sistema de Gestão de Base de Dados (SGBD) MySQL. Neste projeto a base de dados foi criada neste programa e foram definidas todas as tabelas da base de dados e os seus atributos, chaves primárias e estrangeiras para que os dados fiquem organizados e coerentes. Antes de ser feita a criação da base de dados foi feito o diagrama de entidade-relacionamento e a normalização, que serão mencionadas posteriormente no documento.

3.2.5 SQL

SQL é uma linguagem de programação que serve para armazenar e transformar informações numa base de dados e essas informações são organizadas em tabelas, que contêm linhas e colunas para organizar os dados necessários [9]. Neste projeto foi utilizado para fazer inserções, atualizações e pesquisa de informações de maneira que os novos dados a aparecer no *website* sejam inseridos e sejam guardados na base de dados e para que quando fosse necessário apresentar as informações é feita uma pesquisa à base de dados para que os dados sejam apresentados nas páginas *web*.

3.2.6 JS

O JS é uma linguagem de programação leve que é normalmente usada para criar dinamismo no desenvolvimento de aplicações *web*, *websites* e servidores [10]. Neste projeto, o JS foi utilizado no *client-side* para traduzir as páginas estáticas de português para inglês e também para poder mostrar os dados, que estão guardados na base de dados, nas páginas dinâmicas e foi utilizado Node.js para criar um servidor, para que exista ligação entre o *client-side* e o *server-side*.

3.2.7 Express.js

Express.js [11], uma biblioteca de JS, é uma *framework* de Node.js que fornece várias funcionalidades para criar aplicações *web* ou móveis. Esta *framework* permite estruturar servidores de formas mais simples e possui um sistema de rotas muito completo. Foi usado neste projeto, pois permite criar aplicações *fullstack* com mais facilidade.

3.2.8 bcrypt

Para que as palavras-passe fossem guardadas de forma segura na base de dados foi utilizada a biblioteca *bcrypt* [12] que faz o *hash* da palavra-passe, usando *salt*, que é uma sequência de caracteres aleatórios que fica anexada antes ou depois da palavra-passe permitindo assim que cada vez que uma é criada, o seu *hash* seja diferente, mesmo que duas pessoas escolham a mesma palavra-passe.

3.2.9 body-parser

Também foi utilizada a biblioteca *body-parser*[13], um *middleware* do Node.js que processa dados enviados por solicitação de *HyperText Transfer Protocol* (HTTP). Esta biblioteca faz com que todos os *middlewares* preencham a propriedade *req.body* com o corpo (*body*) analisado quando o cabeçalho da requisição corresponder a uma opção *type* ou um objeto vazio se não houver corpo para analisar.

3.2.10 multer

Para guardar os ficheiros que serão inseridos na aplicação foi utilizada a biblioteca *multer* [14] que é um *middleware* que adiciona um objeto *body* e um objeto de *file* ou *files*, ao objeto *request*. Quando se faz *upload* de um ficheiro

via um formulário, adiciona-se `enctype="multipart/form-data"` como um atributo da *tag form*.

3.2.11 *mysql*

Para conseguir realizar a ligação do servidor com a base de dados foi utilizada a biblioteca *mysql* [15] que cria uma conexão com a base de dados e permite fazer operações de consulta, inserção, atualização ou remoção para ser possível armazenar e gerir os dados que são necessários para a aplicação.

3.3 Engenharia de *Software*

A engenharia de *software* é uma área que ajuda a entender as necessidades do cliente e para este tema foram especificados os requisitos funcionais e não funcionais (subsecção 3.3.1), os diagramas UML (subsecção 3.3.2) e a modelação de dados (subsecção 3.3.3).

3.3.1 Análise de Requisitos

Para que a plataforma fosse melhor compreendida e como ia ser construída foram levantados os requisitos funcionais e não funcionais.

3.3.1.1 Requisitos Funcionais

Os requisitos funcionais descrevem as funcionalidades que o sistema deve ter para que as necessidades do cliente sejam atendidas. Estes requisitos vão ser apresentados nas tabelas 3.3.1.1 e 3.3.1.1.

Tabela 3.1: Requisitos funcionais da plataforma.

ID	Requisito Funcional
RF01	O utilizador deve poder aceder às páginas da plataforma.
RF02	O utilizador deve poder aceder aos documentos partilhados.
RF03	O utilizador deve poder aceder às páginas referenciadas numa publicação.
RF04	O utilizador deve poder mudar a linguagem da plataforma de português para inglês e vice-versa.
RF05	O utilizador deve poder criar uma conta.
RF06	O utilizador deve poder iniciar sessão.
RF07	O utilizador deve poder preencher formulários dos vários tipos de publicação, dependendo do privilégio que tiver.
RF08	O utilizador deve poder editar as publicações feitas.

Tabela 3.2: Continuação da tabela requisitos funcionais da plataforma.

ID	Requisito Funcional
RF09	O utilizador deve poder apagar as publicações feitas.
RF10	O utilizador deve poder mudar o <i>email</i> associado à conta.
RF11	O utilizador deve poder mudar a palavra-passe associada à conta.
RF12	O utilizador deve poder terminar sessão.
RF13	O utilizador deve poder aceitar contas, caso tenha o privilégio de um utilizador com todos os privilégios.
RF14	O utilizador deve poder rejeitar contas, caso tenha o privilégio de utilizador com todos os privilégios.
RF15	O utilizador deve poder apagar contas, caso tenha o privilégio de utilizador com todos os privilégios.

3.3.1.2 Requisitos Não Funcionais

Os requisitos não funcionais não estão diretamente ligados às funcionalidades do sistema, mas a outras características como a segurança e confiabilidade. Estes serão mostrados na tabela 3.3.1.2.

Tabela 3.3: Requisitos não funcionais da plataforma.

ID	Requisito Não Funcional
RNF01	A aplicação deve ser desenvolvida utilizando as tecnologias <i>JavaScript</i> , <i>HTML 5.0</i> e <i>CSS</i> .
RNF02	A aplicação deve ter a capacidade de ser executada num <i>browser</i> que interprete <i>JavaScript</i> .
RNF03	A aplicação deve poder armazenar as palavras-passe de forma segura.
RNF04	A plataforma deve seguir a marca gráfica do CACB.
RNF05	A plataforma deve ser responsiva, isto é, deve adaptar-se a diferentes tamanhos de ecrã.
RNF06	A plataforma deve permitir a incorporação de novas funcionalidades.

3.3.2 Diagramas UML

Após fazer o levantamento dos requisitos funcionais e não funcionais foram feitos os diagramas UML que se destinam a representar o sistema de uma forma simples e gráfica, ajudando os clientes a perceber, mesmo sem experiência técnica como a aplicação foi desenhada.

3.3.2.1 Diagramas de Caso de Uso

Os diagramas de caso de uso são diagramas que representam como os atores (utilizadores) interagem com a aplicação, fornecendo, assim, uma visão geral, não muito detalhada, do que o utilizador pode fazer. De seguida, serão apresentados três diagramas de caso de uso representados na figura 3.1, que mostra as funcionalidades gerais de um utilizador sem privilégios, na figura 3.2, que ilustra as funcionalidades gerais de um utilizador com privilégios, ou seja, um utilizador que já consegue efetuar o *login* e, por último, na figura 3.3, o utilizador com todos os privilégios, doravante denominado "super administrador", ou seja, o utilizador que poderá manipular contas e tem acesso a todas as outras funcionalidades da plataforma.

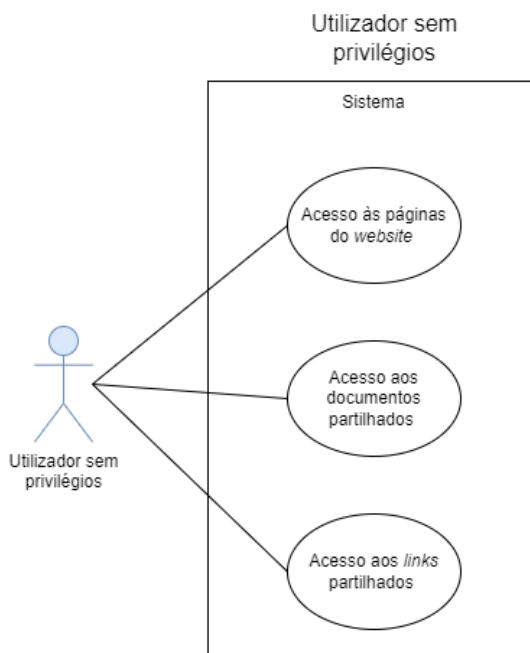


Figura 3.1: Diagrama de caso de uso de um utilizador sem privilégios.

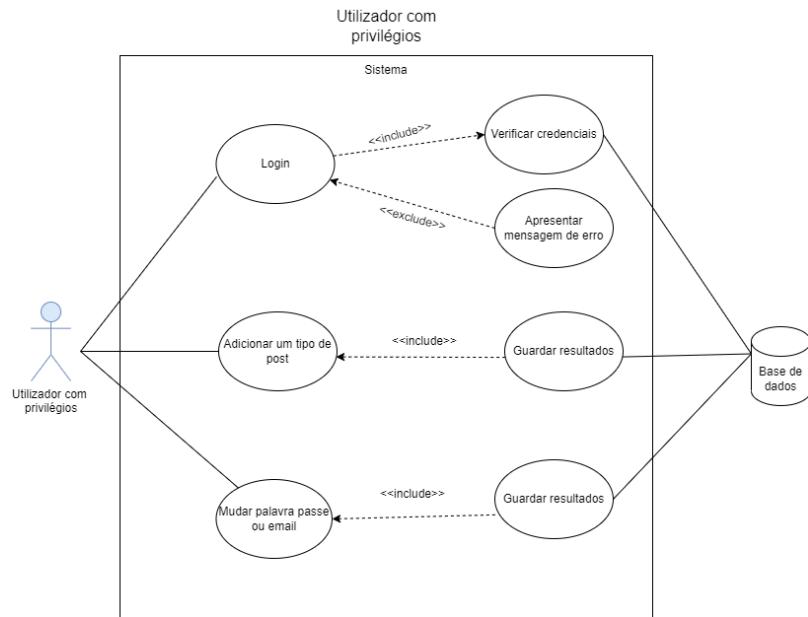


Figura 3.2: Diagrama de caso de uso de um utilizador com privilégios.

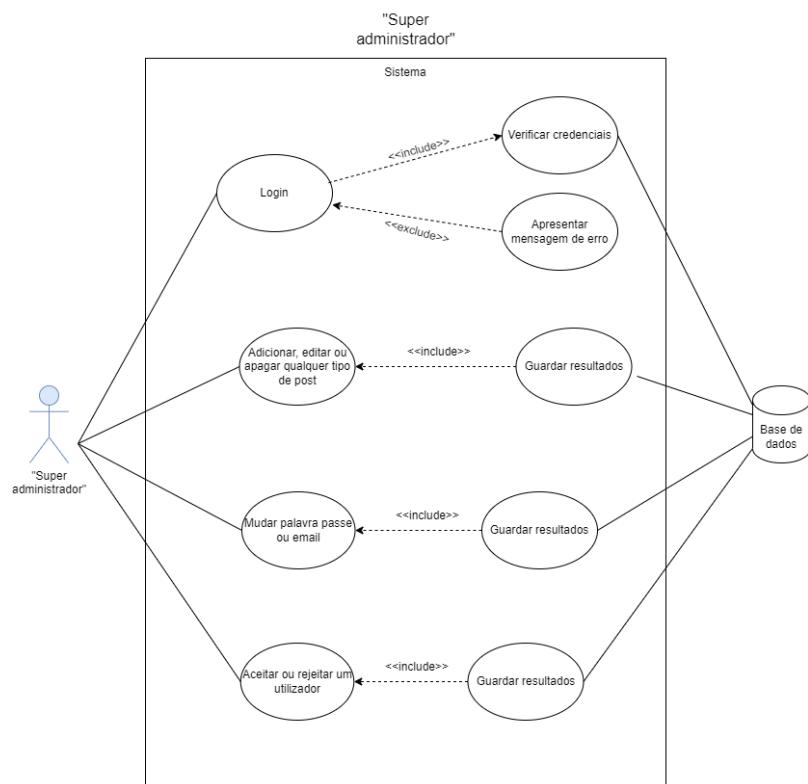


Figura 3.3: Diagrama de caso de uso de um "super administrador".

3.3.2.2 Diagramas de Atividade

Os diagramas de atividade são fluxogramas que ajudam a perceber como funciona uma atividade que descreve uma sequência de ações que se pode realizar numa aplicação.

Na figura 3.4, pode observar-se o diagrama de atividade referente à autenticação no sistema. Este é feito com nome de utilizador e palavra-passe, que caso estejam incorretos, é apresentada uma mensagem de erro e pode tentar iniciar sessão de novo e, caso estejam corretas é apresentada a página pessoal, que é diferente para cada utilizador, pois depende de que privilégio tem, que é definido aquando a criação da conta. No fim de utilizar a plataforma o utilizador pode terminar sessão.

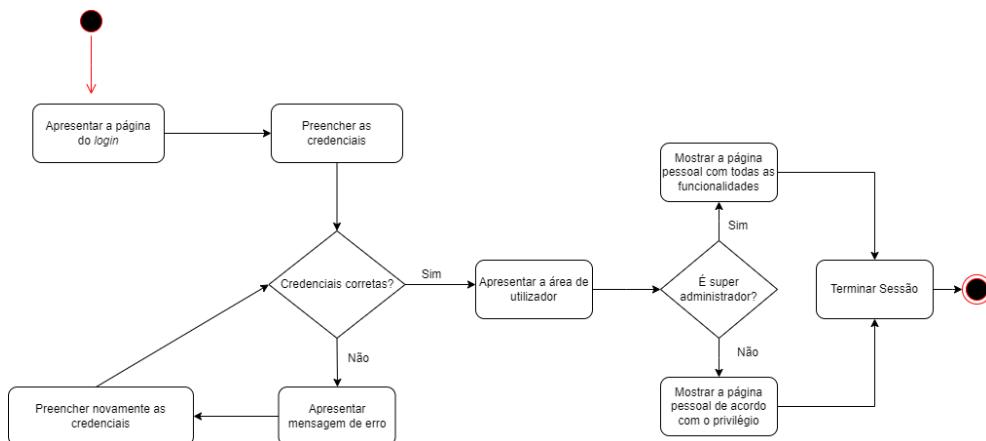


Figura 3.4: Diagrama de caso de atividade.

3.3.3 Modelação de Dados

A modelação de dados permite criar um modelo que representa como os dados serão armazenados num SGBD e qual é a melhor organização que devem ter, garantindo que os dados sejam armazenados de uma maneira lógica e coerente.

3.3.3.1 Modelo Conceptual

No modelo conceptual são identificadas as entidades, os relacionamentos e os atributos de cada entidade. Este modelo está representado na figura 3.5.

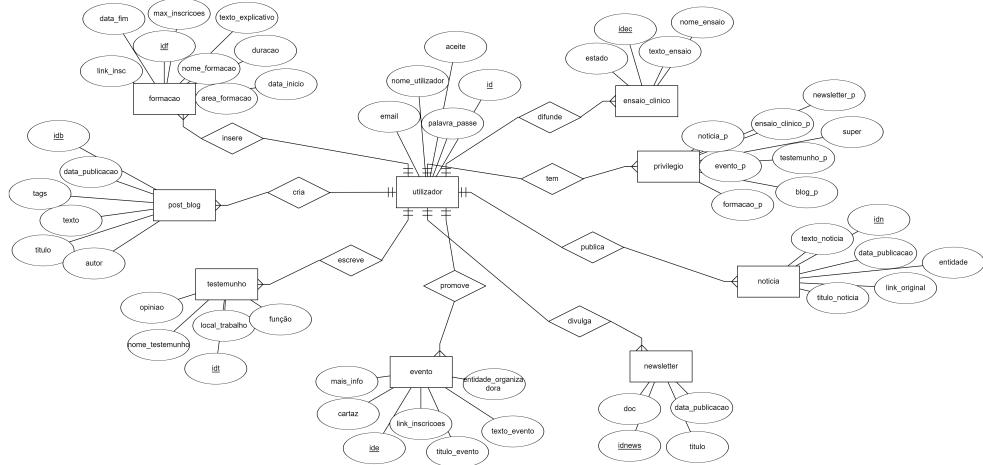


Figura 3.5: Modelo conceptual da aplicação.

3.3.3.2 Modelo Físico

No modelo físico são representadas as tabelas, colunas, as chaves primárias e as chaves estrangeiras de cada tabela, que está representado na figura 3.6.

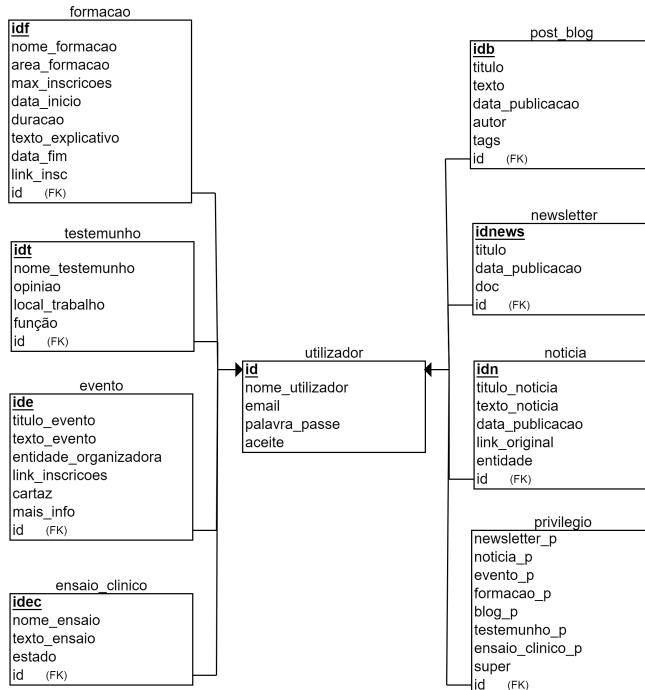


Figura 3.6: Modelo físico da aplicação.

3.4 Conclusões

Em suma, neste capítulo foram abordados os temas das tecnologias utilizadas e a razão pelas quais foram escolhidas e o tema da engenharia de *software*, em que foram explicitados os requisitos funcionais e não funcionais que levaram à criação dos diagramas UML e também da modelação de dados que melhor se adaptava à aplicação.

Capítulo

4

Funcionalidades e Implementação

4.1 Introdução

Este capítulo vai abordar na primeira secção (4.2) o funcionamento da aplicação e na segunda secção (4.3), a implementação da aplicação, apresentando excertos de código para melhor ilustrar o que foi executado.

4.2 Funcionalidades da Aplicação

Esta secção vai explicar as funcionalidades da aplicação *web*. A primeira parte vai ser referente ao que o utilizador sem privilégios vê, ou seja, é o que um utilizador que não tenciona criar uma conta pode visitar. Vão ser apresentadas imagens de certas páginas, mas não todas, pois a lógica de umas é aplicada a outras.

Na home page, representada nas figuras 4.1 e 4.2 pode observar-se duas partes que constituem a *home page*. Na figura 4.1 podem observar-se as três últimas publicações feitas para a página *Blog*, para onde o utilizador poderá navegar clicando no botão "Ver mais". Na figura 4.2 é possível ver os últimos três testemunhos publicados clicando nas setas que aparecem de cada lado do testemunho.



Figura 4.1: Captura de ecrã demonstrativa da primeira parte da *home page* da aplicação.



Figura 4.2: Captura de ecrã demonstrativa da segunda parte da *home page* da aplicação.

Para exemplificar, vai ser apresentada uma das páginas dinâmicas que podem ser visitadas, vai ser utilizado o exemplo da página Eventos, que mostra os eventos organizados pelo CACB ou por outras entidades que podem utilizar a plataforma do CACB para os partilhar. Na figura 4.3 pode observar-se um exemplo de como aparece uma publicação de um evento, nesta pode ver-se que tem três botões que podem levar ao *link* para inscrições, um que mostra a imagem ou o ficheiro do cartaz e outro que contém mais informações acerca do evento.

Também é possível observar a barra de navegação, que está presente em todas as páginas e é por lá que o utilizador muda para outras páginas, muda de linguagem ou encontra a página para fazer *login* ou criar conta.



Figura 4.3: Captura de ecrã demonstrativa da página Evento da aplicação.

De seguida, vai ser apresentada a parte do *backoffice* onde é possível criar conta, conforme representado na figura 4.4. No formulário de criar conta o utilizador tem de colocar um nome de utilizador à sua escolha, um email, a organização e a área da aplicação que pretende manipular. Quando clica no botão podem aparecer duas mensagens, uma caso já exista uma conta com o mesmo *email* ou uma que informa que a conta foi criada e que agora necessita de autorização do "super administrador" para conseguir efetuar o *login*.

Figura 4.4: Captura de ecrã demonstrativa da página Criar Conta da aplicação.

Na figura 4.5 pode observar-se o formulário de *login*, onde o utilizador preenche com o nome de utilizador e com a sua palavra-passe. Caso alguma credencial esteja errada é apresentada uma mensagem de erro e caso estejam corretas avança para a página pessoal.

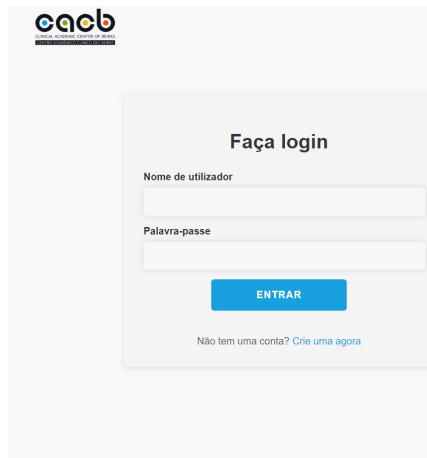


Figura 4.5: Captura de ecrã demonstrativa da página *Login* da aplicação.

Para mostrar um exemplo de uma página pessoal vai ser usada a conta do super administrador, pois é a mais completa. Como se pode observar na figura 4.6 (que se encontra mais pequena para conseguir ser toda colocada numa só figura), podem observar-se várias "caixas" que contêm duas opções para criar uma publicação ou para ver as publicações criadas e, no caso do super administrador, pode ver as contas pendentes e as contas aceites. É importante apontar que a página pessoal varia segundo o privilégio do utilizador, podem aparecer menos caixas caso tenha menos privilégios.

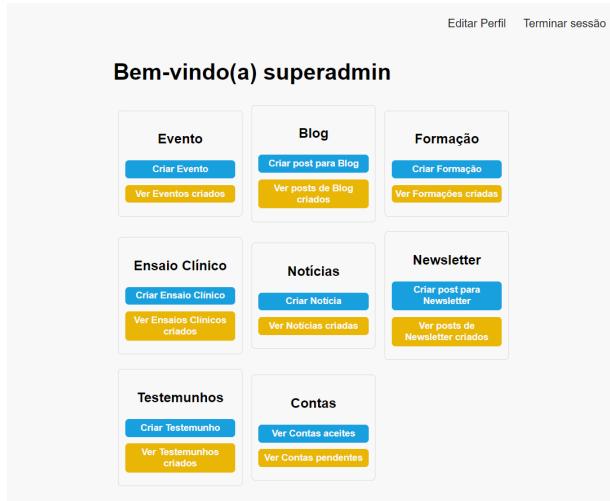


Figura 4.6: Captura de ecrã demonstrativa da página Página Pessoal da aplicação.

Quando se quer criar uma publicação, aparece um formulário adequado para cada um dos tipos de *post* e, neste caso, vai ser apresentado o formulário para criar um evento, conforme demonstrado na figura 4.7. Neste caso o utilizador pode colocar um título, uma descrição do evento, qual a entidade organizadora, a data do evento, link para inscrição, caso tenha, o cartaz e outros tipos de informações, que podem ser um ficheiro Portable Document Format (PDF) ou um link com alguma informação.

Figura 4.7: Captura de ecrã demonstrativa da página Criar Evento da aplicação.

Quando se quer ver os Eventos já criados o utilizador vai à página Ver Eventos criados, apresentada na figura 4.8, e aparece a listagem dos eventos que esse utilizador criou assim como dois botões onde pode escolher apagar ou editar uma publicação. Quando escolhe editar aparece um formulário igual e o utilizador escolhe o que quer editar.

Figura 4.8: Captura de ecrã demonstrativa da página Ver Eventos criados da aplicação.

Também é possível um utilizador alterar a palavra-passe ou o *email*, conforme representado na figura 4.9 onde o utilizador preenche qual dos campos pretende alterar.

A captura de ecrã mostra a interface de utilizador para editar o perfil. No topo, há uma barra com o título "Editar Perfil". Abaixo, uma secção "Alterar Email" com um campo "Novo Email:" e um botão "Alterar Email". Em seguida, uma secção "Alterar palavra-passe" com campos "Nova palavra-passe:" e "Confirmar palavra-passe", e um botão "Alterar palavra-passe".

Figura 4.9: Captura de ecrã demonstrativa da página Editar Perfil da aplicação.

O "super administrador" apresenta uma opção que não está disponível para os outros utilizadores. Esta serve para ver contas pendentes ou para ver as contas existentes. Na página onde pode ver as contas aceites, representada na figura 4.10, o "super administrador" tem a opção de apagar uma conta e de ver alguma informação do utilizador. Na página com as contas pendentes, representada na figura 4.11, o "super administrador" pode aceitar ou rejeitar uma conta.⁵

A captura de ecrã mostra a interface de utilizador para gerir contas aceitas. O topo tem o título "Contas Aceites". Abaixo, há três entradas, cada uma com o nome do utilizador, o email e uma opção "Apagar".

Contas Aceites		
teresa	Email: teresa@gmail.com	Organização: UBI
		<button>Apagar</button>
bruna	Email: brunamarques@gmail.com	Organização:
		<button>Apagar</button>
marta	Email: martaagodinho@gmail.com	

Figura 4.10: Captura de ecrã demonstrativa da página Contas Aceites da aplicação.

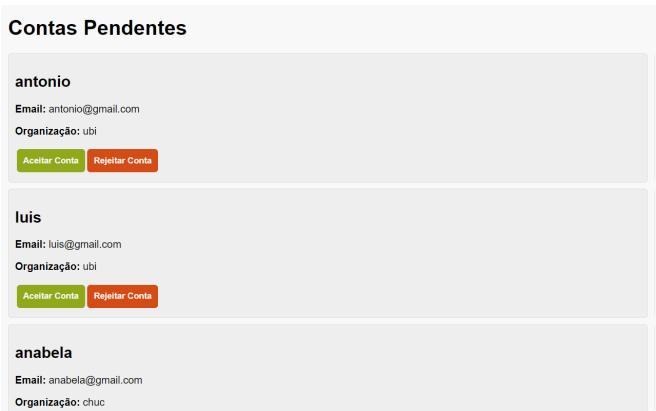


Figura 4.11: Captura de ecrã demonstrativa da página Contas Pendentes da aplicação.

4.3 Implementação

Para começar a implementar a aplicação foi criado um servidor *Node.js* e quando estava configurado foram importadas as várias bibliotecas para ser possível correr a aplicação. Como muitas páginas da aplicação seguem a mesma lógica, vão ser apresentados os excertos de código que representam as páginas apresentadas na secção 4.2.

4.3.1 Home Page

Para aparecerem as informações necessárias foi utilizado o código apresentado no excerto código A.1, que se encontra no A. Neste código encontra-se a uma rota `home_page` para fazer uma pesquisa na base de dados para apresentar as últimas três publicações de *blog* e os últimos três testemunhos. Para isso, foram realizadas duas pesquisas à base de dados, tendo-se especificado quais os campos que eram necessários, que no caso das publicações do *blog* eram só alguns em específico. Estes estão a aparecer pela ordem do último `idb` e `idt` para serem sempre apresentados as últimas três publicações. Caso exista algum erro são apresentadas mensagens a informar que as publicações não foram encontradas.

No excerto de código B.1, inserido no B mostra qual a organização das publicações de *blog* e testemunhos na *home page*. Para os *posts* do *blog* são feitas três "caixas" onde aparecem as informações, neste caso, não aparecem todas as informações como na página com todos os *posts*, pois o utilizador pode observar as informações completas na página *Blog*. Para não aparecer o

texto completo referente à publicação é limitado a 150 caracteres, mostrando só o início do texto.

Para os testemunhos todas as informações são apresentadas, mas de maneira diferente, pois só aparece um de cada vez e o utilizador pode clicar nas setas que aparecem de cada lado, como mostrado na 4.2, para ver os outros testemunhos. Para isso foi feita uma função `showTestimonial()` para mostrar o testemunho atual e ocultar os outros e é inicialmente chamada para mostrar o primeiro testemunho, sendo de seguida adicionado um `addEventListener` para quando se clica numa das setas. A função `showTestimonial()` é inicializada a zero e quando é clicada a variável `currentTestimonial` é atualizada para o índice anterior ou seguinte, dependendo em que seta se clica. Por fim, a função `showTestimonial()` é chamada novamente para atualizar o testemunho a aparecer.

4.3.2 Página Eventos

Para exemplificar o código de uma das páginas dinâmicas vai ser apresentada a página Eventos, pois é uma das que apresenta mais funcionalidades. Do lado do servidor foi feita uma rota `/eventos`, como mostra o excerto de código C.1, para fazer uma pesquisa à base de dados, na tabela eventos, e são apresentados por ordem do último `i de` para aparecer o mais recente primeiro. Para serem apresentados apenas dez eventos por página, as variáveis `perPage` define o número de eventos a aparecer por página, a variável `page` é definida com o valor do parâmetro de consulta "page" do `request`, e se o parâmetro não estiver presente, o valor padrão é um. A variável `offset` calcula quais as publicações que devem aparecer nas páginas, segundo as páginas que já foram passadas.

```
app.get('/eventos', function(req, res) {
  const perPage = 10;
  const page = parseInt(req.query.page) || 1;
  const offset = (page - 1) * perPage;
  connection.query('SELECT * FROM evento ORDER BY ide DESC LIMIT ?',
    [offset], [perPage, offset], function(err, rows) {
    if (err) throw err;
    connection.query('SELECT COUNT(*) AS count FROM evento', function(
      err, countRows) {
      if (err) throw err;
      const totalCount = countRows[0].count;
      const totalPages = Math.ceil(totalCount / perPage);
      res.render('eventos', {
        eventos: rows,
        currentPage: page,
        totalPages: totalPages
      })
    })
  })
})
```

```

    } , function(err , html) {
      if (err) throw err;
      res.send(html);
    }) ; });
  });
}

```

Excerto de Código 4.1: Rota para apresentar a página Eventos.

No excerto de código C.1, que se encontra no C é mostrada a organização do conteúdo da página Eventos, onde todo o *array* eventos é percorrido para depois cada elemento evento seja percorrido e mostrado com o estilo e pela ordem escolhida. Para que o texto sobre o evento apareça com as quebras de linha corretamente apresentadas foi colocada a propriedade `style="white-space: pre-line"`. Como existem alguns botões que podem ou não aparecer é feita uma verificação condicional para que só apareça o que o utilizador inseriu quando criou o evento.

Como é feita a condição de só aparecerem dez publicações por página foi realizado o que se encontra no excerto de código 4.2. Este código verifica se o total de páginas é maior que um, caso seja renderiza os botões "Anterior" ou "Seguinte".

```

<div class="pagination">
  <% if (totalPages > 1) { %>
  <% if (currentPage > 1) { %>
    <a href="/eventos?page=<%= currentPage - 1 %>" class="pagination-button">Anterior</a>
  <% } %>
  <% if (currentPage < totalPages) { %>
    <a href="/eventos?page=<%= currentPage + 1 %>" class="pagination-button">Seguinte</a>
  <% } %>
  <% } %>
</div>

```

Excerto de Código 4.2: Paginação para aparecerem apenas dez publicações por página.

4.3.3 Criar Conta

Quando o utilizador cria uma conta é feito um post para definir uma rota para o endpoint /criar-conta. Como se pode observar no excerto de código 4.3 são definidas as variáveis para depois serem definidos os privilégios de um utilizador. Caso o utilizador preencha a *checkbox* no formulário para criar conta esse valor fica a um, caso não preencha, esse valor fica a zero.

```
app.post('/criar-conta', (req, res) => {
  let { nome_utilizador, email, organizacao, palavra_passe,
        confirm_password, newsletter, news, events, education, blog,
        testimonials, "clinical-trials": clinicalTrials } = req.body;
  const newsletterChecked = newsletter === 'on' ? 1 : 0;
  const newsChecked = news === 'on' ? 1 : 0;
  const eventsChecked = events === 'on' ? 1 : 0;
  const educationChecked = education === 'on' ? 1 : 0;
  const blogChecked = blog === 'on' ? 1 : 0;
  const testimonialsChecked = testimonials === 'on' ? 1 : 0;
  const clinicalTrialsChecked = clinicalTrials === 'on' ? 1 : 0;
  ...
});
```

Exerto de Código 4.3: Rota post para criar uma conta e definição das variáveis para definir o privilégio.

Para que a palavra-passe seja guardada com segurança na base de dados, é utilizada a biblioteca *bcrypt*, como se pode observar no exerto de código 4.4 onde a função *genSalt* gera um *salt*, com a variável *saltRounds* com um número de rondas especificado, no caso são definidas dez rondas. De seguida, a função *hash* gera o *hash* da palavra-passe escolhida utilizando o *salt* gerado anteriormente

```
bcrypt.genSalt(saltRounds, function(err, salt) {
  bcrypt.hash(palavra_passe, salt, function(err, hash) {
    ...
});
```

Exerto de Código 4.4: Uso da biblioteca *bcrypt* para gerar o hash da palavra-passe.

O restante código desta rota verifica se já existe um *email* ou nome de utilizador igual ao que foi escolhido e se a palavra-passe e a sua confirmação são iguais. Se uma destas acontecer é apresentada uma mensagem de erro e o utilizador muda o que necessita, se não acontecer a conta é criada com sucesso e aparece uma mensagem que informa que a conta precisa de ser aceite.

No lado do cliente é feito um formulário simples em HTML com o respetivo estilo, com os campos de *input* para o nome de utilizador, *email*, organização, palavra-passe, confirmação da palavra-passe e as *checkboxes* para escolher quais as áreas da aplicação que pretende manipular.

No exerto de código 4.5, é feita uma verificação condicional para verificar se a variável *message_sucesso* está definida, utilizando o comando *typeof* para verificar o tipo de dado da variável. Se a variável for diferente de *undefined*, o código dentro do bloco *if* é executado.

```
<% if (typeof message_sucesso !== 'undefined') { %>
    <p class="match"><%= message_sucesso %></p>
<% } %>
```

Exerto de Código 4.5: Apresentar mensagem de sucesso ao criar a conta.

Quando é uma mensagem de erro, é executado o que está no exerto de código 4.6, que mostra uma verificação condicional para verificar se a variável message está definida e se estiver, irá aparecer uma mensagem com o que é necessário mudar.

```
<% if (message) { %>
    <p class="error"><%= message %></p>
<% } %>
```

Exerto de Código 4.6: Apresentar mensagem de erro ao criar a conta.

4.3.4 Login

Para realizar o *login*, o utilizador necessita de preencher um formulário com as suas credenciais. Caso uma delas esteja errada é apresentada uma mensagem de erro definida da mesma maneira que no exerto de código 4.6.

No lado do servidor, foi definida uma rota `/login`, como se pode observar no exerto de código 4.7. Aqui é feita uma pesquisa na base de dados para verificar se ele existe, de seguida é verificado se esse utilizador já foi aceite e caso esteja é utilizado o `bcrypt.compare` para comparar a palavra-passe inserida com a que existe na base de dados e se as duas coincidirem a variável `passwordMatch` é verdadeira.

```
app.post('/login', (req, res) => {
    let nome_utilizador = req.body.username;
    let palavra_passe = req.body.password;
    let sql = "SELECT * FROM utilizador WHERE nome_utilizador=?";
    connection.query(sql, [nome_utilizador], (err, result) => {
        if (err) throw err;
        if (result.length > 0) {
            let utilizador = result[0];
            if (utilizador.aceite == 1) {
                bcrypt.compare(palavra_passe, utilizador.palavra_passe, (err,
                    passwordMatch) => {
                    if (err) throw err;
                    ...
                });
            }
        }
    });
});
```

Exerto de Código 4.7: Rota `/login` para efetuar o login e pesquisa na base de dados para encontrar o utilizador.

No excerto de código D.1, localizado no D, mostra como foi feita a verificação de um privilégio de um utilizador. Para isso, quando a variável password-Match, mencionada na subsecção 4.3.3, for verdadeira, o nome de utilizador e a palavra-passe vão ficar armazenadas na `req.session`. A seguir é feita uma consulta à tabela "privilegio" para obter os privilégios do utilizador. Se estes existirem, são extraídos e guardados num objeto `privileges`. De seguida os valores de sessão são configurados com os privilégios do utilizador. Caso a comparação de palavra-passe falhe ou o utilizador não tenha autorização para aceder ou o utilizador não tenha sido encontrado é mostrada uma mensagem de erro. Se tudo estiver correto é renderizada a página pessoal.

4.3.5 Página Pessoal

Quando o login é efetuado com sucesso, é apresentada a página pessoal, são apresentadas apenas as informações segundo o privilégio do utilizador. No excerto de código 4.8 é feita uma verificação para ver se o *id* do utilizador está presente na sessão, caso não esteja presente significa que o utilizador não está autenticado e é redirecionado para a página de *login*. Se estiver presente na sessão o código recupera o *id* do utilizador e de seguida é feita uma consulta à base de dados para recolher as informações do utilizador e de seguida os dados do utilizador são guardados como uma variável local `res.locals.usuario`, que permite que os dados do utilizador sejam acessíveis no modelo da página pessoal quando é renderizada.

```
app.get('/pagina_pessoal', (req, res) => {
  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }
  let userId = req.query.id_usuario;
  connection.query('SELECT * FROM usuario WHERE idu = ?', [userId],
    (error, results) => {
      if (error) throw error;
      let usuario = results[0];
      res.locals.usuario = usuario;
      res.render('pagina_pessoal');
    });
});
```

Exerto de Código 4.8: Rota /pagina_pessoal para apresentar a página pessoal.

Como existem privilégios e podem aparecer certas caixas e outras não é feito uma condição para verificar se a variável `priv_usuario` existe e se o atributo `evento` está presente, como se pode observar no excerto de código

4.9. Esta condição existe para todos os tipos de área que se pretende manipular sempre com a mesma lógica. Depois de fazer esta verificação ambas as condições forem verdadeiras é mostrada a "caixa" dos eventos que tem duas opções, criar um evento ou ver os eventos que esse utilizador criou.

```
<% if (priv_utilizador && priv_utilizador.evento) { %>
    <div class="box" id="eventoBox" style="display: none
        ;">
        <h3 class="h3_pp">Evento</h3>
        <button type="button" onclick="window.location.
            href='/criar_evento'">Criar Evento</button>
        <button type="button" class="bt_editar" onclick=
            "window.location.href='/ver_eventos'">Ver
            Eventos criados</button>
    </div>
<% } %>
```

Excerto de Código 4.9: Condição para mostrar ou não a "caixa" dos eventos.

O excerto de código 4.10 faz a verificação da existência da variável `priv_utilizador` e, para isso o código dentro do bloco `if` é executado e se a variável existir renderiza um `div` com o `id = "user-evento"` e um atributo de dados `data-id` que recebe o atributo `evento` da variável `priv_utilizador`.

```
<% if (priv_utilizador) { %>
    <div id="user-evento" data-id=<%= priv_utilizador.evento %>
        "></div>
<% } %>
```

Excerto de Código 4.10: Verificação da existência da variável `priv_utilizador`.

4.3.6 Criação de um Evento

No excerto de código E.1, apresentado no E, mostra como a rota `novo_evento` foi criada. Primeiro os dados são extraídos do corpo da solicitação (`req.body`) e atribuídos a variáveis individuais e o `id` do utilizador é guardado na variável `id_utilizador`. Depois, é criado um objeto `novoEvento` e é feita a verificação da existência das propriedades `cartaz` e `mais_info1` pois estes são `inputs` de ficheiros. Depois é feita uma inserção dos dados na base de dados, renderizando a página pessoal de novo.

Do lado do cliente, foi feito um formulário simples para se preencher com os dados necessários com os vários campos, sendo alguns opcionais e outros obrigatórios, sendo colocada a condição do input ser `required`.

4.3.7 Ver Eventos Criados

No excerto de código 4.11 é feita uma consulta à tabela eventos para serem mostrados os eventos que aquele utilizador específico criou, vendo qual é o id do utilizador pela chave estrangeira da tabela, fk_idu e os eventos que esse utilizador criou aparecem por ordem do mais recente para o mais antigo.

```
app.get( '/ver_eventos' , function(req , res) {  
    if (!req.session.userID) {  
        res.redirect( '/login' );  
        return;  
    }  
    let userID = req.session.userID;  
    connection.query( 'SELECT * FROM evento WHERE fk_idu = ? ORDER BY ide  
        DESC' , [userID] , function(err , rows) {  
        if (err) throw err;  
        res.render( 'ver_eventos' , {  
            eventos: rows  
        });  
    });  
});
```

Exerto de Código 4.11: Rota ver_eventos para ir buscar a informação dos eventos da base de dados.

Do lado do cliente são mostradas várias "caixas" e cada uma está associada a um evento e o utilizador pode escolher duas opções editar a publicação ou apagá-la.

4.3.8 Editar um Evento

Para fazer a edição de uma publicação é primeiro feita uma consulta à base de dados para selecionar o evento especificado. De seguida, como se pode observar no excerto de código 4.12, os dados do req.body são obtidos e de seguida é criada uma variável updateFields para guardar os campos que serão atualizados no evento. Se algum campo não estiver vazio, esse campo é adicionado ao objeto updateFields, caso os inputs sejam ficheiros é feito um replace do prefixo public ao caminho do ficheiro.

```
let eventoID = req.body.eventoID;  
let { titulo_evento , texto_evento , entidade_organizadora , data_evento ,  
      link_inscricoes , mais_info2 } = req.body;  
let updateFields = {};  
if (titulo_evento !== "") {  
    updateFields.titulo_evento = titulo_evento;  
}  
...  
if (req.files && req.files.cartaz) {
```

```

    updateFields.cartaz = req.files.cartaz[0].path.replace(/ public\\/, '/');
}
}

```

Exerto de Código 4.12: Verificação da edição dos campos do formulário.

Depois, como se pode observar no exerto do código 4.13, é verificado se existem campos a ser atualizados no objeto updateFields e se houver é feita uma atualização na base de dados na tabela evento utilizando também o id do evento selecionado.

```

if (Object.keys(updateFields).length > 0) {
  connection.query(
    'UPDATE evento SET ? WHERE ide = ?',
    [updateFields, eventoID],
    function (err, result) {
      if (err) {
        console.error('Erro ao atualizar o evento:', err);
      } else {
        if (result.affectedRows > 0) {
          console.log('Evento atualizado com sucesso!');
        } else {
          console.log('Nenhum registo encontrado para o ID do evento
            especificado: ' + eventoID);
        }
      }
    }
}

```

Exerto de Código 4.13: Atualização da base de dados.

4.3.9 Editar Perfil

O utilizador tem a opção de alterar o seu *email* ou a sua palavra-passe e pode fazê-lo preenchendo o formulário adequado. Para alterar o *email*, foi feita uma rota `alterar_email`, como pode ser observado no exerto de código F1, que se encontra no F1. O novo email é obtido a partir dos dados enviados pelo formulário para a variável `newEmail` e o id do utilizador é obtido a partir da sessão `userID`, e é feita uma consulta à base de dados para verificar se o novo *email* já existe na base de dados e caso exista é apresentada uma mensagem de erro. Caso contrário o email foi alterado com sucesso.

Para editar a palavra passe, a lógica é semelhante, no entanto para guardar a nova palavra passe é necessário gerar o *hash*, como se pode observar no exerto de código 4.14. Também é feita uma verificação entre a palavra-passe e a sua confirmação, caso não coincidam é apresentada uma mensagem de erro. Se coincidirem é gerado o novo *hash* com a função `bcrypt.hash()`.

```

let { newPassword, confirmPassword } = req.body;
let userId = req.session.userID;
if (newPassword !== confirmPassword) {
  return res.render('editar_perfil', {
    message: 'A nova palavra-passe e a confirmacao nao coincidem. Por favor, tente novamente.'
  });
}
bcrypt.hash(newPassword, 10, (error, hashedNewPassword) => {
  if (error) {
    return res.render('editar_perfil', {
      message: 'Ocorreu um erro ao gerar o hash da nova palavra-passe. Por favor, tente novamente.'
    });
}

```

Excerto de Código 4.14: Alteração da palavra-passe.

4.3.10 Contas Pendentes

Na página das contas pendentes é feita uma consulta à base de dados, como se pode ver no excerto de código 4.15, onde são selecionados os utilizadores que têm o campo aceite = 0, o que significa que as contas ainda não foram aprovadas.

```

app.get('/contas_pendentes', function(req, res) {
  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }
  connection.query('SELECT idu, nome_utilizador, email, organizacao FROM
    utilizador WHERE aceite = 0 ORDER BY idu DESC', function(err,
    rows) {
    if (err) throw err;
    res.render('contas_pendentes', {
      contas: rows
    });
  });
});

```

Excerto de Código 4.15: Consulta à base de dados para ver as contas pendentes.

Do lado do cliente aparece uma listagem com as contas pendentes e o "super administrador" tem duas opções: aceitar ou rejeitar uma conta. No excerto de código 4.16, é apresentada a rota aceitar_conta que atualiza o estado do campo aceite na tabela utilizador, quando se clica no botão

"Aceitar", para ficar com o valor um, ficando assim a conta aceite e o utilizador já possa efetuar login.

```
app.post('/aceitar_conta', (req, res) => {
  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }
  let contaID = req.body.contaID;
  connection.query(
    'UPDATE utilizador SET aceite = 1 WHERE idu = ?',
    [contaID],
    function (err, result) {
      if (err) {
        console.error('Erro ao aceitar a conta:', err);
      } else {
        if (result.affectedRows > 0) {
          console.log('Conta aceite com sucesso!');
          res.redirect('back');
        } else {
          console.log('Nenhum registo encontrado para o ID da conta
                     especificada');
        }
      }
    });
});});
```

Exerto de Código 4.16: Atualização do estado da conta para aceite.

Caso o "super administrador" decida rejeitar a conta foi feito um código com a mesma lógica, no entanto, o campo aceite fica com o valor dois, como se pode observar no exerto de código 4.17, e assim o utilizador fica impedido de realizar o *login*.

```
connection.query(
  'UPDATE utilizador SET aceite = 2 WHERE idu = ?',
  [contaID],
  function (err, result) {
    if (err) {
      console.error('Erro ao rejeitar a conta:', err);
    } else {
      if (result.affectedRows > 0) {
        console.log('Conta rejeitada com sucesso!');
        res.redirect('back');
      } else {
        console.log('Nenhum registo encontrado para o ID da conta
                   especificada');
      }
    }
});
```

Exerto de Código 4.17: Atualização do estado da conta para rejeitado.

4.3.11 Contas Aceites

O "super administrador" pode também ver a listagem das contas aceites, como mostra o excerto de código 4.18, que faz uma consulta à base de dados para mostrar as contas que têm o campo aceite = 1.

```
app.get('/contas_aceites', function(req, res) {
  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }
  connection.query('SELECT idu, nome_utilizador, email, organizacao FROM
    utilizador WHERE aceite = 1 ORDER BY idu DESC', function(err,
    rows) {
    if (err) throw err;
    res.render('contas_aceites', {
      contas: rows
    });
  });
});
```

Exerto de Código 4.18: Consulta à base de dados para ver as contas que já foram aceites.

O "super administrador" pode também apagar uma conta, como se pode observar no excerto de código 4.19.

```
app.post('/apagar_conta', (req, res) => {
  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }
  let contaID = req.body.contaID;
  let sql = 'DELETE FROM utilizador WHERE idu = ${contaID}';
  connection.query(sql, (err, result) => {
    if (err) {
      throw err;
    }
    console.log('Conta apagada');
    res.redirect('back');
  });
});
```

Exerto de Código 4.19: Remoção de uma conta da base de dados.

4.4 Conclusões

Concluindo, algumas páginas seguem a mesma lógica para serem construídas, no entanto, cada uma tem o seu conteúdo específico. Neste capítulo fo-

ram mostradas as funcionalidades da aplicação, assim como a implementação dessas funcionalidades, e quais foram os passos para poder construir esta aplicação.

Capítulo

5

Manual de Instalação e Testes

5.1 Introdução

Este capítulo vai abordar na primeira secção (5.2) o manual de instalação da aplicação, onde é explicado o que fazer para a aplicação funcionar, na segunda secção (5.3), os testes feitos para garantir que todos os requisitos estão a funcionar e, por último, na secção 5.4, onde é explicado como funciona a aplicação.

5.2 Manual de Instalação

Nesta secção vai ser explicitado o que fazer para instalar a aplicação como a inicializar. O manual de instalação fornece os passos necessários para configurar e instalar o servidor.

Para começar é necessário fazer a instalação do *Node.js* [16] e do *npm*, que já vem incluído na instalação do *Node.js*. Como o servidor já está configurado é só descarregar a diretoria com todos os ficheiros e instalar todas as dependências para ser possível inicializar a aplicação. Essa instalação pode ser feita com o comando inserido no excerto de código 5.1 na linha de comandos aberta na diretoria do projeto. Para abrir a pasta específica na linha de comandos, é necessário clicar com o botão direito do rato na opção "Abrir no Terminal".

```
npm install express express-session ejs express-validator multer path
```

Excerto de Código 5.1: Comando para instalar as bibliotecas necessárias.

Depois de se instalarem todas as dependências é necessário inicializar o projeto, também na linha de comandos com o comando descrito no excerto de código 5.2 ou com comando descrito no excerto de código 5.3

```
npm start
```

Excerto de Código 5.2: Comando para inicializar o projeto.

```
node server.js
```

Excerto de Código 5.3: Outra opção de comando para inicializar o projeto.

Para fazer a manutenção da base de dados pode instalar-se um *software* dedicado a isso mesmo. No caso foi instalado o *HeidiSQL* [17] onde foi criado a base de dados utilizando a interface do sistema. Mas antes foi criado um utilizador para aceder à base de dados específica com os comandos no excerto de código 5.4. Os asteriscos servem para representar o nome da base de dados e o segundo, a tabela específica, que neste caso são todas, por isso, fica o asterisco, pois este símbolo representa tudo.

```
CREATE USER 'novo_utilizador'@'localhost' IDENTIFIED BY 'palavra-passe';
CREATE DATABASE database ;
GRANT ALL PRIVILEGES ON base_dados . * TO novo_utilizador '@'localhost';
FLUSH PRIVILEGES;
```

Excerto de Código 5.4: Comandos para criar um novo utilizador e para lhe dar os privilégios.

Depois destes passos é necessário criar as tabelas, isto pode ser feito com o excerto código G.1, presente no G.

5.3 Testes

5.4 Manual de Utilização

5.5 Conclusões

Capítulo

6

Conclusões e Trabalho Futuro

6.1 Conclusões Principais

Esta secção contém a resposta à questão:

Quais foram as conclusões principais a que o(a) aluno(a) chegou no fim deste trabalho?

6.2 Trabalho Futuro

Esta secção responde a questões como:

O que é que ficou por fazer, e porque?

O que é que seria interessante fazer, mas não foi feito por não ser exatamente o objetivo deste trabalho?

Em que outros casos ou situações ou cenários – que não foram estudados no contexto deste projeto por não ser seu objetivo – é que o trabalho aqui descrito pode ter aplicações interessantes e porque?

Apêndice

A

Excerto de código para mostrar a rota da home page

Neste apêndice é apresentado um excerto de código mostrar como foi realizada a rota da *home page* para aparecerem as informações necessárias.

```
app.get('/home_page', (req, res) => {
  const blogQuery = 'SELECT titulo, texto, autor, data_publicacao FROM
    post_blog ORDER BY idb DESC LIMIT 3';
  const testemunhosQuery = 'SELECT * FROM testemunho ORDER BY idt DESC
    LIMIT 3';
  connection.query(blogQuery, (blogError, blogResults) => {
    if (blogError) {
      console.error('Erro ao encontrar posts do blog:', blogError);
      res.render('home_page', { message: 'Erro ao encontrar posts do
        blog' });
    } else {
      connection.query(testemunhosQuery, (testemunhosError,
        testemunhosResults) => {
        if (testemunhosError) {
          console.error('Erro ao encontrar testemunhos:',
            testemunhosError);
          res.render('home_page', { message: 'Erro ao encontrar
            testemunhos' });
        } else {
          res.render('home_page', { blogs: blogResults, testemunhos:
            testemunhosResults });
        }
      });
    }
  });
});
```

Excerto de Código A.1: Rota para apresentar a *home page*.

Apêndice

B

Excerto de código para apresentar a organização dos posts de blog e testemunhos

Neste apêndice é apresentado um excerto de código mostrar como foram organizadas as informações a aparecer na *home page* para as publicações do *blog* e dos testemunhos.

```
<h2 style="display: flex; align-items: center;">Blog</h2>
<div class="blog-row">
<% if (blogs && blogs.length > 0) { %>
<% blogs.forEach(function(blog) { %>
<div class="blog">
<h3 class="titulos_blog" style="color: #333333; font-size: 18px"><%= blog.titulo %></h3>
<div class="autor-data">
<p style="margin-top: 1.5vh; color: #555555; font-size: 14px"><%= blog.autor %> |</p>
<p style="margin-top: 1.5vh; color: #555555; font-size: 14px"><span class="data-publicacao"><%= blog.data_publicacao %></span></p>
</div>
<p style="margin-top: 1vh; color: #777777"><%= blog.texto.substring(0, 150) + '...' %></p>
</div>
<% }); %>
<% } else { %>
<p>Nenhum blog encontrado.</p>
<% } %>
</div>
<div class="ver-mais-button-container">
```

Exerto de código para apresentar a organização dos *posts* de *blog* e testemunhos

```

<button class="ver-mais-button" onclick="window.location.href='/
blog'" data-translate="ver_mais">Ver mais</button>
</div>

<h2 style="display: flex; align-items: center;" data-translate="
dizem">O que dizem sobre nos</h2>
<div class="testimonial-container">
  <div class="arrow arrow-prev"></div>
  <div class="testimonial">
    <% if (testemunhos && testemunhos.length > 0) { %>
      <% testemunhos.forEach(function(testemunho, index) { %>
        <div class="testimonial-item <%= index === 0 ? 'active' : '' %>">
          <h3 class="testimonial-name"><%= testemunho.nome_testemunho
          %></h3>
          <h5 class="testimonial-funcao"><%= testemunho.funcao %> @ <%=
          testemunho.local_trabalho %> </h5>
          <p class="testimonial-text"><%= testemunho.opiniao %></p>
        </div>
      <% } ); %>
    <% } else { %>
      <p>Nenhum testemunho encontrado.</p>
    <% } %>
  </div>
  <div class="arrow arrow-next"></div>
</div>

<script>
  const testimonialItems = document.querySelectorAll('.testimonial-item'
  );
  const arrows = document.querySelectorAll('.arrow');
  let currentTestimonial = 0;

  function showTestimonial() {
    testimonialItems.forEach(function(item, index) {
      item.style.display = 'none';
      if (index === currentTestimonial) {
        item.style.display = 'block';
      }
    });
  }

  showTestimonial();

  arrows.forEach(arrow => {
    arrow.addEventListener('click', () => {
      if (arrow.classList.contains('arrow-prev')) {
        currentTestimonial = (currentTestimonial - 1 + testimonialItems.
        length) % testimonialItems.length;
      }
    });
  });
</script>
```

```
    } else {
        currentTestimonial = (currentTestimonial + 1) % testimonialItems
            .length;
    }
    showTestimonial();
});
});
</script>
```

Excerto de Código B.1: Organização das publicações de *blog* e testemunhos na *home page*.

Apêndice

C

Excerto de código para apresentar a organização conteúdo da página Eventos

Neste apêndice é apresentado um excerto de código mostrar como foi organizado o conteúdo a aparecer na página eventos.

```
<div id="eventos-container">
    <% if (eventos && eventos.length > 0) { %>
    <% eventos.forEach(function(evento) { %>
        <div class="evento">
            <h2><%= evento.titulo_evento %></h2>
            <div class="organizador-data">
                <p style="color: #262626;"><%= evento.
                    entidade_organizadora %> |</p>
                <p><span class="data-evento"><%= evento.data_evento
                    %></span></p>
            </div>
            <p style="white-space: pre-line;"><%= evento.texto_evento
                %></p>

            <% if (evento.link_inscricoes || evento.cartaz || evento.
                mais_info1 || evento.mais_info2) { %>
                <div class="botoes-container">
                    <% if (evento.link_inscricoes) { %>
                        <button class="botao-inscricao" onclick="window.open
                            ('<%= evento.link_inscricoes %>', '_blank', '
                           noopener,noreferrer')">Inscricoes</button>
                    <% } %>
                    <% if (evento.cartaz) { %>
                        <a class="botao-cartaz" href="<%= evento.cartaz %>">
```

Exerto de código para apresentar a organização conteúdo da página

50

```
target="_blank" rel="noopener noreferrer">>Cartaz
</a>
<% } %>

<% if (evento.mais_info1) { %>
    <a class="botao-maisinfo1" href="<%= evento.
        mais_info1 %>" target="_blank" rel="noopener
        noreferrer">Mais Informacao</a>
<% } %>
<% if (evento.mais_info2) { %>
    <a class="botao-maisinfo2" href="<%= evento.
        mais_info2 %>" target="_blank" rel="noopener
        noreferrer">Mais Informacao</a>
<% } %>

</div>
<% } %>

</div>
```

Exerto de Código C.1: Organização do conteúdo página Eventos.

Apêndice

D

Excerto de código para mostrar a verificação de privilégio de um utilizador

Neste apêndice é apresentado um excerto de código mostrar como é feita a verificação do privilégio de um utilizador.

```
if (passwordMatch) {
    req.session.userID = utilizador.idu;
    req.session.userName = utilizador.nome_utilizador;

    let privilegeQuery = "SELECT super_admin, newsletter_p,
        noticia_p, evento_p, formacao_p, blog_p, testemunho_p,
        ensaio_clinico_p FROM privilegio WHERE fkp_idu=?";
    connection.query(privilegeQuery, [utilizador.idu], (err,
        privilegeResult) => {
        if (err) throw err;

        if (privilegeResult.length > 0) {
            let userPrivileges = privilegeResult[0];

            let privileges = {
                super_admin: userPrivileges.super_admin,
                newsletter: userPrivileges.newsletter_p,
                noticia: userPrivileges.noticia_p,
                evento: userPrivileges.evento_p,
                formacao: userPrivileges.formacao_p,
                blog: userPrivileges.blog_p,
                testemunho: userPrivileges.testemunho_p,
                ensaio_clinico: userPrivileges.ensaio_clinico_p
            };
        }
    });
}
```

Exerto de código para mostrar a verificação de privilégio de um utilizador

```
req.session.userPriv = privileges;

res.render('pagina_pessoal', {
  id_utilizador: req.session.userID,
  nome_utilizador: req.session.userName,
  priv_utilizador: privileges,
  message: null
});
} else {
  res.render('pagina_pessoal', {
    id_utilizador: req.session.userID,
    nome_utilizador: req.session.userName,
    priv_utilizador: null,
    message: null
  });
}
});
} else {
  let message = {
    error: 'Password errada'
  };
  res.render('login', {
    message: message.error
  });
}
});
} else {
  let message = {
    error: 'Nao tem autorizacao para aceder a aplicacao'
  };
  res.render('login', {
    message: message.error
  });
}
}
} else {
  let message = {
    error: 'Utilizador nao encontrado'
  };
  res.render('login', {
    message: message.error
  });
}
});
});
```

Exerto de Código D.1: Verificação do privilégio de um utilizador.

Apêndice

E

Excerto de código para apresentar como se cria um evento

Neste apêndice é apresentado um excerto de código a rota novo_evento para ser possível criar um evento.

```
app.post('/novo_evento', upload.fields([
  { name: 'cartaz', maxCount: 1 },
  { name: 'mais_infol', maxCount: 1 }
]), (req, res) => {
  let titulo_evento = req.body.titulo_evento;
  let texto_evento = req.body.texto_evento;
  let entidade_organizadora = req.body.entidade_organizadora;
  let data_evento = req.body.data_evento;
  let link_inscricoes = req.body.link_inscricoes;
  let cartaz = req.files['cartaz'] ? req.files['cartaz'][0] : null;
  let mais_infol = req.files['mais_infol'] ? req.files['mais_infol'][0] : null;
  let mais_info2 = req.body.mais_info2;

  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }

  let id_utilizador = req.session.userID;

  let novoEvento = {
    titulo_evento: titulo_evento,
    texto_evento: texto_evento,
    entidade_organizadora: entidade_organizadora,
    data_evento: data_evento,
    link_inscricoes: link_inscricoes,
```

```
cartaz: cartaz,
mais_info1: mais_info1,
mais_info2: mais_info2,
fk_idu: id_utilizador
};

if (cartaz) {
    novoEvento.cartaz = cartaz.path.replace(/public\//, '/');
}

if (mais_info1) {
    novoEvento.mais_info1 = mais_info1.path.replace(/public\//, '/');
}

connection.query('INSERT INTO evento SET ?',
    novoEvento, (err,
    result) => {
    if (err) {
        console.error('Erro ao inserir dados:', err);
        return;
    }
    console.log('Dados inseridos com sucesso.');
    res.render('pagina_pessoal', {
        id_utilizador: req.session.userID,
        priv_utilizador: req.session.userPriv,
        nome_utilizador: req.session.userName,
        message: null
    });
});
});
```

Exerto de Código E.1: Rota novo_evento para inserir um novo evento na base de dados.

Apêndice

F

Excerto de código para a alteração do email

Neste apêndice é apresentado um excerto de código a rota alterar_email para ser possível alterar o email.

```
app.post('/alterar_email', (req, res) => {
  if (!req.session.userID) {
    res.redirect('/login');
    return;
  }
  let { newEmail } = req.body;
  let userId = req.session.userID;
  connection.query('SELECT idu FROM utilizador WHERE email = ? AND idu
  <> ?', [newEmail, userId], (error, results) => {
    if (error) {
      return res.render('editar_perfil', {
        message: 'Ocorreu um erro ao verificar o email. Por favor,
        tente novamente.'
      });
    }
    if (results.length > 0) {
      return res.render('editar_perfil', {
        message: 'O email ja esta a ser usado por outro utilizador
        . Escolha um email diferente.'
      });
    }
    connection.query('UPDATE utilizador SET email = ? WHERE idu = ?',
      [newEmail, userId], (error, results) => {
      if (error) {
        return res.render('editar_perfil', {
          message: 'Ocorreu um erro ao atualizar o email. Por
          favor, tente novamente.'
        });
      }
    });
  });
});
```

```
        });
    }
    console.log(newEmail)
    res.render('pagina_pessoal', {
        id_utilizador: req.session.userID,
        priv_utilizador: req.session.userPriv,
        nome_utilizador: req.session.userName,
        message: null
    });
});
});
});
```

Excerto de Código F.1: Rota alterar_email para o utilizador alterar o email.

Apêndice

G

Excerto de código para criar as tabelas da base de dados

Neste apêndice é apresentado um excerto de código para criar as tabelas da base de dados.

```
-- Anfitriao: 127.0.0.1
-- Versao do servidor: 5.5.68-MariaDB - mariadb.org binary
-- distribution
-- Server OS: Win64
-- HeidiSQL Versao: 11.0.0.5919

-- Dumping database structure for cabc
DROP DATABASE IF EXISTS `cabc`;
CREATE DATABASE IF NOT EXISTS `cabc` /*!40100 DEFAULT CHARACTER SET
    latin1 */;
USE `cabc`;

-- Dumping structure for table cabc.ensaio_clinico
DROP TABLE IF EXISTS `ensaio_clinico`;
CREATE TABLE IF NOT EXISTS `ensaio_clinico` (
    `idec` int(11) NOT NULL AUTO_INCREMENT,
```

```

`nome_ensaio` varchar(100) NOT NULL,
`patologia` varchar(100) NOT NULL,
`texto_ensaio` varchar(5000) NOT NULL,
`estado` varchar(20) NOT NULL,
`idu` int(11) NOT NULL,
PRIMARY KEY (`idec`),
KEY `idu_idx` (`idu`),
CONSTRAINT `idu` FOREIGN KEY (`idu`) REFERENCES `utilizador` (`idu`)
    ON DELETE NO ACTION ON UPDATE NO ACTION
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cabc.evento
DROP TABLE IF EXISTS `evento`;
CREATE TABLE IF NOT EXISTS `evento` (
`ide` int(11) NOT NULL AUTO_INCREMENT,
`titulo_evento` varchar(200) NOT NULL,
`texto_evento` varchar(1000) NOT NULL,
`entidade_organizadora` varchar(70) NOT NULL,
`data_evento` date NOT NULL,
`link_inscricoes` varchar(200) DEFAULT NULL,
`cartaz` varchar(100) DEFAULT NULL,
`mais_info1` varchar(100) DEFAULT NULL,
`mais_info2` varchar(100) DEFAULT NULL,
`fk_idu` int(11) NOT NULL,
PRIMARY KEY (`ide`),
KEY `fk_idu` (`fk_idu`)
) ENGINE=InnoDB AUTO_INCREMENT=37 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cabc.formacao
DROP TABLE IF EXISTS `formacao`;
CREATE TABLE IF NOT EXISTS `formacao` (
`idf` int(11) NOT NULL AUTO_INCREMENT,
`nome_formacao` varchar(100) NOT NULL,
`texto_explativo` varchar(5000) NOT NULL,
`area_formacao` varchar(50) NOT NULL,
`max_inscricoes` int(11) DEFAULT NULL,
`link_insc` varchar(100) DEFAULT NULL,
`duracao` int(11) NOT NULL,
`data_inicio` date NOT NULL,
`data_fim` date NOT NULL,
`fkf_idu` int(11) NOT NULL,
PRIMARY KEY (`idf`),
KEY `fkf_idu` (`fkf_idu`),
CONSTRAINT `fkf_idu` FOREIGN KEY (`fkf_idu`) REFERENCES `utilizador` (
`idu`)

```

```

) ENGINE=InnoDB AUTO_INCREMENT=3 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cacb.newsletter
DROP TABLE IF EXISTS 'newsletter';
CREATE TABLE IF NOT EXISTS 'newsletter' (
    'idnews' int(11) NOT NULL AUTO_INCREMENT,
    'titulo' varchar(70) NOT NULL,
    'assunto' varchar(500) NOT NULL,
    'data_publicacao' date NOT NULL,
    'doc' varchar(100) DEFAULT NULL,
    'fkne_idu' int(11) NOT NULL,
    PRIMARY KEY ('idnews'),
    KEY 'fkne_idu' ('fkne_idu'),
    CONSTRAINT 'fkne_idu' FOREIGN KEY ('fkne_idu') REFERENCES 'utilizador'
        ('idu')
) ENGINE=InnoDB AUTO_INCREMENT=25 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cacb.noticia
DROP TABLE IF EXISTS 'noticia';
CREATE TABLE IF NOT EXISTS 'noticia' (
    'idn' int(11) NOT NULL AUTO_INCREMENT,
    'titulo_noticia' varchar(100) NOT NULL,
    'texto_noticia' varchar(3000) NOT NULL,
    'link_original' varchar(200) NOT NULL,
    'data_publicacao' date NOT NULL,
    'entidade' varchar(70) NOT NULL,
    'imagem' varchar(200) DEFAULT NULL,
    'fkno_idu' int(11) NOT NULL,
    PRIMARY KEY ('idn'),
    KEY 'fkno_idu' ('fkno_idu'),
    CONSTRAINT 'fkno_idu' FOREIGN KEY ('fkno_idu') REFERENCES 'utilizador'
        ('idu')
) ENGINE=InnoDB AUTO_INCREMENT=6 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cacb.post_blog
DROP TABLE IF EXISTS 'post_blog';
CREATE TABLE IF NOT EXISTS 'post_blog' (
    'idb' int(11) NOT NULL AUTO_INCREMENT,
    'titulo' varchar(200) NOT NULL,
    'texto' varchar(5000) NOT NULL,
    'autor' varchar(50) NOT NULL,
    'data_publicacao' date NOT NULL,
    'tag1' varchar(20) DEFAULT NULL,

```

```

'tag2' varchar(20) DEFAULT NULL,
>tag3' varchar(20) DEFAULT NULL,
'imagem' varchar(200) DEFAULT NULL,
'fkb_idu' int(11) NOT NULL,
PRIMARY KEY ('idb'),
KEY 'fkb_idu' ('fkb_idu'),
CONSTRAINT 'fkb_idu' FOREIGN KEY ('fkb_idu') REFERENCES 'utilizador'
('idu')
) ENGINE=InnoDB AUTO_INCREMENT=28 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cabc.privilegio
DROP TABLE IF EXISTS 'privilegio';
CREATE TABLE IF NOT EXISTS 'privilegio' (
'idp' int(11) NOT NULL AUTO_INCREMENT,
'super_admin' int(11) NOT NULL DEFAULT '0',
'newsletter_p' int(11) NOT NULL DEFAULT '0',
'noticia_p' int(11) NOT NULL DEFAULT '0',
'evento_p' int(11) NOT NULL DEFAULT '0',
'formacao_p' int(11) NOT NULL DEFAULT '0',
'blog_p' int(11) NOT NULL DEFAULT '0',
'testemunho_p' int(11) NOT NULL DEFAULT '0',
'ensaio_clinico_p' int(11) NOT NULL DEFAULT '0',
'media_p' int(11) NOT NULL DEFAULT '0',
'fkp_idu' int(11) NOT NULL,
PRIMARY KEY ('idp'),
KEY 'fkp_idu' ('fkp_idu'),
CONSTRAINT 'fkp_idu' FOREIGN KEY ('fkp_idu') REFERENCES 'utilizador'
('idu')
) ENGINE=InnoDB AUTO_INCREMENT=26 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

-- Dumping structure for table cabc.testemunho
DROP TABLE IF EXISTS 'testemunho';
CREATE TABLE IF NOT EXISTS 'testemunho' (
'idt' int(11) NOT NULL AUTO_INCREMENT,
'nome_testemunho' varchar(50) NOT NULL,
'opiniao' varchar(1000) NOT NULL,
'local_trabalho' varchar(70) NOT NULL,
'funcao' varchar(50) NOT NULL,
'fkt_idu' int(11) NOT NULL,
PRIMARY KEY ('idt'),
KEY 'fkt_idu' ('fkt_idu'),
CONSTRAINT 'fkt_idu' FOREIGN KEY ('fkt_idu') REFERENCES 'utilizador'
('idu')
) ENGINE=InnoDB AUTO_INCREMENT=32 DEFAULT CHARSET=latin1;

```

```
-- Data exporting was unselected.

-- Dumping structure for table cacb.utilizador
DROP TABLE IF EXISTS 'utilizador';
CREATE TABLE IF NOT EXISTS 'utilizador' (
    'idu' int(11) NOT NULL AUTO_INCREMENT,
    'nome_utilizador' varchar(50) NOT NULL DEFAULT '',
    'email' varchar(100) NOT NULL DEFAULT '',
    'organizacao' varchar(150) NOT NULL DEFAULT '',
    'palavra_passe' varchar(500) NOT NULL DEFAULT '',
    'aceite' int(11) NOT NULL DEFAULT '0',
    PRIMARY KEY ('idu')
) ENGINE=InnoDB AUTO_INCREMENT=26 DEFAULT CHARSET=latin1;

-- Data exporting was unselected.

/*!40101 SET SQL_MODE=IFNULL(@OLD_SQL_MODE, '') */;
/*!40014 SET FOREIGN_KEY_CHECKS=IF (@OLD_FOREIGN_KEY_CHECKS IS NULL, 1,
    @OLD_FOREIGN_KEY_CHECKS) */;
/*!40101 SET CHARACTER_SET_CLIENT=@OLD_CHARACTER_SET_CLIENT */;
```

Excerto de Código G.1: Criar as tabelas da base de dados.

Bibliografia

- [1] 2CA Braga Centro Académico Clínico. [Online] <https://ccabraga.org>. Acedido a 19 de abril de 2023.
- [2] ABC - Algarve Biomedical Center, Centro Académico de Investigação/-Formação Biomédica do Algarve. [Online] <https://abcmedicalg.pt>. Acedido a 19 de abril de 2023.
- [3] Wordpress. [Online] <https://pt.wordpress.org>. Acedido a 19 de abril de 2023.
- [4] Advantages and Disadvantages of WordPress website. [Online] <https://www.yashaaglobal.com/blog/advantages-and-disadvantages-of-wordpress-website/>. Acedido a 22 de abril de 2023.
- [5] Django. [Online] <https://www.djangoproject.com>. Acedido a 19 de maio de 2023.
- [6] What is HTML. [Online] <https://www.javatpoint.com/what-is-html>. Acedido a 19 de maio de 2023.
- [7] What is CSS. [Online] <https://www.javatpoint.com/what-is-css>. Acedido a 19 de maio de 2023.
- [8] What is CSS. [Online] <https://ejs.co>. Acedido a 19 de maio de 2023.
- [9] O que é SQL? [Online] <https://aws.amazon.com/pt/what-is/sql/>. Acedido a 19 de maio de 2023.
- [10] What Is JavaScript Used For? [Online] <https://www.hostinger.com/tutorials/what-is-javascript>. Acedido a 19 de maio de 2023.
- [11] Express. [Online] <https://expressjs.com>. Acedido a 20 de maio de 2023.
- [12] bcrypt. [Online] <https://www.npmjs.com/package/bcrypt>. Acedido a 20 de maio de 2023.

- [13] A Complete And Comprehensive Guide To Understand Body Parser In Express JS. [Online] <https://www.simplilearn.com/tutorials/nodejs-tutorial/body-parser-in-express-js>. Acedido a 19 de maio de 2023.
- [14] Multer. [Online] <https://www.npmjs.com/package/multer>. Acedido a 29 de junho de 2023.
- [15] mysql. [Online] <https://www.npmjs.com/package/mysql>. Acedido a 29 de junho de 2023.
- [16] Node.js. [Online] <https://nodejs.org/en>. Acedido a 1 de julho de 2023.
- [17] Download HeidiSQL 12.5, released on 08 May 2023. [Online] <https://wwwheidisql.com/download.php>. Acedido a 1 de julho de 2023.