EventCluster

Projeto II

Juliana Ribeiro Rodrigues (Nº: 20181040)

Micaela Rodrigues dos Santos (Nº: 20200893)

Orientadores

Eduardo Sabina dos Santos Valente

Bruno Miguel Gonçalves Matias

Trabalho de Projeto de Final de Curso, apresentado à Escola Superior de Tecnologia do Instituto Politécnico de Castelo Branco para cumprimento dos requisitos necessários à obtenção do grau de Licenciado em Informática e Multimédia, realizado sob a orientação científica do Professor Doutor Eduardo Valente, e coorientação do Professor Mestre Bruno Matias do Instituto Politécnico de Castelo Branco.

setembro de 2024

Composição do júri

Presidente do júri

Mestre, José Luís Silva Tavares da Cruz

Professor Adjunto, Instituto Politécnico de Castelo Branco

Vogais

Doutor, Eduardo Sabina dos Santos Valente

Professor Adjunto, Instituto Politécnico de Castelo Branco

Doutora, Mónica Isabel Teixeira da Costa

Professora Adjunta, Instituto Politécnico de Castelo Branco

Mestre, José Luís Silva Tavares da Cruz

Professor Adjunto, Instituto Politécnico de Castelo Branco

Agradecimentos

A realização deste projeto representa não só, o desafio e o crescimento pessoal, como também a procura por algo diferente e o empenho depositado no decorrer do mesmo.

Em primeiro lugar, agradecemos aos nossos orientadores, Professor Eduardo Valente e Professor Bruno Matias, por todo o apoio durante a realização do projeto e pelos conhecimentos adquiridos ao longo do projeto.

De uma forma geral, agradecer a todos os restantes, colegas e professores que direta ou indiretamente participaram de forma construtiva para a elaboração deste projeto.

Por último, mas não menos importante, agradecemos às nossas famílias que nos proporcionaram todas as condições para realizar mais uma etapa das nossas vidas.

Resumo

No contexto atual do acesso à informação, foi identificada uma necessidade de automatização da construção de sites para gestão e divulgação de eventos. O grande desafio para os gestores de eventos debate-se com a contratação de um profissional ou empresa que elabore o design web específico para o evento e que se responsabilize pelas questões do seu alojamento e disponibilização *online*. Dada a diversidade de conceitos técnicos, ferramentas e tipos de eventos, torna-se necessária uma abordagem dinâmica, que possibilite uma mais rápida disponibilização do site ao gestor do evento, atendendo às suas necessidades de layout e funcionalidades.

O **EventCluster** tem como objetivo colmatar um conjunto de problemas associados à disponibilização de sites de gestão de eventos, dada a multiplicidade de tipos de eventos. A plataforma encontra-se dividida em duas partes, consistindo a primeira numa aplicação de criação e gestão de *templates* para gestão de eventos. A segunda parte representa uma instância de um *template*, onde o interessado em gerir um evento terá ao seu dispor as funcionalidades necessárias para o fazer.

Para além do gestor do evento, são considerados outros utilizadores da interface do evento, tais como visitante, participante e voluntário, cada um tendo acesso às respetivas funcionalidades.

Esta plataforma pretende demonstrar o conceito da elaboração prévia de *templates* de sites, com possibilidade de algum tipo de configuração por quem os adquire, por forma a disponibilizar mais rapidamente um produto ao cliente, com pouca ou nenhuma intervenção pós-venda específica por parte do programador. Para além da rapidez, esta configuração pode diminuir o valor de venda do produto, tornando-o mais atrativo para o cliente.

Palavras-chave

Gestão, *Templates,* Eventos , Dinâmico.

Abstract

In the current times where there is a large flow of information access, it was identified an automation need for website creation specialised in event management and promotion. The main challenge for event managers lies in hiring a professional or company to design a specific web layout for the event and take responsibility for its hosting and online availability. Given the diversity of technical concepts, tools, and types of events, a dynamic approach is required to allow for quicker availability of the website meeting their layout and functionality needs.

EventCluster aims to address a set of issues related to the availability of event management websites, considering the multiplicity of event types. The platform is divided into two parts, the first being an application for creating and managing templates for event management. The second part represents an instance of a template, where those interested in managing an event will have access to the necessary functionalities to do so.

In addition to the event manager, other users of the event interface are considered, such as visitors, participants, and volunteers, each with access to their respective functionalities.

This platform aims to demonstrate the concept of pre-prepared website templates, allowing some level of configuration by those who acquire them. This approach provides a faster product to the client, with little to no specific post-sale intervention from the programmer. In addition to speed, this configuration can lower the product's price, making it more attractive to the client.

Keywords

Management, *Templates*, Event, Dynamic.

Índice geral

Índice de figuras

Lista de tabelas

Lista de abreviaturas, siglas e acrónimos

CRUD	*Create, read, update and delete*
CSS	*Cascading Style Sheets*
HTML	*Hyper Text Markup Language*
JS	*JavaScript*
Modelo E/R	Modelo Entidade-Relacionamento
PHP	*Hypertext Preprocessor*
UML	*Unified Modeling Language*
SQL	*Structured Query Language*

1. Introdução

Este relatório detalha o desenvolvimento do Projeto II, dando a continuidade ao Projeto I. O Projeto II fica-se na implementação das funcionalidades do EventCluster, idealizada para criar sites de gestão de eventos, através de *templates* personalizados.

Atualmente, a criação de eventos *online* enfrenta limitações no que diz respeito à personalização dos *templates*, o que dificulta a adaptação às necessidades específicas de cada organizador. Esta plataforma surge para preencher essa lacuna, de modo a oferecer uma solução flexível e totalmente personalizável para a criação de eventos.

Durante o desenvolvimento do EventCluster, foram analisadas plataformas como o ViralAgenda [1] e o Eventbrite [2], que serviram como referência e inspiração em termos de funcionalidades e experiência de utilizador. Além disso, cada evento, conta com uma secção de comentários, que permite que obter *feedback* dos participantes. Para os utilizadores que desejam adquirir *templates* personalizados, a plataforma tem a secção de contactos, onde podem enviar seus pedidos diretamente por mensagem para adquirir o *template*.

1.1. Objetivos

Em Projeto I foi feito um estudo do estado da arte, onde foi realizada uma análise aprofundada de artigos e plataforma*s*, com o objetivo de identificar as melhores práticas, funcionalidades e tecnologias que se adequavam melhor para a plataforma que estava por desenvolver no Projeto II. Essa pesquisa permitiu delinear os requisitos e as funcionalidades a implementar na EventCluster. Foram também elaborados diversos diagramas UML, que é a linguagem padrão para visualizar, especificar, construir e documentar os artefactos de um sistema intensamente baseado em software[3], seguindo a metodologia ICONIX, na modelação do comportamento dinâmico e estrutura da plataforma a implementar.

Nesta segunda fase do projeto, no âmbito da unidade curricular Projeto II, o objetivo principal será implementar todas as funcionalidades que foram projetadas anteriormente, ou seja, desempenhar toda a parte funcional. Este projeto dará continuidade ao trabalho iniciado no projeto anterior (Projeto I), transformando os requisitos e modelos desenvolvidos numa plataforma. Serão desenvolvidas as funções de *back-office*, como a gestão de *templates* e de eventos. Também serão elaboradas as funções de *front-office*, que irão permitir que os utilizadores possam interagir com a plataforma, como por exemplo, fazer reservas, escrever comentários, entre outras funcionalidades.

1.1.1. Objetivos Projeto II

Para a realização do Projeto II, foram definidos alguns objetivos específicos que vão guiar o desenvolvimento e garantir que todas as etapas e funcionalidades são implementadas. Estes objetivos foram elaborados com base numa análise já realizada anteriormente, em Projeto I.

- Modelação da base de dados
- Implementação
    - Gestor de *Templates*
    - Gestor de eventos
    - Participante

1.2. Estrutura do relatório

Neste relatório é composto por seis capítulos são eles a introdução, a modelação do sistema, a implementação, a explicação de código, o diagrama de componentes, as tecnologias e linguagens utilizadas e por fim a conclusão.

No primeiro capítulo, Introdução, o objetivo é introduzir o tema do projeto e apresentar os objetivos gerais e específicos do Projeto II, consoante ao que foi desenvolvido no projeto anterior.

No segundo capítulo, Alterações (Projeto I para Projeto II), aborda as alterações que foram realizadas e necessárias na análise de requisitos. Tais como o modelo de classes, o diagrama de classes, o modelo E/R e o modelo de dados (tabelas da base de dados). Neste mesmo capítulo são demonstradas alterações que foram surgindo ao longo da implementação, para o Projeto II.

No terceiro capítulo, Funcionalidades da plataforma, destaca-se na implementação, onde será apresentado todas as funcionalidades da plataforma. Neste mesmo capítulo a divisão é realizada por subcapítulos *back-office* e *front-office.* Nos subcapítulos será apresentada todas as funcionalidades de cada utilizador.

O quarto capítulo, Abordagens de Desenvolvimento, expõe uma explicação detalhada do código, incluindo a descrição de alguns excertos de código, e apresenta-se o diagrama de componentes, com o intuito de facilitar a compreensão da estrutura do sistema.

No quinto capítulo, Tecnologias e Linguagens Utilizadas, são descritas todas as ferramentas utilizadas para o desenvolvimento do Projeto II.

Por fim, no sexto capítulo são as conclusões retiradas e é apresentada a proposta para trabalho futuro.

2. Alterações (Projeto I para Projeto II)

Neste segundo capítulo, serão abordadas todas as alterações que foram necessárias para garantir o funcionamento da plataforma. Foram analisados cada alteração realizada ao longo da implementação, explicando assim o motivo das suas alterações.

As alterações realizadas envolveram o Modelo de Domínio, o Modelo de Classes, o Modelo E/R e Modelo de Dados.

2.1. Modelo de Domínio

A Figura 1 representa o Modelo de Domínio, que é composto pelas entidades (classes) do sistema. Este modelo serve como guia para a elaboração dos restantes diagramas, funcionando como um dicionário de nomes a utilizar em todas as fases do processo ICONIX.[3]

O modelo inclui diversas classes, tais como: "Utilizador", "Evento", "*Template*", "Tipo*Template*", "Comentario", "Reserva", "Bilhete" e "Favorito".

Um utilizador pode realizar várias tarefas, com a carnalidade de **1 - 1…*** (um para muitos) e, conter vários bilhetes, com a carnalidade de **1 - 1…*** (um para muitos). Da mesma forma, um utilizador pode ter diversos *Templates* associados, onde se atribui a carnalidade de **1 - 1…*** (um para muitos).

Um evento pode estar associado a vários comentários, reservas e bilhetes, onde se optou por colocar a carnalidade de **1 - 1…*** (um para muitos). Vários eventos podem ser guardados como favoritos, com a carnalidade de **1…* - 1…*** (muitos para muitos).

O *template* está relacionado a um único tipo de *template*, por consequente a carnalidade ser de **1 - 1** (um para um). Os eventos por sua vez, podem ser criados a partir de mais que um *template*, onde se apresenta com a carnalidade de **1…* - 1…*** (muitos para muitos).

E um utilizador pode fazer várias reservas **1 - 1…*** (um para muitos), e por sua vez uma reserva pode ter vários bilhetes, **1 - 1…*** (um para muitos).

Um utilizador pode escrever várias tarefas, **1 - 1…*** (um para muitos).

Em relação ao Projeto I, houve a necessidade de uma restruturação do Modelo de Domínio. As tabelas TipoUtilizador e o Voluntário foram removidas.

As tabelas que foram criadas foram, "**Tipo de *Template*"**, "**Tarefa"** e "**Favorito**". As tabelas que não foram alteradas, conforme descrito no Projeto I foram, **"Utilizador", "*Template*", "Evento", "Bilhete", "Comentario"** e **"Reserva".**

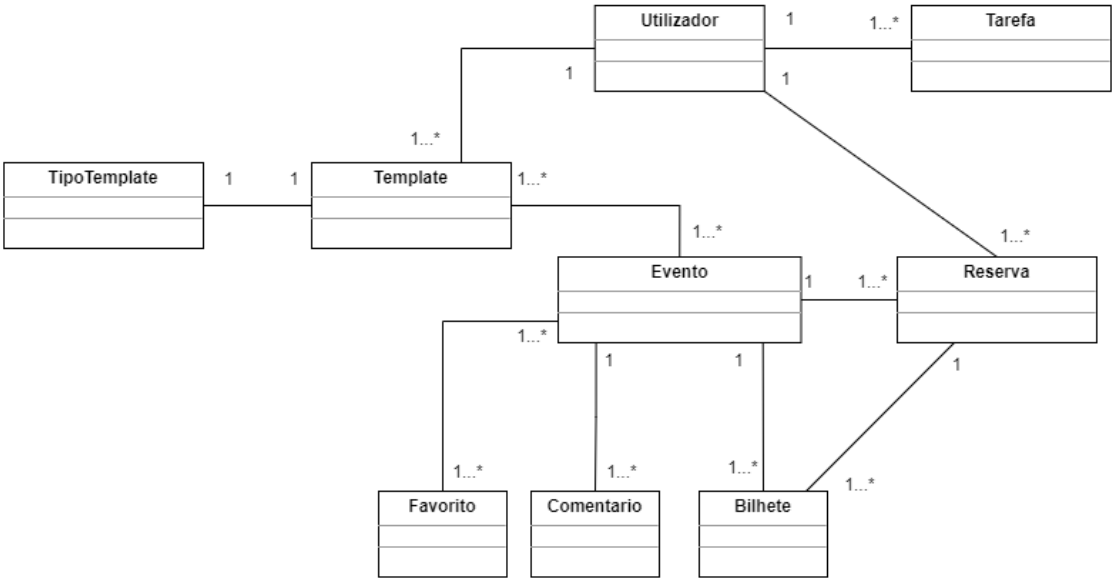

Figura 1 - Modelo de Domínio.

2.2. Diagrama de Classes

O diagrama de classes apresenta a modelação das principais entidades de um sistema. Cada classe representa uma entidade específica com atributos e métodos, bem como as relações entre as mesmas.

O diagrama contém várias classes, como "***Utilizador***", "**Comentario**", "**Tarefa**", "**Reserva**", "**Bilhete**", "**Evento**", "**Favorito**", "***Template***" e "**Tipo*Template***".

Um utilizador pode fazer várias reservas, escrever comentários e tarefas, além de criar *templates*, mas cada reserva, comentário, tarefa e *template* é sempre associado a um único utilizador.

Um tipo de *template* pode ter vários *templates*, mas cada *template* pertence a um único tipo de *template*.

Um evento pode ter várias reservas, receber diversos comentários, ser marcado como favorito, conter vários bilhetes e utilizar vários *templates*, mas cada reserva, comentário e bilhete estão associados a um único evento, enquanto cada *template* pode ser usado por múltiplos eventos.

Em relação do Projeto I, houve a necessidade de reestruturar o modelo de classes. As tabelas "**TipoUtilizador**", "**Voluntário**" e "**Participante**" foram removidas, enquanto novas tabelas foram adicionadas: "**Tipo *Template***", "**Tarefa**" e "**Favorito**". As tabelas que foram mantidas, conforme descrito no Projeto I, foram: "**Utilizadores**", "***Template***", "**Evento**", "**Bilhete**", "**Comentário**" e "**Reserva**".

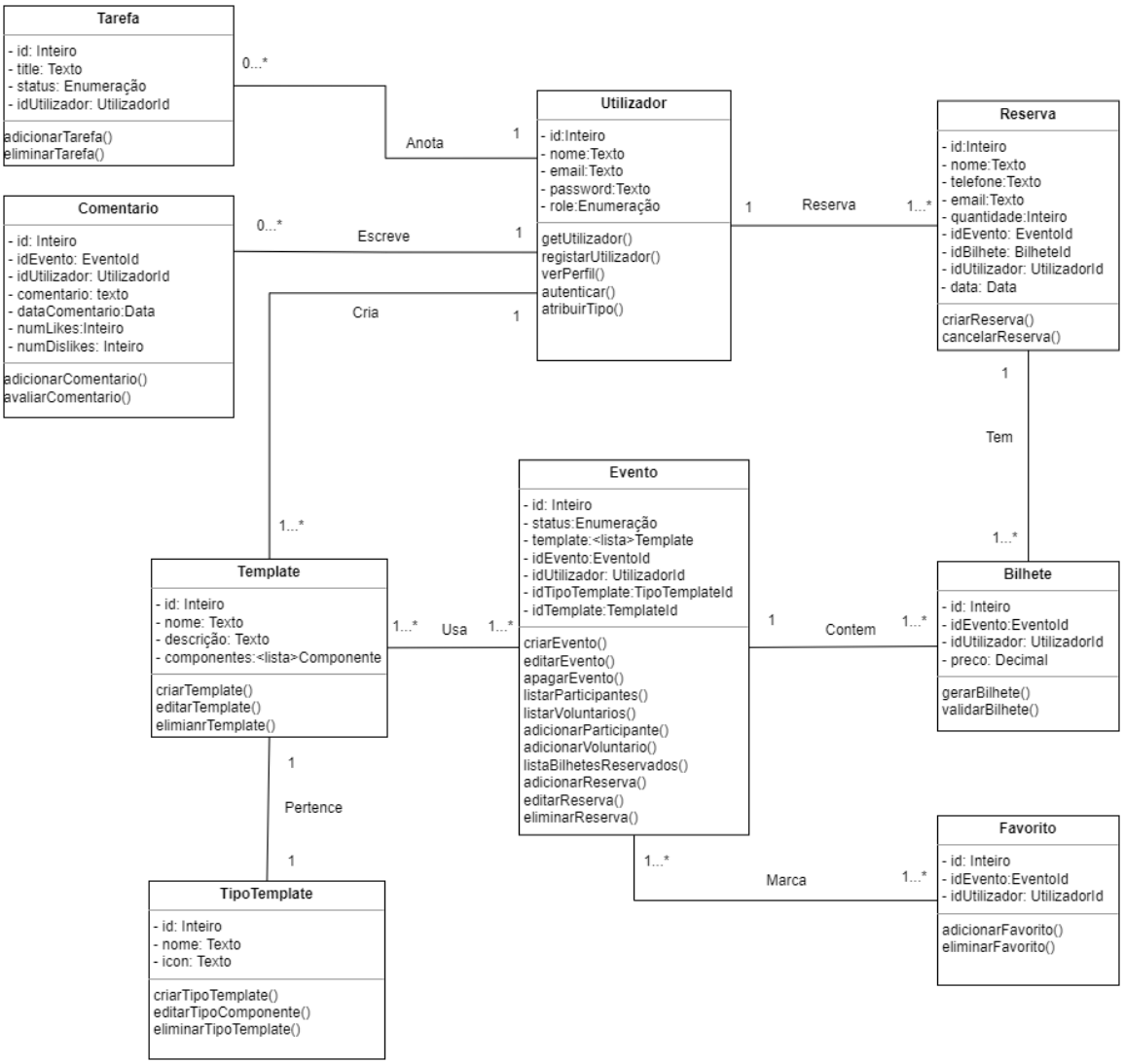

Figura 2 - Diagrama de Classes.

2.4. Modelo E/R

O modelo Entidade/Relacionamento (E/R) tem uma entidade **Administrador *Template*** que está relacionada com a entidade ***Template***, com uma cardinalidade de 1-N (um para muitos). Indica que um **Administrador *Template*** pode criar (A) vários *templates*.

O **Administrador *Template*** também tem um relacionamento com a entidade **Tarefa,** com uma cardinalidade de 1-N (um para muitos), de forma a indicar que o **Administrador *Template*** pode escrever (B) várias tarefas.

Na entidade ***Template* e**stá associado a três outras entidades: **Administrador Evento, Evento** e **Tipo *Template*.** A relação entre a entidade ***Template*** e a entidade **Tipo *Template*** tem uma cardinalidade de 1-1 (um para um), o que significa que um *Template* contém (C) um tipo de *template***.**

A entidade ***Template*** está relacionada com a entidade **Evento**, com a cardinalidade de N-M (muitos para muitos), o que significa que um *template* pode ter (D) vários Eventos.

A entidade ***Template*** está relacionada com a entidade **Administrador Evento** com uma cardinalidade de 1-N (um para muitos). Portanto, um Administrador Evento pode adquirir (E) vários *templates*.

A entidade **Evento** está relacionada com a entidade **Bilhete** por uma cardinalidade de 1-N (um para muitos). Isso indica que um Evento pode ter (F) vários Bilhetes.

A mesma entidade **Evento** está relacionada com a entidade **Administrador Evento** com uma cardinalidade de N-1 (muitos para um). Isso significa que vários Eventos são elaborados (G) por um Administrador Evento.

A entidade **Evento** está relacionada com a entidade **Participante** com duas cardinalidades distintas. A primeira cardinalidade é M-N (muitos para muitos), o que significa que vários Participantes podem comentar (H) vários Eventos. A segunda cardinalidade é N-1 (muitos para um), indicando que um Participante pode reagir (I) a vários Eventos.

Por fim, a entidade **Bilhete** está relacionada com a entidade **Participante** por uma cardinalidade de N-1 (muitos para um). Isso significa que um Participante pode reservar (J) vários Bilhetes.

Em relação ao Projeto I, houve uma reestruturação do modelo Entidade/Relacionamento (E/R). As entidades **Comentário** e **Reserva** foram removidas. Novas entidades denominadas **Tipo *Template*** e **Tarefa** foram adicionadas. Além disso, o nome da entidade **Participante** foi alterado. As entidades que permanecem no Projeto I são: **Evento**, **Bilhete**, ***Template***, **Administrador Evento** e **Administrador *Template***.

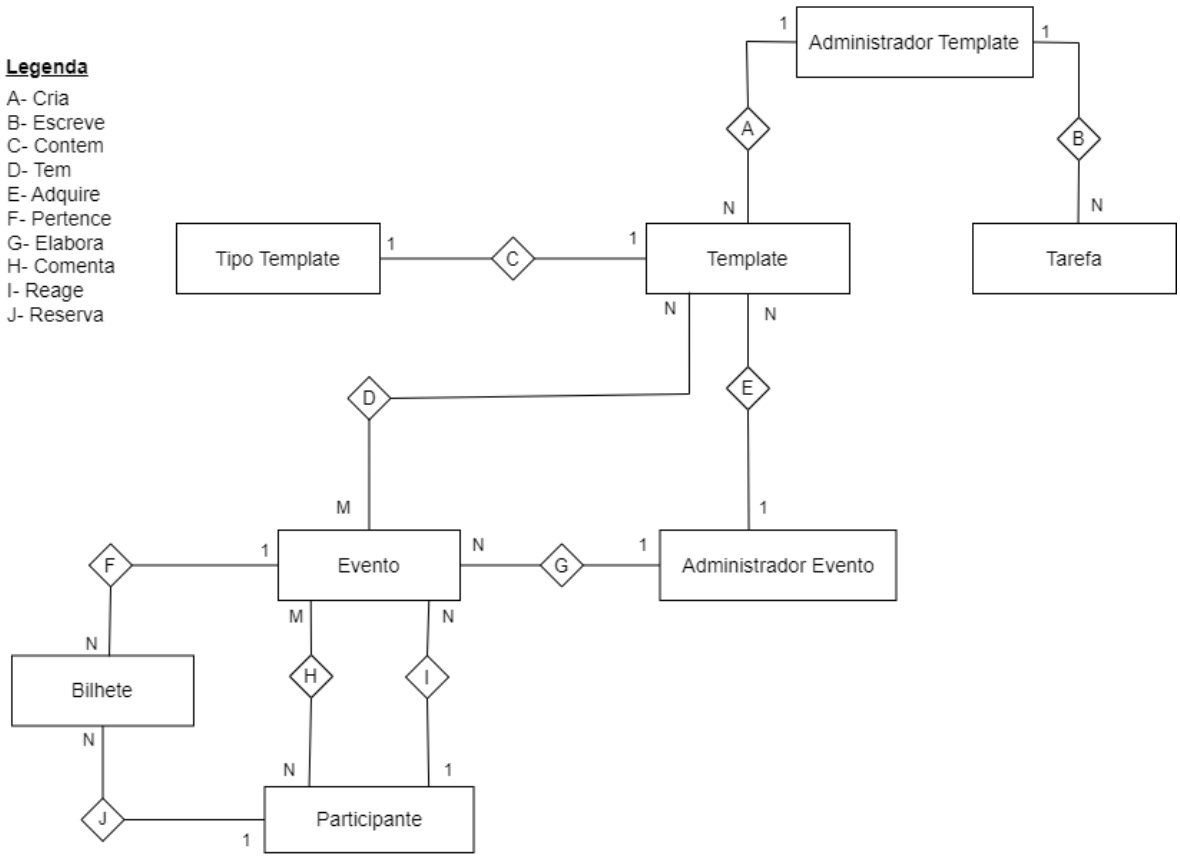

Figura 3 - Modelo E/R

2.5. Modelo de dados

Neste subcapítulo, será apresentada uma visão detalhada das tabelas criadas, assim como as respetivas descrições para cada uma delas. Esta análise, proporciona uma compreensão mais nítida e completa na organização dos dados e as funcionalidades de cada tabela. Cada tabela será descrita individualmente.

Entre as tabelas criadas, duas utilizam o formato JSON para armazenar dados: a Tabela de Eventos e a Tabela de Formulários Dinâmicos. O JSON foi utilizado para armazenar dados de forma estruturada e eficiente. Os dados são organizados em pares de chave e valor, onde as chaves são *strings* entre aspas e os valores podem ser de diversos tipos, incluindo *string*, número, *booleano* ou nulo [4].

A **Tabela 1** é responsável por armazenar os tipos de *template*. O atributo "id" é a chave primária e é gerada automaticamente através do auto-incremento. O atributo "name" representa o nome do tipo de *template* e o atributo "icon" representa o ícone associado ao tipo de *template*. A tabela também guarda a data e a hora de criação, atualização e eliminação do tipo de *template*. Na implementação, esta tabela é representada por "*Template*Types".

Tabela 1 - Descrição da Tabela Tipo de *Template*

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
name	VARCHAR (150)	Nome do tipo de evento	Sim	
icon	VARCHAR (50)	Ícone que representa o evento	Sim	
created_at	DATETIME	Data de criação		
updated_at	DATETIME	Data de atualização		
deleted_at	DATETIME	Data da eliminação		

A **Tabela 2**, referente aos utilizadores, é responsável por armazenar as informações de cada um deles. O atributo id é a chave primária e é gerado automaticamente através de auto-incremento. O atributo email deve ser único para cada utilizador. Os demais atributos incluem "name", que é o nome do utilizador; "phoneNumber", que é o número de telefone do utilizador; "photo", que é a fotografia do utilizador; e "*password*", que é a senha do utilizador. O atributo role é um ENUM que define os diferentes tipos de utilizadores no sistema, como "Participante", "Voluntário", "Administrador de Eventos" e "Administrador de *Templates*".

A tabela também guarda a data e a hora de criação, atualização e eliminação do utilizador. Na implementação, esta tabela é representada por "*users*".

Tabela 2 - Descrição da Tabela Utilizadores

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
name	VARCHAR (500)	Nome do participante	Não	
phoneNumber	VARCHAR (100)	Número de telefone	Sim	
photo	VARCHAR (100)	Fotografia	Não	
photo_blob	LongText	Armazena as fotografias		
email	VARCHAR (100)	Email	Sim	Deve ser único
email_verified_at	DATETIME	Verificação da data do email	Não	
password	VARCHAR	*Password*	Sim	
role	ENUM	Vários utilizadores	Sim	'Participante', 'Voluntário',

			'AdminEventos', 'Admin*Templates*
rememberToken	VARCHAR (100)		
created_at	DATETIME	Data de criação	
updated_at	DATETIME	Data de atualização	
deleted_at	DATETIME	Data da eliminação	

A **Tabela 3** é responsável por armazenar todas as informações relacionadas ao formulário. O atributo "id" é a chave primária e é gerado automaticamente através de auto-incremento. O atributo "name" representa o nome do formulário, enquanto o atributo "content" armazena o conteúdo do formulário em formato JSON. O atributo "description" é para ter uma breve descrição do formulário. A tabela também guarda a data e a hora em que o formulário foi criado, através do atributo created_at, e a data e a hora da última atualização, através do atributo updated_at. Na implementação, esta tabela é representada por "form_builders".

Tabela 3 - Descrição da Tabela Formulário

Campos	**Tipo de dados**	**Descrição**	**Obrigatório**	**Observações**
id	BIGINT	Identificador	Sim	Auto-increment
name	VARCHAR (255)	Nome do formulário	Sim	
description	TEXT	Descrição do formulário	Não	
content	JSON	Conteúdo do formulário	Sim	
created_at	DATETIME	Data de Criação	Sim	
updated_a	DATETIME	Data de Atualização	Sim	

A **Tabela 4** é responsável por armazenar todas as informações relativas a cada evento. O atributo "id" é a chave primária e é gerado automaticamente através de auto-incremento. O atributo "reserve_qtd" indica a quantidade de reservas associadas ao evento, enquanto o atributo "status" define o estado do evento, podendo ser 'Publicado', 'Rascunho' ou 'Arquivado'. O atributo "formContent" armazena dados personalizados do formulário em formato JSON. A tabela também guarda a data e a hora de criação, atualização e eliminação do evento.

As chaves estrangeiras nesta tabela referenciam o identificador único do tipo de *template* da **Tabela 1** "*Template*Type_id" e o identificador do utilizador da **Tabela 2** "user_id", além do identificador do formulário da **Tabela 3** "form_id". Na implementação, esta tabela é representada por "events".

Tabela 4 - Descrição da Tabela Eventos

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
reserve_qtd	INT	Quantidade de reserva	Não	
status	ENUM	Estado do evento		'Publicado', 'Rascunho', e 'Arquivo'
formContent	JSON	Armazena dados personalizados		
created_at	DATETIME	Data de Criação		
updated_at	DATETIME	Data de Atualização		
deleted_at	DATETIME	Data de eliminação	Não	
Templatetype_id	BIGINT	Identificador único do tipo de *template*	Sim	Refere-se a tabela tipo de *template* (id)
user_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela utilizadores (id)
form_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela formulário (id)

Na **Tabela 5**, é responsável por armazenar os bilhetes reservados. O atributo "id" é a chave primária e é gerado automaticamente através de auto-incremento. O atributo "price" representa o preço do bilhete. Os atributos "created_at" e "updated_at" guardam, respetivamente, a data e a hora de criação e da última atualização do bilhete. As chaves estrangeiras nesta tabela são "event_id", que se refere ao identificador único dos eventos na **Tabela 4**, e "user_id", que se refere ao identificador único dos utilizadores na **Tabela 2**. Na implementação, esta tabela é representada por "tickets".

Tabela 5 - Descrição da Tabela de Bilhetes

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
price	DECIMAL (8,2)	Preço do bilhete	Sim	
created_at	DATETIME	Data de Criação		
updated_at	DATETIME	Data de Atualização		
event_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela eventos (id)
user_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela utilizadores (id)

Na **Tabela 6** é responsável por guardar todas as informações relacionadas com as reservas do evento. O atributo "id" é a chave primária e é gerado automaticamente através de auto-incremento. Os atributos desta tabela incluem "name", que representa o nome da pessoa que fez a reserva; "phone", que é o número de telefone da pessoa; e "email", que é o e-mail da pessoa. O atributo "quantity" indica a quantidade de bilhetes reservados. Além disso, a tabela armazena o "status" da reserva, que pode ser "Ativa", "Terminada" ou "Confirmada". Também são guardados a data e a hora da criação da reserva (created_at) e a data e a hora da última atualização (updated_at).

As chaves estrangeiras na tabela são o identificador único do bilhete, "ticket_id" na **Tabela 5**, o identificador único do evento, "event_id" na **Tabela 4,** e o identificador único do utilizador, "user_id" na **Tabela 2.** Na implementação, esta tabela é representada por "reservations".

Tabela 6 - Descrição da Tabela de Reservas

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
name	VARCHAR (100)	Nome do utilizador	Sim	
phone	VARCHAR (100)	Contacto do utilizador	Sim	
email	VARCHAR (100)	Email do utilizador	Sim	
quantity	INT	Quantidade de bilhetes reservados.	Sim	
status	ENUM	Define o estado atual	Sim	'Ativa', 'Terminada', e 'Confirmada'

updated_at	DATETIME	Data de Atualização		
created_at	DATETIME	Data de Criação		
ticket_id	BIGINT	Identificador único do bilhete	Sim	Refere-se a tabela bilhetes (id)
event_id	BIGINT	Identificador único do evento	Sim	Refere-se a tabela eventos (id)
user_id	BIGINT	Identificador único do Participante	Sim	Refere-se a tabela utilizadores (id)

Na tabela dos comentários, **Tabela 7**, que armazena os comentários relacionados aos eventos, o atributo "id" é a chave primária e é gerado automaticamente através de auto-incremento. O atributo "comment" contém o texto do comentário, enquanto "numLikes" e "numDislikes" representam, respetivamente, o número total de *likes* e *dislikes* associados ao comentário. A tabela também guarda a data e a hora de criação, atualização do comentário.

O atributo "event_id" é chave estrangeira que faz referência na **Tabela 4** e o atributo "user_id" é chave estrangeiras que faz referência na **Tabela 2.** Na implementação, esta tabela é representada por "comments".

Tabela 7 - Descrição da Tabela de Comentários

Campos	**Tipo de dados**	**Descrição**	**Obrigatório**	**Observações**
id	BIGINT	Identificador	Sim	Auto-increment
comment	LONGTEXT	Conteúdo	Sim	
numLikes	BIGINT	*Like* do evento	Sim	
numDislikes	BIGINT	*Deslike* do evento	Sim	
created_at	DATETIME	Data de Criação		
updated_at	DATETIME	Data de Atualização		
event_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela eventos(id)
user_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela utilizadores (id)

A **Tabela 8** é responsável por armazenar todas as informações relacionadas às tarefas do administrador de *templates*. O atributo "id" é a chave primária e é gerado automaticamente através de auto-incremento. O atributo "title" representa o título da tarefa, e o atributo "status" define o estado atual da tarefa, que pode ser "Pendente", "A decorrer" ou "Concluída".

A tabela também guarda a data e a hora em que a tarefa foi criada “created_at” e a data e a hora da última atualização “updated_at”. O atributo “user_id” é uma chave estrangeira que faz referência ao identificador único do utilizador da **Tabela 2**.

Na implementação, esta tabela é representada por “tasks”.

Tabela 8 - Descrição da Tabela Tarefas

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
title	VARCHAR (255)	Nome da tarefa	Sim	
status	ENUM	Define o estado atual da tarefa	Sim	‘Pendente’, ‘A decorrer’, e ‘Concluída’
created_at	DATETIME	Data de Criação		
updated_at	DATETIME	Data de Atualização		
user_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela utilizadores (id)

Na **Tabela 9**, é responsável por armazenar todas as interações relacionadas com comentários. O atributo “id” é a chave primária e é gerado automaticamente através de auto-incremento. O atributo “interaction_type” define o tipo de interação, onde pode ser *Like* ou *Dislike*. A tabela também guarda a data e a hora de criação, atualização do Interação Comentários.

O atributo “comment_id” é uma chave estrangeira que referencia o identificador único do comentário na **Tabela 7**, e o atributo “user_id” é uma chave estrangeira que referencia o identificador único do utilizador na **Tabela 2**. da
Na implementação, esta tabela é representada por “comment_interactions”.

Tabela 9 - Descrição da Tabela Interação Comentários

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
interaction_type	ENUM	Estado atual da interação		‘Like’ e ‘Dislike’
created_at	DATETIME	Data de Criação		
updated_at	DATETIME	Data de Atualização		
comment_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela comentários(id)
user_id	BIGINT	Chave Estrangeira	Sim	Refere-se a tabela utilizadores (id)

Na **Tabela 10** é responsável por guardar os eventos que foram marcados como favoritos pelo utilizador. Esta tabela interliga duas outras tabelas: a **Tabela 2**, que contém informações sobre os utilizadores, e a **Tabela 4**, que armazena os eventos. Os atributos da tabela incluem um identificador único que é a chave primária e é gerado automaticamente através de auto-incremento. A tabela também guarda a data e a hora de criação, atualização dos favoritos.

As chaves estrangeiras na tabela são "user_id", que se refere à **Tabela 2** e "event_id", que se refere à **Tabela 4**. Na implementação, esta tabela é representada por "favorites".

Tabela 10 – Descrição da Tabela dos favoritos

Campos	Tipo de dados	Descrição	Obrigatório	Observações
id	BIGINT	Identificador	Sim	Auto-increment
created_at	DATETIME	Data de Criação		
updated_at	DATETIME	Data da Atualização		
user_id	BIGINT	Chave Estrangeira,	Sim	Refere-se a tabela utilizadores(id)
event_id	BIGINT	Chave Estrangeira,	Sim	Refere-se a tabela de eventos(id)

3. Funcionalidades da Plataforma

Neste capítulo será apresentado de forma detalhada o processo de implementação de todo o sistema, aplicando assim as capturas de écrans (*prints*) para demonstrar cada fase do desenvolvimento e as suas devidas descrições. O grande objetivo deste capítulo é proporcionar uma visão mais detalhada. Será apresentado por subcapítulos, com os vários tipos de utilizadores e as suas devidas funcionalidades ao decorrer da plataforma.

Antes de iniciar a programação propriamente dita, começou-se por considerar qual tecnologia seria a mais adequada para o projeto. Após uma análise detalhada, decidiu-se que a tecnologia Laravel seria a mais indicada. Como num ato de resumo, o Laravel, sendo este uma *framework* do PHP, adquire uma estrutura MVC (model, view, controller), o que proporcionou, uma maior facilidade na organização de pastas e ficheiros ao longo do decorrer do projeto.

Em termos de implementação é importante referir que se utilizou os recursos do Laravel, como o sistema de *views*, controladores, rotas, e as funções @extends e @include no blade o que facilitou na organização dos ficheiros no projeto.

Com o uso do @extends, conseguiu-se criar layouts reutilizáveis em várias páginas, ou seja, foi definido um layout inicial com a cabeçalho, rodapé e estrutura geral, e depois em cada *view* específica adiciona-se o conteúdo. A função @include ajuda para não estar a repetir código, como menus, barra superior de navegação, scripts e assim no projeto é só incluir o que se pretende e tem se logo acesso.

De uma forma a contextualizar a plataforma, tem quatro tipos de utilizadores, o Visitante, o Participante, o Administrador de *Templates* e o Administrador de Eventos.

O Visitante consegue visualizar os eventos que existem na página de Eventos e os comentários do mesmo. Caso o visitante deseje participar, comentar em algum evento, terá de efetuar o registo.

O Participante, pode escrever comentários no evento e reagir aos mesmos, reservar um evento e pode ainda adicionar os eventos como favorito.

O Administrador de *templates* gere os *templates*, de modo a criar forma dinâmica e personalizada para cada tipo de evento. Também tem acesso aos clientes que adquiriram o evento escolhido. Este administrador pode visualizar o histórico de todos os eventos já existentes, na listagem mostra apenas aquelas que já foram concluídas ou que estão em rascunho. Caso seja necessário editar algum campo, este administrador tem essa autonomia, assim como editar os dados do utilizador. Existe apenas uma forma de criar o administrador de eventos, por meio do administrador de *templates*, somente assim será possível criar um administrador de eventos. O participante terá que solicitar a alteração do seu perfil ao administrador de *templates*, assim ele terá que editar o perfil do participante para administrador de eventos.

Já no administrador de eventos, gere os eventos que são criados, os tipos de eventos e os utilizadores e as reservas dos participantes. Este administrador também

tem um histórico que lista todos os eventos já terminados, também tem a autonomia de criar um tipo de evento, caso não exista o evento pretendido. Também pode editar o tipo de evento ou mesmo eliminar. O administrador terá de escolher o *template*, criado pelo mesmo. Sendo assim o gestor de eventos terá de preencher todos os campos, que o administrador de *templates* criou para aquele tipo de evento em específico. Este administrador ainda poderá efetuar reservas, onde terá de preencher os dados que forem pedidos. No entanto, ele consegue visualizar todas as reservas de forma detalhada que foram efetuadas de todos os tipos de eventos presentes na plataforma.

3.1. Funcionalidades e Interface do *Front-office*

O *front-office* consiste em toda a área da plataforma que é acessível a todas as pessoas na *Internet*. Ou seja, é a zona de acesso público, que neste caso permite visualizar os Eventos. Nos tópicos seguintes serão expostos e contextualizadas as várias áreas e funcionalidades do EventCluster, na vertente de utilizador não autenticado (visitante).

3.1.1. Menu de Navegação

Na **Figura 4** está representado o menu de navegação do visitante. Nesse menu está indicado o "Início", "Contacto", "*Login*" e por fim o "Registo". Caso o participante seleciona em "Início" será direcionado até à página inicial, onde se encontra os eventos disponíveis, **Figura 6**. Se o participante selecionar em "Contacto" vai ser direcionado para a *Landing page* **Figura 15**. Ao clicar em "*Login*", vai para a página do *login*, Figura 16. O mesmo acontece para o registo, Figura 17.

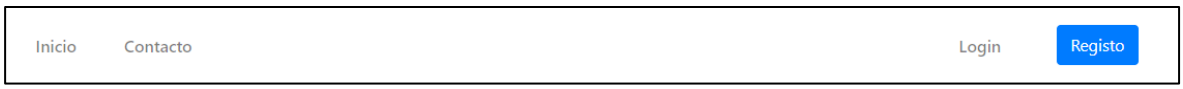

Figura 4 - Menu de navegação

Na Figura 5 está representado o menu de navegação que diz respeito ao participante. Onde está indicado a "Início", "Contacto", as "Minhas Reservas", onde pode visualizar o seu nome e ao lado do botão "Logout". Caso o participante seleciona em "Início" será direcionado até à página inicial onde se encontra os eventos disponíveis, **Figura 6**. Se o participante selecionar em "Contacto" vai ser direcionado para a *Landing page* **Figura 15**. Caso o participante selecionar em "Minhas Reservas" será direcionado para uma listagem onde se encontra todos as reservas do participante, tal como os eventos que o participante colocou nos favoritos, **Figura 14**. Assim que o participante clicar no botão "Logout" a sua sessão será imediatamente encerrada.

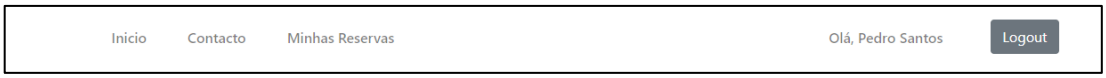

Figura 5 - Menu de navegação (participante)

3.1.2. Eventos

Assim que o utilizador entrar na plataforma vai conseguir visualizar os eventos que estão disponíveis no momento, **Figura 6**. Nessa página tem uma zona de filtragem, assim o utilizador terá mais a facilidade de procurar que tipo de evento deseja. Os eventos irão estar disponíveis, como demonstra na figura abaixo. Assim que o visitante seleciona o evento, será direcionado para o Registo. Pois só assim poderá se tornar um participante. Assim que efetuar o Registo já poderá interagir e participar nos eventos. Para observar um evento em específico o utilizador pode visualizar um *card* com a informação resumida para participar no evento, terá de selecionar o botão de "Ver Mais".

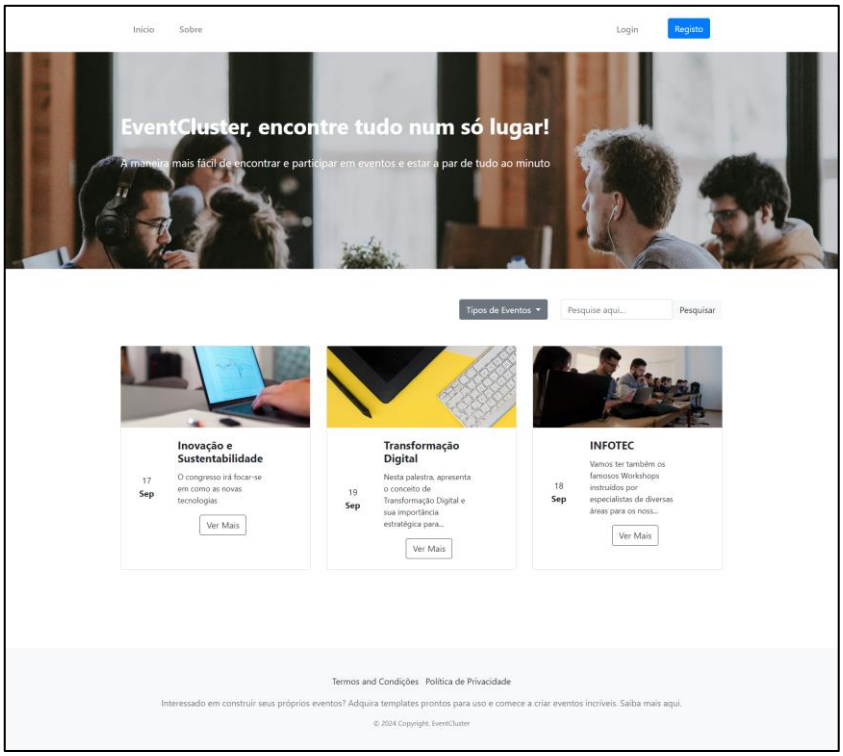

Figura 6 - Página dos Eventos

Na Figura 7 é uma página que contem todas as informações necessárias. No seu rodapé contem um *link* onde o utilizador será direcionado para a *Landing page*, Figura 15.

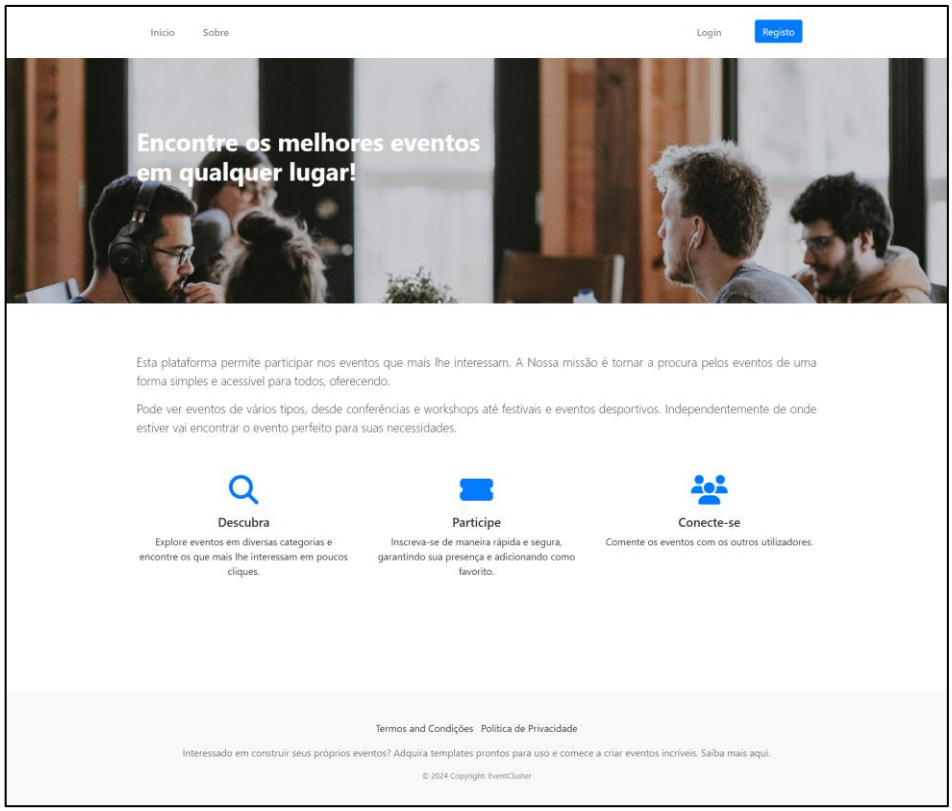

Figura 7 - Página Sobre

O participante ainda poderá colocar o seu evento na listagem dos favoritos, selecionando o coração que estará em cada evento **Figura 8**.

Figura 8 - Selecionar como favorito

3.1.3. Detalhe de Evento

Assim que o visitante selecionar o botão "Ver Mais", como foi apresentado na Figura 6. Será encaminhado para todos os detalhes do evento, Figura 9 .O participante ainda pode comentar o evento, também tem possibilidade de partilhar o evento nas redes sociais. Caso o participante tencione reservar algum bilhete, terá de selecionar no botão "Reservar Bilhete". Caso queria se voluntariar no evento tem uma mensagem para o fazer.

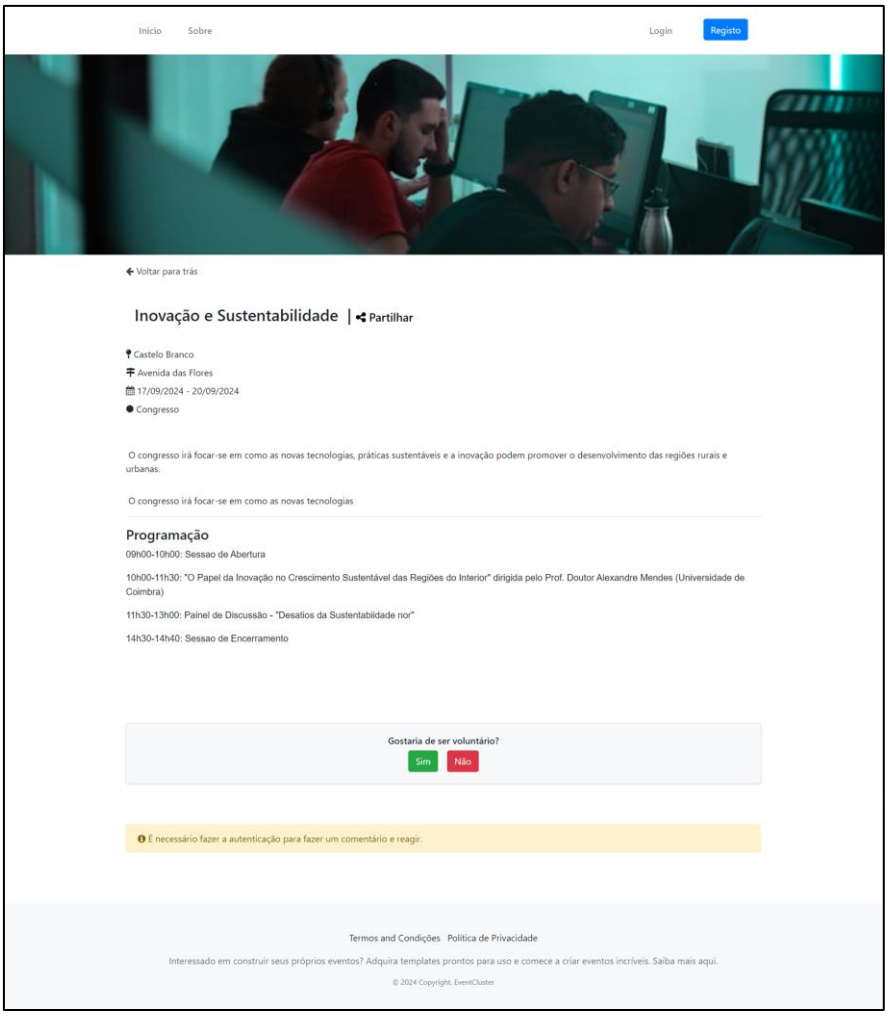

Figura 9 - Visualizar Detalhes do Evento

Caso o visitante queira se voluntariar no evento tem se preencher o formulário que esta na Figura 10, para receber as informações via email.

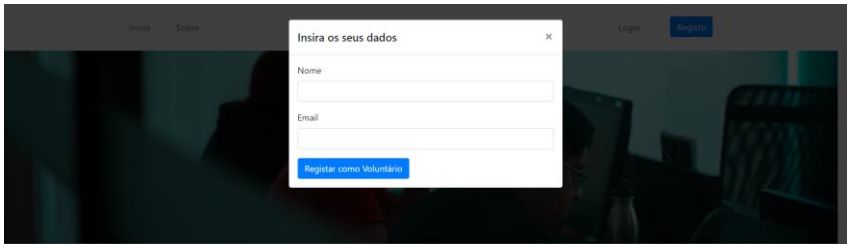

Figura 10 - Inscrever como voluntário

Na Figura 9 será apresentado os detalhes do evento. Nessa mesma página tem uma zona de comentários onde o participante tem a capacidade de escrever um comentário. Como já foi referido anteriormente, o participante terá de efetuar o *login* para conseguir comentar o evento. Também tem a possibilidade de reagir, *like* e *dislike,* aos comentários de outros participantes que já realizaram algum comentário.

No comentário terá o nome do utilizador que comentou, a data que foi realizada e as respetivas reações (*like* e *dislike*), Figura 11.

Figura 11 - Adicionar Comentário e Reagir

Na página de visualizar eventos, Figura 9**,** existe um botão "Partilhar". Assim que o utilizador selecionar o mesmo, irá aparecer um *modal* onde indica qual será o evento, a data e o local. De seguida tem várias opções de partilhar nas redes sociais. São elas o Facebook, X, LinkedIn e o WhatsApp. Caso não pretenda partilhar nas redes sociais basta selecionar o botão "Cancelar", como está representada na Figura 12.

Figura 12 - Partilha do evento das redes sociais

Na Figura 13, apresenta a interface do sistema de reservas de bilhetes de um determinado evento. No *model* apresentado os campos necessários que o participante terá de preencher obrigatoriamente como o telefone e a quantidade de bilhetes. Os campos do nome e do e-mail o sistema já fornece a informação adequada. Estará representado o valor unitário de cada bilhete e também o preço total conforme a quantidade de bilhetes que o utilizador desejar.

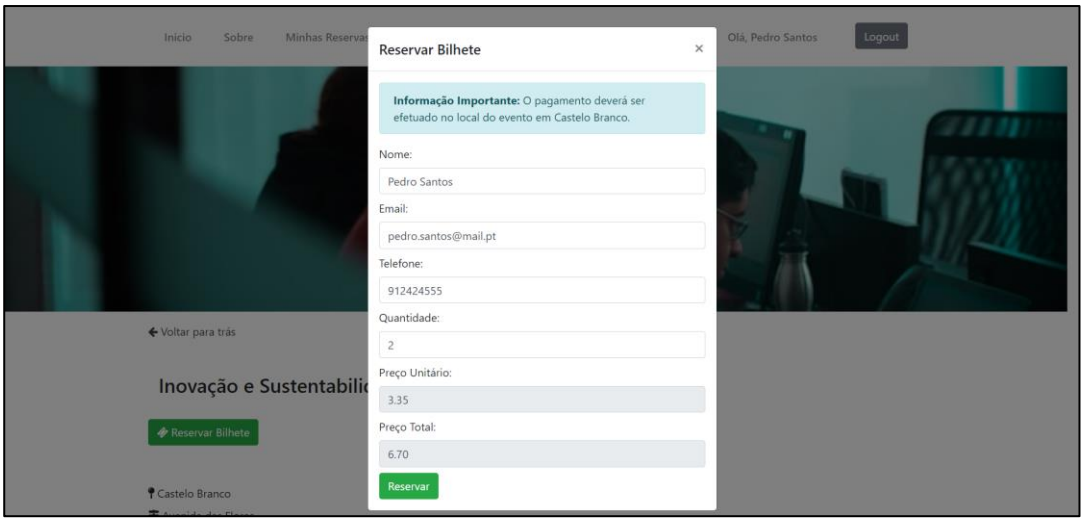

Figura 13 - Efetuar a reserva do participante

Assim que o participante selecionar em "Minhas Reservas", *Figura 13* terá uma área onde pode visualizar as suas reservas dos eventos e onde esta os eventos que o participante marcou como favorito, *Figura 14*.

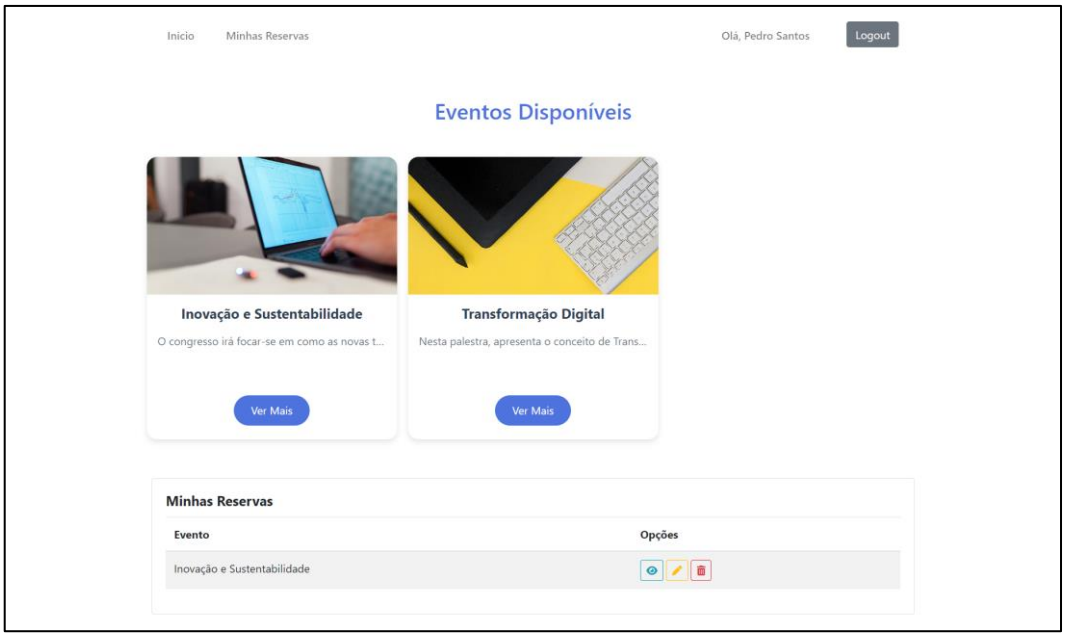

Figura 14 - Listagem dos favoritos dos participantes

3.1.4. Landing Page

O objetivo desta *Landing page* é apresentar exemplos dos *templates* já desenvolvidos, para os utilizadores que pretendem gerir um evento tomarem conhecimento dos *templates*. Na **Figura 15** está representado os modelos que estão disponíveis, por baixo existe uma zona onde o utilizador poderá contactar o administrador de *templates* através de um formulário que o utilizador terá de preencher. Este formulário é o que permite o utilizador comunicar com o

administrador de *templates*, e pedir um *template* personalizável e ter a sua área de gestão de eventos.

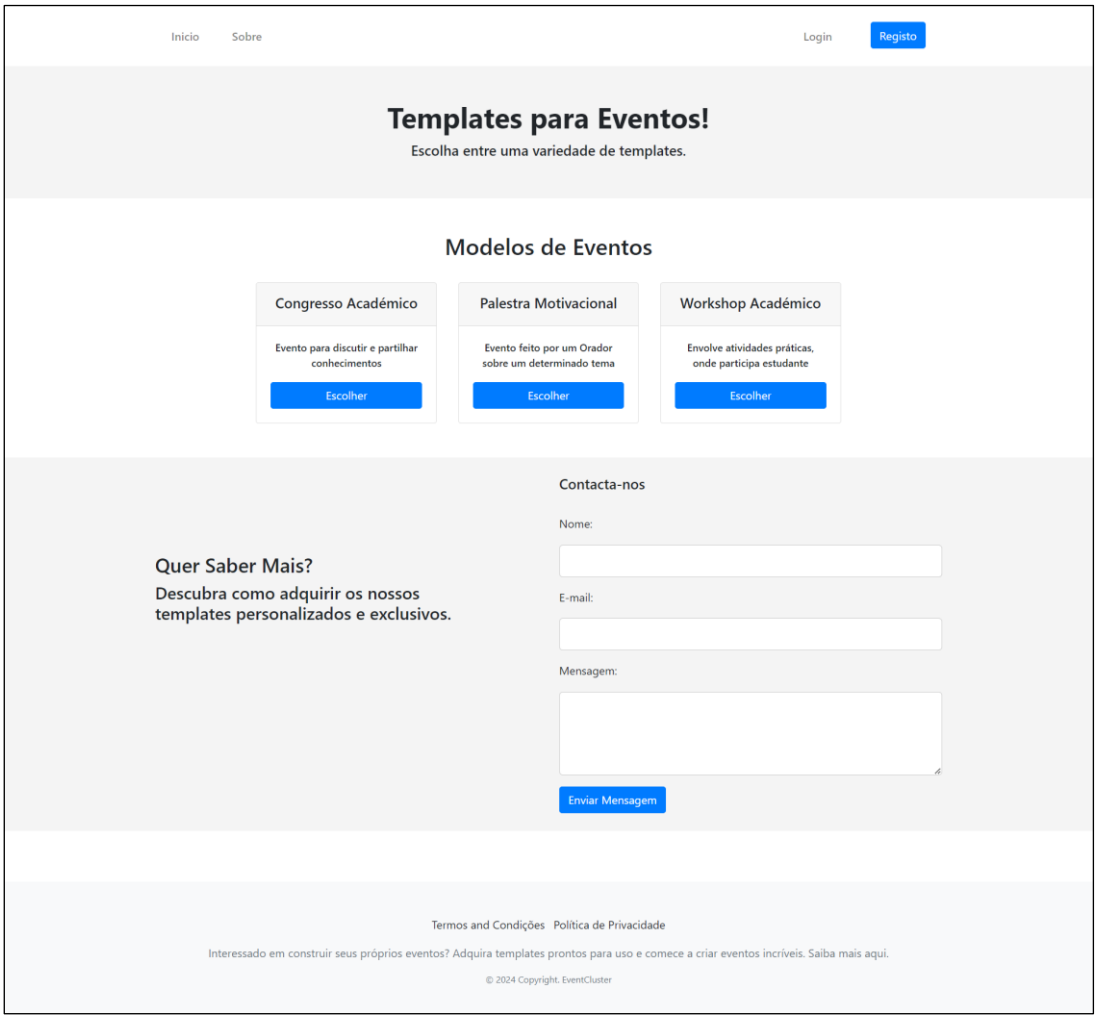

Figura 15 - *Landing page*

3.2. Autenticação

A autenticação no EventCluster é um tópico extremamente importante pois é o responsável por garantir a segurança de dados e a circulação dos mesmos no sistema. De uma forma concreta a autenticação garante que as permissões concedidas aos utilizadores que possuem acesso ao *back-office* são cumpridas e que não são violadas, isto porque, como já foi dito, o *back-office* é uma área de acesso reservado apenas aos utilizadores com acesso ao mesmo.

Assim que o utilizador selecionar na opção *login*, irá aparecer um pequeno formulário como demonstra na Figura 16. Esse formulário tem apenas dois campos para o utilizador possa aceder a sua área. Esses campos são obrigatórios de serem preenchidos, se não aparecerá uma mensagem de erro. Caso o utilizador ainda não possua uma conta, tem a possibilidade de selecionar no *link* "Regista-te", para criar uma conta. Foi ainda adicionado um botão "Visualizar os Eventos" abaixo do formulário, que será direcionado para a página dos eventos Figura 6.

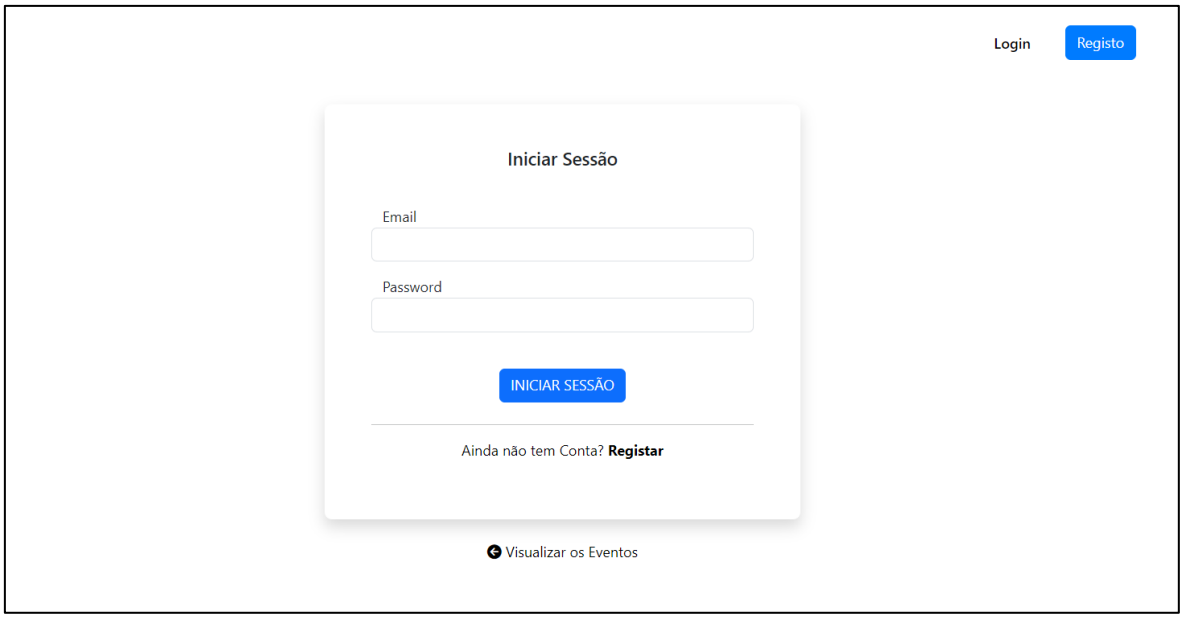

Figura 16 - *Login*

Nesta página, como demonstra a Figura 17 será possível de criar uma conta. Ou seja, o visitante passa a ser participante. Os campos presentes no formulário são obrigatórios de serem preenchidos. Os campos são: o nome do utilizador, o contacto, o e-mail, *password* e por fim a confirmação da mesma, que foi definida anteriormente. Caso não preencha todos os campos será apresentado uma mensagem de erros e não irá deixar avançar com o processo do registo. Se o utilizador já tenha uma conta haverá um *link* que redireciona para o login, Figura 16. Existe um botão "Visualizar os Eventos" foi adicionado abaixo do formulário, que será direcionado para a página dos eventos.

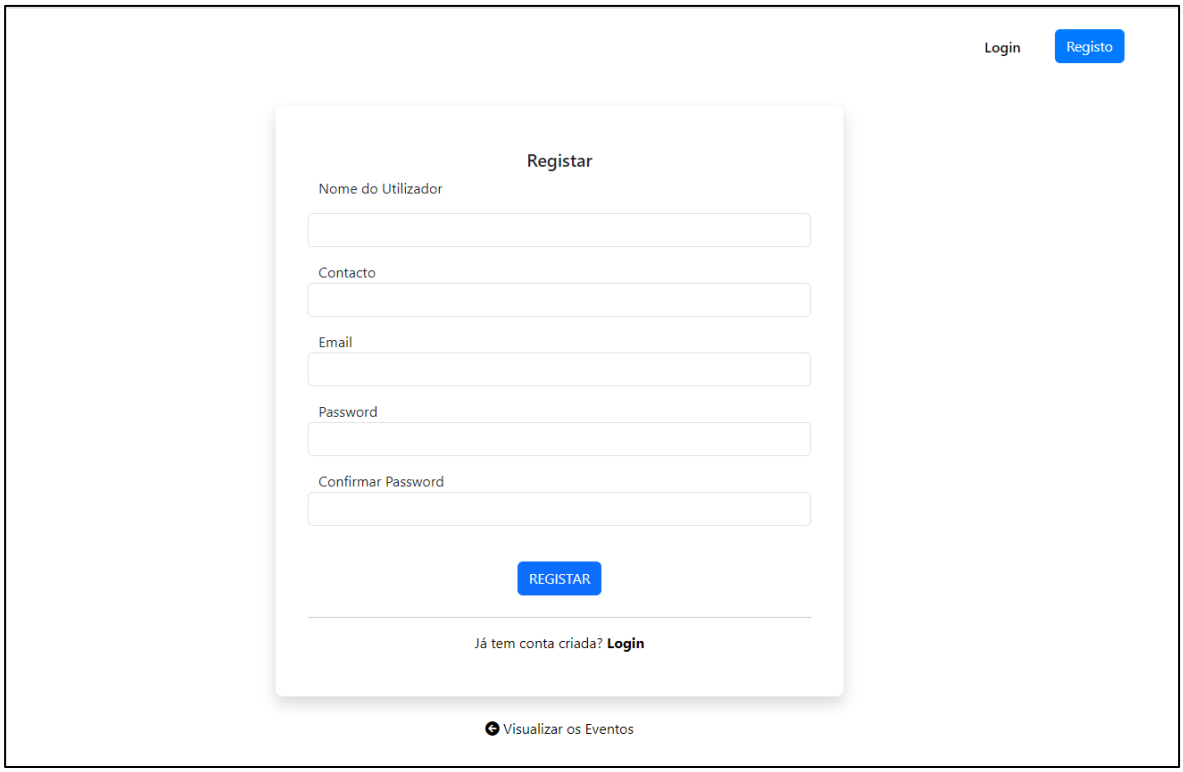

Figura 17 - Registo

3.3. *Funcionalidades e Interface do Back-office*

O *Back-office* consiste em toda a área do sistema de acesso reservado, onde todo o sistema é administrado e mantido. Foi desenvolvido duas grandes áreas de gestão, uma para a gestão de *templates* e outra para a gestão de Eventos.

É lá que se publicam novos eventos, enviam-se mensagens aos voluntários, cria-se os *templates*, entre outras ações de interesse e exclusivas a quem tem acesso ao mesmo.

Nos tópicos seguintes serão abordados os vários focos do *back-office*, onde estes serão contextualizados e explicados a fim de se perceber como foram executados e as funcionalidades que possuem.

3.3.1. Gestão de Templates

A gestão de *templates*, é a área onde ocorre a personalização dos *templates*. Para gerir esta área, o Administrador de *templates* é que é responsável pelo desenvolvimento de *templates* personalizados para cada evento, permitindo adaptar os *templates* às necessidades dos clientes. Com isto, o Administrador é que é responsável pela gestão dos clientes (administradores de eventos). O gestor de *templates*, pode gerir também à sua área pessoal.

3.3.2. Menu Lateral e Barra de Navegação superior

O menu lateral e Barra de Navegação superior está presente em todas as páginas que o Administrador de *templates* pode aceder. No menu de navegação, tem os *links* que são redirecionados para as respetivas páginas.

A páginas são "Dashboard", "*Template*", "Tipos de *Template*" e os "Clientes". O *toggle* é destinado à mudança de tema do ecrã (escuro/claro) a ser visualizado, e o ícone que contém uma imagem associada, é para poder aceder posteriormente à área pessoal do utilizador autenticado, como demonstra a Figura 18.

Figura 18 - Menu do Administrador de *Templates*

3.3.3. Dashboard

Assim que o Administrador de *Templates* fizer a autenticação na página de *Login*, Figura 16, será direcionado para a *dashboard* como se apresenta na Figura 19. Nesta página, são apresentados os *cards* de acesso rápido, para que o processo seja mais facilitado para o administrador. O primeiro *card* apresenta o Total de *templates* que estão desenvolvidos, o mesmo acontece para o segundo *card* que apresenta o Total de Clientes no sistema. Assim que o administrador selecione em "Ver mais", em cada um dos *cards*, será direcionado para as respetivas páginas. O utilizador ainda tem a possibilidade de escrever todas as tarefas que tem de desenvolver.

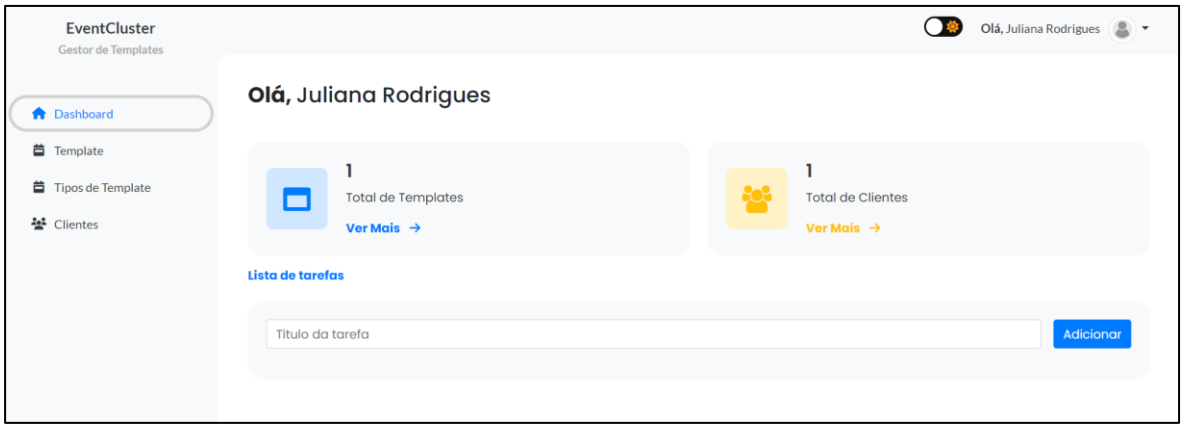

Figura 19 - Dashboard do administrador de *Templates*

Ainda na *Dashboard*, o administrador tem a opção de criar tarefas, Figura 20. Esta funcionalidade foi desenvolvida, pois assim o administrador poderá registar todas as tarefas que desejar. Assim que o administrador escrever a tarefa terá um *dropdown* com o estado da tarefa como: "Em andamento, Pendente e Concluído". Esse estado é apenas para o administrador se guiar. Caso o administrador pretenda eliminar uma tarefa ou várias, terá de a selecionar a(s) tarefa(s) e eliminar.

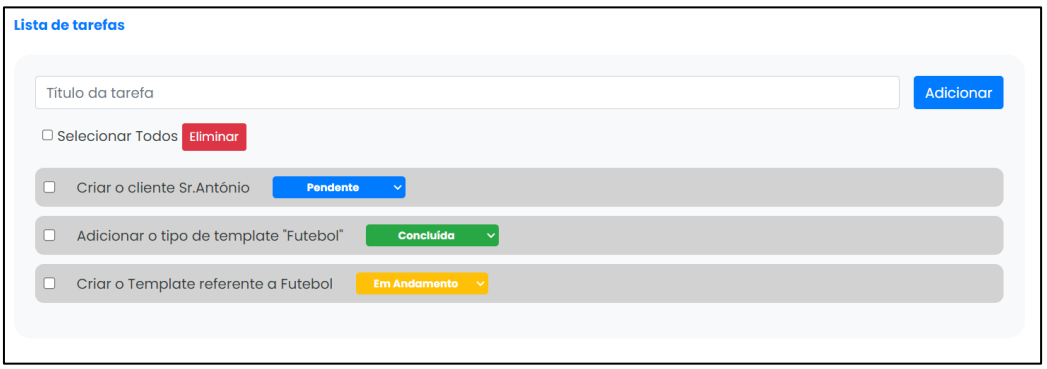

Figura 20 - Lista de Tarefas

3.3.4. *Templates*

A página *template*, apresenta uma tabela com todos os *template* criados**,** Figura 21, onde se pode observar o nome do evento, a data em que foi criado e o tipo de *template*. De seguida contem três opções ícones são eles: Editar, Atribuir Tipo de *template* e eliminar um *template*. No canto superior direito, por cima da listagem contém um botão "Adicionar *template*". Esta secção foi construída com recurso a *Datatables* e os dados estão ordenados por defeito pela ordem de criação.

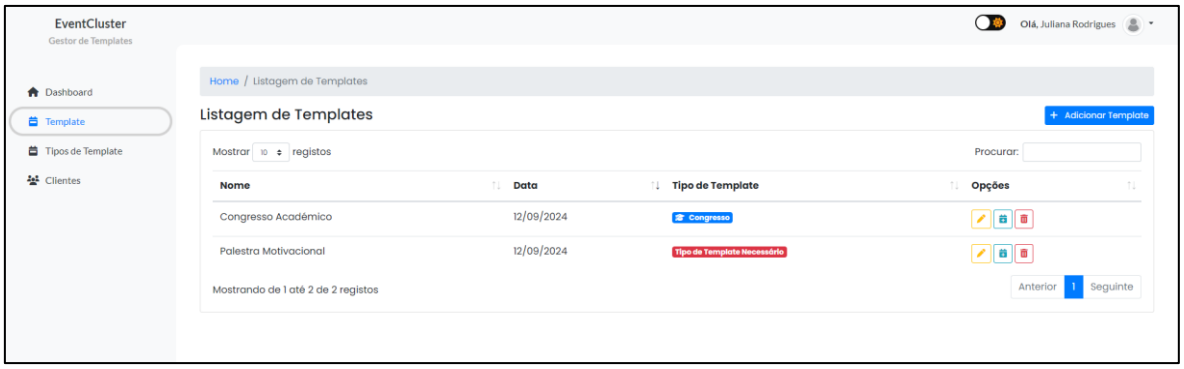

Figura 21 - Listagem de *Templates*

Ao selecionar no botão "Adicionar *Template*", como foi demonstrado na Figura 21, será encaminhado para a página onde será criado o *template* Figura 22. Existe dois campos estáticos no desenvolvimento do *template* será o nome e a descrição.

Todos os outros que são representados na figura abaixo são dinâmicos, ou seja, o utilizador consegue colocar os campos e a ordem que desejar. Os campos que estão disponíveis para criar um *template* são "Data de Início", "Data de Fim", "Cartaz", "Imagem", "País", "Link", "Localização", "Endereço", "Orador", "Título", "Resumo", "Descrição" e por fim "Programação". No entanto, o administrador só pode utilizar cada campo uma única vez, ao criar o *template*. Para se ter os campos dinâmicos recorreu-se ao uso da biblioteca *FormBuilder*.

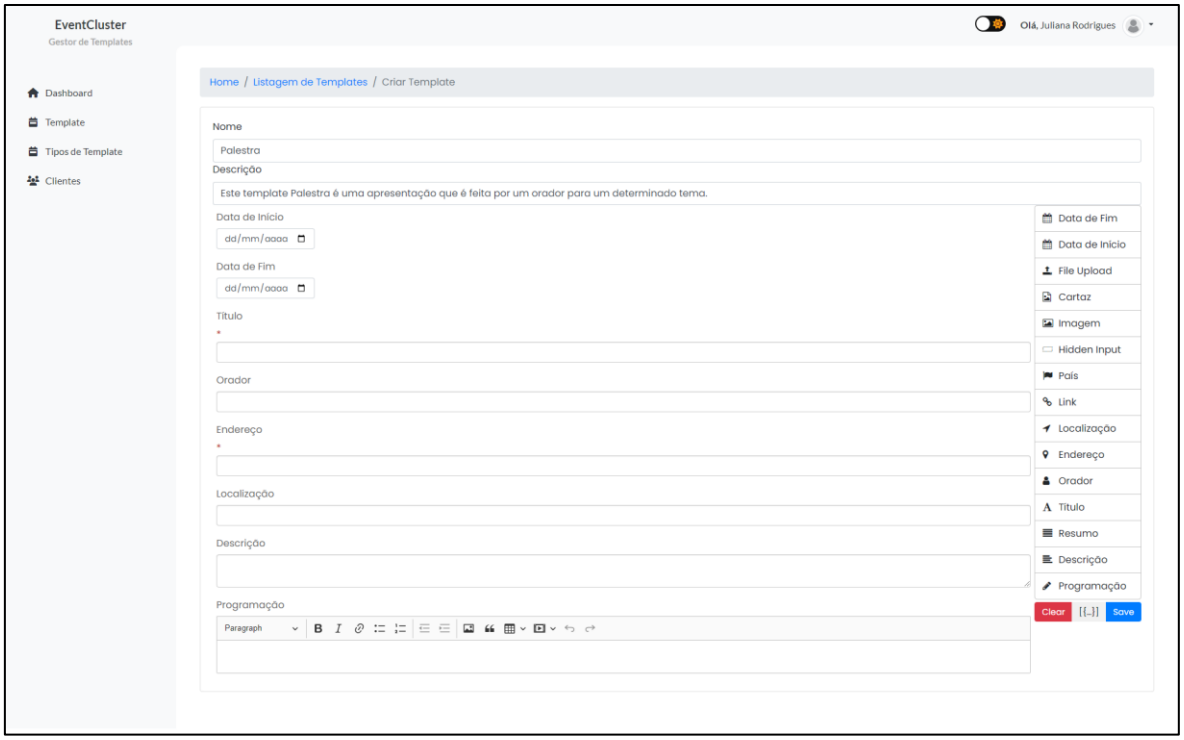

Figura 22 - Criar o *Template*

O administrador de *template* é obrigado a associar um tipo ao *template*, Figura 22. Caso contrário, o *template* não irá estar disponível para o Administrador de Eventos, para atribuir o tipo de específico terá de selecionar no ícone da atribuição, na Figura 23.

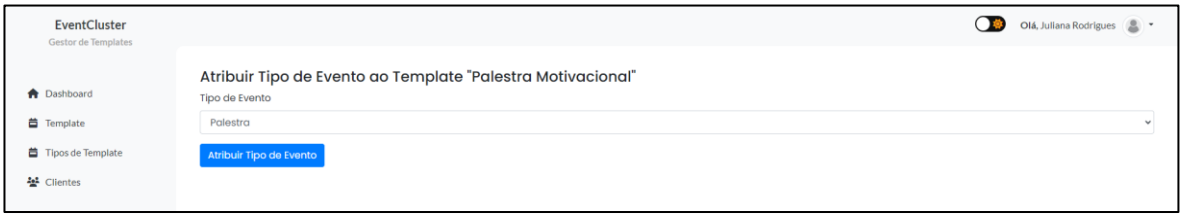

Figura 23 – Escolher Tipo de *Template*

Se o administrador selecionar no botão "Eliminar", que está ilustrado na Figura 21, na listagem de *templates*. Será a exibido um *pop-up* com uma mensagem, para eliminar. O *pop-up* oferece duas opções: o botão "Eliminar", que confirmará a exclusão do *template* na listagem, e o botão "Cancelar", que interrompe a operação.

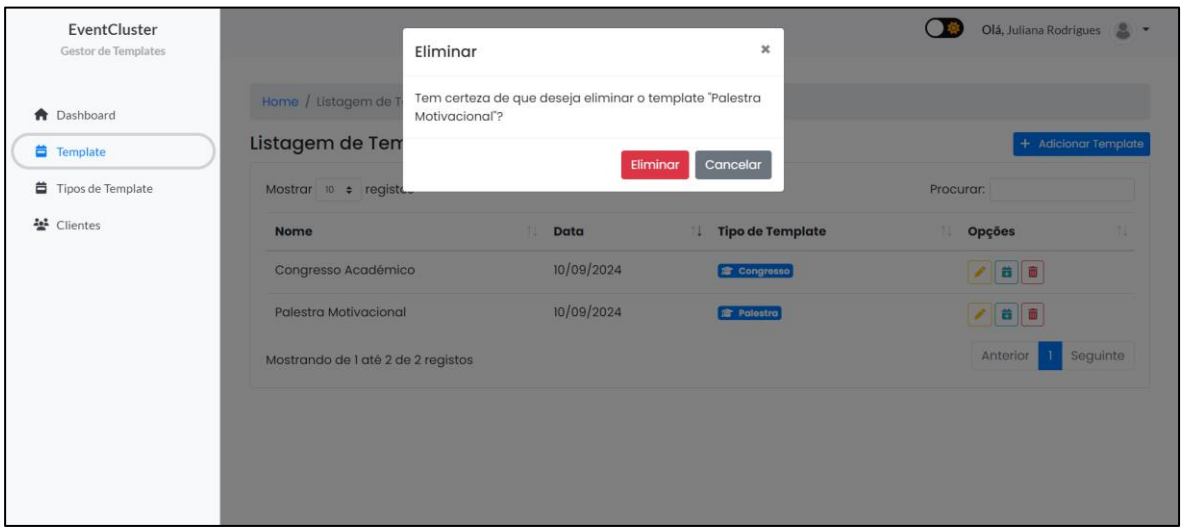

Figura 24 - Eliminar um *Template*

3.3.5. Tipos de *Template*

Quando o administrador seleciona "Tipos de *Templates*" no menu lateral Figura 18, é redirecionado para uma lista de todos os tipos de *templates* previamente criados Figura 25. Nesta página, o administrador pode adicionar um novo tipo de *template*

através do botão "Adicionar Tipo de *Template*" e pode ser editado ou eliminado.

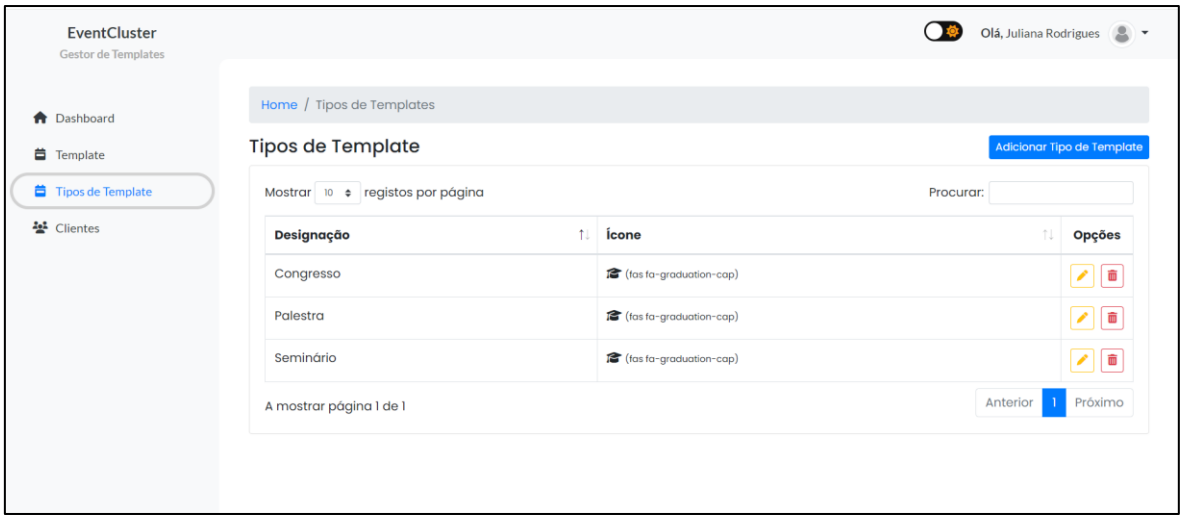

Figura 25 - Listagem de Tipos de *Template*

Para adicionar um tipo de *template*, o administrador terá de selecionar no botão "Adicionar Tipo de *Template*", como demonstra na **Figura 25**. assim que selecionar irá aparecer um *modal*, como está representado na **Figura 26**. O administrador deve atribuir um nome ao *template* e selecionar um ícone que o represente. Após guardar, o *Template* será imediatamente exibido na listagem, com todas as suas informações.

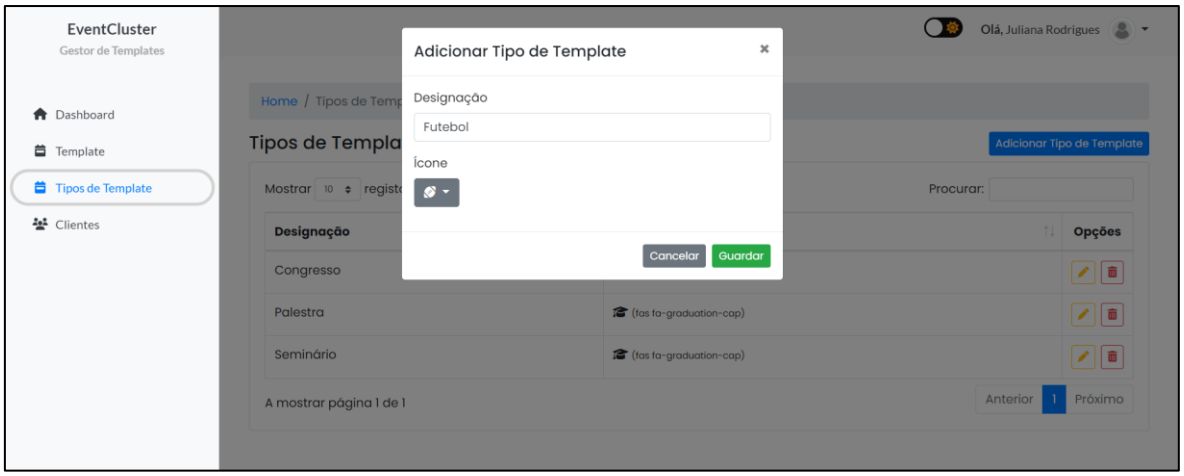

Figura 26 - Adicionar Tipo de *Template*

Para editar um tipo de *template* terá de selecionar no ícone "Editar". De seguida será apresentado um *modal*, como está demonstrado na **Figura 27**. O utilizador altera os campos necessários e guarda. Assim que guardar essa alteração será apresentada na listagem dos tipos de *templates*.

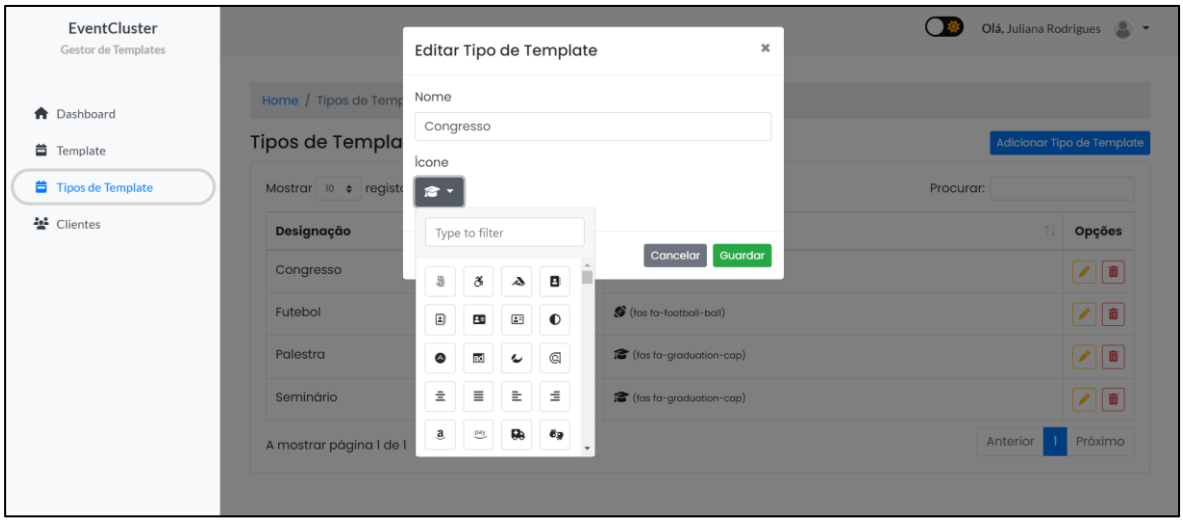

Figura 27 - Editar Tipo de *Template*

O utilizador terá a capacidade de eliminar o tipo de *template* que foi criado. Assim que selecionar no ícone de "eliminar" aparecerá um *modal* onde o utilizador terá de confirmar a exclusão do tipo de *template* específico, como está representado na Figura 28.

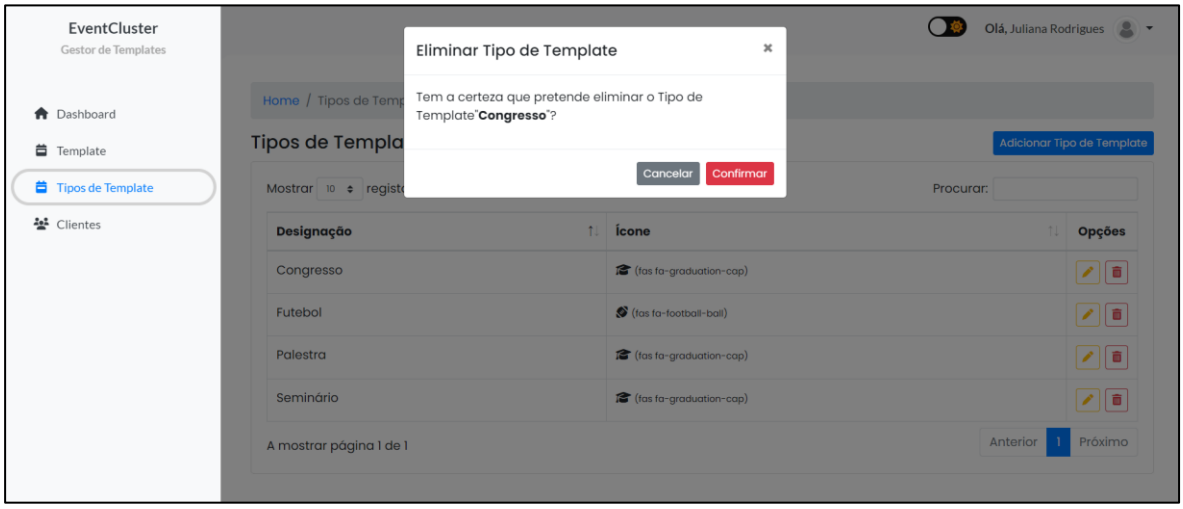

Figura 28 - Eliminar Tipo de *Template*

3.3.6. Clientes

Caso o Administrador de *templates* selecione no menu lateral "Cliente", Figura 18 ou na *dashboard,* Figura 19, será direcionado para uma listagem de todos os clientes. Nessa listagem Figura 29, são exibidos todos os clientes. Na listagem em cada cliente tem as opções de visualizar, editar e eliminar. Nessa mesma listagem pode-se observar a fotografia, o nome, e-mail e o perfil. No canto superior direito da listagem permanece um botão "Adicionar Cliente".

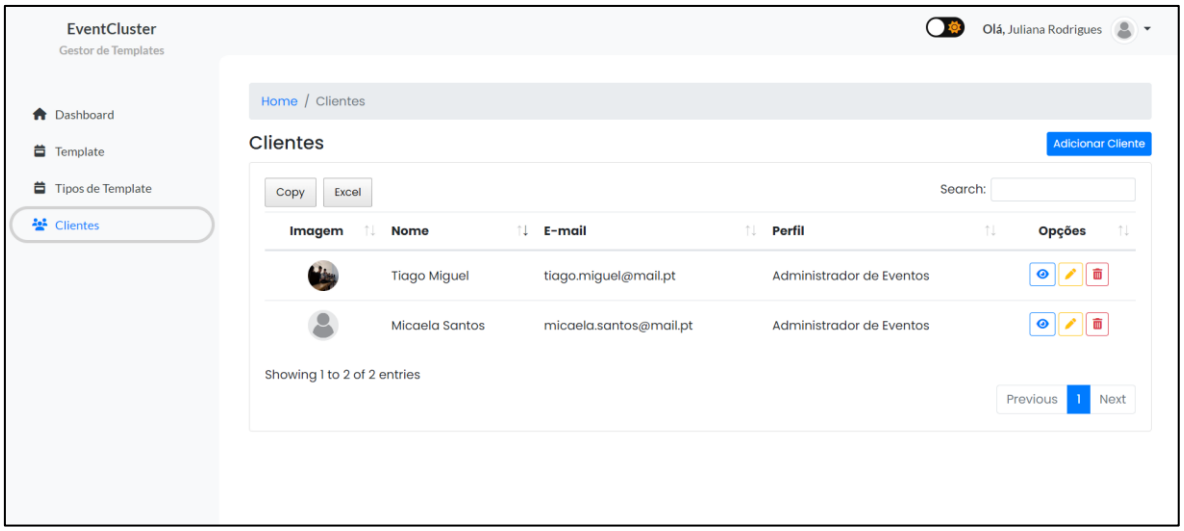

Figura 29 - Listagem de clientes

Ao selecionar o botão "Adicionar Cliente", conforme demonstrado na **Figura 29**, o utilizador será direcionado para a página de criação de clientes, como ilustrado na **Figura 30**. Nesta página, estão disponíveis os campos para inserir o nome, número de telefone, e-mail, palavra-passe, perfil e fotografia do cliente.

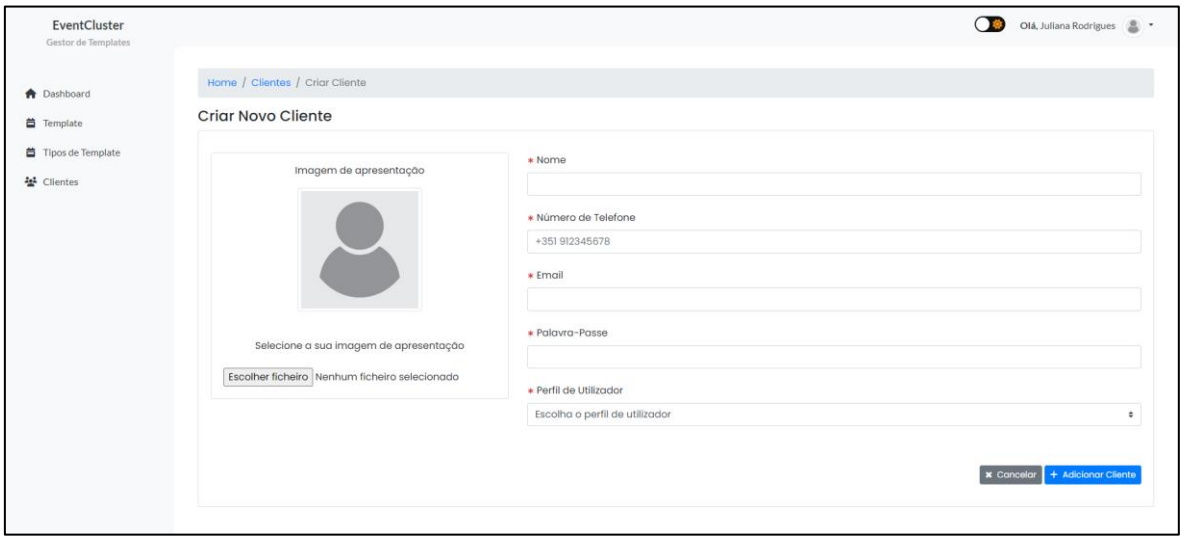

Figura 30 - Criar cliente

Se o Administrador de *templates* selecionar a opção "visualizar", conforme mostrado na Figura 29**,** será redirecionado para a página de detalhes do cliente. As informações que são apresentadas ao administrador incluem o nome do utilizador, e-mail, perfil, data de criação da conta e a fotografia de perfil, como ilustrado na Figura 31.

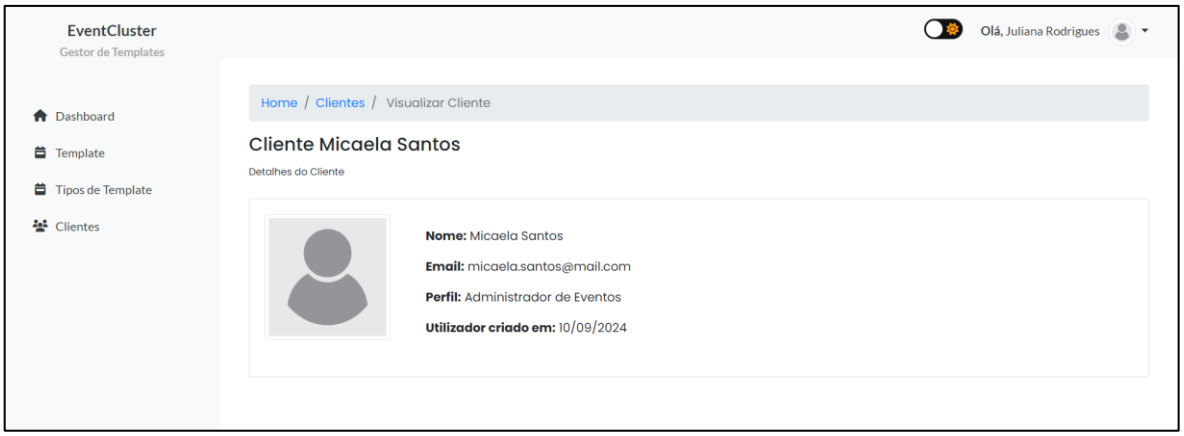

Figura 31 - Visualizar detalhe do cliente

Se o utilizador selecionar na opção "editar", como é representado na Figura 29 será direcionado para uma página onde poderá modificar os campos, como mostrado na Figura 32.

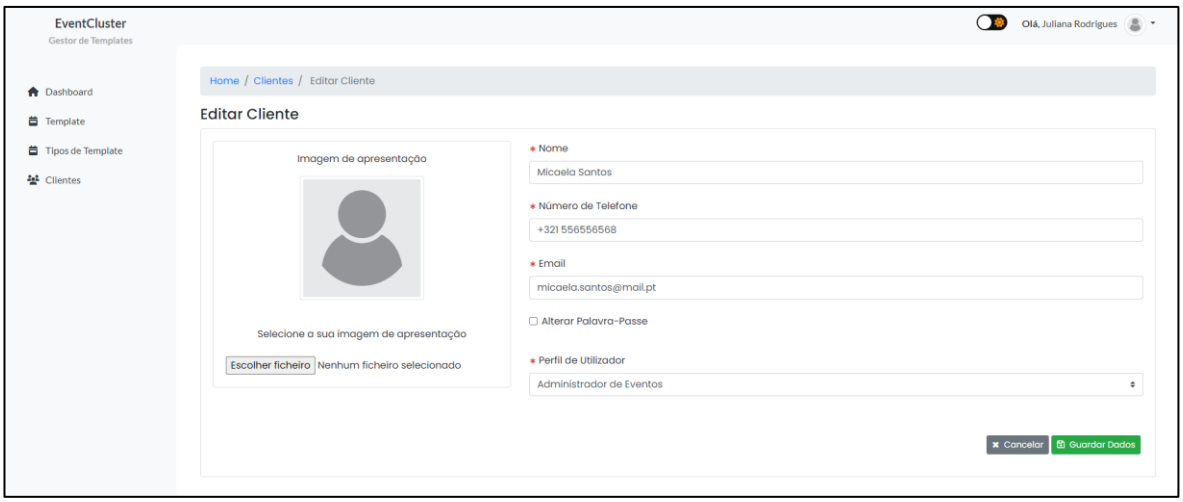

Figura 32 - Editar dados do cliente

Caso o utilizador pretenda eliminar o cliente terá de selecionar em "eliminar", como é representado na Figura 29**.** Assim que selecionar essa opção aparecerá um *modal*, onde o utilizador terá de confirmar se deseja eliminar o cliente, como está representado na Figura 33**.**

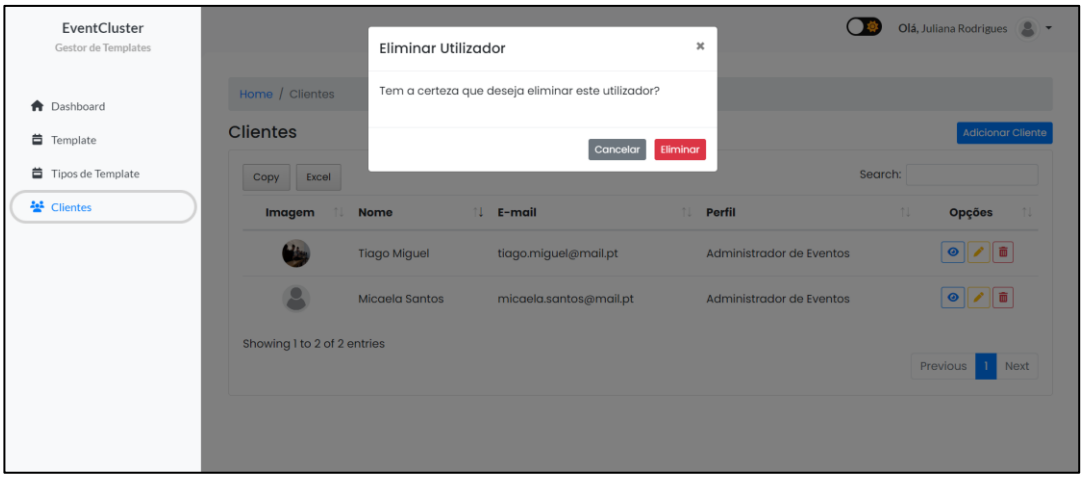

Figura 33 - Eliminar o cliente

3.3.7. Gestão de Eventos

Na gestão de eventos, tanto o menu lateral quanto a barra de navegação superior estão presentes em todas as páginas acessíveis ao Administrador de Eventos, Figura 34. O menu lateral oferece uma navegação consistente, permitindo ao Administrador de Eventos aceder rapidamente as diferentes seções do sistema. O menu é composto por quatro *links*, cada um redirecionando para uma página específica: "Home", "Eventos", "Utilizadores" e "Reservas".

O *link* "Home" leva à página principal do gestor de Eventos, proporcionando uma visão geral das principais funcionalidades. O *link* "Eventos" direciona para a seção onde os eventos podem ser geridos, permitindo a visualização e edição dos mesmos. O *link* "Utilizador" é utilizado para gerir as informações e configurações do utilizador, e o *link* "Reservas" é onde se vai poder gerir as reservas dos eventos.

Figura 34 - Menu do Administrador de Eventos

Assim que o Administrador de Eventos efetuar a autenticação na página de *login*, Figura 16, será direcionado para a *dashboard* como demonstra a Figura 35. As funcionalidades principais estão organizadas em *cards*, facilitando a navegação e

gestão. Cada *card* fornece uma visão geral do total de eventos e rascunhos já desenvolvidos. Além disso, inclui uma área dedicada ao envio de mensagens.

Ao clicar em "Ver mais" em qualquer um dos *cards*, o administrador será redirecionado para a página específica correspondente, permitindo um acesso mais detalhado e direto às funcionalidades e informações relevantes.

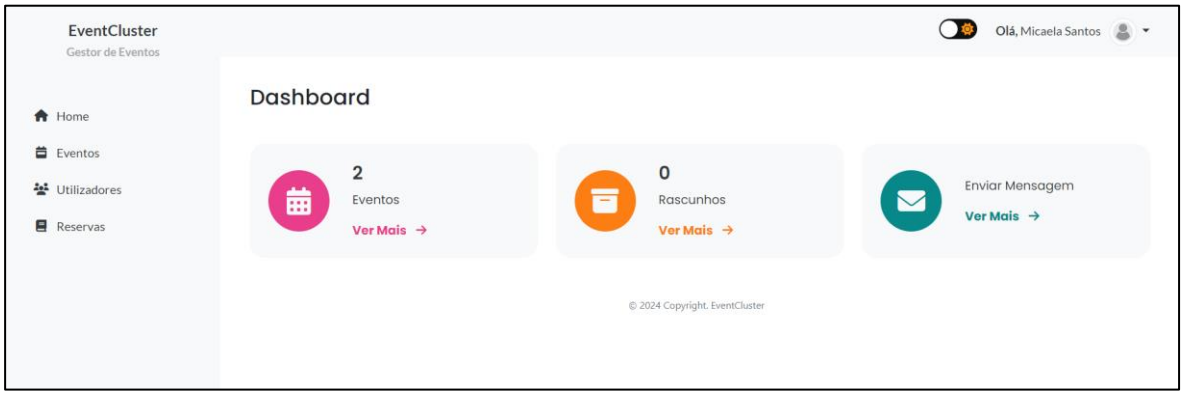

Figura 35 – *Dashboard* do Administrador de Eventos

3.3.8. Eventos

Ao selecionar o botão "Eventos", no menu lateral Figura 34 ou na *dashboard* Figura 35, irá aparecer uma listagem de todos os eventos que estão a decorrer até à data. Na Figura 36, apresenta todos os eventos a data de início, o estado do mesmo e as opções (editar e eliminar). Nesta página ainda tem dois botões na parte superior da lista, de "Histórico de eventos" e outro de "Adicionar Eventos".

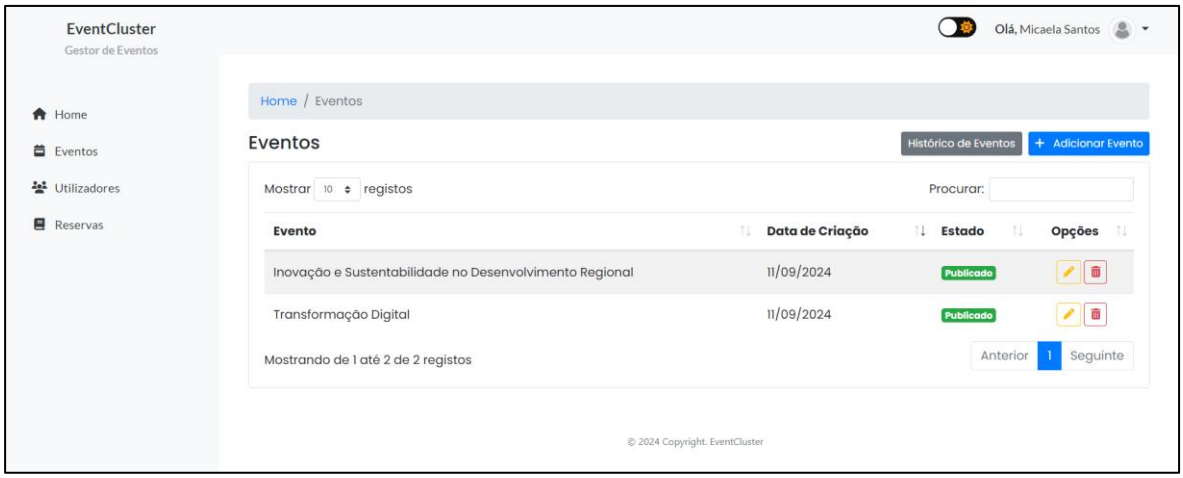

Figura 36 - Listagem de Eventos

Para adicionar um novo evento, o administrador deve clicar no botão “Adicionar Evento”. Primeiro tem de escolher o *template* em que vai criar o evento, conforme apresenta na Figura 37, que exibe todos os *templates* disponíveis fornecidos pelo administrador de *templates*.

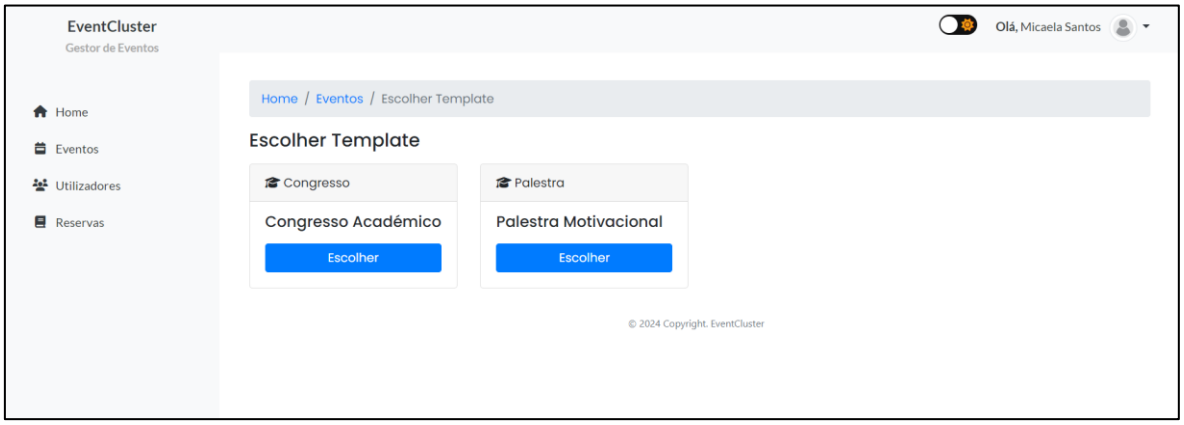

Figura 37 - Escolha do *Template*

Na página "Adicionar Evento", após o administrador selecionar o *template* desejado, conforme ilustrado na Figura 37, será exibido um formulário com todos os campos vazios para preenchimento, como mostrado na Figura 38. Alguns campos são obrigatórios e devem ser preenchidos para que o evento possa ser publicado para os utilizadores. Se o administrador não tiver todas as informações necessárias no momento, pode salvar o evento como rascunho e concluir o evento mais tarde e realizar uma pré-visualização.

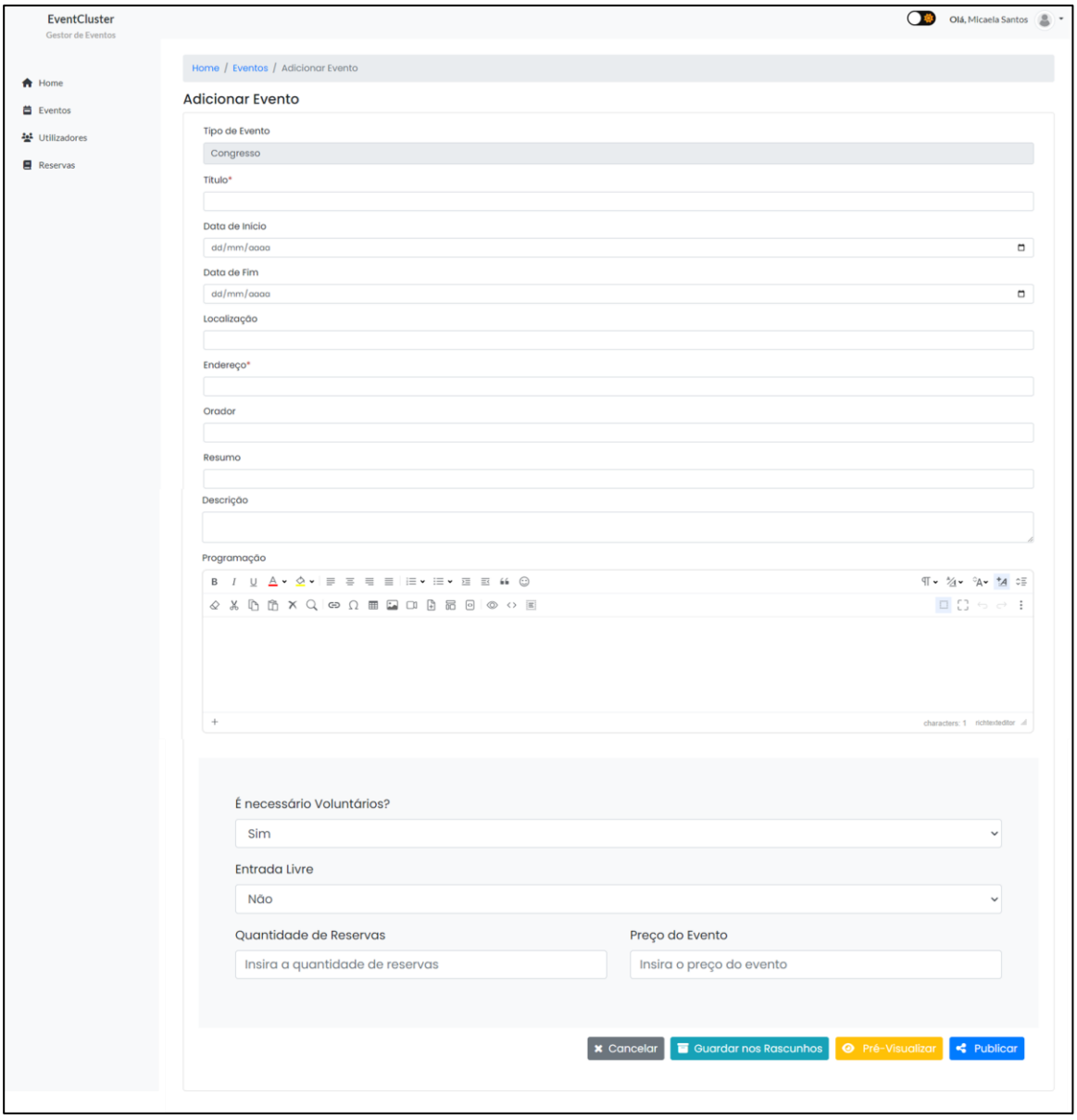

Figura 38 - Adicionar Evento

Assim que o administrador preencha todos os dados necessários do evento, Figura 38, de seguida terá de selecionar o botão "Pré-Visualização". Irá aparecer um *pop-up* em que o utilizador veja como a informação está disposta, Figura 39. Assim que o utilizador verificar se os dados estão todos corretos poderá publicar o evento. Como os dados estão ainda no input e não na base de dados, houve três campos, Título, Localização, e Programação, que não se conseguiu passar na pré-visualização.

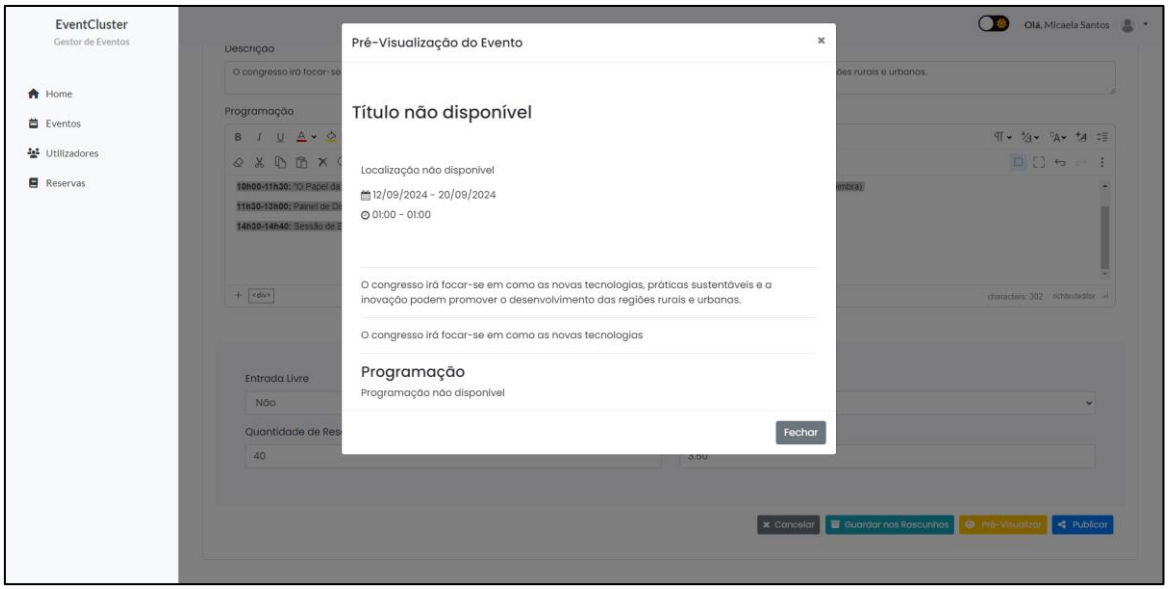

Figura 39 - Pré-Visualização do Evento

Na página "Editar Evento", ao selecionar a opção “editar”, o Administrador de Eventos será redirecionado para a interface apresentada na Figura 40. Nesta página, todos os campos já estarão automaticamente preenchidos com as informações específicas do evento. O administrador poderá então rever e alterar os campos que pretende. Após fazer as alterações necessárias, poderá salvar as alterações para atualizar as informações do evento. Há também uma seção onde o administrador pode especificar se o evento é gratuito ou pago, a quantidade de bilhetes disponíveis e o valor de cada bilhete. O administrador pode rever todas as alterações realizadas clicando em ‘Pré-visualizar’.

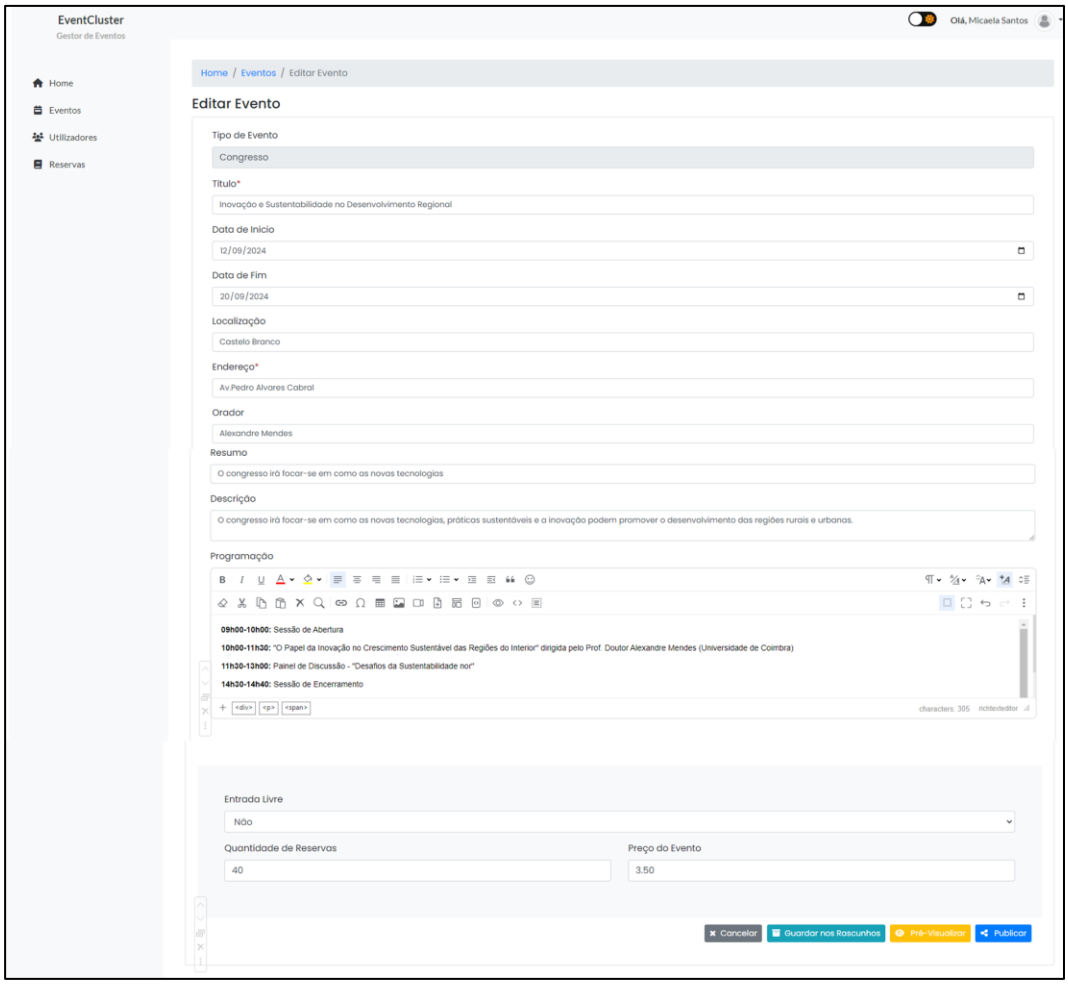

Figura 40 - Editar Evento

Para eliminar um evento, o administrador deve selecionar o botão "eliminar", conforme mostrado na Figura 36. Será então exibido um *pop-up*, ilustrado na Figura 41**.** Se o administrador confirmar, o evento será removido da listagem e transferido para o "Histórico de Eventos", conforme demonstrado na Figura 42.

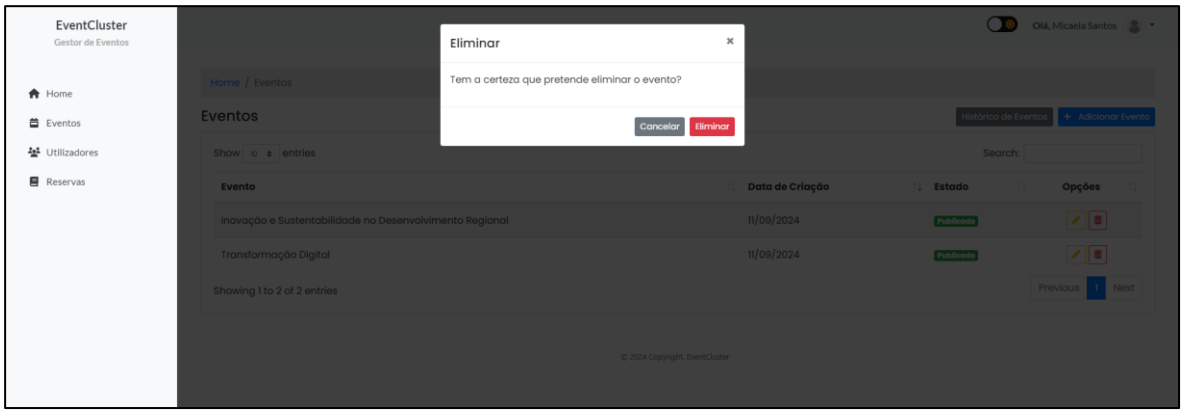

Figura 41 - Eliminar Evento

Se o administrador de eventos selecionar no botão "Histórico de Eventos" Figura 36, será direcionado para a página do histórico de eventos. Nesta página, será exibida uma lista de todos os eventos que já ocorreram ou que foram eliminados. A listagem

incluirá a data de conclusão de cada evento, o seu estado e a opção para visualizar detalhes, conforme mostrado na Figura 42.

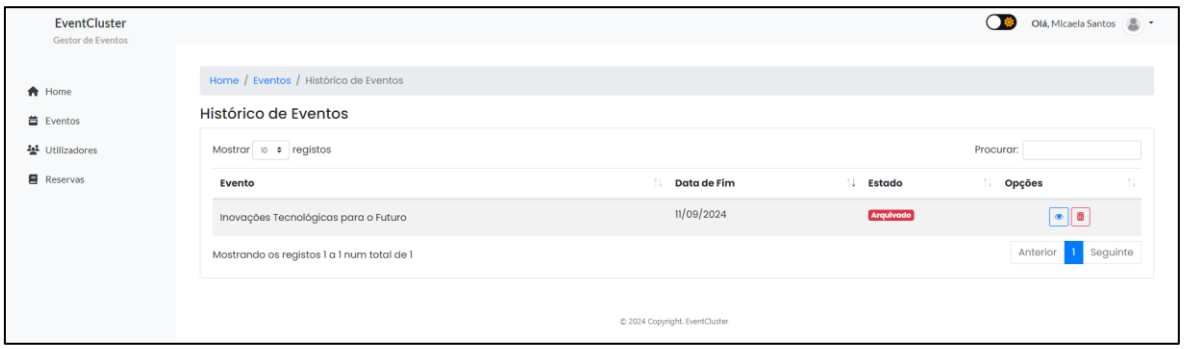

Figura 42 - Histórico de Eventos

Como foi descrito na Figura 42, ao selecionar "Visualizar detalhes". Irá revelar todas as informações presentes do evento em específico, como se demonstra na Figura 43.

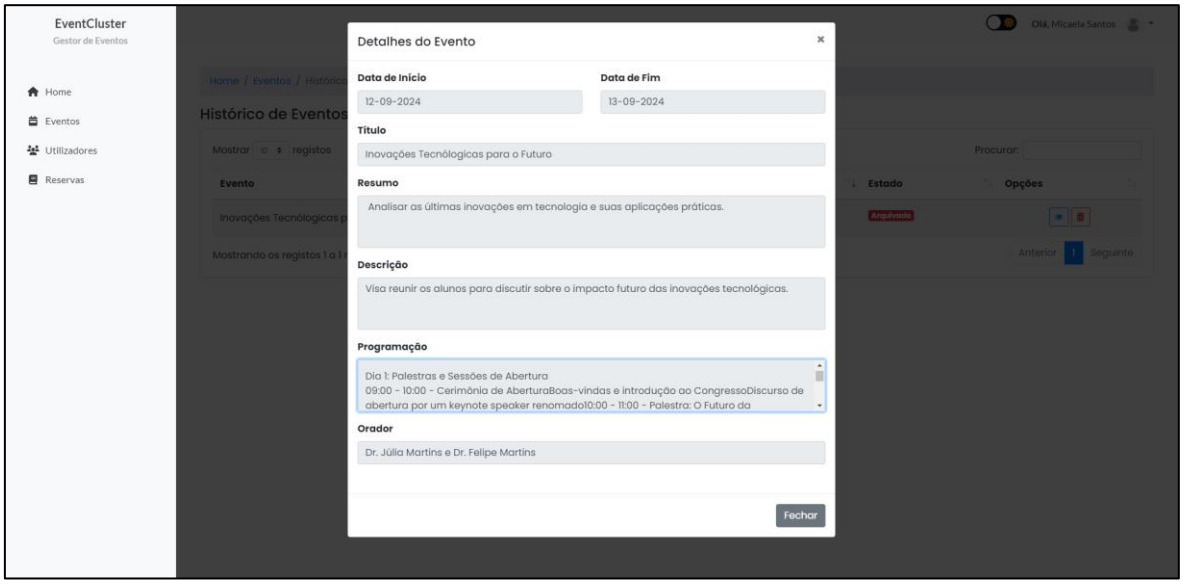

Figura 43 - Visualizar Evento no Histórico

Na Figura 44 está representado um *pop-up* onde o administrador terá de confirmar se quer excluir permanentemente o evento escolhido. Caso confirme o mesmo será excluído da listagem e será eliminado da base de dados.

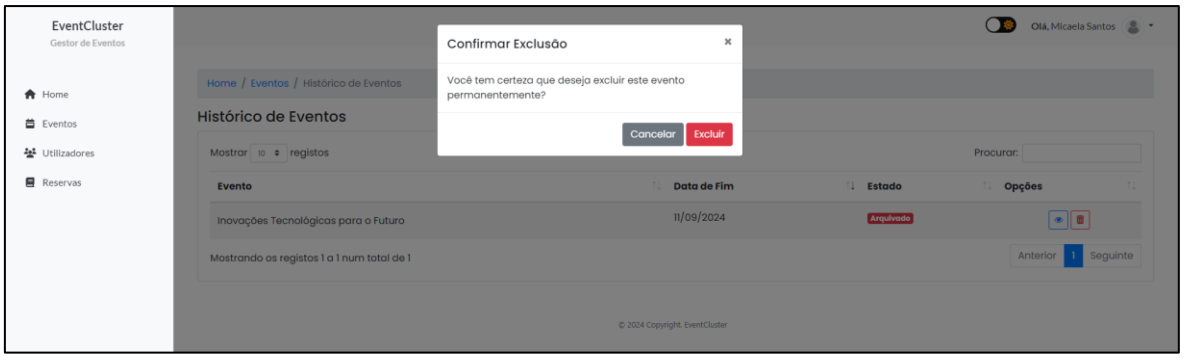

Figura 44 - Eliminar o evento no histórico

3.3.9. Rascunhos

Para aceder aos rascunhos o Administrador de Eventos tem de aceder à *dashboard* na Figura 35. Na Figura 45 aparece uma listagem como todos os rascunhos dos eventos que foram guardados pelo administrador de eventos. Também tem as opções de editar o evento ou eliminar o mesmo caso pretenda.

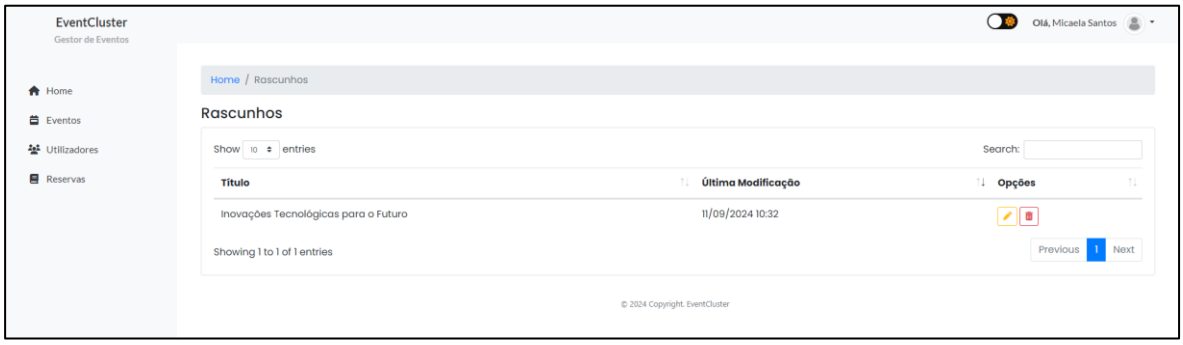

Figura 45 - Listagem dos rascunhos

Como descrito anteriormente, o Administrador de Eventos tem a capacidade de editar o evento que foi guardado nesta listagem dos rascunhos Figura 45. Ao selecionar o botão 'editar', será exibido o formulário já preenchido, permitindo que o administrador modifique o que for necessário. Tal como na Figura 40, referente à edição de eventos, o administrador pode alterar se o evento é gratuito ou pago, a quantidade de bilhetes disponíveis e o valor por cada bilhete. Depois de fazer as alterações, o utilizador pode guardá-las e o evento permanecerá na mesma listagem.

Caso o administrador deseje eliminar o evento dos rascunhos, será exibido um *pop-up* confirmando se realmente deseja eliminar o evento, conforme mostrado na Figura 41, referente à eliminação de eventos. Se o evento for eliminado, ele não aparecerá mais na listagem de rascunhos.

3.3.10. Mensagens

Caso o utilizador selecione em “Enviar Mensagem” na *dashboard*, Figura 35, será direcionada para uma página onde o Administrador de Eventos tem de selecionar o voluntário para quem quer enviar mensagem, o assunto e a mensagem. Como está apresentado na Figura 46.

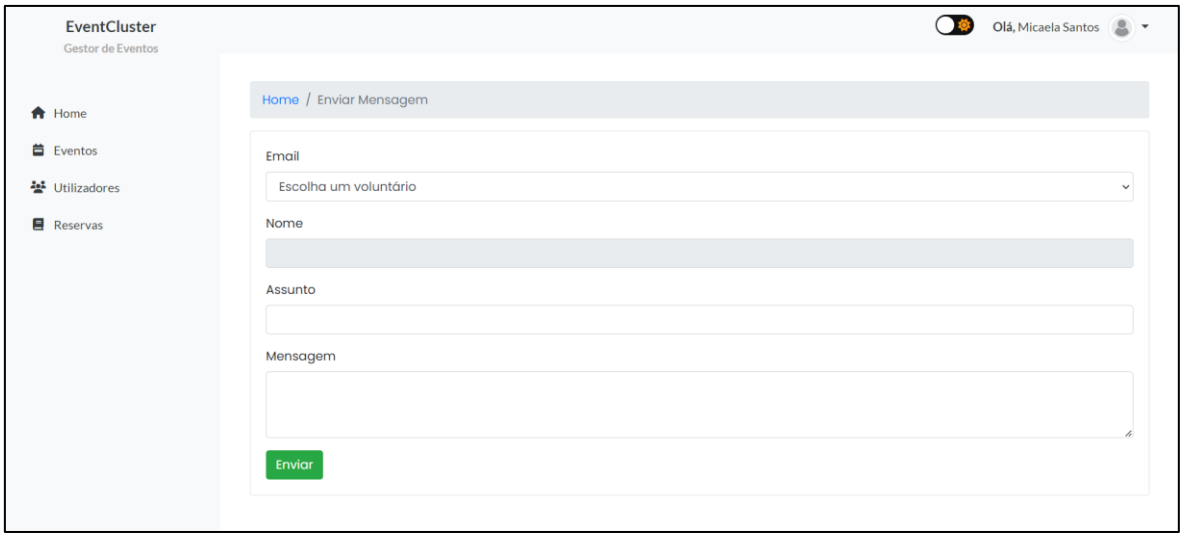

Figura 46 - Enviar Mensagem

3.3.11. Utilizadores

Se o Administrador de Eventos selecionar em "Utilizadores" no menu lateral, Figura 34. Será direcionado para uma listagem de todos os utilizadores que estão registados. Nesta página o administrador tem acesso mais rápido a algumas informações dos utilizadores como a sua fotografia, o nome, o e-mail e o perfil (se é participante ou voluntário). Ainda tem as opções de visualizar com mais detalhe o utilizador, editar o mesmo ou eliminar. Tal como demonstra na Figura 47, ainda existe um botão "Adicionar utilizador" no canto superior direito.

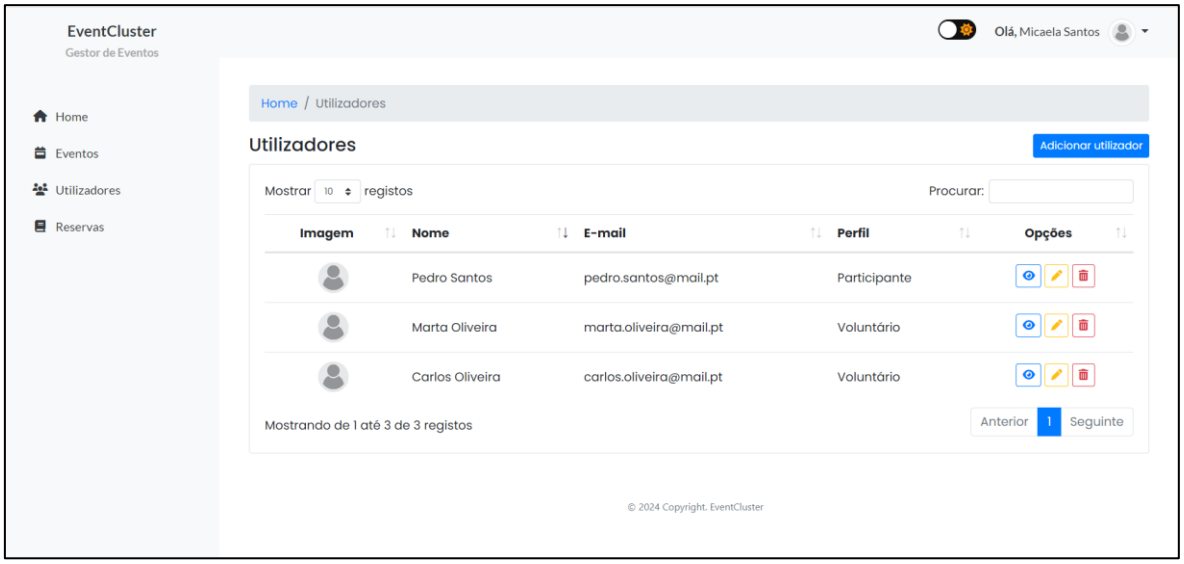

Figura 47 - Listagem de Utilizadores

Ao selecionar o botão "Adicionar Utilizador", como foi apresentado na Figura 47. Será direcionado diretamente para a Figura 48. A Administrador de Eventos poderá inserir os detalhes do novo utilizador, como nome, número de telefone, email, palavra-passe e perfil de utilizador (Participante ou Voluntário).

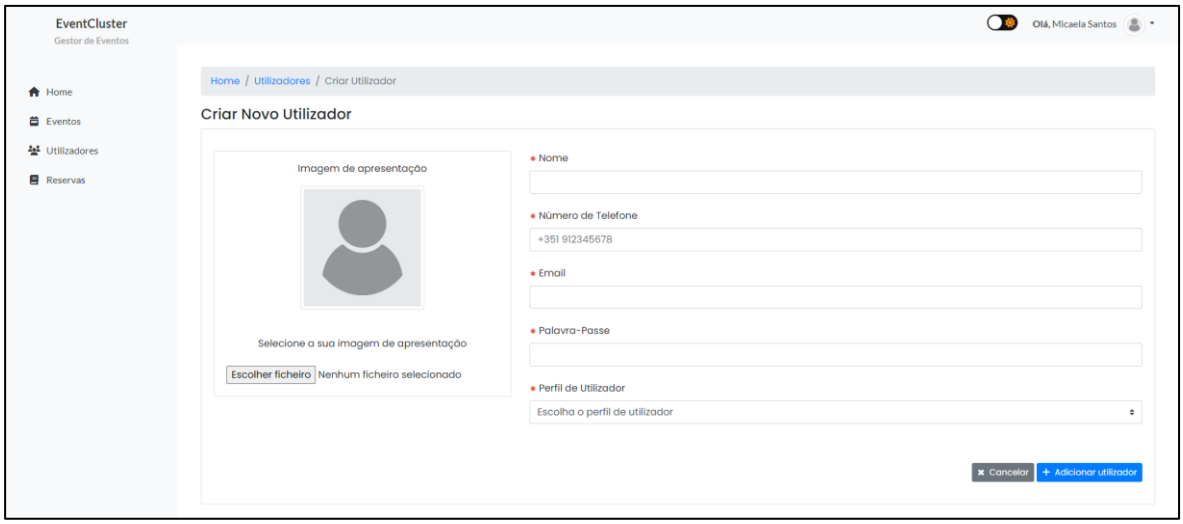

Figura 48 -Criar um Utilizador

Na Figura 47 apresenta uma opção de visualizar com detalhes, os dados do utilizador. Assim que o administrador selecionar esse botão será direcionada para uma página, em que irá apresentar todos os dados pessoais de um determinado utilizador, Figura 49. As informações que vão ser exibidas para o administrador são: o nome, o email, o perfil, a data em que a conta foi criada e a fotografia do utilizador.

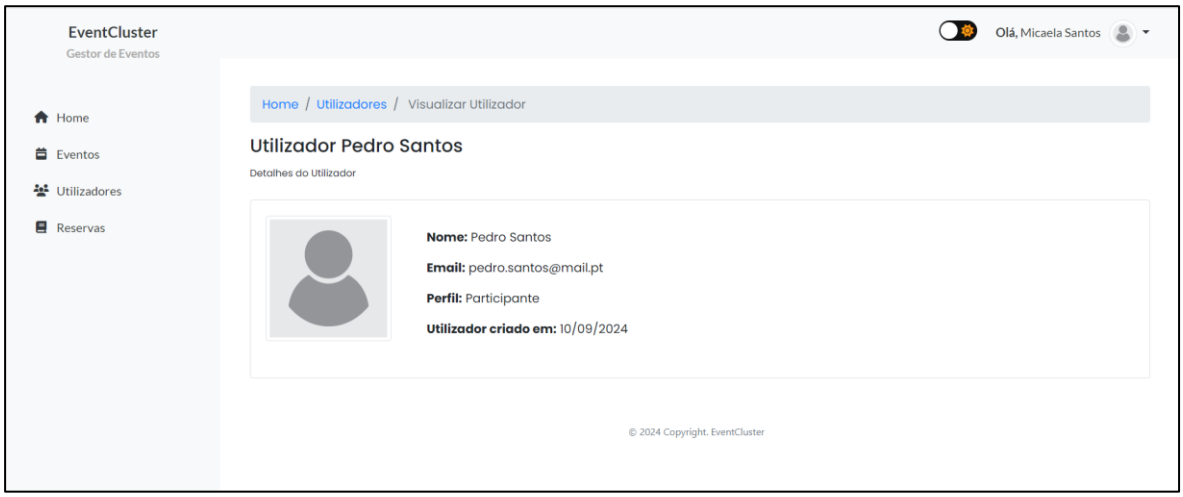

Figura 49 - Visualizar Utilizador

Na Figura 47 apresenta uma opção de editar. Quando o administrador seleciona no botão de “editar” será direcionado para a página de editar utilizador, Figura 50. Todos os campos estão preenchidos, mas o administrador tem a possibilidade de alterar, caso for necessário. Os campos que estão disponíveis para receber essas alterações são: o nome do utilizador, o número de telefone, o e-mail, o perfil do utilizador e a fotografia.

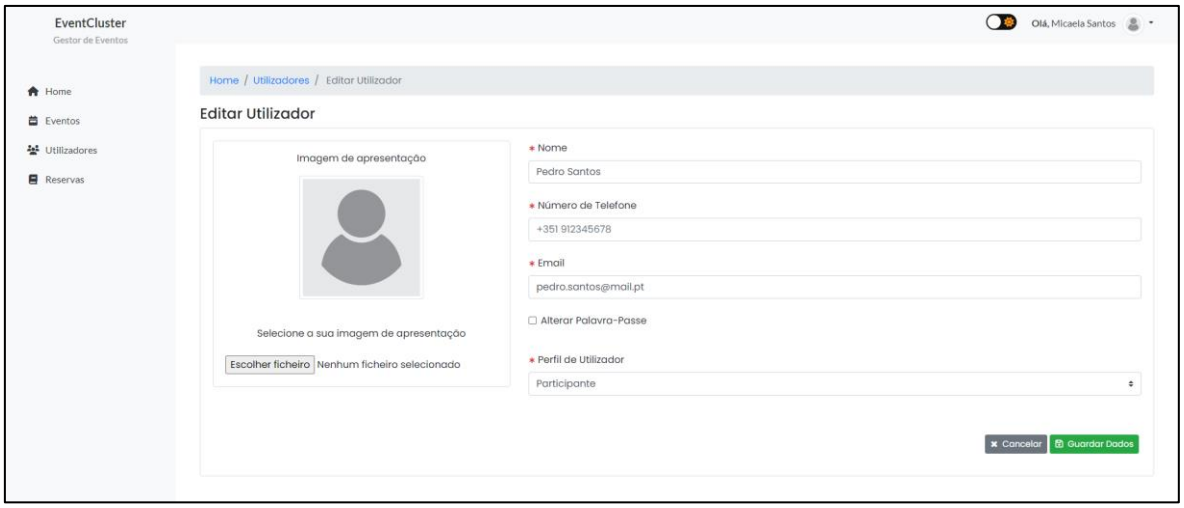

Figura 50 - Editar Utilizador

Na Figura 47 apresenta uma opção de eliminar, assim que o Administrador de Eventos o pressiona irá aparecer um *pop-up*, onde apresenta uma mensagem para saber se deseja eliminar o utilizador. O administrador tem a opção de eliminar ou cancelar, como demonstra na Figura 51. Caso seja confirmada esta operação o utilizador selecionado será eliminado na listagem.

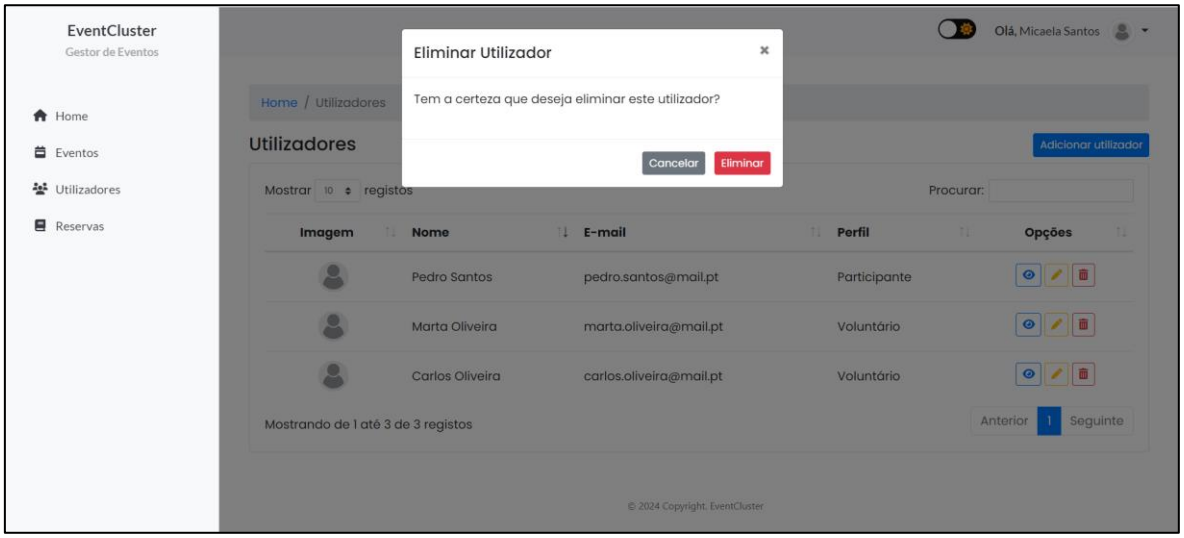

Figura 51 - Eliminar Utilizador

3.3.12. Reservas

O administrador seleciona em "Reservas" no menu lateral, Figura 34**.** Será direcionado para uma listagem de todas as reservas, onde será apresentado o nome do evento e a quantidade de bilhetes para um determinado evento**.** No canto superior direito permanece um botão "Adicionar Reserva", como demonstra na Figura 52.

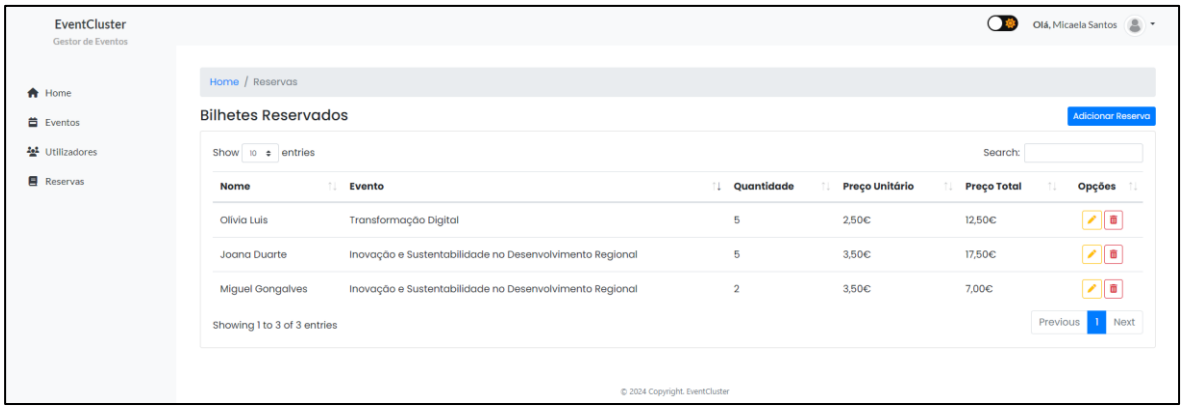

Figura 52 - Listagem dos bilhetes reservados

Assim que o administrador pretenda adicionar uma reserva, terá de selecionar o botão "Adicionar Reserva", Figura 52. Assim que pressionar no botão irá aparecer imediatamente para a página de reservas de bilhetes, como demonstra na Figura 53. O administrador terá de preencher os campos obrigatórios. Terá de escolher qual é o evento que pretende comparecer, terá o nome do utilizador, o contacto, o e-mail e por fim a quantidade de bilhetes. Conforme o número de bilhetes adquiridos será apresentado o valor do bilhete. Assim que todos os campos forem preenchidos o administrador terá de guardar.

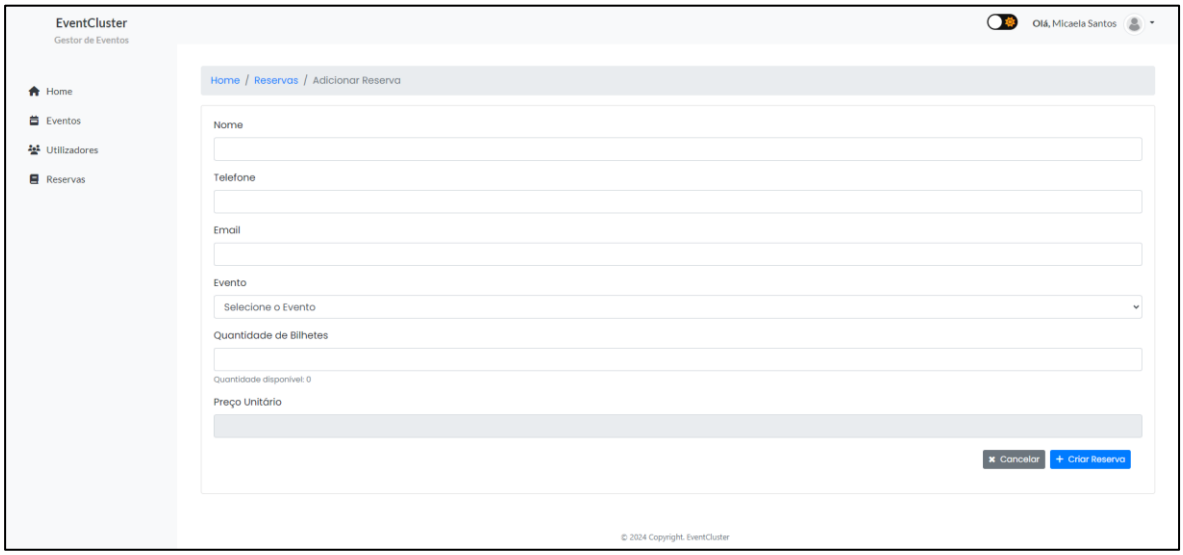

Figura 53 - Adicionar Reserva

Quando o administrador editar uma reserva, deverá selecionar o botão “Editar”, conforme na Figura 52. Depois é direcionada para a página para editar os dados Figura 54, o administrador poderá modificar os detalhes da reserva, como o evento associado, as informações do utilizador nome, contacto, e-mail, e a quantidade de bilhetes reservados.

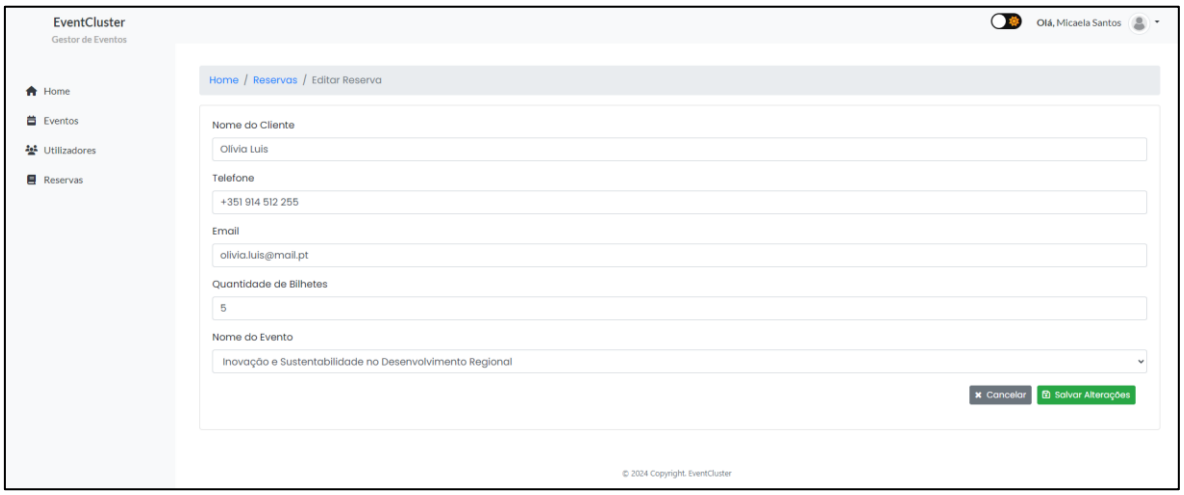

Figura 54 - Editar a reserva

Para eliminar uma reserva, deverá selecionar o botão “Editar”, conforme na Figura 52. Ao clicar no botão, será exibido um *pop-up* de confirmação para garantir que o administrador realmente deseja excluir a reserva na Figura 55.

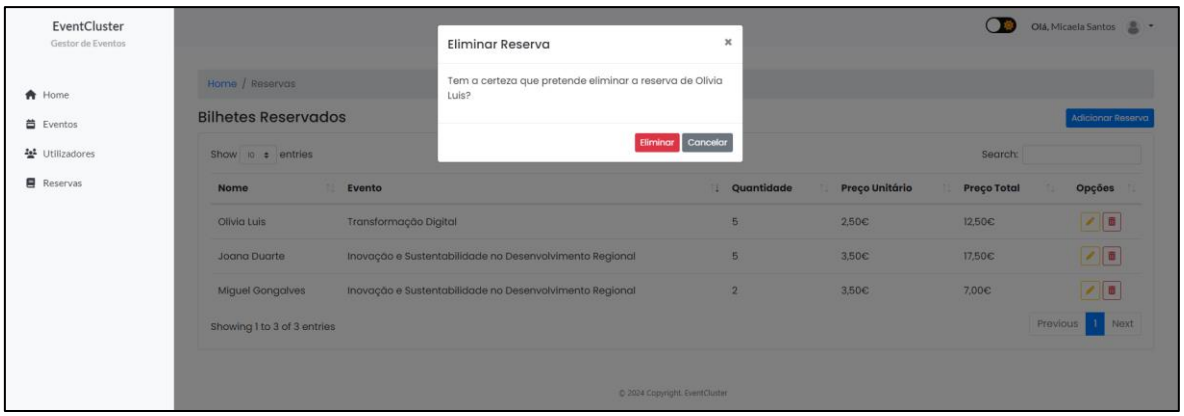

Figura 55 - Eliminar a reserva

3.4. Área pessoal

Assim que o utilizador selecionar na sua fotografia/ícone na barra de navegação, Figura 34 irá aparecer um *dropdown,* como se demonstra na Figura 56**.** O utilizador tem as opções para visualizar o perfil e fazer as alterações necessárias nas definições, e para sair do sistema, sendo redirecionado automaticamente para a página de *Login.*

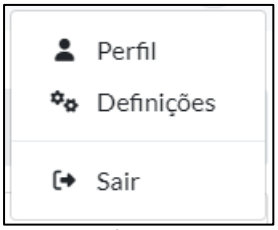

Figura 56 – Área pessoal do Utilizador

Caso o utilizador selecione na opção "Perfil", Figura 56, será direcionado diretamente para a página dos dados pessoais do utilizador, Figura 57. De seguida contem todas as informações pessoais do utilizador tais como: o nome, o e-mail, o perfil e a data em que o utilizador criou a conta.

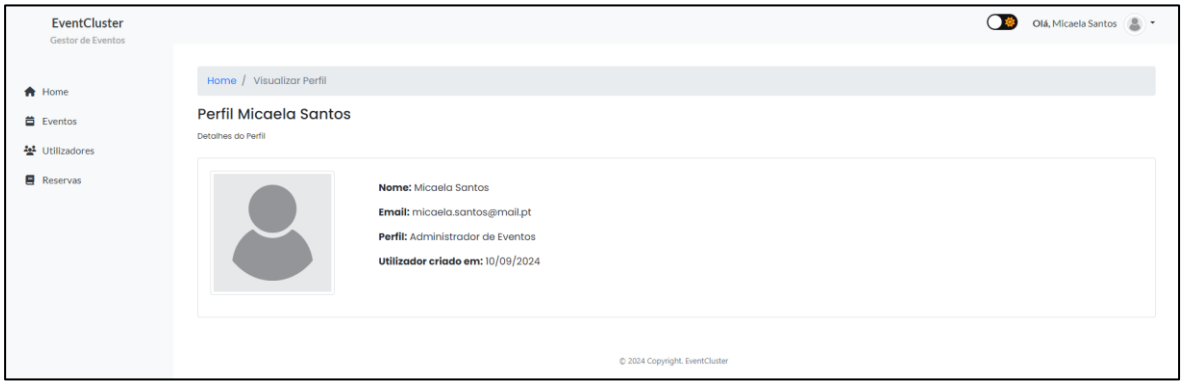

Figura 57 – Visualização dos Dados Pessoais

Caso o utilizador selecione na opção "Definições" Figura 56, será direcionado diretamente para a página dos dados pessoais do utilizador, Figura 58. Nesta página o utilizador irá conseguir editar o nome, o número de telefone, o e-mail, a *password*, o perfil do utilizador e a sua fotografia de perfil.

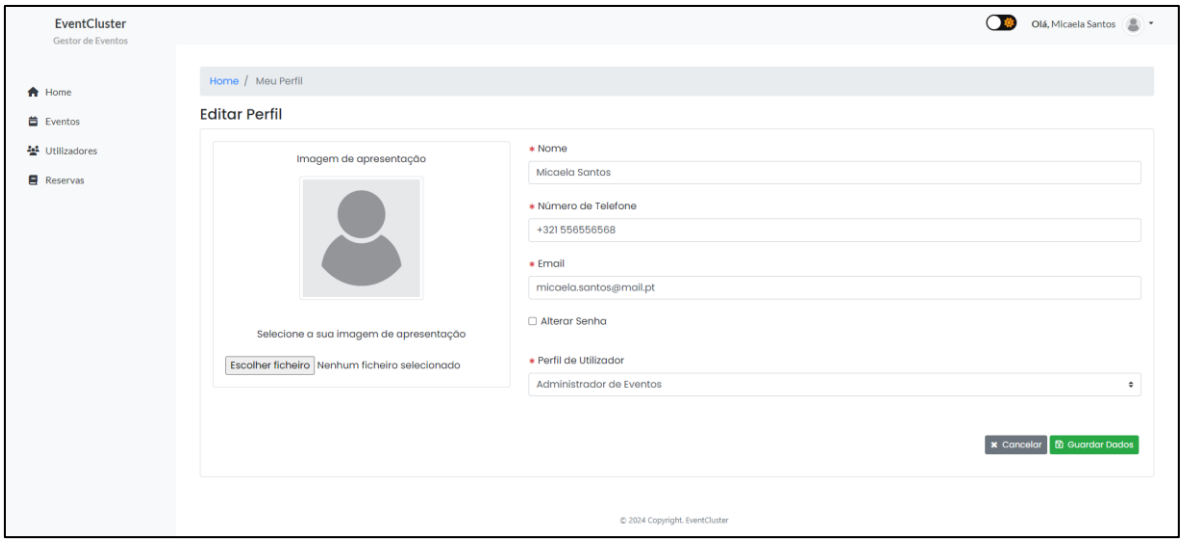

Figura 58 - Definições do utilizador

4. Abordagens de Desenvolvimento

Neste capítulo, serão apresentadas e explicadas as abordagens de desenvolvimento adotadas em partes fundamentais do EventCluster.

4.1. Autenticação

O formulário para os utilizadores de autenticarem, está representado na **Figura 59**. O formulário é enviado através do método POST, para proteger os dados que o utilizador está a inserir no formulário. Na *action* do formulário está definido para o nome da rota que foi criada, e é para onde os dados serão enviados. O *token* CSRF é incluído para proteger contra-ataques (Cross-Site Request Forgery), de forma a proteger o formulário. Dentro do formulário tem as *labels* e os *inputs*, onde tem o email e *password* para colocar. No botão para "iniciar sessão" tem de ser tipo "submit", para enviar os dados do formulário.

```
<div class="card-body">
    @if(session('danger'))
        <div class="alert alert-danger alert-dismissible fade show" role="alert" id="loginAlert">
            {{ session('danger') }}
            <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close">
            </button>
        </div>
    @endif
    <form method="POST" action="{{ route('login') }}">
        @csrf
        <div class="row justify-content-md-center mt-3">
            <label for="email">Email</label>
            <input id="email" type="email"
                   class="form-control @error('email') is-invalid @enderror" name="email"
                   value="{{ old('email') }}" required autocomplete="email" autofocus>
            @error('email')
            <span class="invalid-feedback" role="alert"><strong>{{ $message }}</strong></span>
            @enderror
        </div>
        <div class="row justify-content-md-center mt-3">
            <label for="password">Password</label>
            <input id="password" type="password"
                   class="form-control @error('password') is-invalid @enderror"
                   name="password" required autocomplete="current-password">
            @error('password')
            <span class="invalid-feedback" role="alert"><strong>{{ $message }}</strong></span>
            @enderror
        </div>
        <div class="row mb-0"><div class="text-center mt-4 mb-4">
            <button class="btn btn-primary text-uppercase mt-3" type="submit">Iniciar Sessão
            </button>
            </div><hr>
            <a href="/register" class="font-weight-bold text-black text-decoration-none
            text-center">Ainda não tem Conta?<strong>Registar</strong></a>
        </div>
    </form>
</div>
```

Figura 59 - Página *Login*

O controlador *login* é responsável para autenticar os utilizadores na plataforma. Esta função recebe o e-mail e a *password* do utilizador. De seguida será verificado os dados que foram preenchidos pelo utilizador. Caso os dados que foram preenchidos já estejam na base de dados, o sistema irá perceber que tipo de utilizador pretende efetuar o *login*. Se o utilizador preencher os campos de forma errada, aparecerá um aviso que os campos estão incorretos preenchidos "E-mail ou Palavra-Passe Errados" Figura 60.

```php
<?php

namespace App\Http\Controllers\Auth;

use ...

3 usages
class LoginController extends Controller
{
    use AuthenticatesUsers;

    public function login(Request $request)
    {
        $credentials = $request->only( keys: 'email', 'password');
        if (Auth::attempt($credentials)) {
            $user = Auth::user();
            if ($user->role === 'AdminEvents') {
                return redirect()->route( route: 'dashboard.event.index');
            } elseif ($user->role === 'AdminTemplates') {
                return redirect()->route( route: 'dashboard.templates.index');
            } elseif ($user->role === 'Participant') {
                return redirect()->route( route: 'events');
            } else {
                return redirect()->route( route: 'home');
            }
        }
        return redirect()->back()->with('danger', 'E-mail ou Palavra-Passe Errada. Tente novamente.');
    }
}
```

Figura 60 - Controlador *Login*

Assim que um utilizador fizer *logout*, Figura 61, tem-se outro controlador para remover todos os dados que está na sessão. Assim que efetuar o *logaut*, o utilizador será direcionado para o *login.*

```php
<?php

namespace App\Http\Controllers\Auth;

use ...
2 usages
class LogoutController extends Controller
{
    public function logout(Request $request)
    {
        $request->session()->flush();
        $request->session()->invalidate();
        $request->session()->regenerateToken();

        Auth::logout();

        session()->flash('success', 'Você saiu da sessão com sucesso!');

        return redirect( to: '/login');
    }
}
```

Figura 61 - Controlador Logout

4.2. Utilizador

Para criar e editar os utilizadores no sistema para ser mais rápido, colocou-se os campos essenciais para o mesmo uma *view* e depois consoante o que fosse pretendido, adicionar ou editar acedia-se a esta *view*. A imagem do utilizador está a ser guardada em formato binário, para conseguir guardar diretamente a imagem na Base de Dados e não estar guardada no *storage*, Figura 63.

```
<div class="row">
    <div class="col col-4 mb-3 div-user-photo">
        <div class="card mx-auto">
            <p class="card-text text-center mt-3">Imagem de apresentação</p>
            @if ($user->photo_blob != null)
                <img src="data:" id="picture_preview"
                    class="img-thumbnail shadow mx-auto" alt="Imagem de apresentação"
                    title="Clique para mudar a imagem de apresentação"/>
            @else
                <img src="{{ asset('img/default_user.png') }}" id="preview" class="img-thumbnail shadow mx-auto"
                    alt="Imagem de apresentação" title="Clique para mudar a imagem de apresentação"/>
            @endif

            <div class="text-center">
                <div class="reset_photo mx-auto" id="btn_remove_photo" title="Remover imagem de apresentação">
                    <i class="fas fa-times text-white"></i>
                </div>
            </div>
            <div class="card-body overflow-hidden">
                <p class="text-center">Selecione a sua imagem de apresentação</p>
                <div class="input-group mx-auto mt-4">
                    <input type="file" name="photo" id="inputPhoto" class="file" accept="image/*"
                           aria-describedby="fileHelp">
                </div>
            </div>
        </div>
    </div>
```

Figura 62 - Página Adicionar/Editar Utilizador (Imagem)

Na **Figura 63**, mostra-se alguns campos que fazem parte do formulário para criar/editar utilizador.

```
<label for="name">
    <span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Nome
</label>
<input type="text" class="form-control" name="name" id="name" value="{{ old('name', $user->name) }}" required>
@error('name')<span class="text-danger">{{ $message }}</span>@enderror<br>
<label for="phoneNumber">
    <span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Número de Telefone
</label>
<input type="text" class="form-control" name="phoneNumber" id="phoneNumber"
       value="{{ old('phoneNumber', $user->phoneNumber) }}" required placeholder="+351 912345678">
@error('phoneNumber')<span class="text-danger">{{ $message }}</span>@enderror<br>
<label for="email">
    <span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Email
</label>
<input type="email" class="form-control" name="email" id="email" value="{{ old('email', $user->email) }}" required>
@error('email')<span class="text-danger">{{ $message }}</span>@enderror<br>
@if (!isset($user) || !$user->exists)
    <label for="password"><span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Palavra-Passe</label>
    <input type="password" class="form-control" name="password" id="password" value="{{ old('password') }}">
    @error('password')<span class="text-danger">{{ $message }}</span>@enderror
@else
    <div class="form-check mb-3">
        <input type="checkbox" class="form-check-input" id="changePasswordCheckbox">
        <label class="form-check-label" for="changePasswordCheckbox">Alterar Palavra-Passe</label>
    </div>
    <div id="passwordFields" style="...">
        <label for="password">
            <span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Nova Palavra-Passe
        </label>
        <input type="password" class="form-control" name="password" id="password" value="{{ old('password') }}">
        @error('password')
        <span class="text-danger">{{ $message }}</span>
        @enderror

        <label for="password_confirmation">
            <span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Confirmar Nova Palavra-Passe
        </label>
        <input type="password" class="form-control" name="password_confirmation" id="password_confirmation"
               value="{{ old('password_confirmation') }}">
```

Figura 63 - Página Adicionar/Editar Utilizador

Dependente do tipo de utilizador, caso seja administrador de Evento só pode adicionar Participantes e Voluntários, e o Administrador de *templates*, tem permissão para criar Administrador de Eventos, Figura 64.

```
<div class="form-group">
    <label for="role">
        <span class="campo_obrigatorio text-danger"><small>&#10033;</small></span> Perfil de Utilizador
    </label>
    <select class="custom-select" name="role" id="role" required>
        <option value="" hidden>Escolha o perfil de utilizador</option>
        @if (auth()->user()->role === 'AdminEvents')
            <option value="Participant" {{ old('role', $user->role) == 'Participant' ? 'selected' : '' }}>Participante</option>
            <option value="Voluntary" {{ old('role', $user->role) == 'Voluntary' ? 'selected' : '' }}>Voluntário</option>
        @else
            <option value="AdminEvents" {{ old('role', $user->role) == 'AdminEvents' ? 'selected' : '' }}>Administrador de Eventos</option>
        @endif
    </select>
    @error('role')
    <span class="text-danger">{{ $message }}</span>
    @enderror
</div>
```

Figura 64 - Página Adicionar/Editar Utilizador (Escolha do Tipo)

Na Figura 65**,** está representado a função adicionar, que de uma forma geral, vai adicionar o utilizador à base de dados. O método *fill* já vai preencher com os dados que foram inseridos no formulário, a Palavra-Passe teve-se em atenção para a encriptar, tornando-a mais segura. Relativamente à imagem de perfil do utilizador, guarda a imagem em formato binário, para conseguir guardar diretamente a imagem

na Base de Dados, caso o utilizador não preencha, o sistema vai usar a que esta no *storage*. Como se tem dois gestores optou-se por ter verificações de forma a ambos tenham mensagem personalizadas.

```php
public function store(StoreUserRequest $request)
{
    $fields = $request->validated();
    $user = new User;
    $user->fill($fields);
    if ($request->filled( key: 'password')) {
        $user->password = bcrypt($request->password);
    }
    if ($request->hasFile( key: 'photo')) {
        $photo = $request->file( key: 'photo');
        $profileImg = time() . '.' . $photo->getClientOriginalExtension();
        Storage::disk( name: 'public')->putFileAs( path: 'users_photos/', $photo, $profileImg);
        $user->photo = $profileImg;
        $binaryData = file_get_contents($photo->getRealPath());
        $user->photo_blob = $binaryData;
    } else {
        $user->photo = 'default_user.png';
    }
    $user->save();
    if (Auth::user()->role == 'AdminTemplates')
        return redirect()->route( route: 'gestaotemplates.users.index')->with('success', 'Cliente criado com sucesso');
    else
        return redirect()->route( route: 'gestaoeventos.users.index')->with('success', 'Utilizador criado com sucesso');
}
```

Figura 65 - Controlador Utilizador - Função Adicionar

4.3. *Templates*

Na criação do *template*, o *template* tem dois campos fixos, que é o nome e descrição de *template,* Figura 66**.**

```html
<section>
    <div class="container-fluid">
        <div class="card shadow mb-4">
            <div class="card-body">
                <label for="name">{{__('Nome')}}</label>
                <input type="text" id="name" name="name" class="form-control"/>
                <label for="description">{{__('Descrição')}}</label>
                <input type="text" id="description" name="description" class="form-control"/>
                <div id="fb-editor"></div>

            </div>
        </div>
    </div>
</section>
<script src="https://cdn.ckeditor.com/ckeditor5/36.0.1/classic/ckeditor.js"></script>
```

Figura 66 - Página Criar *Template*

Os campos do *template* vão ter um limite para serem adicionados, para termos um controlo da informação a ser passada. Como a biblioteca *FormBuilder* já fornece campos predefinidos, teve-se de arranjar uma forma de manipular os campos, de forma a colocar os que se pretendia, Figura 67.

```
<script>
    jQuery(function ($) {
        var fieldLimits = {'title': 1, 'resume': 1, 'description': 1, 'schedule': 1, 'speaker': 1,
            'image': 1, 'poster': 1, 'startDate': 1, 'endDate': 1, 'address': 1, 'location': 1, 'link': 1, 'country':1,
        };

        var fieldCounts = {'title': 0, 'resume': 0, 'description': 0, 'schedule': 0, 'speaker': 0, 'image': 0,
            'poster': 0, 'startDate': 0, 'endDate': 0, 'address': 0, 'location': 0, 'link': 0, 'country':0,
        };
        var options = {
            onSave: function (evt, formData) {
                console.log(formData);
                saveForm(formData);
            },
            disableFields: ['file-upload', 'hidden-input', 'autocomplete', 'button', 'textarea', 'checkbox-group', 'date',
                'header', 'paragraph', 'radio-group', 'number', 'select', 'text',
            ],
            fields: [
                { label: 'Resumo', name: 'resume',type: 'text',  icon: '<i class="fa fa-align-justify" aria-hidden="true"></i>', required: false, value: ''},
                { label: 'Descrição',name: 'description',type: 'textarea', icon: '<i class="fa fa-align-left" aria-hidden="true"></i>',required: false, value: '' },
                { label: 'Programação', name: 'schedule', type: 'richTextEditor', icon: '<i class="fa fa-pencil" aria-hidden="true"></i>',required: false,value: ''},
                { label: 'Título', name: 'form-title',type: 'text', icon: '<i class="fa fa-font" aria-hidden="true"></i>', required: true,value: ''},
                { label: 'Orador',name: 'speaker',  type: 'text', icon: '<i class="fa fa-user" aria-hidden="true"></i>',required: false, value: '' },
                { label: 'Imagem',name: 'image', type: 'file', icon: '<i class="fa fa-image" aria-hidden="true"></i>',required: false, value: '' },
                { label: 'Cartaz', name: 'poster', type: 'file',icon: '<i class="fa fa-file-image" aria-hidden="true"></i>',required: false,value: ''  },
                { label: 'Data de Início',name: 'startDate', type: 'date',icon: '<i class="fa fa-calendar" aria-hidden="true"></i>', required: false, value: '' },
                { label: 'Data de Fim', name: 'endDate', type: 'date', icon: '<i class="fa fa-calendar" aria-hidden="true"></i>', required: false, value: ''},
                { label: 'Endereço', name: 'address', type: 'text', icon: '<i class="fa fa-map-marker" aria-hidden="true"></i>', required: true,value: '' },
                { label: 'Localização',name: 'form-location',type: 'text', icon: '<i class="fa fa-location-arrow" aria-hidden="true"></i>', required: false, value: ''},
                { label: 'Link',name: 'link', type: 'text', icon: '<i class="fa fa-link" aria-hidden="true"></i>',required: false, value: ''},
                { label: 'País',name: 'country', type: 'select', icon: '<i class="fa fa-flag" aria-hidden="true"></i>', required: false,
                    values: [{label: 'Portugal', value: 'pt'}, {label: 'Brasil', value: 'br'}, {label: 'Argentina', value: 'ar'}, {label: 'Alemanha', value: 'de'},]
                }
            ],
```

Figura 67 - Página Criar *Template* (jquery)

A **função criar**, é criar o *template* na Base de Dados, é passado o *$request* por parâmetro, o objeto *FormBuilder* é instanciado e o objeto *$item* tem o "name", "description" e o contente associado e são esses dados que vão ser guardados. O "name" é o nome do *template*, a *description* é a descrição do mesmo, e o *content*, é o conteúdo onde vai ser guardados todos os campos que forem preenchidos do *templates*.

```php
public function create(Request $request)
{
    $item = new FormBuilder();
    $item->name = $request->name;
    $item->description = $request->description;
    $item->content = $request->form;
    $item->save();
    return response()->json( data: 'Template adicionado com Sucesso!');
}
```

Figura 68 - Controlador *Template* - Função Criar

A **função atualizar,** é o que permite alterar os campos ou alterar o conteúdo do *template*.

```php
public function update(Request $request)
{
    $item = FormBuilder::findOrFail($request->id);
    $item->name = $request->name;
    $item->description = $request->description;
    $item->content = $request->form;
    $item->update();
    return response()->json( data: 'Template atualizado com Sucesso!');
}
```

Figura 69 - Controlador *Template* - Função Atualizar

Esta **função atribuir tipo de *template*,** permite que ao *template* criado seja atribuído um tipo de *template*.

```php
public function assignEventType($id)
{
    $form = FormBuilder::findOrFail($id);
    $eventTypes = EventType::all();
    $assignedEventTypeId = $form->eventType ? $form->eventType->id : null;
    return view( view: 'FormBuilder.assign-event-type', compact( var_name: 'form',  ...var_names: 'eventTypes', 'assignedEventTypeId'));
}
```

Figura 70 - Controlador *Template* - Função Atribuir Tipo *Template*

Para visualizar o *template* a partir do *formbuider*, no jquery faz-se a verificação e com o método *formRender*, apresenta o que está na *data*.content, que por sua vez é apresentado na *div* onde tem o id=”fb-*reader*” Figura 71.

```
<section>
    <div class="container-fluid ">
        <div class="card shadow mb-4">
            <div class="card-body">
                <form method="POST" action="{{ URL('save-form-transaction') }}" enctype="multipart/form-data">
                    @csrf
                    <input type="number" id="form_id" name="form_id" hidden/>
                    <div id="fb-reader"></div>
                    <input type="submit" value="Save" class="btn btn-success"/>
                </form>
            </div>
        </div>
    </div>
</section>
<script>
    $(function () {
        $.ajax({
            type: 'get',
            headers: {
                'Authorization': 'Bearer ' + localStorage.getItem('token')
            },
            url: '{{ URL('get-form-builder') }}',
            data: {
                'id': {{ $id }}
            },
            success: function (data) {
                $("#form_id").val(data.id);
                $('#fb-reader').formRender({
                    formData: data.content
                });
            }
        });
    });
</script>
```

Figura 71 - Visualizar *Template*

4.4. Diagrama de componentes

Nesta secção é apresentada a estrutura de relacionamento entre os vários artefactos do sistema, através de um diagrama de componentes pois contribui-o para a organização e uma comunicação para o desenvolvimento do mesmo. Foi discutido quais foram as principais características do diagrama.

Os diagramas de componentes UML (Unified Modeling Language) oferecem uma visão conceitual das interações entre os vários sistemas. Assim pode incluir aspetos de modelagem lógica e física. Os diagramas contêm uma estrutura complexa, interligam-se com outros componentes apenas por ligações específicas. Cada componente pode conectar operações e serviços de outros componentes, também são expostas às dependências [5].

Será apresentado as notações utilizadas no diagrama, explicando assim os símbolos e os seus devidos significados de cada um deles.

Na Figura 72, é representada por um componente que tem a forma de um retângulo com dois retângulos menores na lateral. O símbolo que está representado na figura, é um componente que vai interagir com outros [3].

Figura 72 – Componente

Na Figura 73**,** é demonstrado como é representado o *package*, por uma pasta de arquivo. Esta pasta, compila diversos elementos do sistema. Assim pode juntar os elementos de uma forma mais organizada [6].

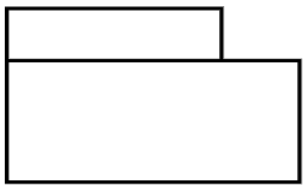

Figura 73 - *Package*

A Figura 74**,** ilustra a dependência entre os elementos do sistema, representada por linhas tracejadas. Essas linhas indicam que uma parte do sistema depende de outra para funcionar corretamente [6].

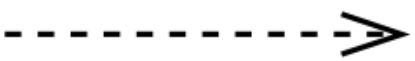

Figura 74 - Dependência

Na Figura 75, é representado o Diagrama de Componentes do EventCluster. O diagrama está divido em quatro seções onde se pode observar as diferentes etapas que fazem parte do sistema, como as *Views*, Controllers, Models e a Base de Dados.

Nas ***Views*** do EventCluster, há diversas seções. A diretoria **Auth** inclui as *views* "*login*.blade.php" e "register.blade.php". No BackOffice, tem a diretoria **Layouts** inclui as *views* "main_manage_*Templates*.blade.php" e "main_manage_events.blade.php". A diretoria **partials** inclui as *views* "errors.blade.php" e "success.blade.php"

Em **Manage Events** tem a *view* "index.blade.php", e as *views* estão organizadas em categorias como "Events", "Drafts" e "Reservations", cada uma com seus próprios ficheiros de blade.

O ***FormBuilder*** inclui *views* como "index.blade.php", "create.blade.php", "edit.blade.php", e "read.blade.php", enquanto a diretoria **Mail** inclui a *view* "contact.blade.php".

Em **Manage_*Templates*** tem a *view* "index.blade.php" e uma diretoria ***Template*Types** com as respetivas *views*. Na diretoria **Manage_Participants** e **Users**, cada uma tem as suas respectivas *views*.

Nos **Controllers** do EventCluster, contém uma diretoria Auth que contém os controladores "*Login*Controller.php" e "RegisterController.php". Os restantes controladores que fazem parte do sistema para que ele funcione são os seguintes:

- "DraftController.php";
- "ContactController.php";
- "HomeIndexController.php";
- "UserController.php";
- "ParticipantController.php";
- "EventController.php";
- "EventDetailController.php";
- "EventTypeController.php";
- "GEventsController.php";
- "GEventsIndexController.php";
- "DraftController.php";
- "CommentController.php";
- "ReservationController.php";
- "*FormBuilder*Controller.php";
- "G*Templates*IndexController.php";
- "FavoriteController.php";
- "GEvetsIndexController.php";
- "EventController.php";
- "ReservationController.php";
- "ParticipantController.php";
- "ContactController.php";

- "*FormBuilder*Controller.php";
- "CommentController.php".

Os Models do EventCluster incluem View.php, Ticket.php, Event.php, Reservation.php, *FormBuilder*.php, *Template*Type.php, User.php, Comment.php, Favorite.php e Task.php. Esses modelos representam as principais entidades do sistema e estão diretamente ligados às tabelas da base de dados.

A Base de Dados do EventCluster contém várias tabelas, como *views*, tickets, reservations, events, form_builders, *Template*Types, users, comments, comment_interactions, favorites e tasks. Cada uma dessas tabelas armazena dados respetivos para funcionalidades que o sistema oferece.

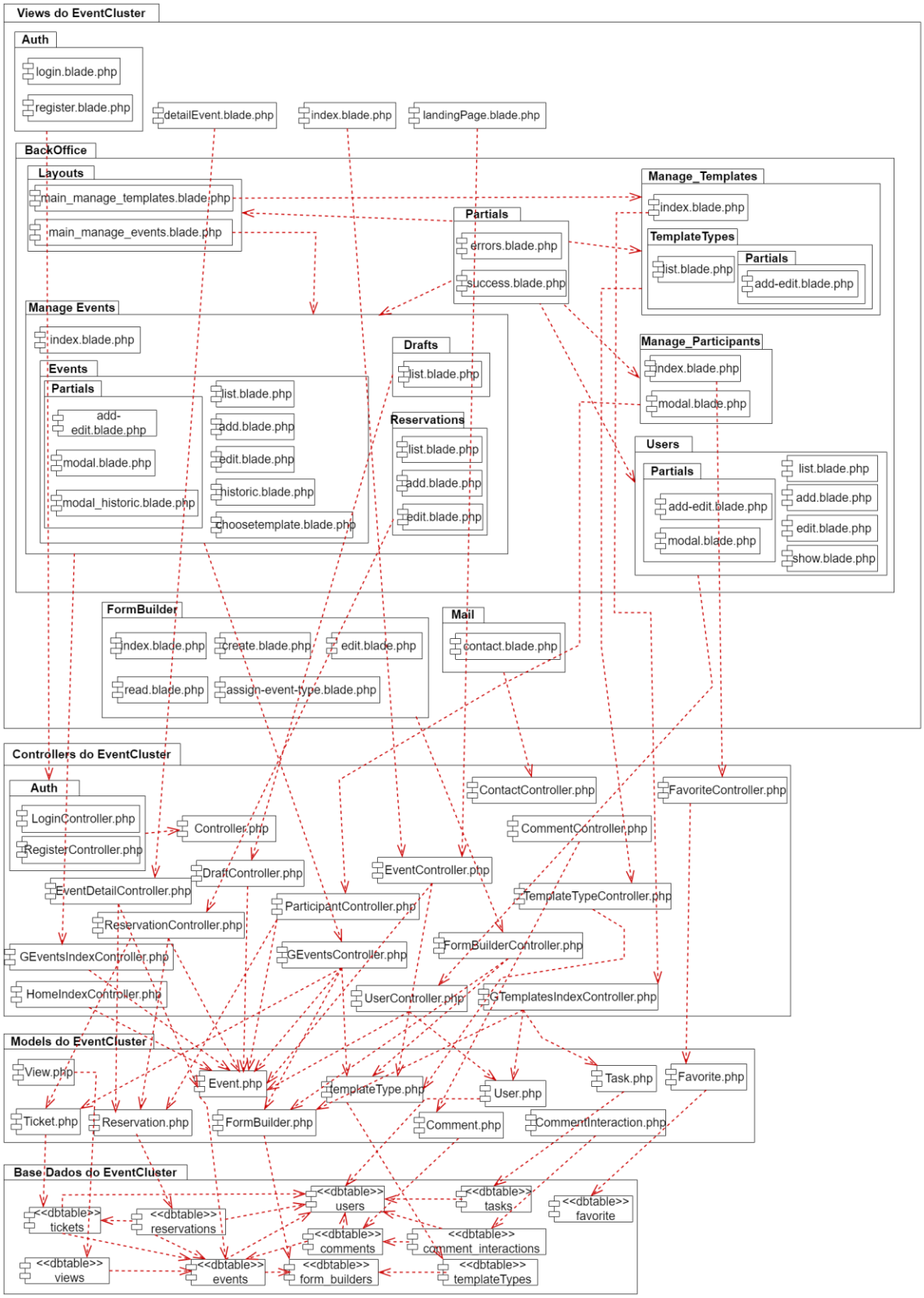

Figura 75 - Diagrama de Componentes

5. Tecnologias e Linguagens Utilizadas

Durante toda a execução deste projeto, muitas foram as vezes, em que se teve que recorrer a ferramentas adicionais para prestar algum tipo de auxílio no desenvolvimento do trabalho. Na lista que se encontra abaixo deste parágrafo pode-se verificar todas as ferramentas utilizadas:

- HTML
- CSS
- PHP
- Laravel
- Bootstrap 4
- MySql
- JavaScript
- Ajax
- JQuery
- JSON
- GIT
- Icon Picker
- Rich Text Editor
- DataTables
- *FormBuilder*
- EmailJS
- Draw.io
- UML
- Laragon
- Visual Studio Code

HTML - HTML (HyperText Markup Language) sendo uma linguagem de marcação, é maioritariamente utilizada para a construção de páginas Web. Como será óbvio, foi uma das tecnologias mais utilizadas neste projeto, visto que toda a aplicação corre na *web*.

CSS - Sendo que o HTML se ocupa com a construção de uma página *web*, o CSS ocupa-se com o estilo da mesma, ou seja, todo o design da aplicação construída é possível através desta ferramenta.

PHP - Todo o conteúdo dinâmico presente no nosso projeto deve-se a esta ferramenta, PHP. Têm como objetivo pegar em dados do lado do servidor e retornar para o lado do utilizador, podendo gerar conteúdo dinâmico sem que este seja mudado nos ficheiros de HTML e CSS.[4].

Bootstrap - O bootstrap, na sua versão 4, é uma ferramenta que ajuda a construir *website*s de uma forma mais simples, trabalhando com um sistema de grelhas que visa a ajudar no desenho responsivo do mesmo [5].

MySql - O MySQL é open *source* e é um dos sistemas de gestão de bases de dados para fim profissional mais utilizado e conhecido a nível mundial. O MySQL usa a linguagem SQL a mais conhecida para inserir conteúdo na base de dados [9]. Todos os dados estão a ser armazenados no MySql.

JavaScript - O JavaScript mais conhecida por JS, é uma linguagem de programação que possui a possibilidade de realizar elementos mais complexos numa página *web.* Com esta linguagem é possível realizar a informação de forma mais dinâmica e as animações do *website* [7] .

JQuery - jQuery, sendo uma biblioteca de funções JavaScript, presta auxílio na simplificação dos scripts outrora construídos por JavaScript. [11].

JSON – O JSON é um formato de dados com base em texto (*string*) seguido a sintaxe do objeto JavaScript. O seu usso pode ser independente. Em muitas projetos de programação têm a capacidade de ler e gerar JSON [9]. Foi utilizado para armazenar os dados específicos dos campos do *templates* e do evento.

Ajax – O AJAX foi uma das mais importantes ferramentas no desenvolvimento deste projeto, pois permitiu que a atualização de dados numa página sem que esta sofresse uma atualização "visual", de modo a ser mais interativo com o utilizador e experienciar uma navegação bastante melhor.

GIT – Com alguma semelhança ao AJAX, em termos de importância, o GIT foi claramente uma das tecnologias que nos ajudou em termos de organização. Com o seu controlo de versões e todas as suas possibilidades o trabalho e a partilha de ficheiros entre grupo tornou-se bastante mais fácil [13].

Icon Picker - Icon Picker, é um plugin apesar da sua redundância, facilitou de igual modo a escolha de ícones do Font Awesome, para os tipos de *template* [10].

Rich Text Editor – Oferece uma maneira flexível e acessível de criar documentos. Pois é um formato que permite obter vários estilos de formatação e recursos ao texto, com fontes, cor, tamanhos, sublinhado, itálico, negrito, etc [11]. Foi utilizada especificamente para um campo do formulário do *template*.

DataTables – A semelhança de todas as tecnologias anteriormente referidas, a Datatables tem um impacto bastante grande neste projeto, sendo que ajudou bastante na construção de tabelas e tudo o que as mesmas envolvem, como por exemplo a pesquisa e valores inseridos nas tabelas [12].

***FormBuilder* -** O *FormBuilder* permite criar formulários através de uma interface intuitiva, onde o utilizador arrasta e solta campos disponíveis [17]. Como foi referido esta tecnologia foi usada para criar formulários, neste caso o administrador de *templates* cria os eventos através desses formulários.

EmailJS – O EmailJS é uma ferramenta que permite enviar e-mails diretamente de um *website* [18]. Usa apenas a tecnologia do lado do cliente (como HTML, CSS e JavaScript), com estas não é necessário um servidor para enviar e-mails. O EmailJS usou-se por exemplo para o administrador de eventos enviar mensagens para os voluntários.

Visual studio Code - O Visual studio Code, mais conhecido por VS Code, é um editor de código-fonte gratuito e de código aberto desenvolvido pela Microsoft. Ele é usado principalmente por programadores e desenvolvedores de software para escrever e editar código. O VS Code é conhecido por ser leve e rápido, o que torna um dos editores mais popular para o desenvolvimento em várias plataformas.

Draw.io - Devido à metodologia que foi escolhida, foi necessário criar vários diagramas, optou-se por usar o draw.io, como é uma *website* gratuita e tem todos os elementos UML ao dispor de qualquer utilizador. Esta plataforma foi necessária para elaborar modelos e diagramas ao logo neste projeto [19].

UML - A linguagem UML permite modelar a estrutura e o comportamento dinâmico de um sistema informático sob várias perspetivas, foi utilizada para a realização da modelação do projeto, usado especificamente os seguintes diagramas: Diagramas de Componentes, Diagrama de Classes, Modelo E/R e Modelo de Domínio.,

Laragon - O laragon é um ambiente de desenvolvimento de desenvolvimento local. Esta ferramenta já vem incluído alguns recursos como por exemplo Git, PHP, Extensões PHP, phpMyAdmin, Apache entre outros. Também tem a facilidade de configurar servidor de e-mail localmente [20].

Laravel - É uma framework de PHP para o desenvolvimento de sistemas web onde se utiliza o padrão MVC. Todo o conteúdo dinâmico presente no projeto deve-se a esta ferramenta.

- Model: Responsável pela gestão dos dados. O Model interage com a base de dados, garante que as informações estão corretas e fornece ao Controller quando solicitado.

- *View*: São as páginas onde os dados são apresentados aos utilizadores. No Laravel, as *Views* utilizam *Blade*, uma ferramenta que permite criar estruturas condicionais e funções para controlar a informação que é apresentada ao utilizador.

- Controller: Organiza a lógica do *website*. Ele recebe dados do *Model* e envia para as *Views*. Utilizar Controllers torna o código mais organizado e evita sobrecarregar as rotas com lógica desnecessária.

- Rotas: Definem o caminho dos pedidos dos utilizadores para os Controladores. No Laravel, as rotas são configuradas no ficheiro web.php, quando se está a criar a rota tem de ter o caminho específico, o método e o nome do *Controller* e opcional colocar o nome da rota.

6. Conclusão e Trabalho Futuro

Este relatório descreve o desenvolvimento da plataforma EventCluster, destinada à criação de sites de gestão de eventos baseada em *templates* personalizados.

Na realização do Projeto II, o foco foi a implementação das funcionalidades que foram descritas anteriormente. No entanto, houve a necessidade de alterar algumas partes de Projeto I, com as alterações no Modelo de Domínio, Modelo de Classes, o Modelo E/R e o Modelo de Dados.

Foram desenvolvidas funcionalidades em *back-office* como a Gestão de *templates* e a Gestão de Eventos, onde foi atribuído administradores a cada área de gestão. Para cada utilizador é descrito detalhadamente as suas respetivas funcionalidades, tanto no *front-office* como no *back-office*. As funcionalidades de *front-office,* permite que os participantes interagem com a plataforma, como, efetuar reservas, aceder às informações dos eventos e ainda a possibilidade de escrever comentários.

As tecnologias que foram utilizadas para desenvolver este projeto foram HTML, CSS, Bootstrap, PHP, MySQL JavaScript, JQuery, Visual Studio Code, Draw.oi, Laragon GitHub, Laravel e entre outras.

A *framework* Laravel com a sua arquitetura MCV(Model-View-Controller) foram essenciais para a organização do código e facilitar a manutenção futura da plataforma.

Como trabalho futuro, apresentam-se algumas funcionalidades que poderiam melhorar a usabilidade da EventCluster:

- Tradução para um outro idioma (Inglês)
- Compra *online* dos bilhetes
- Criar o Administrador Geral

Referências

[1] "VIRAL – Agenda Cultural – Eventos em destaque." Accessed: Sep. 13, 2024. [Online]. Available: https://www.viralagenda.com/pt/home

[2] "Eventbrite - Descubra eventos fantásticos ou crie os seus próprios eventos e venda bilhetes." Accessed: Sep. 13, 2024. [Online]. Available: https://www.eventbrite.pt/

[3] E. S. dos S. Valente, "Modelação de Sistemas," 2020.

[4] "JSON." Accessed: Sep. 11, 2024. [Online]. Available: https://www.json.org/json-en.html

[5] "O que é um diagrama de componentes UML: símbolos e tutorial." Accessed: Sep. 12, 2024. [Online]. Available: https://www.mindonmap.com/pt/blog/uml-component-diagram/

[6] E. S. dos S. Valente, "Diagramas de Pacotes Diagramas de Componentes Diagramas de Distribuição," 2020.

[7] "O que é PHP (Hypertext Preprocessor), e para que é utilizado?" Accessed: Sep. 11, 2024. [Online]. Available: https://www.one.com/pt/alojamento/o-que-e-php

[8] "Introduction · Bootstrap v4.6." Accessed: Sep. 11, 2024. [Online]. Available: https://getbootstrap.com/docs/4.6/getting-started/introduction/

[9] "O que é e como usar o MySQL?" Accessed: Sep. 11, 2024. [Online]. Available: https://www.techtudo.com.br/noticias/2012/04/o-que-e-e-como-usar-o-mysql.ghtml

[10] "O que é JavaScript? - Aprendendo desenvolvimento web | MDN." Accessed: Sep. 11, 2024. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/First_steps/What_is_JavaScript

[11] "jQuery." Accessed: Sep. 11, 2024. [Online]. Available: https://jquery.com/

[12] "Trabalhando com JSON - Aprendendo desenvolvimento web | MDN." Accessed: Sep. 12, 2024. [Online]. Available: https://developer.mozilla.org/pt-BR/docs/Learn/JavaScript/Objects/JSON

[13] "GitHub." Accessed: Sep. 14, 2024. [Online]. Available: https://github.com/

[14] "Icon Picker | Use the Font Awesome Icons (Font Awesome Free v5.11.2) in your HTML forms. (1544 icons)." Accessed: Sep. 11, 2024. [Online]. Available: https://furcan.github.io/IconPicker/

[15] "WYSIWYG HTML Editor | Javascript Rich Text Editor | RichTextEditor." Accessed: Sep. 11, 2024. [Online]. Available: https://richtexteditor.com/

[16] "DataTables | Javascript table library." Accessed: Sep. 11, 2024. [Online]. Available: https://datatables.net/

[17] “jQuery formBuilder | Drag & Drop Form Creation.” Accessed: Sep. 13, 2024. [Online]. Available: https://formbuilder.online/

[18] “How does EmailJS work | EmailJS.” Accessed: Sep. 14, 2024. [Online]. Available: https://www.emailjs.com/docs/introduction/how-does-emailjs-work/

[19] “Diagrama sem nome - draw.io.” Accessed: Sep. 14, 2024. [Online]. Available: https://app.diagrams.net/

[20] “Why Laragon? | Laragon - portable, isolated, fast & powerful universal development environment for PHP, Node.js, Python, Java, Go, Ruby.” Accessed: Sep. 11, 2024. [Online]. Available: https://laragon.org/why-laragon/