

Instituto Politécnico de Viseu  
Escola Superior de Tecnologia e Gestão de Viseu  
Departamento de Informática

Unidade Curricular: Projeto Integrado

## Relatório Relativo ao Trabalho Prático

Tema: SoftShares

Realizado por:

[Redacted]

[Redacted]

[Redacted]

Relatório relativo ao Trabalho Prático

Curso de Licenciatura em Engenharia Informática

Unidade Curricular de Projeto Integrado

## Projeto Integrado – The SoftShares

Ano Letivo 2023/24

Viseu, 2024

---

# 1. Índice

1.	Índice.....	3
2.	Introdução.....	10
2.1.	Contexto do Projeto.....	10
2.2.	Objetivos .....	10
3.	Descrição Geral do Projeto.....	11
3.1.	Visão Geral da Solução .....	11
3.2.	Funcionalidades Principais.....	11
3.2.1.	Aplicação para Dispositivos Móveis .....	11
3.2.2.	Aplicação Web com <i>Backoffice</i> .....	11
3.2.3.	API.....	12
3.3.	Arquitetura do Sistema.....	13
4.	Desenvolvimento do Projeto .....	14
4.1.	Planeamento e Metodologia .....	14
4.2.	Tecnologias Utilizadas .....	14
5.	Desenvolvimento da Aplicação para Dispositivos Móveis .....	15
5.1.	Requisitos Funcionais e Não Funcionais.....	15
5.1.1.	Requisitos Funcionais.....	15
5.1.2.	Requisitos Não Funcionais .....	15
5.2.	Arquitetura Organizacional .....	15
5.3.	Desafios Principais.....	16
5.4.	Desenvolvimento em Dart/Flutter .....	16
5.4.1.	Interface do Usuário .....	17
5.4.2.	Prototipagem.....	17
5.4.3.	Principais Componentes de UI .....	18
5.4.3.1	Página <i>Sign-in</i> :.....	18
5.4.3.2	Página Sign-up.....	19
5.4.3.3	Página Homepage .....	20
5.4.3.4	Página Categoria:.....	21
5.4.3.5	Página Evento:.....	21
5.4.3.6	Página Fórum:.....	22
5.4.3.7	Página Post: .....	23
5.4.3.8	Página Criação Contéudo:.....	24
5.4.3.9	Página <i>Profile</i> : .....	26
5.4.3.10	Página Calendar .....	26
5.4.4.	Aplicação em Light Mode .....	27
6.	Desenvolvimento da Aplicação Web com <i>backend</i> .....	28
6.1.	Arquitetura Organizacional .....	28
6.2.	Estrutura logica .....	28
6.3.	Desafios Principais.....	26
6.4.	Desenvolvimento em Node.js e Express .....	26
6.4.1.	Estrutura da Aplicação.....	26

---

6.5.	React UI.....	26
6.5.1.	Páginas.....	27
6.5.1.1	Login.....	27
6.5.1.2	Registo .....	27
6.5.1.3	Seleção de Centro.....	25
6.5.1.4	Email Enviado .....	25
6.5.1.5	Definição de Palavra-Passe.....	26
6.5.1.6	Página Inicial .....	26
6.5.1.7	Posts/Eventos .....	27
6.5.1.8	Adicionar Post .....	28
6.5.1.9	Criação de Evento .....	29
6.5.1.10	Criação de Centro.....	30
6.5.1.11	Dashboard .....	30
6.5.1.12	Perfil.....	31
6.5.1.13	Detalhe do Evento .....	31
6.5.1.14	Detalhe do Post .....	32
6.5.1.15	Centros .....	32
6.5.1.16	Areas .....	33
6.5.1.17	SubAreas .....	33
6.5.1.18	Forums .....	34
6.5.1.19	Avisos .....	34
6.5.1.20	Formulários Lista .....	34
6.5.1.21	Formulários Criar .....	35
6.5.1.22	Editar Formulários .....	35
6.5.1.23	Administradores .....	36
6.5.1.24	Albums .....	37
6.5.2.	Componentes Reutilizaveis .....	37
6.5.3.	Componentes Criados.....	37
6.5.3.1	ButtonWithIcon.js.....	37
6.5.3.2	Card.js .....	37
6.5.3.3	ParentComponent.js .....	38
6.5.3.4	BarChart.js .....	38
6.5.3.5	CategoryCard.js .....	38
6.5.3.6	PieChart.js.....	38
6.5.3.7	ShowEventCalendar.js .....	38
6.5.3.8	Navbar.js .....	38
6.5.3.9	Calendar.js .....	39
6.5.3.10	PostCard.js .....	39
6.5.4.	Reutilização de Componentes.....	39
7.	Desenvolvimento da API.....	40
7.1.	Controladores (controllers).....	40
7.1.1.	authController.....	41
7.1.1.1	register .....	41

---

---

7.1.1.2	setupPassword.....	41
7.1.1.3	updatePassword .....	41
7.1.1.4	startRecoveryPassword .....	42
7.1.1.5	resetPassword .....	42
7.1.1.6	login_web e login_mobile.....	42
7.1.1.7	login_SSO.....	42
7.1.1.8	getUserByToken .....	43
7.1.1.9	refreshToken .....	43
7.1.1.10	updateFcmToken.....	43
7.1.2.	adminController.....	43
7.1.2.1	validate_content.....	43
7.1.2.2	reject_content.....	44
7.1.2.3	getUserEngagementMetrics .....	44
7.1.2.4	getContentValidationStatusByadmin.....	44
7.1.2.5	getContentValidationStatus .....	44
7.1.2.6	getActiveDiscussions .....	44
7.1.2.7	getActiveWarnings e getAllWarnings .....	45
7.1.2.8	getContentCenterToBeValidated.....	45
7.1.2.9	createCenter .....	45
7.1.2.10	deleteCenter .....	45
7.1.2.11	getCenters .....	45
7.1.2.12	makeCenterAdmin .....	45
7.1.2.13	updateCenter .....	45
7.1.2.14	validate_user e deactivate_user.....	46
7.1.2.15	register_admin.....	46
7.1.2.16	getReports e deleteReport .....	46
7.1.2.17	createWarnings e updateWarning .....	46
7.1.3.	dashboardController .....	47
7.1.3.1	toValidate.....	47
7.1.3.2	validated.....	47
7.1.3.3	postsbycity .....	47
7.1.3.4	eventsbycity .....	47
7.1.3.5	comments_by_city .....	48
7.1.4.	Static_contentController .....	48
7.1.4.1	create_category .....	48
7.1.4.2	create_sub_category.....	49
7.1.4.3	get_all_areas .....	49
7.1.4.4	get_all_sub_areas.....	49
7.1.4.5	update_category .....	49
7.1.4.6	delete_category .....	50
7.1.4.7	update_sub_category .....	50
7.1.4.8	delete_sub_category.....	50
7.1.4.9	getAllCenters .....	50

---

---

7.1.5.	dynamic_contentController .....	51
7.1.5.1	getAllContent.....	51
7.1.5.2	getAllContentByCity .....	51
7.1.5.3	getPostsByCity, getForumsByCity, getEventsByCity .....	51
7.1.5.4	getPostById, getForumById, getEventById .....	52
7.1.5.5	getUserInfo .....	52
7.1.5.6	getUsers .....	52
7.1.5.7	updateUserOffice .....	52
7.1.5.8	getEventByDate .....	53
7.1.5.9	getPosts, getForums, getEvents .....	53
7.1.6.	postController .....	53
7.1.6.1	create_post .....	53
7.1.6.2	edit_post.....	54
7.1.6.3	get_post_state .....	54
7.1.6.4	delete_post .....	54
7.1.6.5	getPostScoreByID.....	55
7.1.7.	forumController .....	55
7.1.7.1	create_forum .....	55
7.1.7.2	get_forum_state .....	56
7.1.7.3	edit_forum.....	56
7.1.7.4	change_forum_state .....	56
7.1.7.5	delete_forum .....	57
7.1.8.	eventController .....	57
7.1.8.1	create_event .....	57
7.1.8.2	register_user_for_event.....	57
7.1.8.3	unregister_user_from_event .....	58
7.1.8.4	get_event_state.....	58
7.1.8.5	edit_event.....	58
7.1.8.6	get_participants.....	58
7.1.8.7	get_participants_adm .....	59
7.1.8.8	getEventScoreByID .....	59
7.1.9.	formsController .....	59
7.1.9.1	create_event_form .....	59
7.1.9.2	add_fields_event_form .....	60
7.1.9.3	edit_fields_event_form .....	60
7.1.9.4	get_event_form .....	60
7.1.9.5	get_event_json_form .....	60
7.1.9.6	add_answers.....	61
7.1.9.7	delete_field_from_form .....	61
7.1.9.8	get_event_answers .....	61
7.1.9.9	get_event_answers_for_user.....	61
7.1.9.10	get_event_answers_for_users .....	62
7.1.10.	commentsController .....	62

---

---

7.1.10.1	add_comment.....	62
7.1.10.2	get_comments_tree .....	63
7.1.10.3	like_comment.....	63
7.1.10.4	unlike_comment.....	63
7.1.10.5	report_comment .....	63
7.1.10.6	likes_per_content .....	64
7.1.10.7	delete_comment .....	64
7.1.10.8	likes_per_user .....	64
7.1.11.	ratingController .....	64
7.1.11.1	add_evaluation .....	65
7.1.12.	mediaController.....	65
7.1.12.1	create_album.....	65
7.1.12.2	add_photograph_area_album .....	66
7.1.12.3	add_photograph.....	66
7.1.12.4	add_photograph_event .....	66
7.1.12.5	get_albums .....	67
7.1.12.6	get_album_photo.....	67
7.1.12.7	get_event_photos .....	67
7.1.12.8	get_area_photos .....	67
7.1.12.9	get_albums_of_areas.....	67
7.1.12.10	get_photos_of_areas_albums .....	68
7.1.13.	uploadController.....	68
7.1.14.	userController .....	68
7.1.14.1	get_user_preferences.....	68
7.1.14.2	create_user_preferences .....	68
7.1.14.3	update_user_preferences .....	69
7.1.14.4	get_user_role.....	69
7.1.14.5	get_user_by_role.....	69
7.1.14.6	add_bookmark.....	69
7.1.14.7	remove_bookmark .....	70
7.1.14.8	get_user_bookmarks .....	70
7.1.14.9	update_acc_status.....	70
7.1.14.10	get_users_to_validate .....	70
7.1.14.11	update_profile .....	70
7.1.14.12	get_user_content .....	71
7.1.14.13	get_user_registeredEvents.....	71
7.1.15.	jwt_middlewareController .....	71
7.1.15.1	Passos .....	71
7.1.16.	emailController.....	72
7.2.	Rotas (routes) .....	74
7.2.1.	authRoutes.js .....	74
7.2.2.	adminRoutes.js .....	74
7.2.3.	dashboardRoutes.js.....	74

---



---

7.2.4.	static_ContentRoute.js .....	74
7.2.5.	dynamic_contentRoute.js.....	75
7.2.6.	postRoutes.js.....	75
7.2.7.	forumRoutes.js.....	75
7.2.8.	eventRoutes.js.....	75
7.2.9.	formsRoutes.js .....	75
7.2.10.	commentsRoutes.js .....	76
7.2.11.	mediaRoutes.js.....	76
7.2.12.	ratingRoutes.js .....	76
7.2.13.	uploadRoutes.js .....	76
7.2.14.	userRoutes.js.....	76
7.3.	Arquivos de Inicialização (start) .....	77
7.3.1.	syncModels.js .....	77
7.3.2.	insertAdmins.js .....	77
7.3.3.	deleteDB.js e insertFakeData.js .....	78
7.4.	Utilitários.....	78
7.4.1.	Validadores de Entradas .....	78
7.4.2.	Notificações em Tempo Real.....	78
7.4.3.	Registo de Erros .....	79
7.5.	Endpoints.....	79
7.5.1.	/api/categories.....	79
7.5.2.	/api/forum .....	80
7.5.3.	/api/post .....	80
7.5.4.	/api/event .....	80
7.5.5.	/api/media .....	80
7.5.6.	/api/rating.....	80
7.5.7.	/api/administration .....	81
7.5.8.	/api/form .....	81
7.5.9.	/api/comment .....	81
7.5.10.	/api/user .....	81
7.5.11.	/api/dynamic .....	81
7.5.12.	/api/notification.....	82
7.5.13.	/api/auth .....	82
7.5.14.	/api/dashboard.....	82
7.5.15.	/api/upload .....	82
7.5.16.	/api/uploads.....	82
7.6.	Conclusão backend.....	83
8.	Integração com Base de Dados.....	84
8.1.	Estrutura da Base de Dados .....	84
8.2.	Modelagem de Dados .....	84
8.2.1.	ActiveDiscussions .....	84
8.2.2.	ContentValidationStatus .....	84
8.2.3.	Offices .....	84

---

---

8.2.4.	OfficeWorkers .....	67
8.2.5.	Comments.....	67
8.2.6.	Likes .....	67
8.2.7.	Participation.....	67
8.2.8.	Reports.....	68
8.2.9.	Warnings.....	68
8.2.10.	Events .....	68
8.2.11.	Forums.....	68
8.2.12.	Posts.....	68
8.2.13.	Albuns e Photographs.....	69
8.2.14.	Scores e Ratings .....	69
8.2.15.	Users .....	69
8.3.	Triggers .....	69
8.3.1.	Trigger para Criar Álbum Após a Criação de Área .....	69
8.3.2.	Trigger para Incrementar e Decrementar Contagem de Curtidas em Comentários .....	70
8.3.3.	Trigger para Moderação de Conteúdo de Evento .....	70
8.3.4.	Trigger para Validação de Conteúdo .....	70
8.3.5.	Trigger para Criação de Álbum Após Validação de Evento.....	71
8.3.6.	Trigger para Atualização da Pontuação Média de Publicações e Eventos.....	71
8.3.7.	Trigger para Gestão de Participação em Eventos .....	71
8.3.8.	Triggers para Notificações.....	71
8.4.	Procedimentos armazenados .....	72
9.	Avaliação dos Objetivos.....	73
10.	Observações.....	75
10.1.	Dificuldades .....	75
10.2.	Pontos Fortes e Fracos.....	75
10.2.1.	Pontos Fortes .....	75
10.2.2.	Pontos Fracos.....	75
11.	Conclusão .....	76
12.	Bibliografia.....	77
13.	Referências .....	78

---

## 2. Introdução

### 2.1. Contexto do Projeto

No mundo moderno, marcado por avanços tecnológicos e uma crescente globalização dos locais de trabalho, surgem desafios únicos para os funcionários em diversas áreas. O projeto "**The SoftShares**" enfrenta esses desafios, oferecendo uma solução inovadora que auxilia novos colaboradores a adaptarem-se em ambientes urbanos desconhecidos. O presente relatório tem como objetivo analisar detalhadamente o desenvolvimento de uma plataforma digital essencial para os colaboradores da Softinsa, facilitando a sua rápida adaptação e integração em todas as cidades onde a empresa está presente.

A plataforma procura fornecer informações e orientações sobre vários aspetos das cidades, incluindo restaurantes, alojamento, desporto, lazer e transporte. Este repositório é mais do que uma mera coleção de dados; é uma estrutura interativa que fomenta a colaboração e a partilha de experiências entre os colaboradores. Este aspeto colaborativo é particularmente importante no contexto atual, em que o trabalho remoto e as equipas distribuídas por diferentes locais tornam a integração social e profissional um grande desafio.

Este relatório explora minuciosamente os aspetos funcionais e técnicos do projeto, passando pelas componentes Web (*frontend* e *backend*), componentes da aplicação para dispositivos móveis, e por fim pela base de dados.

Por fim, aborda-se as **dificuldades** obtidas em todas as fases do projeto, terminando o relatório com uma **conclusão** geral sobre o mesmo.

Assim, este relatório aborda não só os avanços técnicos da plataforma "The SoftShares", mas também a coloca em contexto com as práticas atuais de gestão de recursos humanos e a integração de colaboradores em ambientes de negócios geograficamente dispersos e dinâmicos, de forma a promover uma maior entreeajuda e colaboração dentro da empresa como um todo.

### 2.2. Objetivos

O principal objetivo deste projeto é desenvolver uma aplicação integrada que inclua uma aplicação mobile e uma aplicação web com *backoffice* administrativo, além de uma API conectada a uma base de dados. Esta solução visa agregar e partilhar informações úteis sobre a cidade e a organização, promovendo uma integração ágil e fácil para os novos colaboradores.

Os objetivos específicos incluem:

- Desenvolver uma aplicação para dispositivos móveis para os colaboradores.
- Criar uma aplicação web com um *backoffice* para administração.
- Desenvolver uma API robusta.
- Facilitar a partilha de informações sobre saúde, desporto, formação, gastronomia, habitação, transportes e lazer.

---

## 3. Descrição Geral do Projeto

### 3.1. Visão Geral da Solução

O projeto "The SoftShares" consiste em desenvolver uma aplicação para dispositivos móveis, uma aplicação web com *backoffice* administrativo, e uma API. A aplicação para dispositivos móveis, desenvolvida em Dart, permitirá que os colaboradores acessem a informações úteis e interajam com outros membros da empresa. A aplicação web fornecerá uma interface administrativa para a gestão de conteúdos e utilizadores, enquanto a API facilitará a comunicação entre as aplicações e a base de dados.

### 3.2. Funcionalidades Principais

#### 3.2.1. Aplicação para Dispositivos Móveis

❖ **Acesso a Informações sobre a Cidade**

Os colaboradores poderão aceder a informações detalhadas relativas as diversas áreas (saúde, desporto, formação, gastronomia, habitação, transportes e lazer) na cidade onde irão (ou estão a) trabalhar.

Cada uma dessas categorias incluirá recomendações e avaliações de outros colaboradores, ajudando os novos colaboradores a encontrar rapidamente os melhores serviços e atividades.

❖ **Criação e Participação em Eventos**

A aplicação permitirá aos colaboradores criar e gerir eventos, como atividades desportivas, encontros sociais, e sessões de formação.

Os utilizadores poderão visualizar um calendário de eventos, inscrever-se em atividades, e receber notificações sobre atualizações ou novas interações nos eventos em que estão inscritos.

❖ **Partilha de Recomendações**

Os colaboradores poderão partilhar recomendações sobre locais e serviços, como restaurantes, ginásios, clínicas, e opções de alojamento.

As recomendações incluirão descrições detalhadas, avaliações, e fotos, proporcionando uma visão abrangente para os novos colaboradores.

❖ **Interação com Outros Colaboradores**

A aplicação incluirá fóruns de discussão para cada categoria, permitindo aos colaboradores fazer perguntas, partilhar experiências, e dar conselhos.

Será possível comentar em posts e responder a outros comentários, promovendo uma interação contínua e colaborativa.

#### 3.2.2. Aplicação Web com *Backoffice*

❖ **Gestão de Utilizadores e Permissões**

---

Administradores poderão criar e remover utilizadores, assim como atribuir permissões baseadas em perfis (Administrador/Gestor e Utilizador), sendo que apenas o Administrador do Sistema (conta única) poderá gerir contas de administradores dos diversos centros.

Os utilizadores receberão um email de confirmação de registo com um URL com um JWT (*JSON Web Token*) embebido para a criação da password para a conta destes. No caso das contas dos administradores de centros criadas pelo administrador do sistema, terão de alterar a password no primeiro login, uma vez que esta é definida previamente.

❖ **Moderação de Conteúdos**

Todos os conteúdos partilhados pelos utilizadores, como recomendações, eventos, e fóruns, serão moderados pelos administradores antes de serem publicados.

Administradores poderão editar, aprovar ou rejeitar conteúdos, garantindo que as informações partilhadas sejam relevantes e apropriadas. Após a criação de qualquer tipo de conteúdo por parte destes, será automaticamente validado para as cidades onde estes são administradores (no caso do administrador do sistema, será automaticamente validado em qualquer centro).

❖ **Criação de Formulários e Eventos**

A plataforma permitirá a criação dinâmica de formulários para inscrições em atividades desportivas, eventos, e outros, com diversos tipos de campos (TextBox, ComboBox, Checkbox, etc.).

Administradores poderão alterar o estado dos formulários (ativos/inativos) a qualquer momento e partilhar eventos através de redes sociais ou na plataforma colaborativa da empresa (ex: Microsoft Teams).

❖ **Visualização de Dashboards**

Um painel de controlo permitirá aos administradores visualizar uma visão geral das atividades mais comentadas, atividades mais vistas, número de tópicos abertos, e conteúdos por validar.

As métricas incluirão a percentagem de ofertas em cada área de atuação, número de utilizadores registados, e outras estatísticas importantes para a gestão da plataforma.

### 3.2.3. API

❖ **Gestão de Dados de Utilizadores**

A API fornecerá *endpoints* para operações CRUD (*Create, Read, Update, Delete*) para gerir dados de utilizadores, incluindo registo, autenticação, e recuperação de passwords.

A integração com serviços de *Single Sign-On* (SSO) como Google e Facebook será suportada.

❖ **Gestão de Eventos, Recomendações, Posts e Fóruns**

*Endpoints* específicos permitirão a criação, atualização, visualização e eliminação dos diversos tipos de conteúdo possível.

A API suporta a associação de geolocalização aos eventos, recomendações e posts, utilizando Google Maps.

❖ **Integração com Base de Dados**

A API será responsável pela comunicação com a base de dados PostgreSQL, utilizando Sequelize como ORM (Object-Relational Mapping).

As operações incluirão a gestão de categorias, subcategorias, diversos conteúdos, comentários, e muito mais.

---

#### ❖ **Notificações e Interações**

A API gerará notificações para interações importantes, como novas inscrições em eventos, novos comentários, e atualizações de estado.

As notificações serão enviadas aos utilizadores através da aplicação mobile , garantindo que estejam sempre informados sobre as atividades relevantes. (utilizando o **Firebase**)

Com estas funcionalidades, o projeto "The SoftShares" proporcionará uma plataforma integrada e eficiente para facilitar a integração de novos colaboradores, promovendo a partilha de informações e a interação contínua entre os membros da empresa.

### **3.3. Arquitetura do Sistema**

O sistema será composto por três principais componentes interconectados:

- Aplicação para Dispositivos Móveis: Desenvolvida em Dart/Flutter.
- Aplicação Web com *Backoffice*: Desenvolvida utilizando Node.js e Express.
- API e Base de Dados: Implementada com Node.js, Express, Sequelize e PostgreSQL.

---

## 4. Desenvolvimento do Projeto

### 4.1. Planeamento e Metodologia

O projeto foi desenvolvido fazendo primeiro uma divisão de tarefas por membros do grupo de forma a proporcionar as funcionalidades requeridas para o projeto de forma a aproveitar as melhores qualidades, capacidades, conhecimento e áreas de interesse dos diversos elementos. Para além de tal organizou-se os requisitos por dificuldade e importância ao longo do desenvolvimento de forma a priorizar os requisitos da melhor forma possível. Ao longo do desenvolvimento foram executados diversos testes dos diversos cenários visualizados. De notar, no entanto, que ocorreram situações que não forem previstas nem planeadas, que acabaram por causar certos percalços.

### 4.2. Tecnologias Utilizadas

Das diversas tecnologias utilizadas, as de maior relevância foram:

- Linguagens de Programação: Dart, JavaScript
- Frameworks e Bibliotecas: Flutter, Node.js, Express, Axios, Sequelize, Firebase.
- Base de Dados: PostgreSQL, previamente MicrosoftSQL Server.
- Ferramentas de Desenvolvimento: Git, GitHub, Visual Studio Code, Postman, pgAdmin, Firebase Console.

O desenvolvimento foi centrado em criar uma interface intuitiva e responsiva, facilitando a navegação e a interação dos utilizadores com a aplicação.

---

## 5. Desenvolvimento da Aplicação para Dispositivos Móveis

### 5.1. Requisitos Funcionais e Não Funcionais

A aplicação desenvolvida nesta parte do projeto tem como principal objetivo proporcionar uma interface amigável e funcional que permita aos colaboradores da Softinsa aceder a informações sobre a cidade onde trabalham (ou irão trabalhar), criar e participar em eventos, e interagir com outros colaboradores. A seguir, os requisitos funcionais e não funcionais são detalhados:

#### 5.1.1. Requisitos Funcionais

- **Acesso a Informações:** Os utilizadores devem poder visualizar informações sobre saúde, desporto, formação, gastronomia, habitação, transportes e lazer, categorizadas para facilitar a navegação.
- **Criação de Eventos:** A aplicação permite a criação de eventos, onde os utilizadores podem definir detalhes como local, data, e tipo de atividade, e compartilhá-los com outros colaboradores.
- **Participação em Eventos:** Os colaboradores podem visualizar um calendário de eventos e inscrever-se nas atividades de seu interesse. Eles também podem comentar nos eventos (após registo) e interagir com outros participantes.
- **Partilha de Recomendações:** Função para os utilizadores compartilharem recomendações sobre locais e serviços, incluindo descrição, fotos e avaliações.
- **Interação com Outros Colaboradores:** Fóruns de discussão e áreas de comentários permitem a interação contínua entre os utilizadores, promovendo a colaboração e troca de experiências.

#### 5.1.2. Requisitos Não Funcionais

Os requisitos não funcionais incluem a usabilidade, desempenho, segurança, e compatibilidade da aplicação.

Seguindo estes requisitos, tivemos um grande foco no layout da aplicação para tentar chegar a um design que fosse tão estético como intuitivo, passando por uma fase inicial de **protótipos de baixa fidelidade**, seguindo para uma primeira versão do design da app e, finalmente, chegando ao design final com os protótipos de alta-fidelidade.

Outro grande requisito não funcional que demos bastante atenção, foi a segurança da app, implementando diversos mecanismos de proteção, como por exemplo, os **tokens**.

### 5.2. Arquitetura Organizacional

O código foi dividido da seguinte maneira:



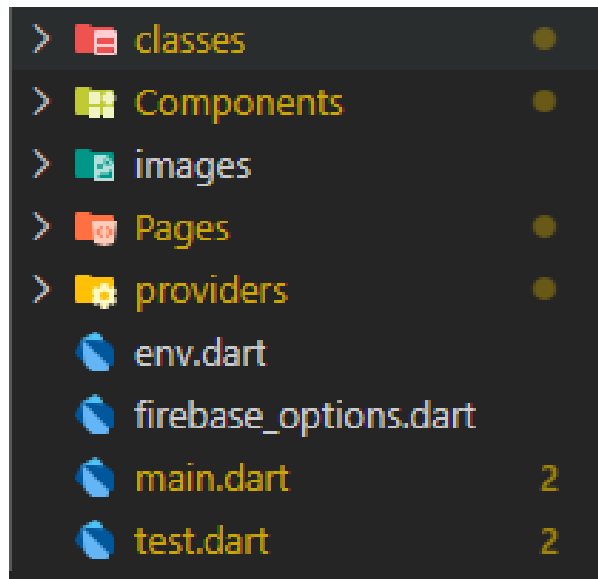


Figura 5-1 Estrutura do Código

- **Classes:** Todas as classes criadas para o bom funcionamento da aplicação. Alguns exemplos seriam *commentsClass.dart*, *fieldClass.dart*, bem com o código da base de dados, *db.dart*, e da **API**, *ClasseAPI.dart* (e ainda algumas exceções);
- **Components:** Esta pasta contém os **widgets** reutilizáveis. Exemplos destes **widgets** são, *formAppBar.dart* e *forumCard.dart*;
- **Images:** Pasta onde as imagens utilizadas estão guardadas;
- **Pages:** Todas as páginas criadas para o funcionamento da aplicação;
- **Providers:** Provedor de autenticação, responsável pela gestão das operações de login, logout, armazenamento seguro de dados e integração com diversos serviços de autenticação.

### 5.3. Desafios Principais

- **Integração com API:** A integração da aplicação com a API desenvolvida em Node.js exigiu um tratamento cuidadoso das requisições HTTP, utilizando a biblioteca **http** para gerenciar as operações de rede de forma eficiente e segura.
- **Gestão de Estados:** Garantir que os estados da aplicação (e.g., carregamento de dados, interações dos utilizadores) fossem geridos de forma eficiente e reativa foi um desafio superado com o uso do Provider.

### 5.4. Desenvolvimento em Dart/Flutter

A aplicação foi desenvolvida utilizando o Flutter, um *framework* UI *open-source* da Google que permite a criação de aplicações nativas para Android e iOS a partir de um único código base escrito em Dart. O uso do Flutter proporcionou uma rápida iteração no desenvolvimento, permitindo a criação de interfaces modernas e responsivas com menor esforço de manutenção.

O processo de desenvolvimento envolveu a criação de várias “páginas” interativas, como a “página” inicial com acesso às diversas categorias de informações e todo o conteúdo (de determinado centro) do mais recente para o mais antigo, a “página” de eventos por área, “página” do calendário interativo, “página” dos posts, “página” dos fóruns, e a “página” dos pontos de interesse, onde os utilizadores podem visualizar e compartilhar experiências, recomendações, entre outros.

### 5.4.1. Interface do Usuário

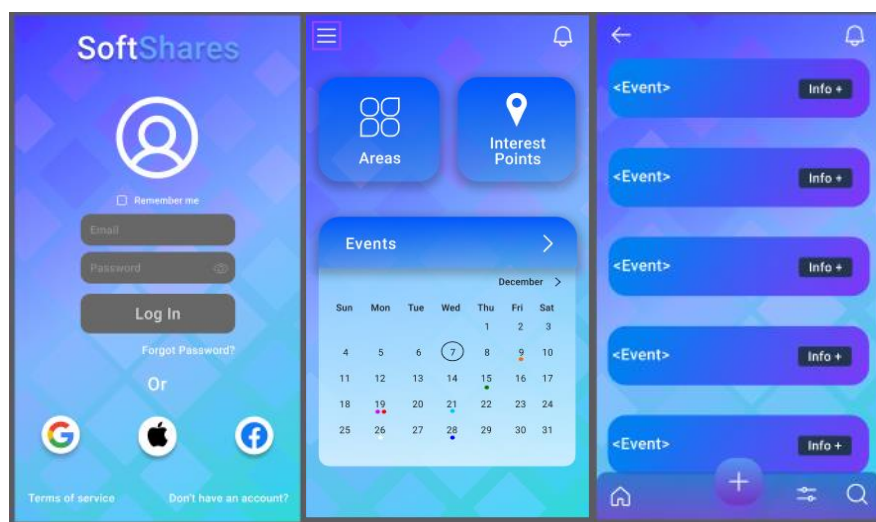
A interface da aplicação foi projetada para ser limpa e intuitiva, com uma navegação simples entre as diferentes categorias e funcionalidades. A paleta de cores e a tipografia foram escolhidas para proporcionar uma experiência visual agradável, consistente e relacionada com a empresa Softinsa.

### 5.4.2. Prototipagem

Os protótipos foram essenciais para o desenvolvimento da aplicação “SoftShares”. Estes ajudaram desde o início a fazer designs elegantes e, quando necessário, alterar os mesmos, sem ter de recorrer à alteração de código.

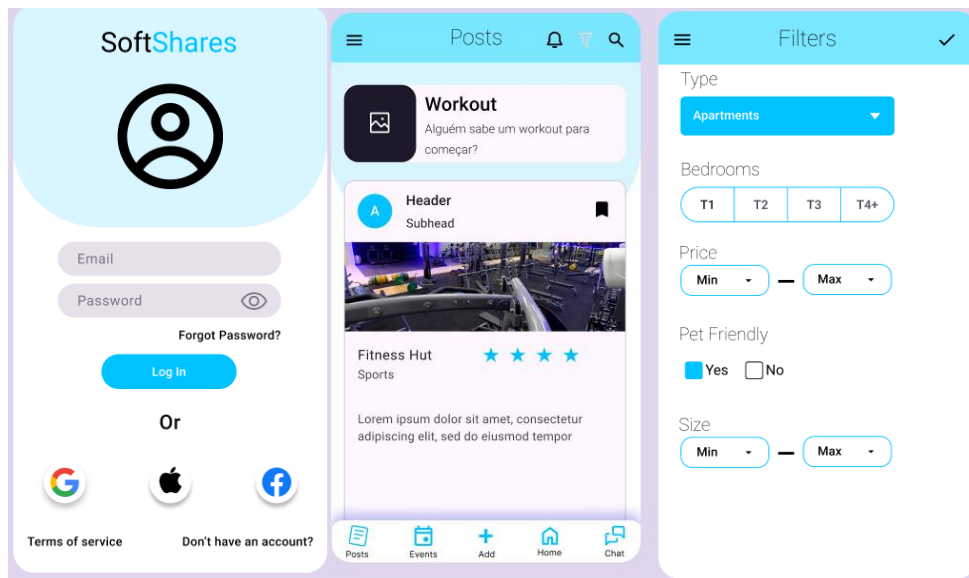
De início, desenvolvemos o protótipo de baixa fidelidade (PBF) para obter uma ideia preliminar dos tipos de *widgets* que a aplicação necessitaria.

Após termos uma ideia inicial, passamos à fase seguinte com os protótipos de alta-fidelidade. Com estes protótipos, a interação utilizador-máquina já era mais relevante no desenvolvimento do design.



**Figura 5-2:** Login, Homepage, Event page (PBF)

Estes ecrãs forneceram uma visão inicial sobre o design da aplicação. Muitas das ideias apresentadas nestes *mockups* foram posteriormente revistas e refinadas, servindo como base para o desenvolvimento da versão final do design da aplicação. Este processo de iteração permitiu melhorar a experiência do utilizador e alinhar o design com os objetivos funcionais e estéticos da plataforma.

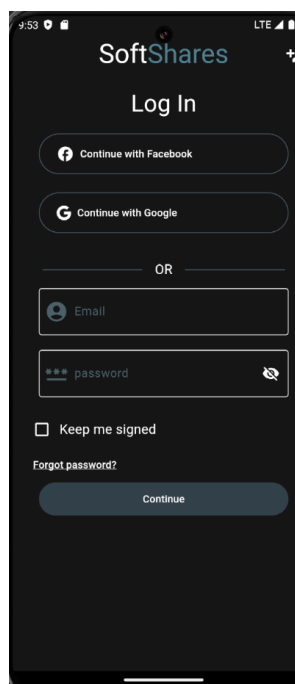


**Figura 5-3:** Login, Post page, Filter (PAF)

Esta segunda versão do design da aplicação teve como objetivo explorar a interação do cliente com o sistema, procurando o equilíbrio entre o design e a usabilidade. Nesta fase, procurámos incorporar de forma mais acentuada as cores que identificam a Softinsa (azul e branco), integrando-as de maneira a complementar o design sem comprometer a funcionalidade. Na nova e última versão, os *screenshots* presentes neste relatório foram capturados no *dark mode*. Posteriormente, são apresentados *screenshots* adicionais que ilustram o aspeto geral da aplicação no modo de cor normal.

### 5.4.3. Principais Componentes de UI

#### 5.4.3.1 Página *Sign-in*:



**Figura 5-4:** Sign in Page

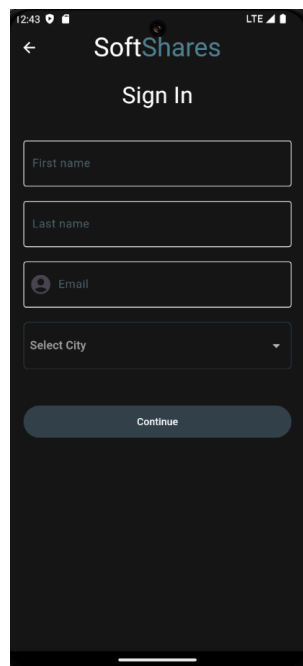
---

Esta é a tela inicial da aplicação, que o utilizador encontra ao abrir a mesma pela primeira vez. Nesta página, há dois campos de texto onde o utilizador pode inserir o seu email e palavra-passe para efetuar o login. Abaixo dos campos de texto, há uma *checkbox* que permite ao utilizador optar por manter a sessão iniciada.

Para além disso, o utilizador tem a opção de iniciar sessão utilizando as suas contas Google ou Meta, através de dois botões localizados acima dos campos de texto.

Por fim, o botão no canto superior esquerdo direciona o utilizador para a página de registo, onde pode criar uma conta.

#### 5.4.3.2 Página Sign-up



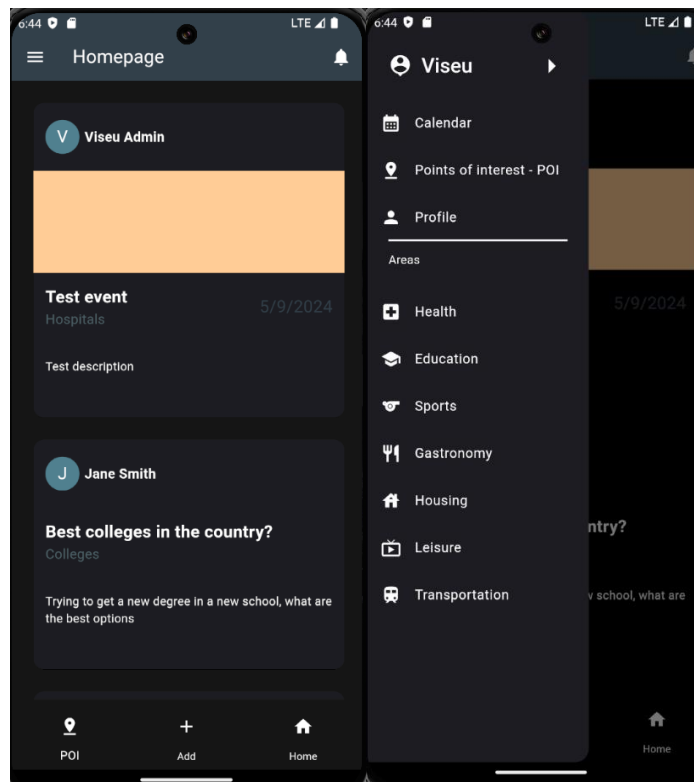
**Figura 5-5:** Sign Up Page

A página de criação de conta contém três campos de texto onde o utilizador insere o primeiro e último nome e o email para a criação da conta.

Contêm também um botão *dropdown* para o utilizador seleccionar o centro onde se situa.

---

### 5.4.3.3 Página Homepage



**Figura 5-6:** Homepage, Sidebar

Nas figuras acima, podemos observar a página principal da aplicação e a *sidebar*.

Na homepage, encontram-se todos os “posts”, “events” e “forums” criados por ordem do mais recentemente criado, bem como uma *bottomNavigationBar* que dá acesso a duas páginas diferentes (*Points of interest* e *Create publication*).

Na *sidebar* observamos um botão para selecionar a cidade de qual o conteúdo está a ser retirado, um botão para a página **calendar** onde se encontra um calendário com os eventos marcados pelos dias, outro para aceder à página **points of interest** e um para aceder à página pessoal do utilizador.

Temos ainda sete botões diferentes para aceder a cada uma das áreas diferentes.

#### 5.4.3.4 Página Categoria:

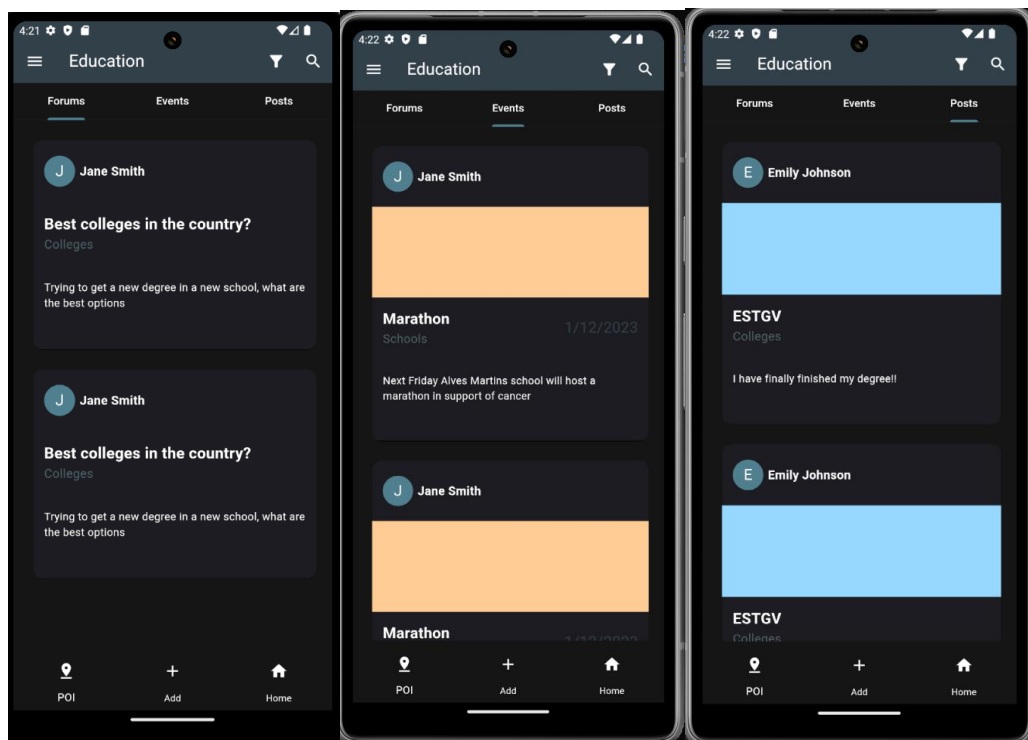


Figura 5-7: Forums, Events, Posts ( por Categoria)

Cada uma das páginas das áreas contém três abas: Fóruns, Eventos e Posts, cada uma exibindo os respectivos conteúdos. Na aba de Posts, também é possível filtrar os posts por preço e avaliação.

#### 5.4.3.5 Página Evento:

A página de conteúdo do evento é composta por três abas:

- **Overview:** Toda a informação do evento está exposta nesta página (título, descrição, localização, etc)
- **Forum:** Fórum do evento, apenas visível para quem se encontra inscrito no evento.
- **Gallery:** Galeria de imagens. Semelhante ao fórum, apenas quem se encontra inscrito pode visualizar o conteúdo e inserir outros

Na aba de *overview*, existe também um botão no fundo da página que muda dependendo do utilizador que está a visualizar o evento.

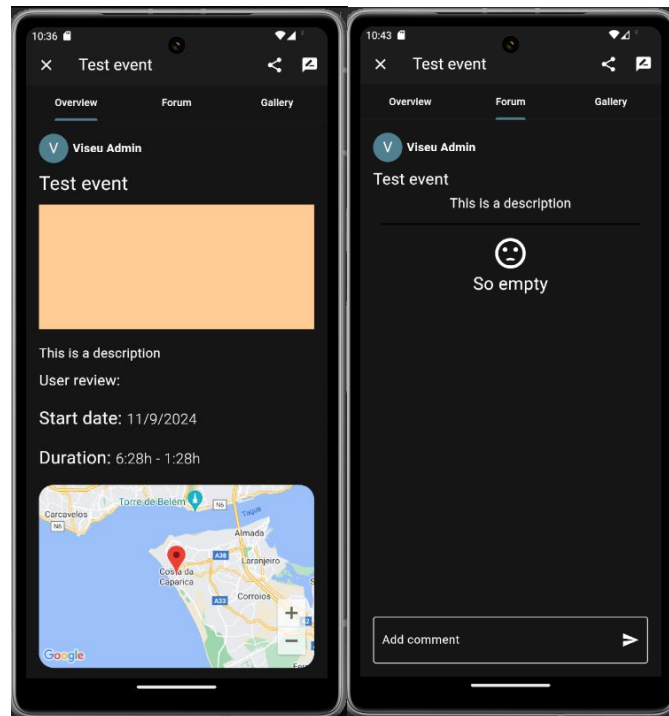
Se o utilizador for o criador do evento, este botão vai redirecionar o utilizador para uma nova página onde pode ver a lista dos utilizadores inscritos e as suas respostas aos formulários.

Se o utilizador não se encontrar inscrito no evento, o botão redireciona- o para a página de registo. Se o utilizador já tiver registado, este botão deixa de aparecer.

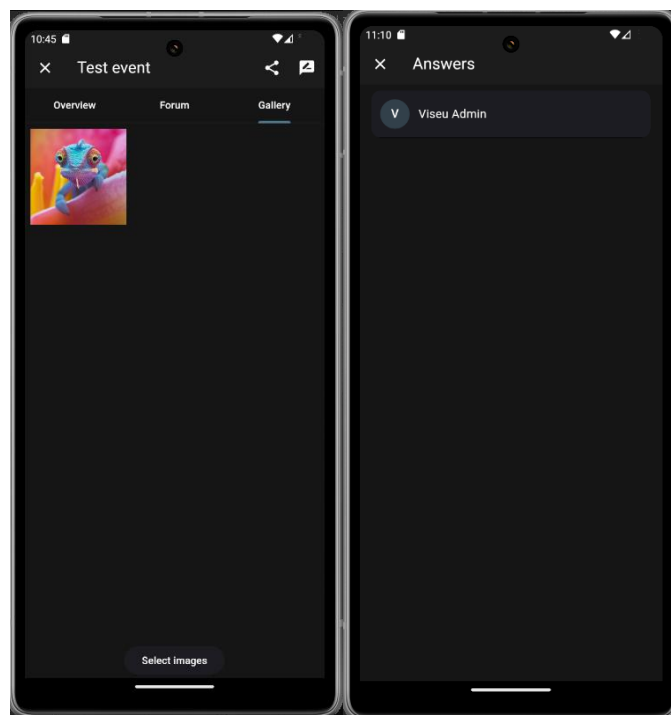
No top da página, existem dois botões:

- **Share:** Partilha o conteúdo numa outra app;
- **Rate:** Avaliação do conteúdo (Se o conteúdo já tiver sido validado);

- **Edit:** Editar o conteúdo (Se o conteúdo ainda tiver sido validado);



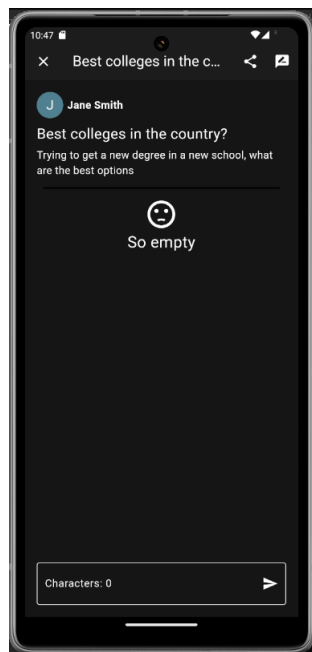
**Figura 5-8:** Event Overview, Event Forum



**Figura 5-9:** Event Gallery, Event checkAnswers

#### 5.4.3.6 Página Fórum:

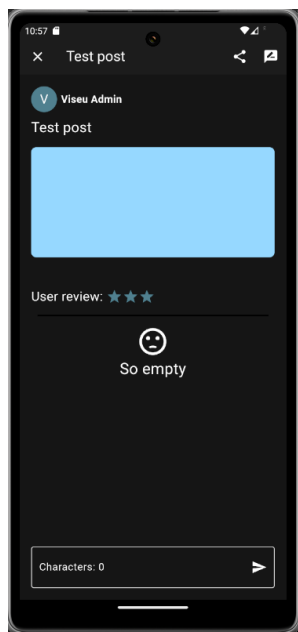
Esta página é semelhante á aba “*fórum*” na página dos eventos com a única diferença ser que é independente e não está associado a nenhum evento.



**Figura 5-10:** Forum page

#### 5.4.3.7 Página Post:

A página de “*post*” é semelhante às outras. Esta página tem diferentes informações dependendo de como o “*post*” é criado.



**Figura 5-11:** Post page



---

#### 5.4.3.8 Página Criação Contéudo:

The image displays two side-by-side screenshots of a mobile application's 'Create Publication' screen. The left screenshot shows the form with fields for Title, Description, Location, Recurrent, and Date. The right screenshot shows the form with fields for Search, Recurrent, Date, Time, Area, Sub Area, and an Advance button.

**Figura 5-12:** Criação de evento

Na página de criação de eventos, encontram-se todos os campos necessários para a criação do mesmo.

Estes campos são, por exemplo, o título, a *thumbnail*, dia do evento, entre outros.

The image shows a screenshot of the 'Create Form' screen in a mobile application. It features a list of form elements: 'Radio Button', 'Checkbox', and 'Text/Numeric Input'. At the bottom, there is an 'Add +' button.

**Figura 5-13:** Criação de Form

Após os campos obrigatórios da criação de eventos serem preenchidos, passamos para a criação do formulário que os participantes têm de preencher.

O criador tem a opção entre três campos:

- Radio Button;
- Checkbox;
- Campo de texto;

Os campos costumáveis partilham semelhança na maneira de serem criados, mudando apenas algumas opções.

A criação de outro tipo de conteúdos é semelhante, com algumas alterações nos campos obrigatórios e a ausência de formulários.

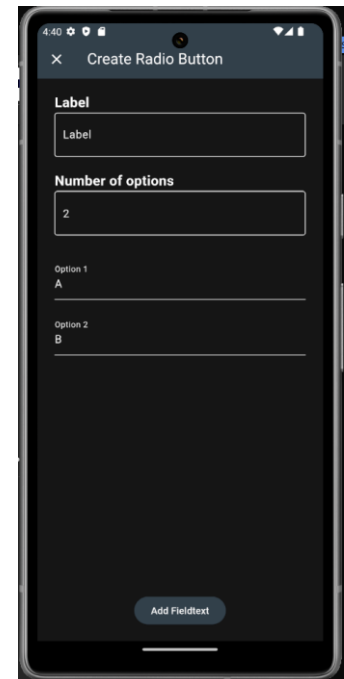


Figura 5-14: Criação de um campo

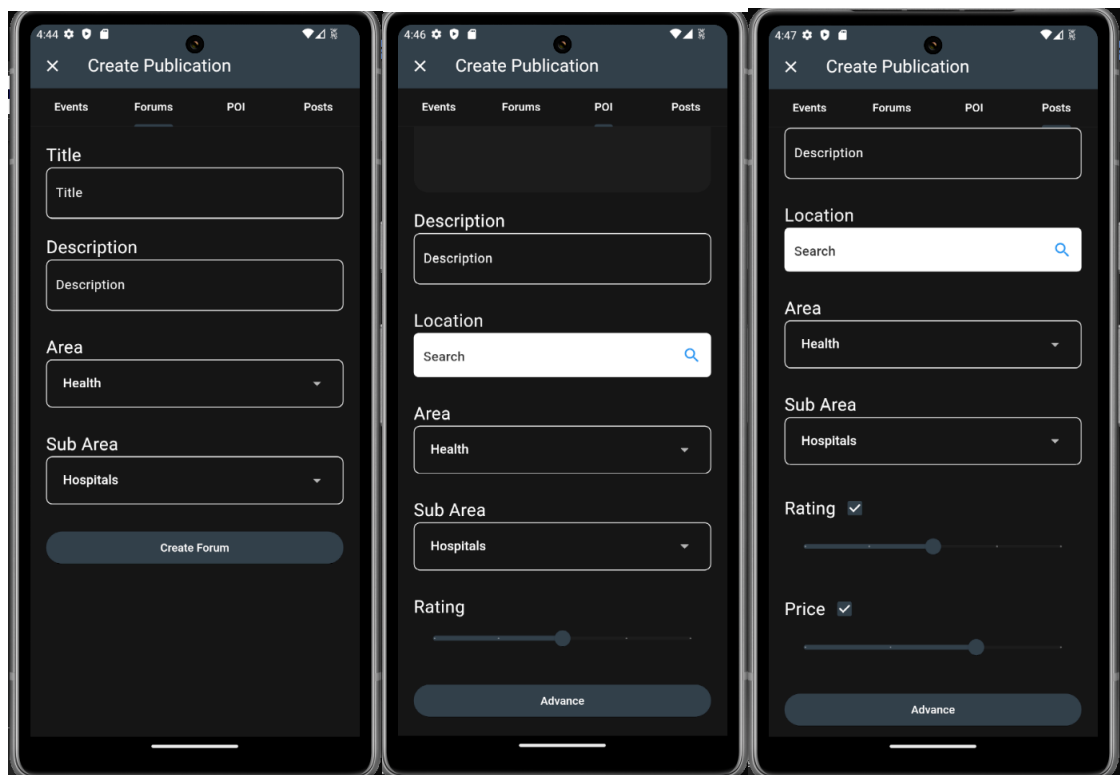


Figura 5-14: Criação de Forums, POIs, Posts

---

#### 5.4.3.9 Página *Profile*:

Nesta página, os utilizadores conseguem ver as publicações criadas e os eventos onde se encontram registados. Conseguem, também, se a publicação estiver á espera de validação, editar a mesma.

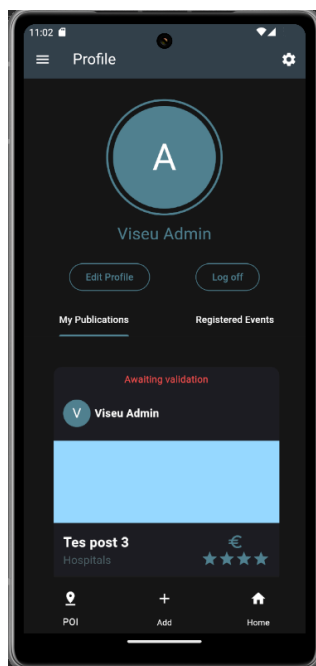


Figura 5-15: Página myProfile

#### 5.4.3.10 Página *Calendar*

Na página *calendar*, existe um calendário com um marcador nos dias em que existem eventos.

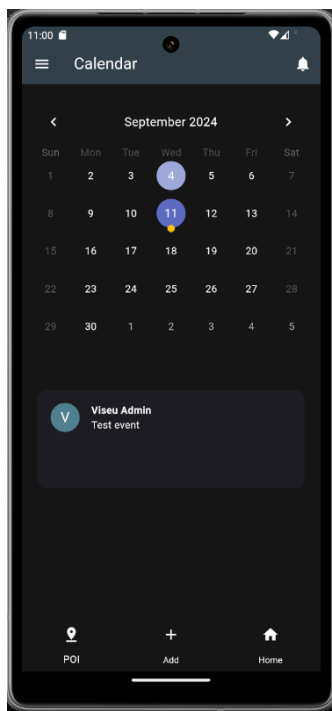


Figura 5-16: Calendar page

#### 5.4.4. Aplicação em Light Mode

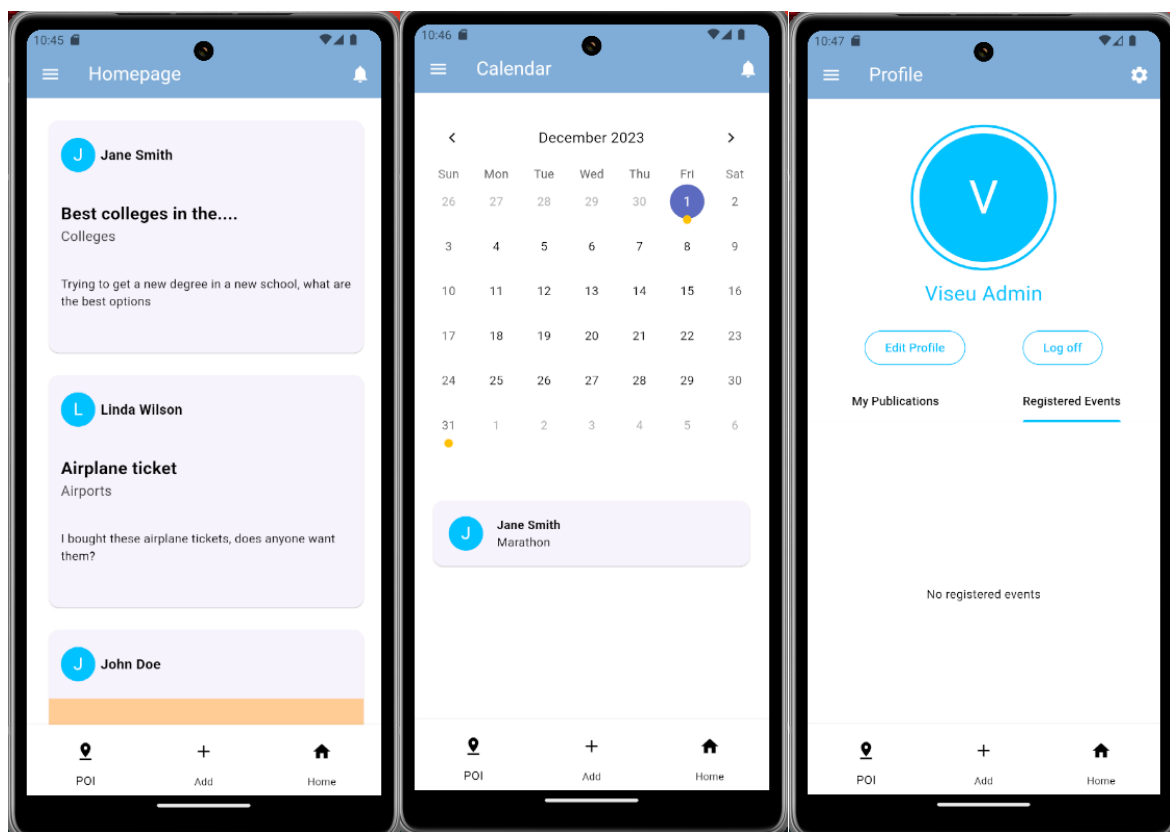


Figura 5-17: Screenshots light mode

---

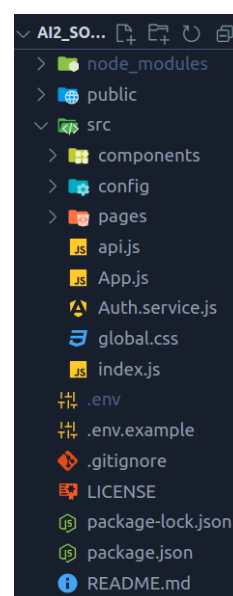
## 6. Desenvolvimento da Aplicação Web com *backend*

A arquitetura do *frontend* (aplicação web) é baseada em componentes, onde cada componente representa uma parte específica da interface, e esses componentes comunicam entre si através de props e estados.

A comunicação entre o frontend e o backend é feita de forma assíncrona, utilizando Axios para enviar e receber dados. O Axios simplifica a realização de chamadas HTTP, oferecendo uma API limpa e fácil de usar para lidar com pedidos GET, POST, PUT, DELETE, OPTIONS e PATCH.

### 6.1. Arquitetura Organizacional

Este projeto está organizado de forma modular, facilitando a sua manutenção e escalabilidade. Na pasta `src`, encontramos a pasta `components`, que contém componentes reutilizáveis como cartões, mapas, calendários ... , promovendo a reutilização de código. A pasta `pages` armazena as páginas dinâmicas da aplicação, enquanto o ficheiro `App.js` define as rotas do projeto. O ficheiro `api.js` lida com o envio de tokens JWT para autenticação, e o `Auth.service.js` é responsável pela gestão e utilização desses tokens. Para evitar a necessidade de modificar ficheiros diretamente, utiliza-se um ficheiro `.env` onde estão armazenadas variáveis de ambiente, permitindo uma configuração centralizada e segura.



**Figura 6-1:** Estrutura do código

### 6.2. Estrutura logica

No *frontend*, a organização dos ficheiros está dividida entre componentes e páginas. As páginas servem como containers principais que podem, ou não, incorporar vários componentes, conforme necessário. Esta abordagem modular visa evitar a repetição excessiva de código, promovendo a reutilização eficiente e facilitando a manutenção do projeto. Ao separar a lógica e a apresentação em componentes distintos, é possível criar uma estrutura mais clara e organizada, que não só melhora a legibilidade do código, como também simplifica o processo de desenvolvimento.

---

## 6.3. Desafios Principais

- **Integração com API:** A integração da aplicação com a API desenvolvida em Node.js exigiu um tratamento cuidadoso dos pedidos HTTP, utilizando a biblioteca **axios** para gerir as operações de rede de forma eficiente e segura.
- **Layout Gráfico:** Garantir que tudo fica bem em todos os ecrãs não foi uma tarefa fácil, pois, além da responsividade, foi também necessário criar uma interface atrativa e intuitiva para o utilizador.

## 6.4. Desenvolvimento em Node.js e Express

A aplicação web foi desenvolvida utilizando Node.js e Express, *frameworks* que oferecem uma plataforma robusta e escalável para a construção de aplicações web. O Express foi utilizado para estruturar o servidor web e as rotas, enquanto o Node.js forneceu um ambiente eficiente para o manuseio de requisições assíncronas e integração com a API e a base de dados.

### 6.4.1. Estrutura da Aplicação

- **Rotas e Controladores:** Foram implementadas rotas RESTful para gerir as operações CRUD de utilizadores, eventos e conteúdos. Os controladores foram responsáveis por processar as requisições e interagir com a camada de serviço.
- **Middleware de Autenticação:** Utilizou-se *middleware* para garantir que apenas utilizadores autenticados e autorizados pudessem aceder certas áreas do backoffice, aumentando a segurança.
- **Integração com a API:** O backoffice comunica diretamente com a API para buscar e atualizar dados, garantindo que as informações visualizadas e manipuladas estejam sempre atualizadas.

Esta parte do projeto será explorada mais minuciosamente na secção (7) da API .

## 6.5. React UI

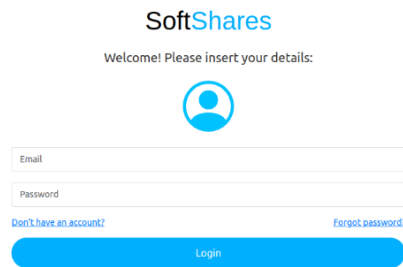
O desenvolvimento focou-se na criação de uma interface administrativa eficiente, permitindo aos administradores gerir facilmente o conteúdo e os utilizadores. Foram utilizadas práticas de desenvolvimento seguro e escalável.

O processo de desenvolvimento da versão web envolveu a criação de várias páginas interativas e dinâmicas, como a página de posts/eventos, que permite aceder a diferentes categorias de informação e apresenta o conteúdo adequado com base nos parâmetros de entrada (*props*). Foi também implementado um calendário interativo, e criadas páginas dedicadas a diferentes tipos de conteúdo. O código foi estruturado de forma a facilitar futuras intervenções, permitindo alterações sem grandes mudanças à base existente.

---

## 6.5.1. Páginas

### 6.5.1.1 Login

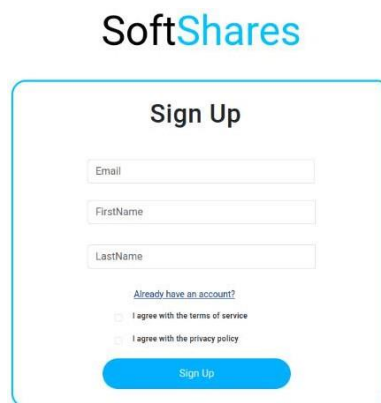


The image shows the login page for 'SoftShares'. At the top, the logo 'SoftShares' is displayed in blue. Below it, the text 'Welcome! Please insert your details:' is centered. A blue circular icon with a white person silhouette is positioned above the input fields. There are two text input fields: 'Email' and 'Password'. Below the 'Email' field is a link 'Don't have an account?'. Below the 'Password' field is a link 'forgot password?'. At the bottom, there is a large blue button with the text 'Login' in white.

**Figura 6-2:** Página Login

Na Página de Login, o administrador insere o e-mail e a *password*. O sistema verifica se os dados inseridos correspondem aos registados na base de dados. Se a verificação for bem-sucedida, o sistema retorna um *token* JWT, que será armazenado no `localStorage` e usado para verificar a validade em todas as páginas.

### 6.5.1.2 Registo

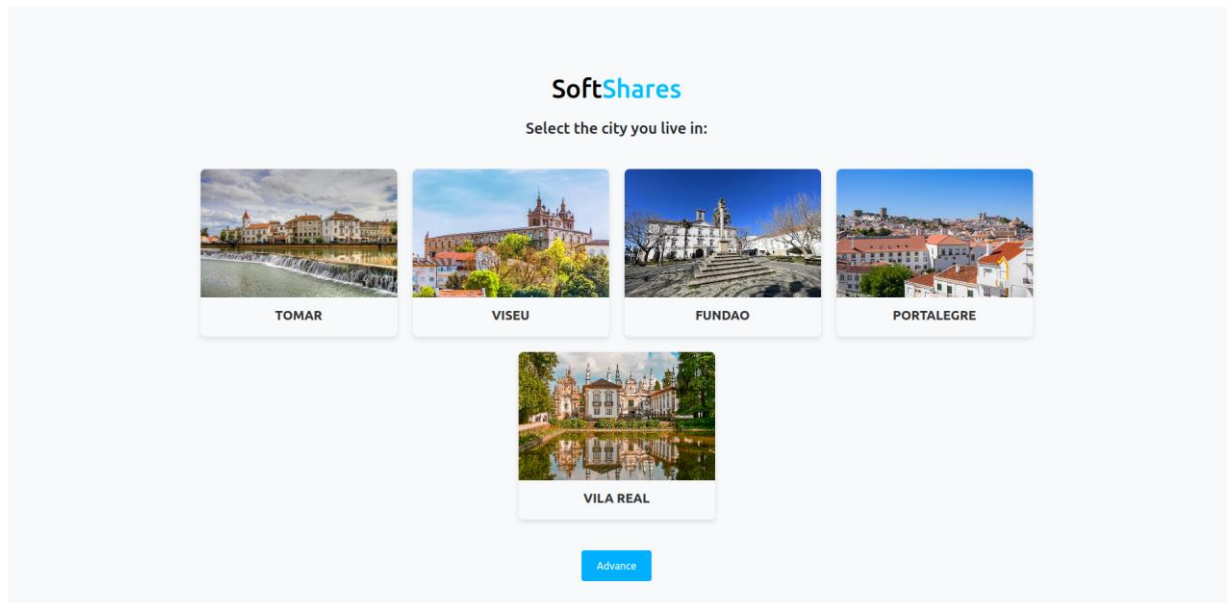


The image shows the sign-up page for 'SoftShares'. At the top, the logo 'SoftShares' is displayed in blue. Below it, the title 'Sign Up' is centered. There are three text input fields: 'Email', 'FirstName', and 'LastName'. Below the 'Email' field is a link 'Already have an account?'. Below the 'FirstName' and 'LastName' fields are two checkboxes: 'I agree with the terms of service' and 'I agree with the privacy policy'. At the bottom, there is a large blue button with the text 'Sign Up' in white.

**Figura 6-3:** Página *Sign Up*

Na página de Registo, são apresentados três campos ao utilizador: Email, Primeiro Nome e Último Nome. O utilizador deve preencher cada um destes campos com as suas informações pessoais. Após o preenchimento e submissão do formulário, os dados inseridos são armazenados no `localStorage` do browser. Este armazenamento local permite que as informações sejam facilmente acedidas na página seguinte.

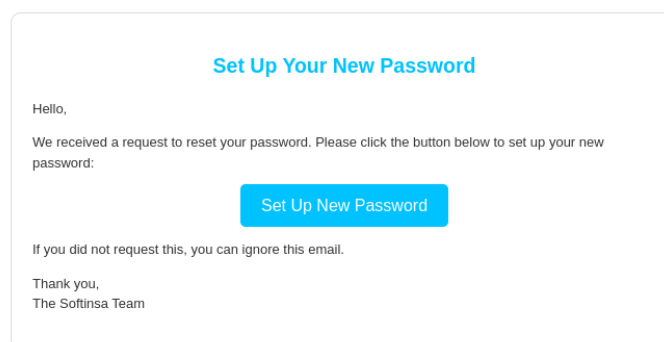
### 6.5.1.3 Seleção de Centro



**Figura 6-4:** Seleção de Centro - *Sign Up*

Nesta página, o utilizador escolhe o centro ao qual pertence. Após seleccionar o centro e avançar para o próximo passo, o sistema recupera os dados previamente armazenados no `localStorage`, nomeadamente o e-mail, o Primeiro Nome e o Último Nome do utilizador. Com estas informações, o sistema envia um e-mail de verificação para o endereço de e-mail fornecido. Este e-mail contém um link que permitirá ao utilizador definir a sua palavra-passe, completando assim o processo de registo, embora a conta fique inativa até um administrador existente a verificar.

### 6.5.1.4 Email Enviado



**Figura 6-5:** Email - *Sign Up*

Aqui está o e-mail que foi enviado para validar o *token*. O URL inclui um *token* que serve para identificar o utilizador que está a definir a palavra-passe.



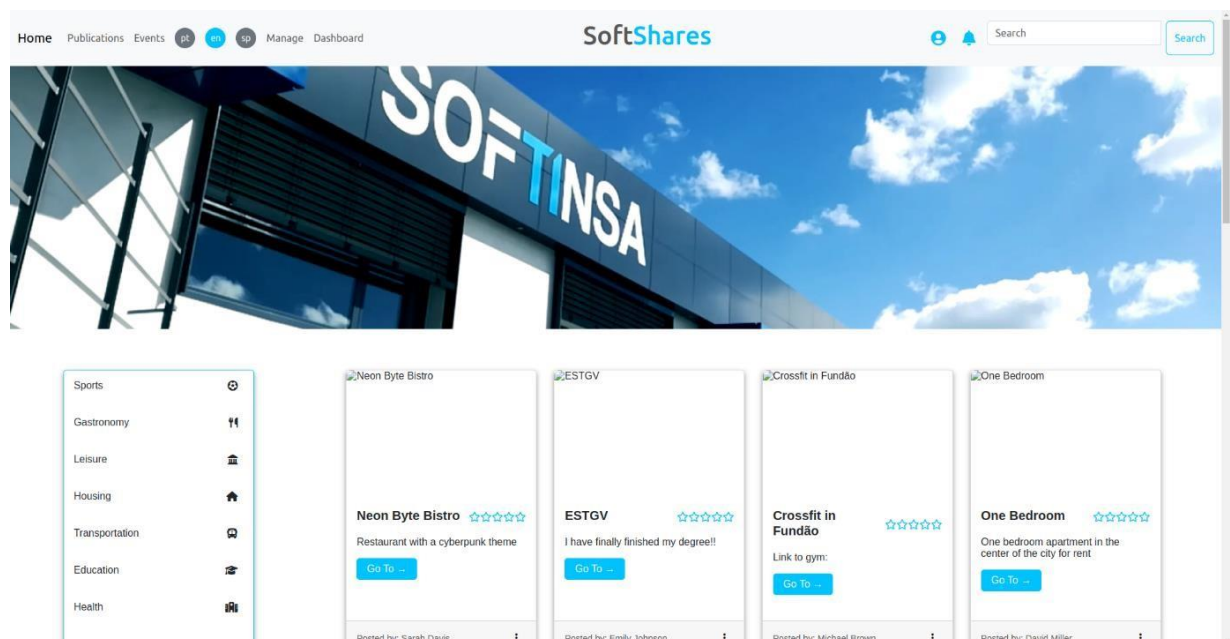
### 6.5.1.5 Definição de Palavra-Passe



**Figura 6-6:** Definição *Password* - *Sign Up*

Nesta página, a identidade do utilizador é verificada com base no *token* apresentado no URL. A interface apresenta dois campos para o utilizador preencher: "*Password*" e "*Confirmar Password*". O utilizador deve introduzir a nova *password* em ambos os campos. Se ambas as entradas coincidirem, o sistema considera que a nova *password* é válida e procede à atualização da mesma na base de dados. Caso as *passwords* não coincidam, o utilizador é notificado do erro e solicitado a corrigir a discrepância. Este processo garante que a nova *password* é corretamente definida e a segurança da conta do utilizador é mantida.

### 6.5.1.6 Página Inicial



**Figura 6-7:** *Homepage*

Na página Inicial, são apresentados tanto os Posts como os Eventos criados na plataforma, organizados de forma clara e acessível. Nos Posts, são exibidos o Nome, uma Descrição concisa, o nome do Publicador e a classificação de Rating, proporcionando uma visão completa e avaliativa do conteúdo. Na secção de Eventos, são mostrados o Nome do evento,

uma Descrição detalhada e a Data em que ocorrerá, permitindo aos utilizadores planear e participar nas atividades de interesse. Esta organização facilita a navegação e o acesso rápido às informações mais relevantes para os utilizadores.

6.5.1.7 Posts/Eventos

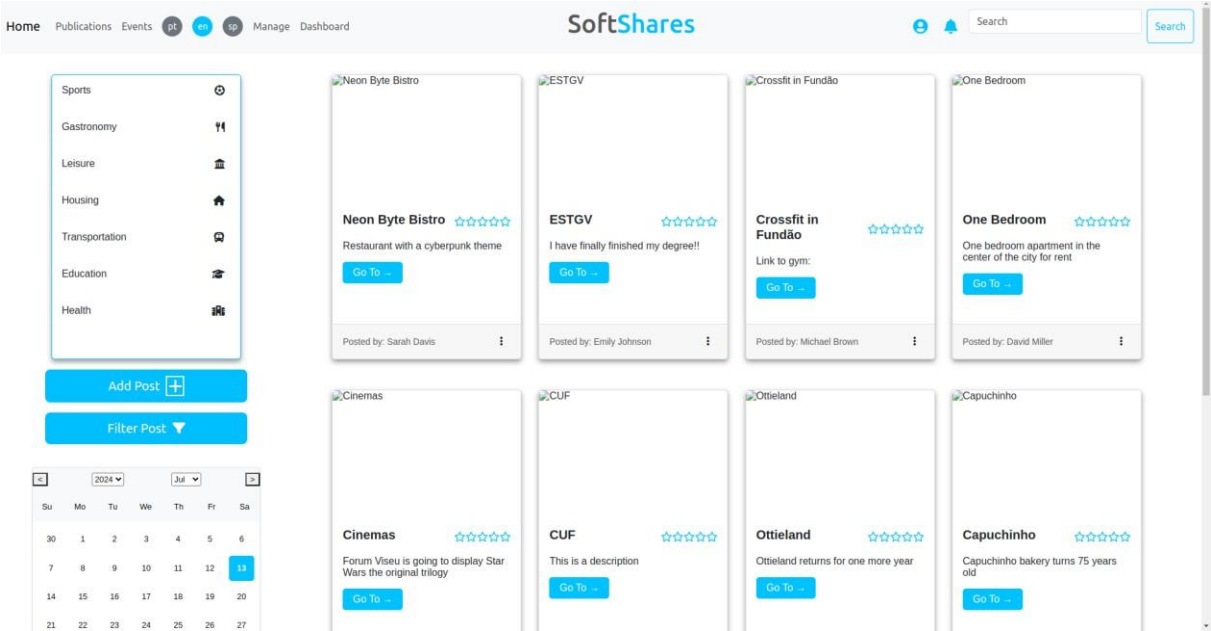
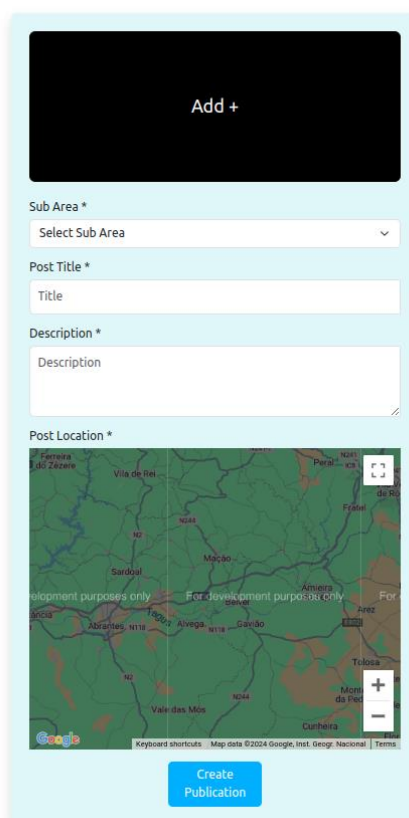


Figura 6-8: Publications

Esta página é chamada duas vezes, uma para exibir os Posts e outra para exibir os Eventos, diferenciando-se apenas pelos dados passados (Posts ou Eventos). Em ambas as chamadas, a página contém uma lista que mostra os detalhes relevantes: para Posts, exibe o Nome, Descrição, Publicador e Rating; para Eventos, mostra o Nome, Descrição e Data. Além disso, a página inclui dois botões funcionais: um botão "Adicionar" que permite criar um Post ou Evento, conforme o contexto da página, e um botão "Filtro".

---

### 6.5.1.8 Adicionar Post



The form is titled "Add +" and is set against a light blue background. It contains the following elements:

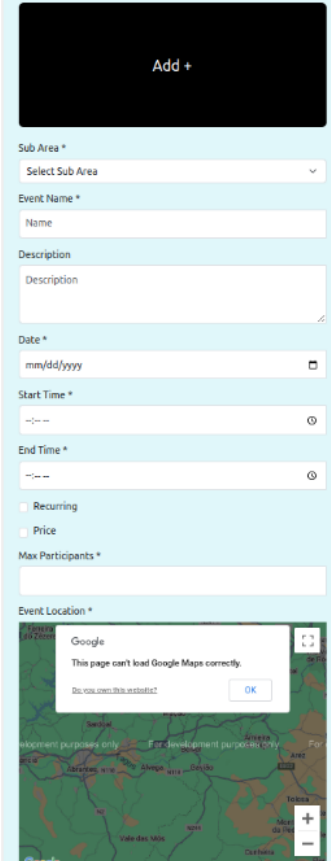
- A large black rectangular area at the top for adding an image, with the text "Add +" in white.
- A "Sub Area \*" dropdown menu with the placeholder text "Select Sub Area".
- A "Post Title \*" text input field with the placeholder text "Title".
- A "Description \*" text input field with the placeholder text "Description".
- A "Post Location \*" section featuring a map of a region with various place names (e.g., Vila de Rei, Macão, Amora, Fátima, Vale das Mós, Curbeira). The map includes a scale bar and a "Keyboard shortcuts" link.
- A blue "Create Publication" button at the bottom.

**Figura 6-9:** Adicionar Post

Esta página contém um formulário dedicado à criação de um novo Post, projetado para ser intuitivo e fácil de usar. O formulário inclui um campo para selecionar a imagem desejada, permitindo que os utilizadores adicionem um elemento visual ao seu Post. Além disso, há um menu para escolher a subárea à qual o Post pertence. O utilizador também tem de preencher o Título do Post, que dará uma visão imediata do assunto abordado, e uma Descrição detalhada, onde pode fornecer mais informações e contexto sobre o conteúdo do Post. Seleciona ainda uma área no mapa para identificar o local do estabelecimento presente no Post (opcional).

---

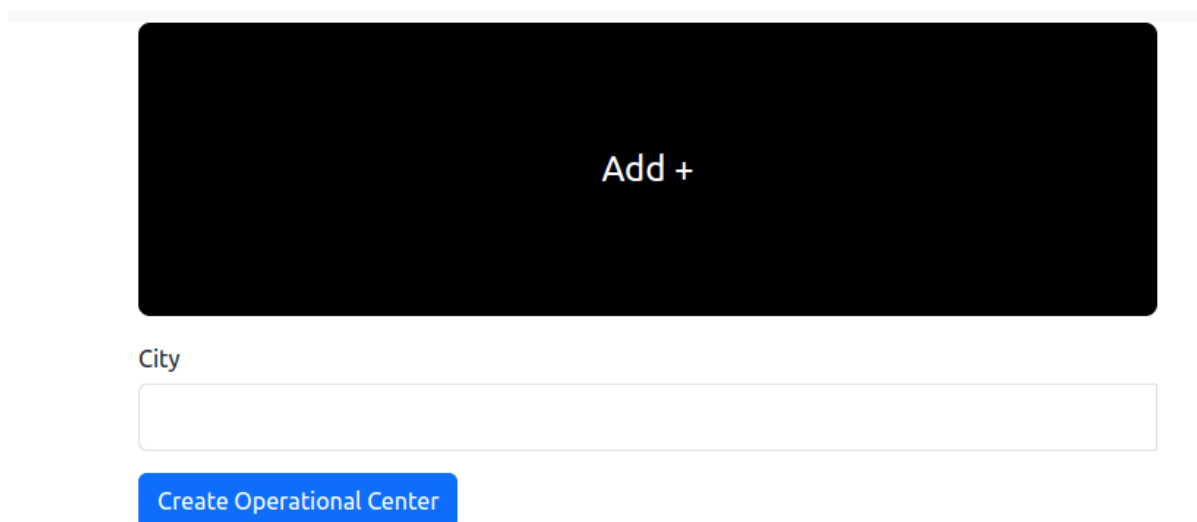
### 6.5.1.9 Criação de Evento



**Figura 6-10:** Criar Evento

A página de Criação de Evento está concebida para facilitar a entrada de todas as informações necessárias para a organização de um evento de forma eficiente e detalhada. O formulário inclui um campo para a seleção de uma imagem representativa do evento, bem como uma opção para escolher a subárea pertinente. O utilizador deve preencher o nome do evento, fornecendo uma descrição detalhada para informar os participantes sobre o que esperar. A data do evento é obrigatória, permitindo a programação precisa do evento, com adição da data de início e de fim para um espaço temporal mais conciso. Há também opções para indicar se o evento é recorrente, bem como campos opcionais para especificar o preço e o número máximo de participantes. Além disso, a localização do evento é escolhida num mapa interativo, proporcionando uma forma visual e precisa de definir onde o evento terá lugar.

### 6.5.1.10 Criação de Centro



Add +

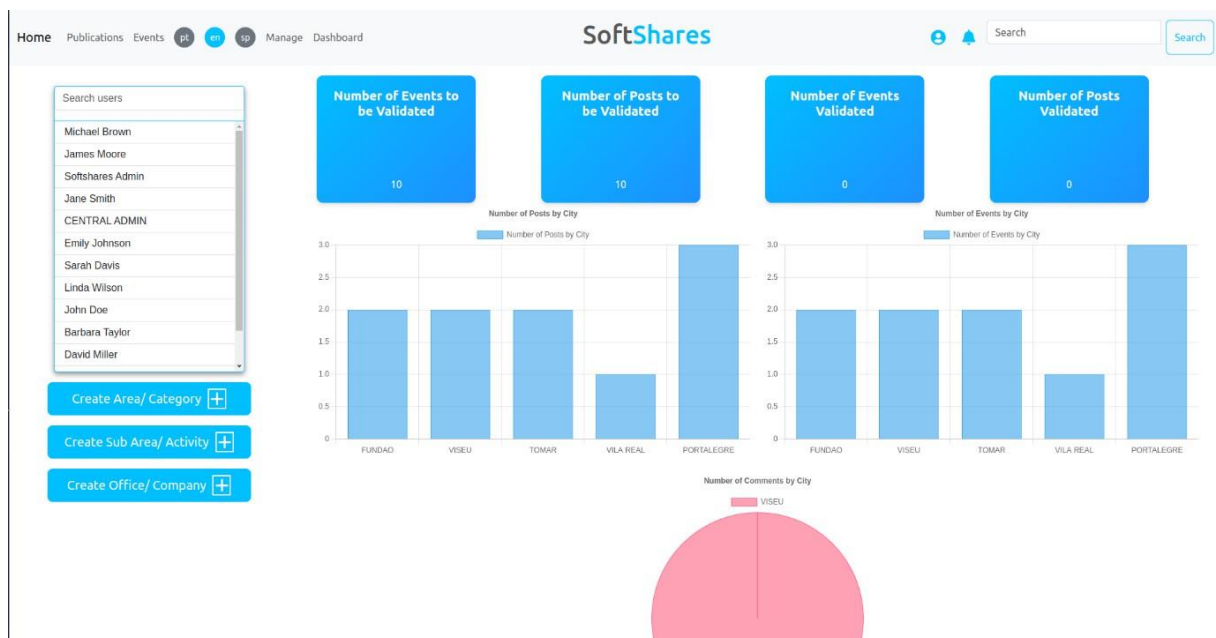
City

Create Operational Center

**Figura 6-11:** Criar centro

Esta página permite a criação de um novo Centro. O formulário inclui um campo para adicionar uma imagem representativa do Centro, ajudando a visualizá-lo facilmente. Há também um campo para especificar a cidade onde o Centro está localizado, assegurando que a sua localização geográfica está claramente identificada.

### 6.5.1.11 Dashboard



**Figura 6-12:** Dashboard

Na *Dashboard*, os administradores têm acesso a uma visão abrangente e detalhada das atividades e estatísticas da plataforma. É possível visualizar informações sobre o número de eventos e posts que foram validados e os que ainda estão pendentes de validação. Além disso, a *Dashboard* apresenta uma análise por cidade, mostrando o número de posts e eventos

associados a cada localidade, assim como a quantidade de comentários realizados em cada cidade, facilitando a compreensão da distribuição geográfica das interações.

Adicionalmente, há uma lista de utilizadores, onde se pode alterar o Centro ao qual cada utilizador está associado e ativar ou desativar a sua conta, oferecendo uma gestão flexível e eficiente dos membros da plataforma. A *Dashboard* também dispõe de três botões específicos para a criação de novas Áreas, Subáreas e Centros que são exclusivamente acessíveis ao **Server Admin**, garantindo que apenas utilizadores com permissões apropriadas podem realizar esta ação.

### 6.5.1.12 Perfil

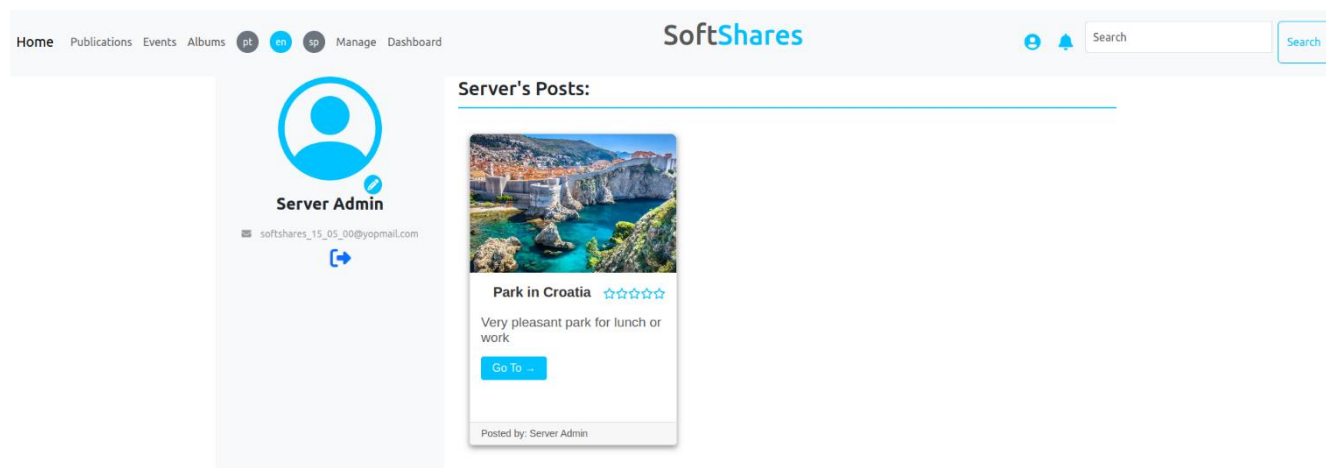


Figura 6-13: Página de Perfil do administrador

Na Página do Perfil, são exibidos o Nome e o E-mail do utilizador, proporcionando uma visão clara e direta das informações pessoais básicas. Além disso, todos os posts que o utilizador criou são listados nesta página, permitindo uma fácil revisão e gestão do conteúdo publicado. Cada Post é apresentado com detalhes como o título, a data de criação e um excerto da descrição. Caso o utilizador ainda não tenha criado nenhum Post, a página exibe uma mensagem representativa.

### 6.5.1.13 Detalhe do Evento

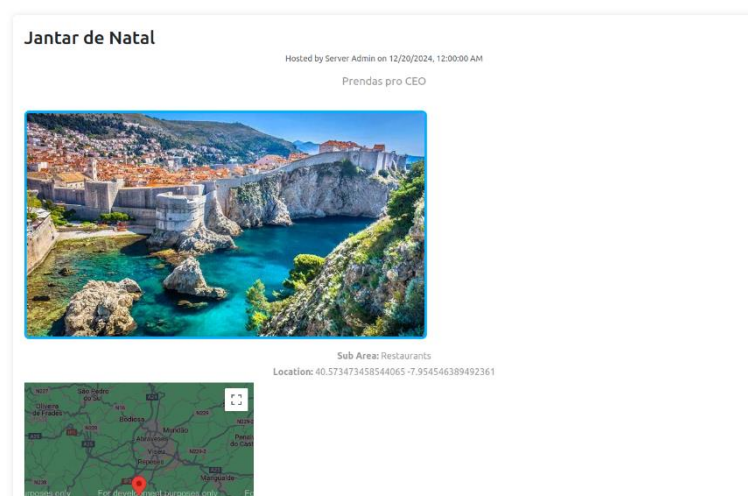


Figura 6-14: Detalhes Evento

Na página do evento obtém-se a informação de um determinado evento, tendo acesso ao conteúdo desse evento, e do fórum relacionado com tal.

6.5.1.14      Detalhe do Post

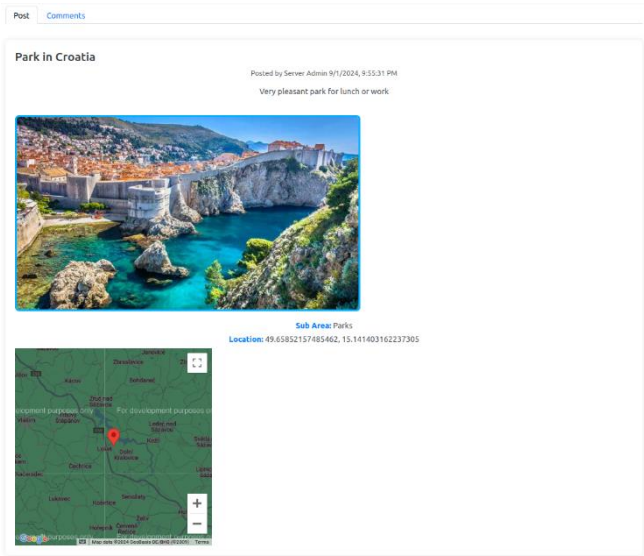


Figura 6-15: Detalhes Post

Na página de detalhes do post, o título do post é destacado no topo, seguido pelo nome do publicador e a data de publicação ou alteração. Abaixo dessas informações, encontra-se a descrição detalhada do post, oferecendo uma visão completa do conteúdo. É visível a imagem do post e a subárea à qual o post pertence. Por fim, a localização do post é exibida.

6.5.1.15      Centros

Operational Centers			
<div>+ Add Operational Center</div>			
City	Office Image	nins	Actions
Server Admin	No image	<ul style="list-style-type: none"><li>Guilherme Pedrinho</li><li>Jose Machado</li><li>Filipe Correia</li><li>Tio Patinhas</li><li>Server Admin</li></ul>	<div>EditDelete</div>
TOMAR		<ul style="list-style-type: none"><li>Jane Smith</li><li>Tomar Admin</li></ul>	<div>EditDelete</div>
UISEU		<ul style="list-style-type: none"><li>Emily Johnson</li><li>Viseu Admin</li><li>test Admin</li></ul>	<div>EditDelete</div>
FUNDAO		<ul style="list-style-type: none"><li>Michael Brown</li><li>Fundao Admin</li></ul>	<div>EditDelete</div>
PORTALEGRE		<ul style="list-style-type: none"><li>Sarah Davis</li><li>Portalegre Admin</li></ul>	<div>EditDelete</div>
VILA REAL		<ul style="list-style-type: none"><li>James Moore</li><li>VilaReal Admin</li></ul>	<div>EditDelete</div>

Figura 6-16: Centros

A página de "Lista de Centros" exibe todos os centros, mostrando a imagem do centro, a sua cidade e os administradores associados. Nessa página, é possível editar o nome e a imagem do centro ao clicar no botão de editar, permitindo manter as informações sempre atualizadas. Além disso, a página oferece a opção de eliminar um centro por meio do botão de eliminar, uma ação que remove permanentemente o centro selecionado do sistema. Essas funcionalidades tornam a gestão dos centros simples e eficiente.

### 6.5.1.16 Areas



Title	Icon	Actions
Health		<a href="#">Edit</a> <a href="#">Delete</a>
Education		<a href="#">Edit</a> <a href="#">Delete</a>
Sports		<a href="#">Edit</a> <a href="#">Delete</a>
Gastronomy		<a href="#">Edit</a> <a href="#">Delete</a>
Housing		<a href="#">Edit</a> <a href="#">Delete</a>
Leisure		<a href="#">Edit</a> <a href="#">Delete</a>
Transportation		<a href="#">Edit</a> <a href="#">Delete</a>
Area51		<a href="#">Edit</a> <a href="#">Delete</a>

Figura 6-17: Gestão das áreas

A página de "Lista de Áreas" exibe todas as áreas, apresentando as suas imagens. Nesta página, o administrador tem duas opções de edição: é possível alterar tanto a imagem quanto o nome de uma área, garantindo que as informações estejam sempre atualizadas e refletindo corretamente a identidade visual e o nome desejado para cada área. Ainda é possível eliminar uma área ao utilizar o botão de Eliminar

### 6.5.1.17 SubAreas

Area Title	SubArea Title	Actions
Health	Hospitals	<a href="#">Edit</a> <a href="#">Delete</a>
Health	Clinics	<a href="#">Edit</a> <a href="#">Delete</a>
Health	Pharmacies	<a href="#">Edit</a> <a href="#">Delete</a>
Education	Schools	<a href="#">Edit</a> <a href="#">Delete</a>
Education	Colleges	<a href="#">Edit</a> <a href="#">Delete</a>
Education	Libraries	<a href="#">Edit</a> <a href="#">Delete</a>
Sports	Gyms	<a href="#">Edit</a> <a href="#">Delete</a>
Sports	Stadiums	<a href="#">Edit</a> <a href="#">Delete</a>
Sports	Parks	<a href="#">Edit</a> <a href="#">Delete</a>
Gastronomy	Restaurants	<a href="#">Edit</a> <a href="#">Delete</a>
Gastronomy	Cafes	<a href="#">Edit</a> <a href="#">Delete</a>
Gastronomy	Bars	<a href="#">Edit</a> <a href="#">Delete</a>

Figura 6-18: Lista de SubAreas

A página de "Lista de SubÁreas" apresenta todas as subáreas, juntamente com suas respectivas áreas. Nessa página, o usuário dispõe de duas opções: editar, onde é possível



modificar o nome de uma subárea, e eliminar, permitindo a exclusão de uma subárea específica do sistema.

### 6.5.1.18 Forums

A página de "Forums" apresenta uma lista de todos os fóruns disponíveis, exibindo o título, uma pequena descrição, e o estado (ativo ou desativo) de cada fórum. Nessa página, o administrador tem à disposição dois botões: um para alterar o estado do fórum, permitindo ativá-lo ou desativá-lo conforme necessário, e outro para eliminar o fórum, removendo-o permanentemente do sistema.

### 6.5.1.19 Avisos

Warnings		
<a href="#">+ Add Warning</a>		
Warning	OfficeID	State
teste leve l 1	WISEU	Active <a href="#">Change State</a>
teste lvl3	WISEU	Active <a href="#">Change State</a>
teste lvl5	WISEU	Active <a href="#">Change State</a>

Figura 6-19: Lista de Avisos

A página de "Avisos" exibe uma lista de todos os avisos, juntamente com o estado atual de cada um (ativo ou desativado). Nessa página, há um botão que permite alterar o estado de um aviso, possibilitando a troca entre ativo e desativo conforme necessário.

### 6.5.1.20 Formulários Lista

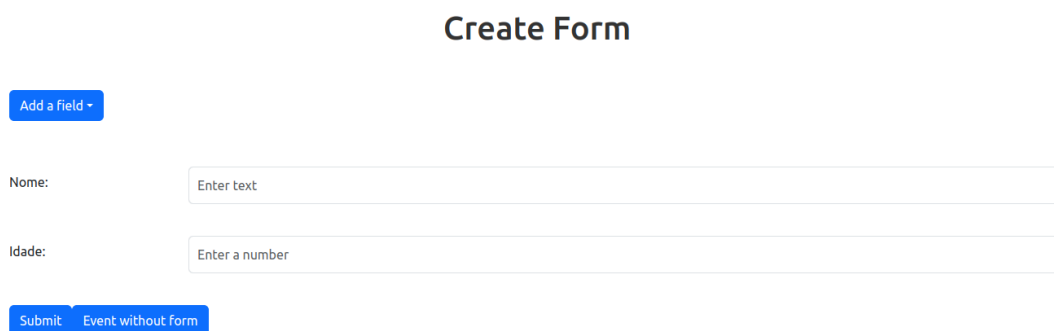
Forms	
Event	Actions
Test event 2	<a href="#">Edit Form</a> <a href="#">Check Answers</a>
Test event 2	<a href="#">Edit Form</a> <a href="#">Check Answers</a>
Football game	<a href="#">Edit Form</a> <a href="#">Check Answers</a>

Figura 6-20: Lista de Formulários

Nesta página são apresentados todos os formulários existentes, onde apenas é possível editar um formulário de um evento que ainda não foi validado. Ao clicar no botão "Check Answers", podemos visualizar as respostas dos utilizadores ao formulário.

---

### 6.5.1.21 Formulários Criar



Create Form

Add a field ▾

Nome:

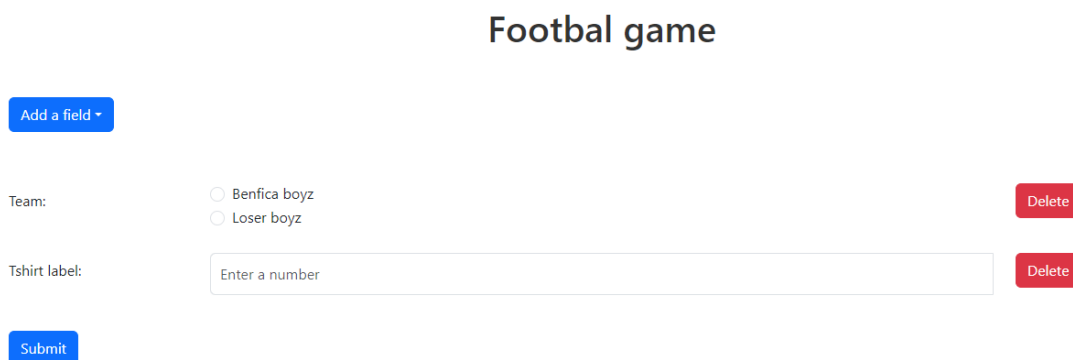
Idade:

Submit Event without form

**Figura 6-21:** Criar Formulário

Nesta página é possível criar um formulário para um evento. Ao clicar no *dropdown* “*Add a field*”, é possível selecionar entre diferentes tipos de campos (Campo de Texto, Campo Numérico, Grupo de Botões de Opção e Caixa de Seleção). Também é possível criar um evento sem formulário ao clicar no botão “*Event Without Form*”.

### 6.5.1.22 Editar Formulários



Football game

Add a field ▾

Team: ☐ Benfica boyz ☐ Loser boyz Delete

Tshirt label:  Delete

Submit

**Figura 6-22:** Editar Formulário

Nesta página é possível editar um formulário que ainda não foi validado. Existem botões de Eliminar para remover campos, e o *dropdown* “*Add a Field*” funciona da mesma forma que na página de criação de formulário.



---

### 6.5.1.24 Albums

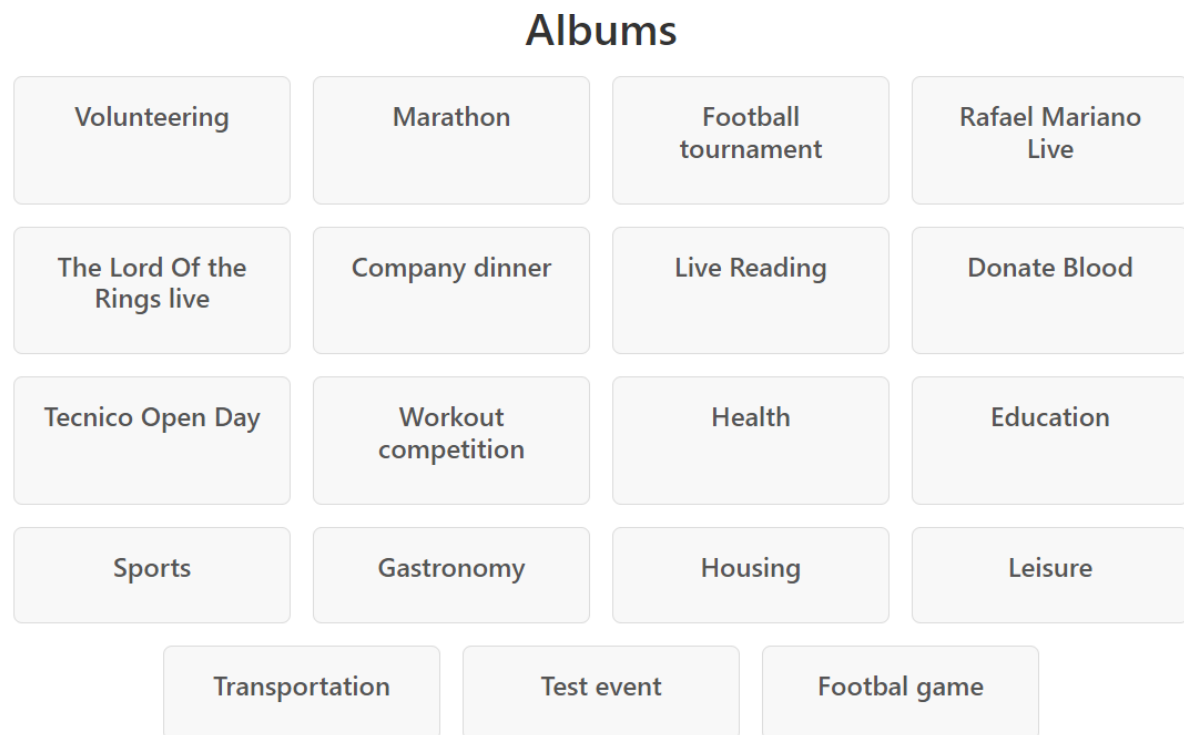


Figura 6-24: Albums

Nesta página é possível visualizar todos os álbuns disponíveis. Ao clicar em qualquer álbum, é possível explorar as fotografias que ele contém, visualizar os detalhes de cada imagem e, se necessário, adicionar novas fotografias ao álbum. Além disso, existe a opção de editar a ordem das fotos, remover aquelas que já não são relevantes e gerir os detalhes do álbum, como o título e a descrição. Este sistema facilita a organização e atualização contínua do conteúdo visual.

### 6.5.2. Componentes Reutilizáveis

O *frontend* do projeto foi desenvolvido com foco na reutilização de componentes, garantindo uma estrutura modular e eficiente. A seguir, apresento um resumo detalhado dos componentes criados e como foram reutilizados em diferentes páginas da aplicação.

### 6.5.3. Componentes Criados

#### 6.5.3.1 ButtonWithIcon.js

- **Descrição:** Um botão estilizado que inclui um ícone FontAwesome.
- **Utilização:** Amplamente utilizado em toda a aplicação para botões de ação que requerem ícones, melhorando a usabilidade e a estética das interfaces.

#### 6.5.3.2 Card.js

- **Descrição:** Um componente de cartão flexível que exibe uma imagem e um conteúdo textual.

- 
- **Utilização:** Utilizado em várias páginas, incluindo a apresentação de posts e eventos, proporcionando uma aparência consistente e organizada para diferentes tipos de conteúdo.

#### 6.5.3.3 ParentComponent.js

- **Descrição:** Um componente pai que administra a exibição de pop-ups para validação de itens.
- **Utilização:** Usado para encapsular lógica de estado e exibição condicional de componentes filhos, como `UserComponent` e `ValidateItemPopup`.

#### 6.5.3.4 BarChart.js

- **Descrição:** Um gráfico de barras construído com a biblioteca “react-chartjs-2”.
- **Utilização:** Implementado na *Dashboard* para exibir o número de posts e eventos por cidade, fornecendo uma visualização clara e intuitiva dos dados estatísticos.

#### 6.5.3.5 CategoryCard.js

- **Descrição:** Um cartão que lista várias categorias com ícones representativos.
- **Utilização:** Utilizado para navegar entre diferentes categorias de conteúdos, como desportos, gastronomia, lazer, etc., melhorando a navegação e acessibilidade da aplicação.

#### 6.5.3.6 PieChart.js

- **Descrição:** Um gráfico de pizza que exhibe dados em formato visual.
- **Utilização:** Implementado na *Dashboard* para mostrar a quantidade de comentários por cidade, facilitando a análise da distribuição geográfica das interações dos utilizadores.

#### 6.5.3.7 ShowEventCalendar.js

- **Descrição:** Um *modal* que exhibe eventos numa data selecionada.
- **Utilização:** Integrado no componente `Calendar` para mostrar detalhes dos eventos numa data específica quando o utilizador interage com o calendário.

#### 6.5.3.8 Navbar.js

- **Descrição:** Uma barra de navegação responsiva que inclui links para diferentes seções da aplicação e opções de mudança de idioma.
- **Utilização:** Usado em todas as páginas para fornecer navegação fácil e acesso rápido a diferentes funcionalidades da aplicação.

---

### 6.5.3.9 Calendar.js

- **Descrição:** Um calendário interativo que permite selecionar datas e ver eventos agendados.
- **Utilização:** Fornece uma interface intuitiva para visualizar e aceder eventos em datas específicas, com integração direta ao componente `ShowEventCalendar`.

### 6.5.3.10 PostCard.js

- **Descrição:** Um cartão específico para exibir posts e eventos, incluindo detalhes como título, descrição, imagem e opções de interação.
- **Utilização:** Reutilizado para listar tanto posts como eventos, ajustando-se dinamicamente para cada tipo de conteúdo, garantindo uma apresentação consistente e funcional.

## 6.5.4. Reutilização de Componentes

Os componentes acima foram projetados para serem altamente reutilizáveis, permitindo sua aplicação em diversas partes da interface de forma consistente. Por exemplo:

`ButtonWithIcon` é utilizado em múltiplos formulários e páginas de ação, como adicionar novos posts ou eventos.

`Card` e `PostCard` são reutilizados para exibir diferentes tipos de conteúdo, como posts, eventos e categorias, mantendo uma aparência unificada.

Gráficos (`BarChart` e `PieChart`) são implementados na *Dashboard* para fornecer uma visão clara das estatísticas relevantes.

`Navbar` e `Calendar` oferecem navegação e interação intuitivas, sendo componentes centrais na estrutura da aplicação.

Este enfoque na criação e reutilização de componentes assegura que o código é mais fácil de manter e expandir, enquanto proporciona uma experiência de utilizador coerente e agradável.

---

## 7. Desenvolvimento da API

A API do projeto foi desenvolvida utilizando Node.js e Express, sendo responsável por fornecer os dados e operações necessárias para as aplicações mobile e web. A seguir, detalhamos os principais componentes da API, incluindo a estrutura da base de dados, os controladores, as rotas, e as funcionalidades implementadas.

A pasta "models" no *backend* contém as definições dos modelos responsáveis pela criação das tabelas no *PostgreSQL*, utilizando o *Sequelize* como ORM (*Object-Relational Mapping*). O *Sequelize* simplifica a manipulação de dados, permitindo que as interações com a base de dados sejam realizadas por meio de objetos JavaScript, em vez de comandos SQL diretos. Contudo, devido ao uso de *schemas* para uma organização mais eficiente das tabelas e maior controlo das operações, optou-se pela utilização de consultas SQL diretas (*raw queries*) em diversas interações com a base de dados. A pasta "database" contém o ficheiro responsável pela configuração da conexão com a base de dados, incluindo os procedimentos armazenados, que são programados como funções e utilizados nos diversos controladores. Nesta pasta, também está presente a subpasta "triggers", onde são implementados os *triggers* que atuam automaticamente sobre a base de dados.

Os "controllers" são responsáveis pela lógica de interações com a API. Estes recebem os pedidos HTTP dos clientes, interagem com os modelos de dados e retornam as respostas apropriadas. Cada *controller* trata de uma parte específica do sistema, como a gestão de utilizadores ou de recursos específicos da aplicação. Por exemplo, o "authController" gere as operações relacionadas com a autenticação e registo dos utilizadores.

As "routes" definem os *endpoints* da API. Cada rota corresponde a um caminho específico no URL que direciona a o pedido HTTP para o controlador apropriado. Por exemplo, uma rota para */users* pode estar ligada ao "UserController" para gerir pedidos HTTP GET, POST, PUT, PATCH ou DELETE.

Os ficheiros presentes na pasta "start" são responsáveis pela configuração da inicialização da aplicação antes do início do servidor. Esses ficheiros executam operações como a limpeza da base de dados, a criação dos *schemas* e suas respectivas tabelas, além da inserção dos dados iniciais.

Por fim, a pasta "utils" contém funções utilitárias que apoiam o desenvolvimento da API, como funções de notificações, geração de *tokens* de autenticação (JWT – JSON Web Tokens), ou validações específicas que são usadas por várias partes da aplicação.

Esta estrutura modular e bem organizada assegura que a API seja fácil de manter e expandir, permitindo uma integração suave entre a aplicação mobile, a aplicação web e a base de dados.

### 7.1. Controladores (controllers)

Os controladores são responsáveis por lidar com a lógica de operações, processando os pedidos recebidos pelas rotas e interagindo com os modelos da base de dados.

---

### 7.1.1. authController

O `authController.js` é o controlador responsável por lidar com todas as funcionalidades de autenticação e gestão de contas de utilizadores. Este ficheiro contém várias funções dedicadas ao registo, login, recuperação de passwords, atualização de passwords e manipulação de *tokens* de autenticação, integrando-se com outros módulos e serviços da aplicação. A seguir, vamos abordar cada função importante do ficheiro:

#### 7.1.1.1 register

O controlador *register* trata do processo de registo de um novo utilizador na aplicação. O fluxo desta função é o seguinte:

- **Validação de Dados:** Através da função “`validateInput_register`”, é validado se o email, primeiro nome e último nome estão corretos.
- **Verificação de Email:** Utiliza a função “`sp_findUserByEmail`” para verificar se o email já está associado a uma conta existente.
- **Criação do Utilizador:** Se o email for único, chama “`spRegisterNewUser`” para adicionar o novo utilizador à base de dados.
- **Token para Definir Password:** Gera um *token* utilizando “`generateTokenAccountCreation_resetpassword`”, que será enviado ao utilizador por email.
- **Envio de Email:** Envia um email ao utilizador com um link para configurar a password, utilizando a função “`sendMail`”.

Este controlador assegura que o utilizador inicie o processo de criação da conta de forma segura, sem expor sua password diretamente no início.

#### 7.1.1.2 setupPassword

Este controlador é chamado quando o utilizador segue o link enviado por email para configurar a sua password.

- **Validação de Password:** Utiliza o “`validator`” para garantir que a nova password seja forte.
- **Criação da Password:** A password é então guardada na base de dados através da função “`spCreatePassword`”.
- **Confirmação:** O utilizador recebe um email de confirmação após a password ser definida com sucesso, e um log é feito para registar a ação.

Para processar as passwords é utilizada a biblioteca “`bcrypt`” para fazer o *hashing* das passwords e futuras validações das mesmas na autenticação.

#### 7.1.1.3 updatePassword

Este controlador permite que os administradores gerados no “`backoffice`” alterem as suas passwords no primeiro login.

- **Validação de Password:** Semelhante ao “`setupPassword`”, a nova password é validada quanto à sua segurança.
- **Alteração de Password:** Utiliza “`spChangeUserPassword`” para atualizar a password na base de dados.



- 
- **Reutilização de Password:** Se a password inserida já tiver sido usada anteriormente (detetada pela exceção “PasswordReuseError”), o utilizador recebe uma mensagem de erro.
  - **Envio de Email:** Após a password ser alterada, o utilizador recebe um email confirmando a mudança.

#### 7.1.1.4 startRecoveryPassword

Este controlador inicia o processo de recuperação de password quando o utilizador se esquece da mesma.

- **Verificação de Email:** Utiliza “findUserByEmail” para garantir que o email está registado.
- **Geração de Token:** Um *token* é gerado e enviado por email para que o utilizador possa redefinir a password com o mesmo.
- **Envio de Email:** Um email contendo o link de redefinição de password é enviado ao utilizador.

Desta forma o utilizador pode continuar o processo de recuperação na aplicação móvel ou utilizar a página web para o mesmo efeito.

#### 7.1.1.5 resetPassword

Esta função é usada quando o utilizador acede ao link enviado por email para redefinir sua password ou continua na aplicação móvel, tendo de inserir o *token* recebido por email.

- **Validação de Password:** Garante que a nova password seja segura.
- **Verificação de Token:** O *token* de redefinição de password é verificado utilizando “verifyToken”.
- **Alteração da Password:** Se o *token* for válido, a password é atualizada utilizando “spChangeUserPassword”, e um email de confirmação é enviado ao utilizador.

#### 7.1.1.6 login\_web e login\_mobile

Estes controladores tratam do *login* para as versões web e mobile da aplicação, respetivamente.

- **Validação de Credenciais:** Os dados do login são validados por “validateInput\_Login”.
- **Autenticação de Utilizador:** Utiliza “authenticateUser” para verificar se as credenciais estão corretas.
- **Tokens de Autenticação:** Se o utilizador for autenticado com sucesso, são gerados um *token* de acesso e um *refresh token*.
- **Restrições de Acesso:** Caso o utilizador tenha restrições ou o login seja inválido, o sistema retornará mensagens apropriadas.

#### 7.1.1.7 login\_SSO

Este controlador trata do login via *Single Sign-On* (SSO), utilizando Google ou Facebook como provedores (em conjunto com o Firebase).

- 
- **Verificação de Token:** O *token* do SSO é verificado utilizando as APIs do Google ou Facebook.
  - **Autenticação ou Criação de Conta:** Se o utilizador já tiver uma conta, ele é autenticado. Caso contrário, uma nova conta é criada.

#### 7.1.1.8 getUserByToken

Este controlador permite obter as informações de um utilizador a partir de um *token* de autenticação. Ele é utilizado para validar se o *token* de um utilizador é válido e retornar os detalhes do utilizador.

#### 7.1.1.9 refreshToken

Este controlador permite que o utilizador obtenha um novo *token* de acesso quando o seu *token* atual expirar, utilizando um *refresh token* previamente gerado.

#### 7.1.1.10 updateFcmToken

Este controlador atualiza o *token* FCM (*Firebase Cloud Messaging*) associado a um utilizador, permitindo o envio de notificações *push*.

O `authController.js` é essencial para gerir todo o ciclo de autenticação e segurança de utilizadores. Ele garante que os processos de login, registo e recuperação de password sejam feitos de forma segura e eficiente, utilizando validação de dados, criptografia de passwords, e integração com serviços de email para confirmar ações importantes. Cada função é projetada para lidar com um aspeto específico da gestão de contas e tokens de autenticação, garantindo uma experiência de utilizador segura e fluida.

### 7.1.2. adminController

O `adminController.js` é o controlador que gere as operações administrativas da aplicação, como validação de conteúdos, administração de utilizadores, criação de centros, gestão de alertas, e mais. Este controlador está focado em funções específicas para administradores e inclui a interação com a base de dados para a execução de várias tarefas. Abaixo está a análise detalhada de cada função:

#### 7.1.2.1 validate\_content

Este controlador é responsável por validar conteúdos como posts, eventos ou fóruns:

- **Validação de Inputs:** O controlador utiliza o *validator* para assegurar que o tipo de conteúdo (*contentType*), o ID do conteúdo (*contentID*), e o ID do administrador (*adminID*) são válidos.
- **Execução de Validação:** Uma vez que os parâmetros são verificados, o controlador chama o procedimento “`spValidateContent`” para validar o conteúdo na base de dados.

- 
- **Resposta:** Se o conteúdo for validado com sucesso, uma mensagem de sucesso é retornada. Caso contrário, é exibida uma mensagem de erro.

#### 7.1.2.2 reject\_content

Esta função rejeita conteúdos, como posts, eventos ou fóruns, que não cumprem os requisitos estabelecidos, ou outro qualquer motivo:

- **Validação de Inputs:** Verifica se os parâmetros fornecidos são válidos.
- **Rejeição de Conteúdo:** Utiliza “rejectContent” para processar a rejeição na base de dados.
- **Resposta:** Similar ao controlador anterior, uma resposta de sucesso ou erro é enviada de acordo com o resultado da operação.

#### 7.1.2.3 getUserEngagementMetrics

Esta função recupera métricas de “engagement” dos utilizadores, como atividades realizadas e participação em eventos:

- **Execução da Consulta:** Utiliza “getUserEngagementMetrics” para recuperar os dados de “engagement” da base de dados.
- **Resposta:** Os dados são retornados em formato JSON com uma mensagem de sucesso. Caso haja erro, uma mensagem explicativa é exibida.

#### 7.1.2.4 getContentValidationStatusByadmin

Obtém o status de validação de conteúdos, mas filtrados pelo administrador específico:

- **Validação de Inputs:** Verifica se o “adminID” é válido.
- **Consulta:** O controlador consulta “getContentValidationStatusByadmin” para obter os dados.
- **Resposta:** Retorna os dados em caso de sucesso, ou uma mensagem de erro caso haja falha.

#### 7.1.2.5 getContentValidationStatus

Esta função é similar à anterior, mas sem filtrar por administrador, retornando o status de todos os conteúdos:

- **Execução:** Utiliza “getContentValidationStatus” para obter os dados de validação de conteúdos.
- **Resposta:** Os resultados são retornados ou uma mensagem de erro é exibida.

#### 7.1.2.6 getActiveDiscussions

Obtém discussões ativas na plataforma:

- **Consulta:** A função recupera as discussões ativas através de “getActiveDiscussions”.
- **Resposta:** Retorna as discussões ou exibe uma mensagem de erro.

---

### 7.1.2.7 getActiveWarnings e getAllWarnings

Estas funções recuperam alertas ativos ou todos os alertas:

- **Consulta:** As consultas são realizadas através das funções “getActiveWarnings” ou “getAllWarnings”.
- **Resposta:** Os dados são retornados com sucesso ou uma mensagem de erro é exibida.

### 7.1.2.8 getContentCenterToBeValidated

Esta função recupera os conteúdos de um centro específico que precisam ser validados:

- **Validação de Inputs:** Verifica se o “center\_id” é válido.
- **Consulta:** A função usa “getContentCenterToBeValidated” para obter os dados.
- **Resposta:** Os resultados são retornados ou um erro é exibido.

### 7.1.2.9 createCenter

Cria um centro administrativo:

- **Validação de Inputs:** Verifica se os dados obrigatórios (cidade, ID do administrador e imagem do escritório) estão presentes e são válidos.
- **Criação do Centro:** Utiliza “createCenter” para inserir o centro na base de dados.
- **Resposta:** Uma mensagem de sucesso ou erro é enviada.

### 7.1.2.10 deleteCenter

Remove um centro existente:

- **Validação de Inputs:** Verifica o “center\_id” antes de executar a exclusão.
- **Remoção:** A função utiliza “deleteCenter” para excluir o centro.
- **Resposta:** Uma mensagem de sucesso ou erro é enviada.

### 7.1.2.11 getCenters

Recupera todos os centros disponíveis:

- **Execução:** Consulta “getCenters” para obter a lista de centros.
- **Resposta:** Retorna os centros ou uma mensagem de erro.

### 7.1.2.12 makeCenterAdmin

Designa um utilizador como administrador de um centro específico:

- **Execução:** Utiliza “spSetCenterAdmin” para definir um administrador de centro.
- **Resposta:** Retorna o resultado da operação ou um erro.

### 7.1.2.13 updateCenter

Atualiza os dados de um centro:

- **Validação de Inputs:** Verifica se os dados fornecidos são válidos.

- 
- **Atualização:** A função utiliza “updateCenter” para atualizar as informações do centro na base de dados.
  - **Resposta:** Uma mensagem de sucesso ou erro é exibida.

#### 7.1.2.14 validate\_user e deactivate\_user

Estas funções ativam ou desativam utilizadores na plataforma:

- **Validação de Inputs:** Verifica se o “user\_id” é válido.
- **Execução:** Utiliza “spActivateUser” ou “spDeactivateUser” para ativar ou desativar um utilizador.
- **Resposta:** Uma mensagem de sucesso ou erro é exibida.

#### 7.1.2.15 register\_admin

Regista um novo administrador:

- **Validação de Inputs:** Valida as entradas e verifica o papel do utilizador que faz a solicitação.
- **Registo:** Utiliza “spRegisterNewUser” para criar o administrador e definir as permissões apropriadas.
- **Resposta:** Uma mensagem de sucesso ou erro é enviada.

#### 7.1.2.16 getReports e deleteReport

Estas funções permitem a recuperação e exclusão de denúncias:

- **Execução:** Utilizam “getReports” para listar as denúncias e “deleteReport” para excluir uma denuncia específica.
- **Resposta:** Retornam os dados ou mensagens de sucesso/erro.

#### 7.1.2.17 createWarnings e updateWarning

Criam ou atualizam alertas (*warnings*) na plataforma:

- **Validação de Inputs:** Verificam se os dados de alerta fornecidos são válidos.
- **Execução:** Utilizam “createWarnings” ou “updateWarnings” para criar ou atualizar os alertas.
- **Resposta:** Retornam mensagens de sucesso ou erro.

O adminController.js é uma peça fundamental para a gestão administrativa da plataforma, fornecendo as ferramentas necessárias para validar conteúdos, gerir utilizadores, centros e alertas, e gerar alertas/avisos. Este faz uso extensivo de consultas à base de dados, garantindo que os administradores tenham controlo total sobre as operações que afetam a integridade e a organização da plataforma.

---

### 7.1.3. dashboardController

O dashboardController.js é um controlador simples, que gere a recuperação de dados para exibição em *dashboards* administrativos. Estas operações incluem informações sobre validação de conteúdos, posts, eventos, fóruns e comentários, agrupados por cidade. Todas as funções utilizam “procedimentos armazenados” para obter dados diretamente da base de dados.

Abaixo está uma análise detalhada de cada função presente no controlador:

#### 7.1.3.1 toValidate

Esta função recupera os conteúdos que ainda precisam ser validados.

- **Descrição:** A função utiliza o “procedimento armazenado” “sptoValidate” para obter todos os conteúdos pendentes de validação.
- **Resposta:** O resultado do procedimento é retornado como resposta em formato JSON.
- **Aplicação:** Esta função é útil para administradores, permitindo que visualizem rapidamente os conteúdos que necessitam de revisão antes de serem publicados.

#### 7.1.3.2 validated

Essa função recupera os conteúdos que já foram validados.

- **Descrição:** Utiliza o procedimento “spValidated” para listar todos os conteúdos que já passaram pelo processo de validação.
- **Resposta:** O resultado é enviado em formato JSON.
- **Aplicação:** Permite ao administrador ter uma visão dos conteúdos que já estão ativos na plataforma, proporcionando transparência sobre o que foi aprovado.

#### 7.1.3.3 postsbycity

Recupera posts por cidade, agrupando os posts com base no escritório.

- **Descrição:** A função chama o procedimento “sppostsbycity”, que retorna um agrupamento de posts associadas a diferentes cidades.
- **Resposta:** Os resultados são retornados como JSON.
- **Aplicação:** Esta função oferece uma visão geográfica dos posts, sendo útil para entender as interações e atividades dos colaboradores em diferentes localidades.

#### 7.1.3.4 eventsbycity

Recupera eventos realizados em cada cidade.

- 
- **Descrição:** A função utiliza “speventsbycity” para obter os eventos que foram criados, agrupados por cidade.
  - **Resposta:** O resultado da consulta é retornado como JSON.
  - **Aplicação:** Esta função é útil para monitorizar a distribuição de eventos nas diferentes localidades onde a empresa tem presença, ajudando a entender o *engagement* regional dos colaboradores.

#### 7.1.3.5 comments\_by\_city

Recupera comentários feitos por cidade.

- **Descrição:** A função “spcomments\_by\_city” é utilizada para obter os comentários feitos nos posts e fóruns, agrupados por cidade.
- **Resposta:** O resultado é enviado como JSON.
- **Aplicação:** Ajuda a mapear o envolvimento dos utilizadores em discussões e feedbacks nas várias cidades onde a empresa atua, proporcionando uma visão clara da interação em diferentes localidades.

O dashboardController.js é um controlador direto e funcional, focado na recuperação de dados específicos para fins de visualização e análise em *dashboards* administrativos. Todas as suas funções são baseadas em chamadas de “procedimentos armazenados”, que fornecem dados agregados sobre conteúdos (posts, eventos, comentários) e o estado de validação desses conteúdos. Estas funções são essenciais para dar aos administradores uma visão geral da atividade e do *engagement* dos colaboradores, filtrada por cidade, permitindo uma gestão mais informada.

#### 7.1.4. Static\_contentController

O Static\_contentController.js é responsável por gerir as operações relacionadas com as categorias e subcategorias na aplicação, incluindo a criação, leitura, atualização e exclusão (CRUD). Este também fornece funcionalidades para a recuperação de centros administrativos. As operações principais envolvem consultas SQL diretas utilizando “procedimentos armazenados” e *query builders* do *Sequelize*.

A seguir, analisamos cada função em detalhes:

##### 7.1.4.1 create\_category

Esta função cria uma categoria no sistema.

- **Entrada:** Recebe os dados da nova categoria, como “title” (título da categoria) e “icon” (ícone da categoria) do corpo do pedido HTTP.

- 
- **Execução:** A função chama o “procedimento armazenado” “spCreateCategory” para inserir os dados na base de dados.
  - **Resposta:** Se a criação for bem-sucedida, retorna uma mensagem de sucesso com o código de status 201. Caso ocorra um erro, retorna um status 500 com a mensagem de erro.

#### 7.1.4.2 create\_sub\_category

Cria uma subcategoria associada a uma área específica.

- **Entrada:** Recebe o “areaId” (ID da categoria principal) e o “title” (título da subcategoria) do corpo do pedido HTTP.
- **Execução:** Utiliza o “procedimento armazenado” “spCreateSubArea” para criar a subcategoria na base de dados.
- **Resposta:** Retorna uma mensagem de sucesso ou erro, dependendo do resultado da operação.

#### 7.1.4.3 get\_all\_areas

Recupera todas as categorias existentes no sistema.

- **Execução:** Faz uma consulta SQL direta para obter todas as áreas da tabela “static\_content”. “area”.
- **Resposta:** Retorna a lista de áreas em formato JSON, ou uma mensagem de erro em caso de falha.

#### 7.1.4.4 get\_all\_sub\_areas

Obtém todas as subcategorias, associando-as com as suas respectivas categorias.

- **Execução:** A função executa uma “*join query*” para obter subcategorias e suas categorias correspondentes.
- **Resposta:** Retorna uma lista das subcategorias e suas áreas associadas. Se ocorrer um erro, uma mensagem explicativa é exibida.

#### 7.1.4.5 update\_category

Atualiza as informações de uma categoria existente.

- **Entrada:** Recebe o “categoryID” como parâmetro do URL e os dados a serem atualizados (“title” e “icon”) do corpo do pedido HTTP.
- **Execução:** Utiliza uma consulta SQL para atualizar a categoria na base de dados, usando *COALESCE* para garantir que apenas os campos fornecidos sejam atualizados.



- 
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.4.6 delete\_category

Exclui uma categoria com base no “categoryID”.

- **Entrada:** O “categoryID” é passado como um parâmetro no URL.
- **Execução:** A consulta SQL DELETE remove a categoria correspondente na tabela "static\_content"."area".
- **Resposta:** Se a exclusão for bem-sucedida, retorna uma mensagem de sucesso; caso contrário, um erro é exibido.

#### 7.1.4.7 update\_sub\_category

Atualiza as informações de uma subcategoria específica.

- **Entrada:** O “subCategoryId” é passado como parâmetro do URL, e os dados a serem atualizados (“title”) são recebidos no corpo do pedido HTTP.
- **Execução:** Faz uma consulta SQL para atualizar a subcategoria na tabela "static\_content"."sub\_area".
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.4.8 delete\_sub\_category

Exclui uma subcategoria com base no “subCategoryId”.

- **Entrada:** O “subCategoryId” é passado como parâmetro do URL.
- **Execução:** A consulta SQL DELETE remove a subcategoria da tabela "static\_content"."sub\_area".
- **Resposta:** Se a exclusão for bem-sucedida, retorna uma mensagem de sucesso; caso contrário, um erro é exibido.

#### 7.1.4.9 getAllCenters

Recupera uma lista de todos os centros (escritórios) da aplicação.

- **Execução:** Faz uma consulta SQL para obter todos os escritórios na tabela "centers"."offices", ordenando-os por cidade.
- **Resposta:** A lista de escritórios é retornada em formato JSON, ou uma mensagem de erro se houver falha.

---

O `Static_contentController.js` desempenha um papel crucial na gestão de categorias e subcategorias, permitindo que administradores adicionem, atualizem e removam essas entidades conforme necessário. As consultas SQL e os “procedimentos armazenados” garantem que as operações sejam eficientes e seguras. Além disso, a consulta de centros é essencial para a gestão geográfica das áreas cobertas pela empresa.

### 7.1.5. `dynamic_contentController`

O `dynamic_contentController.js` é responsável por gerir o conteúdo dinâmico da aplicação, que inclui posts, fóruns e eventos. Este implementa operações CRUD e consultas para exibir e manipular conteúdo agrupado por cidade, data e tipo de conteúdo, além de fornecer detalhes de utilizadores e participação em eventos.

Aqui está uma análise detalhada de cada função:

#### 7.1.5.1 `getAllContent`

Essa função recupera todos os tipos de conteúdo dinâmico (posts, fóruns, eventos).

- **Execução:** Faz três consultas SQL para obter posts, fóruns e eventos, ordenando-os pela data de criação.
- **Resposta:** Retorna uma estrutura JSON com todas os posts, fóruns e eventos. Em caso de erro, uma mensagem explicativa é exibida.

#### 7.1.5.2 `getAllContentByCity`

Recupera posts, fóruns e eventos filtrados por cidade.

- **Entrada:** Recebe o “city\_id” como parâmetro no URL.
- **Validação:** Verifica se o “city\_id” é válido utilizando o *validator*.
- **Execução:** Realiza três consultas SQL que obtêm posts, fóruns e eventos para a cidade especificada, retornando apenas conteúdo validado.
- **Resposta:** Os resultados são retornados como JSON, ou uma mensagem de erro em caso de falha.

#### 7.1.5.3 `getPostsByCity`, `getForumsByCity`, `getEventsByCity`

Estas funções recuperam posts, fóruns, e eventos, respetivamente, filtrados por cidade.

- **Entrada:** Recebem o “city\_id” como parâmetro.
- **Validação:** Cada função valida o “city\_id” e, em seguida, executa a consulta SQL específica.

- 
- **Resposta:** Cada função retorna os dados correspondentes em formato JSON ou exibe uma mensagem de erro.

#### 7.1.5.4 `getPostById`, `getForumById`, `getEventById`

Estas funções recuperam um conteúdo específico por ID (post, fórum ou evento).

- **Entrada:** Recebem o “post\_id”, “forum\_id”, ou “event\_id” como parâmetro no URL.
- **Validação:** Cada ID é validado com o *validator* para garantir que seja um inteiro válido.
- **Execução:** As consultas SQL recuperam os dados detalhados de cada conteúdo, incluindo informações sobre o autor, administrador, avaliações e tópicos relacionados.
- **Resposta:** O conteúdo é retornado em JSON. Se o conteúdo não for encontrado, uma mensagem de erro é exibida.

#### 7.1.5.5 `getUserInfo`

Recupera informações detalhadas sobre um utilizador específico.

- **Entrada:** Recebe o “user\_id” como parâmetro no URL.
- **Validação:** O ID do utilizador é validado para garantir que seja um número inteiro válido.
- **Execução:** A consulta SQL retorna informações sobre o utilizador, incluindo nome, email, última data de acesso, e a cidade onde trabalha.
- **Resposta:** Os dados do utilizador são retornados em JSON ou uma mensagem de erro é exibida.

#### 7.1.5.6 `getUsers`

Recupera uma lista de todos os utilizadores da plataforma.

- **Execução:** Faz uma consulta SQL para listar utilizadores e associar os seus escritórios.
- **Resposta:** A lista de utilizadores é retornada em formato JSON.

#### 7.1.5.7 `updateUserOffice`

Atualiza o escritório de um utilizador.

- **Entrada:** Recebe o “user\_id” e “office\_id” no corpo do pedido HTTP.
- **Execução:** A função verifica se o utilizador e o escritório existem. Caso o utilizador já esteja associado a um escritório, a associação antiga é removida antes de criar uma nova.
- **Resposta:** A operação é concluída com sucesso, ou uma mensagem de erro é exibida.

---

#### 7.1.5.8 `getEventByDate`

Recupera eventos que ocorrem em uma data específica.

- **Entrada:** Recebe a date como parâmetro no URL, no formato ISO 8601 (YYYY-MM-DD).
- **Execução:** Obtém eventos que ocorreram na data fornecida, comparando o “event\_date” de forma precisa, e retorna apenas eventos validados.
- **Resposta:** Os eventos são retornados ou uma mensagem de erro é exibida.

#### 7.1.5.9 `getPosts`, `getForums`, `getEvents`

Estas funções obtêm listas de posts, fóruns, e eventos, ordenados pela data de criação.

- **Execução:** Cada função realiza uma consulta SQL para obter os dados relevantes, incluindo as pontuações de avaliações.
- **Resposta:** Os dados são retornados em JSON ou uma mensagem de erro é exibida.

O `dynamic_contentController.js` permite a gestão e recuperação eficiente de conteúdo dinâmico na plataforma, incluindo posts, fóruns, e eventos, filtrando por cidade, ID e data. Este também oferece funcionalidades para gerir utilizadores e seus escritórios, garantindo que os administradores possam visualizar e gerenciar dados de forma eficaz e rápida. As consultas SQL são otimizadas para obter detalhes completos e precisos, proporcionando uma visão detalhada de cada tipo de conteúdo e a interação dos utilizadores com esses dados.

#### 7.1.6. `postController`

O `postController.js` é responsável por gerir as operações relacionadas com os posts no sistema, incluindo criação, edição, exclusão e obtenção de informações sobre o estado e a pontuação (*score*) de um post. Este utiliza “procedimentos armazenados” para manipular os dados diretamente na base de dados.

Abaixo está uma análise detalhada de cada função:

##### 7.1.6.1 `create_post`

Esta função lida com a criação de um novo post.

- **Entrada:** Recebe os seguintes dados no corpo do pedido HTTP:
  - **subAreaId:** ID da subárea associada.
  - **officeId:** ID do escritório que o post pertence.

- 
- **publisher\_id:** ID do publicador do post.
  - **title:** Título do post.
  - **content, pLocation, filePath:** Conteúdo opcional, localização e caminho da imagem.
  - **type:** Tipo do post (default: "N" - normal, ou "P" – Ponto de Interesse).
  - **rating, price:** Avaliação e preço, se aplicável.
  - **Execução:** Utiliza "spCreatePost" para criar o post na base de dados.
  - **Resposta:** Se o post for criada com sucesso, retorna uma mensagem de sucesso. Caso contrário, retorna um erro 500 com detalhes da falha.

#### 7.1.6.2 edit\_post

Edita um post existente.

- **Entrada:** O "postId" é passado como parâmetro no URL, e o corpo do pedido contém os dados a serem editados, incluindo:
  - **subAreaId, officeId, adminId:** IDs da subárea, escritório e administrador responsáveis pelo post.
  - **title, content, pLocation, filePath, type, price, rating:** Dados opcionais para edição.
- **Execução:** A função chama a "spEditPost" para atualizar o post com os dados fornecidos.
- **Resposta:** Se a operação for bem-sucedida, retorna uma mensagem de sucesso. Caso contrário, retorna uma mensagem de erro.

#### 7.1.6.3 get\_post\_state

Obtém o estado atual de um post.

- **Entrada:** O "postId" é passado como parâmetro no URL.
- **Execução:** Utiliza a função "fnGetPostState" para obter o estado do post (ex: pendente, aprovado, rejeitado).
- **Resposta:** Retorna o estado do post, ou uma mensagem de erro caso não consiga obter as informações.

#### 7.1.6.4 delete\_post

Exclui um post com base no seu "postId".

- **Entrada:** O "postId" é passado como parâmetro no URL.

- 
- **Execução:** Utiliza “spDeletePost” para excluir o post da base de dados.
  - **Resposta:** Se a operação for bem-sucedida, retorna uma mensagem de sucesso. Caso contrário, retorna uma mensagem de erro.

#### 7.1.6.5 getPostScoreByID

Obtém a pontuação de um post específico.

- **Entrada:** O “post\_id” é passado como parâmetro no URL.
- **Execução:** Utiliza uma consulta SQL para obter a pontuação do post na tabela “dynamic\_content”. “scores”.
- **Resposta:** Retorna a pontuação do post, ou uma mensagem de erro caso o post não seja encontrado.

O postController.js é essencial para a gestão de posts dentro da plataforma, fornecendo funções para criar, editar e excluir posts, além de obter informações detalhadas sobre o estado e a pontuação. Este utiliza “procedimentos armazenados” para garantir que as operações sejam feitas de maneira eficiente e segura na base de dados, assegurando consistência nos dados e performance.

#### 7.1.7. forumController

O forumController.js é responsável por gerir as operações relacionadas com fóruns no sistema. As funcionalidades incluem criar, editar, excluir, alterar o estado de um fórum, e recuperar seu estado atual. Este utiliza “procedimentos armazenados” para garantir que as operações sejam realizadas diretamente na base de dados de forma eficiente.

Abaixo está uma análise detalhada de cada função:

##### 7.1.7.1 create\_forum

Esta função cria um fórum na plataforma.

- **Entrada:** Recebe os seguintes parâmetros no corpo do pedido HTTP:
  - **officeID:** ID do escritório.
  - **subAreaId:** ID da subárea associada ao fórum.
  - **title:** Título do fórum.
  - **publisher id:** ID do publicador do fórum.
  - **description:** (Opcional) Descrição do fórum.
- **Execução:** A função utiliza a “spCreateForum” para criar o fórum na base de dados.

- 
- **Resposta:** Retorna uma mensagem de sucesso se o fórum for criado com êxito. Em caso de erro, retorna uma mensagem de erro com o detalhe da falha.

#### 7.1.7.2 get\_forum\_state

Obtém o estado atual de um fórum específico.

- **Entrada:** O “forumId” é passado como parâmetro no URL.
- **Execução:** Utiliza a função “fnGetForumState” para obter o estado atual do fórum (ex.: pendente, aprovado, rejeitado).
- **Resposta:** Retorna o estado do fórum. Caso haja erro, é retornada uma mensagem de falha.

#### 7.1.7.3 edit\_forum

Edita um fórum existente.

- **Entrada:** O “forumId” é passado como parâmetro no URL. O corpo do pedido HTTP contém os dados a serem editados:
  - **subAreaId:** (Opcional) ID da subárea.
  - **officeId:** (Opcional) ID do escritório.
  - **adminId:** (Opcional) ID do administrador.
  - **title:** (Opcional) Título do fórum.
  - **content:** (Opcional) Conteúdo do fórum.
  - **eventId:** (Opcional) ID do evento associado ao fórum.

**Execução:** Utiliza “spEditForum” para atualizar as informações do fórum na base de dados.

**Resposta:** Se a edição for bem-sucedida, retorna uma mensagem de sucesso. Caso contrário, exibe uma mensagem de erro.

#### 7.1.7.4 change\_forum\_state

Altera o estado de um fórum, por exemplo, de ativo para inativo.

- **Entrada:** O “forumId” é passado como parâmetro no URL, e o novo *state* (estado) é enviado no corpo do pedido HTTP.
- **Execução:** A função chama “spChangeForumState” para atualizar o estado do fórum na base de dados.
- **Resposta:** Se a alteração for bem-sucedida, retorna uma mensagem de sucesso. Caso ocorra um erro, a função retorna uma mensagem de falha.

---

### 7.1.7.5 delete\_forum

Exclui um fórum específico da base de dados.

- **Entrada:** O “forumId” é passado como parâmetro no URL.
- **Execução:** Utiliza “spDeleteForum” para excluir o fórum da base de dados.
- **Resposta:** Se a operação for bem-sucedida, retorna uma mensagem de sucesso. Caso contrário, retorna uma mensagem de erro.

O forumController.js desempenha um papel fundamental na gestão de fóruns na plataforma, permitindo a criação, edição, exclusão e alteração de estado de fóruns. As operações são realizadas de forma eficiente com o uso de “procedimentos armazenados”, garantindo que as mudanças sejam refletidas na base de dados de maneira consistente.

### 7.1.8. eventController

O eventController.js é responsável por gerir as operações relacionadas com eventos na plataforma. As funcionalidades incluem a criação, edição, exclusão, registo e remoção do registo de utilizadores para eventos, além da obtenção de informações sobre os participantes e o estado de um evento. Este utiliza “procedimentos armazenados” e *queries* para garantir operações eficientes e seguras na base de dados.

Abaixo está uma análise detalhada de cada função:

#### 7.1.8.1 create\_event

Cria um evento na plataforma.

- **Entrada:** Recebe os seguintes parâmetros no corpo do pedido:
  - officeId, subAreaId, name, description, eventDate, startTime, endTime, recurring, recurring\_pattern, max\_participants, location, publisher\_id, filePath.
- **Validação:** Verifica se o “publisher\_id”, “officeId” e “subAreaId” são inteiros válidos.
- **Execução:** Chama “spCreateEvent” para criar o evento, o fórum relacionado ao evento em conjunto com a inicialização do controlo de participantes para o evento e ativa o *trigger* que vai criar um álbum para esse evento.
- **Resposta:** Se o evento for criado com sucesso, retorna o “eventId” e uma mensagem de sucesso. Caso contrário, retorna uma mensagem de erro.

#### 7.1.8.2 register\_user\_for\_event

Regista um utilizador para participar no evento.



- 
- **Entrada:** Recebe o “userId” e “eventId” como parâmetros no URL.
  - **Execução:** Chama “spRegisterUserForEvent” para registrar o utilizador no evento e envia uma notificação ao criador do evento sobre o novo registo.
  - **Resposta:** Retorna uma mensagem de sucesso. Caso ocorra um erro, retorna uma mensagem com o detalhe da falha.

#### 7.1.8.3 unregister\_user\_from\_event

Remove um utilizador de um evento.

- **Entrada:** Recebe o “userId” e “eventId” como parâmetros no URL.
- **Execução:** Chama “spUnregisterUserFromEvent” para remover o utilizador do evento e envia uma notificação ao criador sobre o sucedido.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.8.4 get\_event\_state

Obtém o estado de um evento.

- **Entrada:** Recebe o “eventId” como parâmetro no URL.
- **Execução:** Utiliza a função “fnGetEventState” para obter o estado atual do evento (ex.: pendente, aprovado, cancelado).
- **Resposta:** Retorna o estado do evento ou uma mensagem de erro.

#### 7.1.8.5 edit\_event

Edita os detalhes de um evento existente.

- **Entrada:** O “eventId” é passado como parâmetro no URL, e os dados do evento a serem editados são fornecidos no corpo do pedido HTTP (como subAreaId, officeId, name, description, eventDate, etc.).
- **Execução:** Utiliza “spEditEvent” para atualizar os detalhes do evento na base de dados. Notifica os participantes sobre as alterações no evento.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.8.6 get\_participants

Obtém os participantes registados para um evento.

- **Entrada:** Recebe o “eventId” como parâmetro no URL.
- **Execução:** Chama “spGetParticipants” para obter a lista de participantes.

- 
- **Resposta:** Retorna a lista de participantes ou uma mensagem de erro.

#### 7.1.8.7 get\_participants\_adm

Obtém os participantes de um evento para fins administrativos.

- **Entrada:** Recebe o “eventId” como parâmetro no URL.
- **Execução:** Chama “spGetParticipants\_adm” para obter informações detalhadas dos participantes.
- **Resposta:** Retorna a lista de participantes para fins administrativos ou uma mensagem de erro.

#### 7.1.8.8 getEventScoreByID

Obtém a pontuação de um evento específico.

- **Entrada:** O “event\_id” é passado como parâmetro no URL.
- **Execução:** Utiliza uma *query* para obter a pontuação do evento na tabela “dynamic\_content”. “scores”.
- **Resposta:** Retorna a pontuação do evento ou uma mensagem de erro caso o evento não seja encontrado.

O eventController.js desempenha um papel essencial na gestão de eventos, permitindo que utilizadores e administradores criem, editem, e gerenciem eventos de forma eficiente. Além disso, este automatiza a comunicação com os participantes através de notificações em tempo real, garantindo que o criador esteja informado sobre registos, novos comentários e todos os participantes de alterações nos eventos. Os “procedimentos armazenados” garantem que as operações sejam seguras e bem organizadas na base de dados.

### 7.1.9. formsController

O formsController.js é responsável por gerir a criação, edição, exclusão e obtenção de formulários associados a eventos, além de permitir que os utilizadores respondam a esses formulários. Este também lida com notificações aos participantes do evento sempre que o formulário é alterado. Abaixo, você encontra uma explicação detalhada de cada função no controlador.

#### 7.1.9.1 create\_event\_form

Cria um formulário para um evento específico.

- 
- **Entrada:** Recebe “eventID” e “customFieldsJson” no corpo do pedido HTTP.
  - **Execução:** Chama “createEventForm” para criar um formulário associado ao evento.
  - **Resposta:** Retorna uma mensagem de sucesso ou erro dependendo do resultado.

#### 7.1.9.2 add\_fields\_event\_form

Adiciona novos campos ao formulário de um evento antes de este ser aprovado.

- **Entrada:** Recebe “eventID” e “customFieldsJson” no corpo do pedido HTTP.
- **Execução:** Utiliza “addCustomFieldsToEventForm” para adicionar campos ao formulário. Em seguida, notifica os participantes do evento sobre a alteração no formulário.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.9.3 edit\_fields\_event\_form

Edita campos já existentes de um formulário de evento.

- **Entrada:** O “eventID” é passado como parâmetro no URL, e os novos campos são enviados no corpo do pedido HTTP.
- **Execução:** Utiliza “editEventFormField” para editar os campos do formulário. Notifica os participantes sobre as alterações no formulário.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.9.4 get\_event\_form

Obtém o esquema bruto do formulário de um evento.

- **Entrada:** Recebe o “eventID” como parâmetro no URL.
- **Execução:** Chama “getFormSchema” para obter o esquema do formulário.
- **Resposta:** Retorna o esquema do formulário ou uma mensagem de erro.

#### 7.1.9.5 get\_event\_json\_form

Obtém o formulário do evento em formato JSON.

- **Entrada:** O “eventID” é passado como parâmetro no URL.
- **Execução:** Utiliza “getFormSchemaAsJson” para converter o formulário em JSON.
- **Resposta:** Retorna o formulário em formato JSON ou uma mensagem de erro.

---

### 7.1.9.6 add\_answers

Adiciona as respostas do utilizador ao formulário de um evento e regista o utilizador no evento.

- **Entrada:** Recebe o “userID” e “eventID” como parâmetros no URL, e as respostas do formulário (answersJson) no corpo do pedido HTTP.
- **Execução:** Insere as respostas no formulário com “insertFormAnswers” e regista o utilizador no evento usando “spRegisterUserForEvent”. Notifica o criador do evento sobre o novo participante.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

### 7.1.9.7 delete\_field\_from\_form

Remove um campo específico do formulário de um evento.

- **Entrada:** Recebe o “eventID” e “fieldID” como parâmetros no URL.
- **Execução:** Chama “deleteEventFormField” para excluir o campo e notifica os participantes do evento sobre a alteração no formulário.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

### 7.1.9.8 get\_event\_answers

Obtém todas as respostas fornecidas para o formulário de um evento específico.

- **Entrada:** O “eventID” é passado como parâmetro no URL.
- **Execução:** Chama “getFormAnswersByEvent” para obter todas as respostas do formulário de um evento.
- **Resposta:** Retorna as respostas ou uma mensagem de erro.

### 7.1.9.9 get\_event\_answers\_for\_user

Obtém as respostas fornecidas por um utilizador específico para o formulário de um evento.

- **Entrada:** O “eventID” é passado como parâmetro no URL, e o “userID” é extraído do token JWT.
- **Execução:** Chama “getFormAnswersByEventAndUser” para obter as respostas do utilizador para o formulário de um evento.
- **Resposta:** Retorna as respostas ou uma mensagem de erro.

---

### 7.1.9.10 get\_event\_answers\_for\_users

Obtém as respostas fornecidas por um utilizador específico para o formulário de um evento, com o “userID” explicitamente passado no URL.

- **Entrada:** Recebe o “eventID” e “userID” como parâmetros no URL.
- **Execução:** Utiliza “getFormAnswersByEventAndUser” para obter as respostas do utilizador.
- **Resposta:** Retorna as respostas ou uma mensagem de erro.

O formsController.js facilita a criação e manipulação de formulários associados a eventos, permitindo que administradores personalizem os formulários e que os utilizadores enviem suas respostas. Além disso, o controlador automatiza o envio de notificações aos participantes sempre que um formulário é alterado, mantendo-os atualizados sobre mudanças.

### 7.1.10. commentsController

O commentsController.js é responsável por gerir as operações relacionadas aos comentários em posts e fóruns. As funcionalidades incluem a criação, exclusão, *likes*, denúncias de comentários e obtenção da árvore de comentários. Este controlador utiliza “procedimentos armazenados” para aceder e modificar os dados relacionados a comentários de forma eficiente e segura na base de dados.

Abaixo está uma análise detalhada de cada função:

#### 7.1.10.1 add\_comment

Adiciona um novo comentário a um conteúdo (Post ou Fórum). Esta funcionalidade está implementada com a possibilidade de comentários hierárquicos.

- **Entrada:** Recebe os seguintes parâmetros no corpo do pedido HTTP:
  - **parentCommentID:** (Opcional) ID do comentário pai (para respostas).
  - **contentID:** ID do conteúdo (Post ou Fórum) ao qual o comentário será adicionado.
  - **contentType:** Tipo do conteúdo (Post ou Fórum).
  - **commentText:** Texto do comentário.
- **Validação:** Valida os IDs de comentário e conteúdo, além de verificar se o texto do comentário não está vazio.
- **Execução:** Chama “spAddComment” para adicionar o comentário. Envia notificações ao criador do conteúdo ou participantes do evento (se aplicável).

- 
- **Resposta:** Retorna uma mensagem de sucesso ou erro, dependendo do resultado da operação.

#### 7.1.10.2 get\_comments\_tree

Obtém a árvore de comentários de um Post ou Fórum.

- **Entrada:** Recebe o “contentID” e “contentType” como parâmetros no URL.
- **Validação:** Verifica se os IDs e o tipo de conteúdo são válidos.
- **Execução:** Chama a função “getCommentTree” para recuperar a hierarquia de comentários.
- **Resposta:** Retorna a árvore de comentários ou uma mensagem de erro.

#### 7.1.10.3 like\_comment

Adiciona um *like* a um comentário.

- **Entrada:** Recebe o “commentID” no corpo do pedido HTTP.
- **Execução:** Chama “likeComment” para registrar o *like* na base de dados. Envia uma notificação ao autor do comentário. Ativa um *trigger* que atualiza o número total de likes no comentário.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.10.4 unlike\_comment

Remove um *like* de um comentário.

- **Entrada:** Recebe o “commentID” no corpo do pedido HTTP.
- **Execução:** Chama “unlikeComment” para remover o *like*.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.10.5 report\_comment

Denuncia um comentário por comportamento inadequado ou outro qualquer motivo que um utilizador ache pertinente.

- **Entrada:** Recebe o “commentID”, “reporterID” (ID de quem está denunciando) e “observation” (observação sobre a denúncia) no corpo do pedido HTTP.
- **Execução:** Utiliza “reportComment” para registrar a denúncia.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

---

#### 7.1.10.6 likes\_per\_content

Obtém a lista de comentários de um conteúdo, juntamente com os *likes*.

- **Entrada:** Recebe o “contentID” e “contentType” como parâmetros no URL.
- **Execução:** Recupera a árvore de comentários com “getCommentTree\_forlikes” e usa “likes\_per\_content”.
- **Resposta:** Retorna os comentários com o estado dos likes ou uma mensagem de erro.

#### 7.1.10.7 delete\_comment

Exclui um comentário específico.

- **Entrada:** Recebe o “commentID” como parâmetro no URL.
- **Execução:** Chama “deleteComment” para remover o comentário da base de dados.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.10.8 likes\_per\_user

Obtém os comentários “liked” por um utilizador específico.

- **Entrada:** Recebe o “userID” como parâmetro no URL.
- **Validação:** Verifica se o “userID” é válido.
- **Execução:** Chama “likes\_per\_user” para recuperar os comentários “liked” pelo utilizador.
- **Resposta:** Retorna os comentários com gostos ou uma mensagem de erro.

O commentsController.js oferece funcionalidades essenciais para a gestão de comentários em posts e fóruns, incluindo a criação, moderação (gostos e denúncias), e recuperação de árvores de comentários. Este também lida com notificações automáticas para os criadores do conteúdo e participantes de eventos, mantendo a comunicação em tempo real dentro da plataforma. O uso de “procedimentos armazenados” garante operações rápidas e consistentes na base de dados.

#### 7.1.11. ratingController

O ratingController.js é responsável por permitir que os utilizadores avaliem conteúdos como Post e Event. A função central é a adição de uma avaliação, que é então base de dados.

Abaixo está uma análise detalhada da função:

---

#### 7.1.11.1 add\_evaluation

Adiciona uma avaliação a um conteúdo (Post ou Evento).

- **Parâmetros no URL:**
  - **contentType:** Tipo de conteúdo (pode ser "Post" ou "Event").
  - **contentId:** ID do conteúdo a ser avaliado.
- **Corpo do pedido HTTP:**
  - **evaluation:** Valor da avaliação (um número representando a pontuação de 0 a 5).
- **Autenticação:** O user\_id é extraído do token JWT do utilizador autenticado.
- **Execução:** Chama “spInsertEvaluation” para registrar a avaliação na base de dados, associando o conteúdo (Post ou Event) com o utilizador e a avaliação fornecida.
- **Resposta:** Retorna uma mensagem de sucesso se a avaliação for adicionada com sucesso. Em caso de erro, retorna uma mensagem detalhada do erro.

O ratingController.js oferece uma funcionalidade direta para avaliar conteúdos na plataforma. A avaliação é associada ao conteúdo e ao utilizador que a realizou, e é inserida na base de dados por meio de um “procedimento armazenado”. A parte para atualização da pontuação média do conteúdo é efetuada através de um *trigger*.

#### 7.1.12. mediaController

O mediaController.js gere operações relacionadas a álbuns de fotografias associados a eventos e áreas na plataforma. Este permite a criação de álbuns, adição de fotos a esses álbuns, e recuperação de álbuns e fotos. Abaixo está uma análise detalhada de cada função no controlador.

##### 7.1.12.1 create\_album

Cria um álbum de fotos associado a um evento ou subárea.

- **Entrada:** Recebe os seguintes parâmetros no corpo do pedido HTTP:
  - **eventId:** ID do evento (opcional).
  - **subAreaId:** ID da subárea (opcional).
  - **title:** Título do álbum.
- **Execução:** Chama “spCreateAlbum” para criar o álbum.
- **Resposta:** Retorna uma mensagem de sucesso ou erro, dependendo do resultado.



---

### 7.1.12.2 add\_photograph\_area\_album

Adiciona uma foto a um álbum associado a uma área específica.

- **Parâmetros no URL:**
  - **area\_id:** ID da área.
  - **publisherId:** ID do utilizador que publica a foto.
- **Corpo do pedido HTTP:**
  - **filePath:** Caminho do arquivo da foto.
- **Execução:** Obtém o “albumId” da área com “getAlbumOfAreaID”, e depois usa “spAddPhotograph” para adicionar a foto ao álbum.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

### 7.1.12.3 add\_photograph

Adiciona uma foto a um álbum específico.

#### **Entrada:**

- **Parâmetros no URL:**
  - **albumId:** ID do álbum.
  - **publisherId:** ID do utilizador que publicar a foto.
- **Corpo do pedido HTTP:**
  - **filePath:** Caminho do ficheiro da fotografia.
- **Execução:** Usa “spAddPhotograph” para adicionar a foto ao álbum.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

### 7.1.12.4 add\_photograph\_event

Adiciona uma foto a um álbum de evento.

- **Parâmetros no URL:**
  - **eventID:** ID do evento.
  - **publisherId:** ID do utilizador que publica a foto.
- **Corpo do pedido HTTP:**
  - **filePath:** Caminho do arquivo da foto.
- **Execução:** Obtém o “albumId” do evento com “getAlbumIdByEventId” e usa “spAddPhotograph” para adicionar a foto ao álbum.

- 
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.12.5 get\_albums

Obtém todos os álbuns.

- **Execução:** Chama “spGetAlbums” para obter todos os álbuns da plataforma.
- **Resposta:** Retorna a lista de álbuns ou uma mensagem de erro.

#### 7.1.12.6 get\_album\_photo

Obtém uma foto específica de um álbum.

Entrada:

- **Parâmetro no URL:**
  - **photo\_id:** ID da foto.
- **Execução:** Chama “spGetAlbumPhoto” para obter a foto associada ao “photo\_id”.
- **Resposta:** Retorna a foto ou uma mensagem de erro.

#### 7.1.12.7 get\_event\_photos

Obtém todas as fotos de um evento específico.

- **Entrada:** Recebe o “eventID” como parâmetro no URL.
- **Execução:** Usa “getAlbumIdByEventId” para obter o “albumID” do evento, e depois obtém as fotos com “getPhotosByAlbumId”.
- **Resposta:** Retorna as fotos ou uma mensagem de erro.

#### 7.1.12.8 get\_area\_photos

Obtém todas as fotos de uma área específica.

- **Entrada:** Recebe o “area\_id” como parâmetro no URL.
- **Execução:** Usa “getAlbumOfAreaID” para obter o “albumID” da área e obtém as fotos com “getPhotosByAlbumId”.
- **Resposta:** Retorna as fotos ou uma mensagem de erro.

#### 7.1.12.9 get\_albums\_of\_areas

Obtém IDs dos álbuns associados a áreas que não possuem null no campo areaId.

- 
- **Execução:** Usa “getAlbumsWithNonNullAreaId” para obter os álbuns.
  - **Resposta:** Retorna os IDs dos álbuns ou uma mensagem de erro.

#### 7.1.12.10 get\_photos\_of\_areas\_albums

Obtém as fotos de um álbum específico de uma área.

- **Entrada:** Recebe o “albumID” como parâmetro no URL.
- **Execução:** Usa “getPhotosByAlbumId” para obter as fotos do álbum.
- **Resposta:** Retorna as fotos ou uma mensagem de erro.

O mediaController.js oferece funcionalidades essenciais para a gestão de *media* na plataforma, permitindo a criação de álbuns, adição de fotos e recuperação de fotos e álbuns associados a eventos e áreas. O controlador lida com múltiplos cenários, como adicionar fotos a eventos ou áreas, e garantir que as operações de *media* sejam realizadas de forma eficiente e segura.

#### 7.1.13. uploadController

O uploadController.js é responsável por gerir o *upload* de ficheiros na aplicação, utilizando a biblioteca *Multer* para manipular *multipart/form-data*. Este define as configurações de armazenamento e nomeação dos ficheiros e lida com a resposta após o upload.

#### 7.1.14. userController

O usersController.js gere as funcionalidades relacionadas aos utilizadores da plataforma, incluindo preferências, atualizações de perfil, eventos registados, e gerenciamento de conteúdo e marcações. A seguir está uma análise detalhada das funções presentes neste controlador.

##### 7.1.14.1 get\_user\_preferences

Obtém as preferências do utilizador com sessão iniciada.

- **Entrada:** Nenhum parâmetro adicional, pois o “user\_id” é extraído do token JWT.
- **Execução:** Chama “getUserPreferences” para obter as preferências do utilizador.
- **Resposta:** Retorna as preferências do utilizador ou um erro se não as encontrar.

##### 7.1.14.2 create\_user\_preferences

Cria preferências para o utilizador com sessão iniciada.

- 
- **Entrada:** Nenhum parâmetro adicional, pois o “user\_id” é extraído do token JWT.
  - **Corpo do pedido HTTP:** Inclui tópicos de notificação, idioma preferido, entre outros campos opcionais.
  - **Execução:** Usa “createUserPreferencesv2” para criar as preferências com base nos dados enviados.
  - **Resposta:** Retorna uma mensagem de sucesso ou erro, dependendo do resultado.

#### 7.1.14.3 update\_user\_preferences

Atualiza as preferências do utilizador com sessão iniciada.

- **Entrada:** Nenhum parâmetro adicional, pois o “user\_id” é extraído do token JWT.
- **Corpo do pedido HTTP:** Inclui preferências, como tópicos de notificação e opções adicionais.
- **Execução:** Usa “updateUserPreferencesv2” para atualizar as preferências do utilizador com os novos dados fornecidos.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.14.4 get\_user\_role

Obtém o cargo (role) do utilizador com base no seu ID.

- **Parâmetro no URL:** “userID”.
- **Execução:** Chama “getUserRole” para obter o “role” do utilizador com o ID fornecido.
- **Resposta:** Retorna o cargo do utilizador ou um erro.

#### 7.1.14.5 get\_user\_by\_role

Obtém uma lista de utilizadores com base no seu cargo.

- **Parâmetro no URL:** role (nome do cargo).
- **Execução:** Usa “getUserByRole” para obter todos os utilizadores com o cargo especificado.
- **Resposta:** Retorna a lista de utilizadores com o cargo ou um erro.

#### 7.1.14.6 add\_bookmark

Adiciona uma marcação (*bookmark*) a um conteúdo específico.

- **Corpo do pedido HTTP:** Inclui o “userID”, “contentID” e “contentType”.

- 
- **Execução:** Chama “addBookmark” para adicionar a marcação ao conteúdo para o utilizador.
  - **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.14.7 remove\_bookmark

Remove uma marcação de um conteúdo.

- **Parâmetros no URL:** “userID”, “contentID” e “contentType”.
- **Execução:** Chama “removeBookmark” para remover a marcação do conteúdo.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.14.8 get\_user\_bookmarks

Obtém as marcações (*bookmarks*) do utilizador.

- **Parâmetro no URL:** “userID”.
- **Execução:** Usa “getUserBookmarks” para obter todas as marcações do utilizador.
- **Resposta:** Retorna as marcações ou uma mensagem de erro.

#### 7.1.14.9 update\_acc\_status

Atualiza o estado da conta de um utilizador.

- **Corpo do pedido HTTP:** Inclui o “user\_id” e o novo “status” da conta.
- **Execução:** Chama “updateAccStatus” para atualizar o estado da conta.
- **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.14.10 get\_users\_to\_validate

Obtém os utilizadores que precisam ser validados.

- **Execução:** Usa “getUsersToValidate” para obter os utilizadores pendentes de validação.
- **Resposta:** Retorna a lista de utilizadores ou uma mensagem de erro.

#### 7.1.14.11 update\_profile

Atualiza o perfil do utilizador com sessão iniciada.

- **Corpo do pedido HTTP:** Inclui os campos “firstName”, “lastName”, e “profile\_pic”.

- 
- **Execução:** Usa `updateProfile` para atualizar os dados do perfil.
  - **Resposta:** Retorna uma mensagem de sucesso ou erro.

#### 7.1.14.12 `get_user_content`

Obtém o conteúdo publicado pelo utilizador com sessão iniciada.

- **Execução:** Chama “`getUserPublications`” para obter todas as publicações do utilizador.
- **Resposta:** Retorna a lista de conteúdos ou uma mensagem de erro.

#### 7.1.14.13 `get_user_registeredEvents`

Obtém os eventos registados pelo utilizador com sessão iniciada.

- **Execução:** Usa “`getUserRegisteredEvents`” para obter os eventos nos quais o utilizador está registado.
- **Resposta:** Retorna os eventos ou uma mensagem de erro.

O `usersController.js` oferece uma ampla gama de funcionalidades para gerir perfis de utilizadores, suas preferências, eventos, marcações e conteúdos publicados. As operações são protegidas pelo uso de tokens JWT, garantindo que apenas o utilizador autenticado possa realizar modificações em seus próprios dados. O controlador também oferece um suporte robusto para operações administrativas, como validação de utilizadores e gerenciamento de cargos (*roles*).

### 7.1.15. `jwt_middlewareController`

O `jwt_middlewareController.js` é um *middleware* responsável por validar *tokens* JWT (*JSON Web Tokens*) em pedidos HTTP. O *middleware* garante que apenas utilizadores autenticados possam aceder a rotas protegidas. Ele também lida com diferentes cenários de erro relacionados ao *token*, como *tokens* expirados ou inválidos.

- **validation:** *Middleware* Principal para Validação de *Tokens* JWT

Esta função é o principal *middleware* de autenticação baseado em JWT. Este verifica se o *token* JWT presente no cabeçalho do pedido HTTP é válido, e se for, permite que o pedido prossiga para a próxima função.

#### 7.1.15.1 Passos

1. Captura o cabeçalho de autorização:

- 
- 1.1. O *middleware* obtém o *header* de autorização usando `req.headers["authorization"]`, que normalmente segue o padrão “Bearer <token>”.
  - 1.2. O *token* é extraído da *string* “Bearer <token>”.
  - 1.3. Verifica se o *token* está presente:
    - 1.3.1. Se o *token* estiver ausente, responde com o código de erro 401 (não autorizado) e uma mensagem indicando que o *token* não foi fornecido.
  2. Verifica o *token* com “verifyToken”:
    - 2.1. O *token* é verificado pela função “verifyToken” do módulo “tokenUtils”.
    - 2.2. Se o *token* for inválido, retorna um erro 403 (proibido).
  3. Erros de Expiração ou Formatação de *Tokens*:
    - 3.1. *Token* Expirado: Se o erro for do tipo “TokenExpiredError”, retorna 401 com uma mensagem de “token expired”.
    - 3.2. Erro de Formatação de *Token*: Se houver problemas de formatação, como um *token* inválido ou corrompido, retorna 401 com uma mensagem de “Token error”.
  4. Atribui o utilizador ao “req”:
    - 4.1. Caso o *token* seja válido, as informações do utilizador são anexadas ao objeto “req.user”, permitindo que rotas subsequentes acessem os dados do utilizador.

Este *middleware* garante que apenas utilizadores autenticados possam aceder a rotas protegidas, com tratamento adequado para erros como *tokens* ausentes, inválidos ou expirados. (Foram criados mais dois *middlewares* semelhantes ao “validation” de modo a garantir que as rotas para os administradores só conseguem ser acedidas por contas com as permissões necessárias).

### 7.1.16. emailController

O `emailController.js` é responsável por gerir o envio de e-mails na aplicação utilizando a biblioteca *Nodemailer*. Este oferece duas funcionalidades principais: o envio de e-mails síncronos (via *Express*) e o envio assíncrono com promessa (*Promise*). O controlador usa variáveis de ambiente para armazenar as credenciais do serviço de e-mail.

#### 1. Configuração de nodemailer

O *Nodemailer* é configurado com as credenciais de um serviço de e-mail, neste caso o Mail.ru. A configuração é feita através do objeto “*transporter*”, que especifica:

- **Host**: O servidor SMTP do Mail.ru (`smtp.mail.ru`).
- **Porta**: A porta segura 465 para conexões TLS.
- **Segurança**: O uso de uma conexão segura (TLS) com `secure: true`.

- 
- **Autenticação:** Autenticado com as credenciais MAILRU\_USER e MAILRU\_PASS, que são carregadas do arquivo .env via process.env.

## 2. **sendEmail: Função Síncrona para Envio de E-mails**

Esta função é utilizada para o envio de e-mails através de um pedido HTTP POST via *Express*. Os parâmetros do e-mail (destinatário, assunto, corpo) são recebidos diretamente no corpo do pedido HTTP.

- **Passos:**
  - **Validação:** Verifica se os campos “to”, “subject” e “body” foram fornecidos. Se algum estiver em falta, retorna um erro 400.
  - **Criação das Opções de E-mail:** O objeto “mailOptions” contém as configurações do e-mail, incluindo remetente, destinatário, assunto e corpo da mensagem.
  - **Envio do E-mail:** Usa o método “transporter.sendMail()” para enviar o e-mail.

Se houver um erro no envio, responde com o código 500 e a mensagem "Error sending email".

Se o e-mail for enviado com sucesso, responde com 200 e a mensagem "Email sent successfully".

## 3. **sendMail: Função Assíncrona para Envio de E-mails com Promessas (Promises)**

Essa função é semelhante à anterior, mas projetada para ser usada em outras partes do código de forma assíncrona, retornando uma *Promise*.

- **Passos:**
  - **Configuração das Opções de E-mail:** O objeto “mailOptions” contém as mesmas configurações, mas aqui o corpo do e-mail é enviado como HTML (html), o que permite e-mails formatados.
- **Envio do E-mail:** Usa uma promessa para lidar com o envio assíncrono:
- **Sucesso:** A promessa é resolvida com a mensagem "Email sent successfully".
- **Erro:** A promessa é rejeitada com "Error sending email".

Este método é útil quando você deseja integrar o envio de e-mails em funções mais complexas que utilizam promessas ou *async/await*.

O emailController.js oferece uma solução simples e eficaz para o envio de e-mails em uma aplicação Node.js. Utiliza o *Nodemailer* configurado com credenciais de serviço (Mail.ru neste caso) e permite o envio de e-mails em formato de texto ou HTML, seja de forma síncrona com rotas *Express* ou de forma assíncrona com *Promises*. As funções são protegidas contra erros comuns, como falta de parâmetros ou problemas com autenticação.



---

## 7.2. Rotas (routes)

As rotas definem os *endpoints* da API e são responsáveis por mapear os pedidos HTTP para os controladores correspondentes.

A estrutura de rotas programadas é uma implementação organizada de APIs RESTful utilizando Express.js para diferentes funcionalidades de uma aplicação. Cada rota chama um controlador específico para executar a lógica e, na maioria dos casos, usa um *middleware* JWT para validação de autenticação.

Aqui está uma visão geral de cada um dos ficheiros de rotas:

### 7.2.1. authRoutes.js

Este ficheiro gere todas as rotas relacionadas à autenticação e gestão de passwords:

- Registo de utilizadores (**/register**).
- *Login* (tanto para web quanto para mobile) e *Single Sign-On* (SSO).
- Alteração de password e recuperação da mesma.
- Atualização de *tokens* (**/refresh-token**).
- Atualização do *token* de FCM (*Firebase Cloud Messaging*) para notificações.

### 7.2.2. adminRoutes.js

Aqui são geridas as ações administrativas:

- Validação e rejeição de conteúdo.
- Métricas de *engagement* de utilizadores e *status* de validação do conteúdo.
- Gestão de centros (criar, atualizar e remover).
- Validação e desativação de utilizadores.
- Criação de avisos e relatórios administrativos.

### 7.2.3. dashboardRoutes.js

Este ficheiro contém rotas relacionadas à *dashboard*, retornando conteúdos para análise:

- Rotas para conteúdo que necessita de validação (**/toValidate**) e já validado (**/validated**).
- Posts, eventos e comentários por cidade.

### 7.2.4. static\_ContentRoute.js

Rotas para gerir conteúdo estático (categorias e subcategorias):

- 
- Criação, atualização e remoção de categorias e subcategorias.
  - Listagem de áreas e subáreas disponíveis.
  - Busca de centros registados.

### **7.2.5. dynamic\_contentRoute.js**

Estas rotas são responsáveis por interações dinâmicas de conteúdo como posts, fóruns e eventos:

- Obter posts, fóruns e eventos por cidade.
- Detalhes de posts, eventos e fóruns por ID.
- Registo e atualização de utilizadores.

### **7.2.6. postRoutes.js**

Gere os posts:

- Criação, edição e exclusão de posts.
- Obtém o estado e pontuação de um post específico.

### **7.2.7. forumRoutes.js**

Controla as rotas para fóruns:

- Criação, edição, alteração de estado e exclusão de fóruns.
- Gestão de fóruns relacionados a eventos.

### **7.2.8. eventRoutes.js**

Rotas focadas em eventos:

- Criação, edição e gestão de eventos.
- Registo e remoção de utilizadores em eventos.
- Gestão de participantes e a pontuação de eventos.

### **7.2.9. formsRoutes.js**

Para formulários relacionados a eventos:

- Criação e gestão de campos personalizados nos formulários de eventos.
- Inserção e obtenção de respostas dos formulários.

- 
- Rotas para excluir campos de formulários.

#### 7.2.10. **commentsRoutes.js**

Gestão de comentários:

- Adicionar, remover e visualizar árvore de comentários.
- *Likes* e *reports* de comentários.
- Consultar *likes* por conteúdo e usuário.

#### 7.2.11. **mediaRoutes.js**

Rotas para gerenciamento de fotografias:

- Criação de álbuns e adição de fotografias.

#### 7.2.12. **ratingRoutes.js**

Rotas para avaliação de conteúdo:

- Inserir uma avaliação para um conteúdo (post ou evento).

#### 7.2.13. **uploadRoutes.js**

Para gerir uploads de ficheiros (imagens):

- Upload de imagens associadas ao conteúdo.

#### 7.2.14. **userRoutes.js**

Rotas focadas na gestão de utilizadores:

- Gestão de *bookmarks* e preferências do utilizador.
- Recuperação de informações e cargos de utilizadores.
- Atualização de perfis e *status* de conta.
- Obtenção de conteúdo associado ao utilizador (eventos registados, conteúdo criado, etc.).

Cada ficheiro de rotas tem um propósito claro e está bem organizado para separar as diferentes funcionalidades da aplicação. Utiliza JWT para proteger as rotas, garantindo que apenas utilizadores autenticados possam aceder às operações permitidas. Isto mantém a segurança da aplicação.

---

## 7.3. Arquivos de Inicialização (start)

Os ficheiros de inicialização são essenciais para garantir que o ambiente de *backend* seja configurado corretamente, criando esquemas da base de dados e inserindo dados críticos para o funcionamento da aplicação. No caso deste projeto, existem alguns scripts principais responsáveis por essa tarefa, tais como `syncModels.js`, `insertAdmins.js`, `deleteDB.js` e `insertFakeData.js`.

### 7.3.1. syncModels.js

O ficheiro “`syncModels.js`” é responsável por sincronizar os modelos de dados com a base de dados. Este cria os esquemas necessários, sincroniza as tabelas e assegura que os modelos estejam alinhados com a estrutura da base de dados. Além disso, ele garante a inserção de dados padrões, como permissões de acesso e escritórios (offices). Os principais passos incluem:

- **Criação de Esquemas:** O script cria diversos esquemas (como “admin”, “centers”, “forms”, entre outros), garantindo que o ambiente da base de dados esteja preparado para receber os dados.
- **Sincronização de Modelos:** Uma ordem específica de sincronização é seguida para assegurar que dependências entre as tabelas sejam respeitadas, como “Users”, “Posts”, “Events”, “Comments” e mais.
- **Inserção de Dados Padrão:** Após a criação dos esquemas e tabelas, o script insere permissões de acesso padrão na tabela “security.access\_permissions” e escritórios na tabela “centers.offices”.

### 7.3.2. insertAdmins.js

Este ficheiro foca-se na inserção das contas administrativas necessárias para a administração e moderação do sistema. O script cria tanto o **Server Admin** quanto os **Admins de Centros**, associando-os aos respetivos centros. Após a criação dos administradores, passwords são definidas e os acessos são ativados. Esse processo garante que os administradores tenham as permissões corretas para administrar o sistema.

- **Criação do Server Admin:** O administrador principal do sistema é criado com acesso total, permitindo a administração de todos os centros e criação de mais administradores de centros.
- **Criação dos Admins de Centros:** Cada centro tem um administrador específico associado, como `softshares_tomar@yopmail.com`, `softshares_viseu@yopmail.com`, entre outros.

---

### 7.3.3. deleteDB.js e insertFakeData.js

Estes scripts complementares são utilizados para:

- **Remover Dados (deleteDB.js)**: Limpar a base de dados em um ambiente de desenvolvimento ou testes, preparando-a para receber novos dados e ou alterações aos modelos.
- **Inserir Dados de Teste (insertFakeData.js)**: Popular a base de dados com dados fictícios para testar funcionalidades e validar o comportamento do sistema. Isto é essencial em ambientes de desenvolvimento, onde é necessário realizar testes sem comprometer dados reais.

Estes ficheiros de inicialização são fundamentais para assegurar que o sistema esteja configurado corretamente desde o início, com permissões apropriadas, administradores criados e dados de teste para simulação de cenários do mundo real.

## 7.4. Utilitários

A pasta “utils” abrange um conjunto de ferramentas que facilitam operações específicas do *backend*. Estas utilidades simplificam processos como a validação dos dados de entrada, o registo de erros e as notificações em tempo real, tornando o código mais modular e de fácil gestão.

### 7.4.1. Validadores de Entradas

O ficheiro “inputValidators.js” concentra-se na validação de dados de entrada dos utilizadores em várias funcionalidades. Este processo garante que esses dados, como endereços de e-mail, nomes e *passwords* seguem os formatos esperados antes de serem processados pelo sistema. O ficheiro disponibiliza funções como “validateInput\_Login”, que verifica se o e-mail e a palavra-passe fornecidos são válidos, e “validateInput\_register”, que valida os formatos de e-mail e verifica a integridade dos nomes de utilizador durante o processo de registo.

Esta utilidade desempenha um papel crucial na manutenção da integridade dos dados e na prevenção de entradas erróneas ou maliciosas que possam afetar o sistema.

### 7.4.2. Notificações em Tempo Real

As notificações são um aspeto fundamental para o envolvimento dos utilizadores. O ficheiro “realTimeNotifications.js” gere notificações relacionadas com interações de utilizadores, como **comentários em publicações**, **gostos em comentários** e **inscrições em eventos**. As notificações são suportadas pelo *Firebase Cloud Messaging*, que envia notificações *push* para os dispositivos dos utilizadores.

As principais funcionalidades incluem:

- **Envio de Notificações Baseadas em Eventos**: Por exemplo, quando um utilizador se inscreve num evento, o criador do evento é notificado através da função `sendEventRegistrationNotification`.

- 
- **Gestão de Notificações de Comentários:** Quando os utilizadores comentam numa publicação, as notificações são enviadas ao proprietário do conteúdo ou aos participantes, informando-os sobre a atividade. Esta funcionalidade é gerida pelas funções “sendNewCommentNotification” e “sendNewCommentNotificationForEventsParticipants”.

Adicionalmente, é utilizado o mecanismo “LISTEN/NOTIFY” do PostgreSQL no ficheiro “server.js” para escutar eventos de validação de conteúdos na base de dados. Quando um novo conteúdo é validado, uma notificação é enviada aos utilizadores subscritos:

- Quando uma nova notificação de validação de conteúdos é recebida, o *listener* “pgClient.on('notification')” desencadeia uma notificação *push* em tempo real para os utilizadores subscritos no tópico correspondente.

### 7.4.3. Registo de Erros

Em sistemas complexos, uma gestão eficaz de erros é essencial. O ficheiro logError.js ajuda a capturar e registar erros em toda a aplicação, proporcionando um mecanismo centralizado para garantir que todas as exceções são devidamente documentadas. Este processo melhora a depuração e a monitorização do sistema, aumentando a fiabilidade geral e a experiência do utilizador.

Estas utilidades asseguram que as funcionalidades centrais do *backend*, como validação, comunicação em tempo real e registo de erros, são eficientes e escaláveis, contribuindo para uma arquitetura de *backend* robusta e fácil de manter.

## 7.5. Endpoints

A listagem e descrição dos *endpoints* incluirão operações CRUD para utilizadores, eventos, recomendações e categorias.

Abaixo, explica-se o propósito e a funcionalidade de cada *endpoint* exposto pela aplicação utilizando “Express.js”. Estes *endpoints* estão estruturados para cobrir uma gama de funcionalidades como autenticação, gestão de conteúdo dinâmico, fóruns, eventos, upload de imagens e mais.

### 7.5.1. /api/categories

- **Rotas associadas:** categoryRoutes
- **Função:** Gestão de categorias e subcategorias.
  - Criar categorias e subcategorias.
  - Atualizar e remover categorias e subcategorias.
  - Obter informações sobre áreas e subáreas.

---

### 7.5.2. /api/forum

- **Rotas associadas:** forumRoutes
- **Função:** Gestão de fóruns.
  - Criar, editar e remover fóruns.
  - Alterar o estado de um fórum (ativo/inativo).
  - Obter informações de fóruns específicos, como o estado ou detalhes do fórum.

### 7.5.3. /api/post

- **Rotas associadas:** postRoutes
- **Função:** Gestão de posts.
  - Criar, editar, remover posts.
  - Obter o estado de um post ou sua pontuação.

### 7.5.4. /api/event

- **Rotas associadas:** eventRoutes
- **Função:** Gestão de eventos.
  - Criar, editar, registar e cancelar a inscrição em eventos.
  - Gerir participantes e obter pontuações dos eventos.
  - Consultar o estado dos eventos ou obter participantes.

### 7.5.5. /api/media

- **Rotas associadas:** mediaRoutes
- **Função:** Gestão de álbuns e fotos.
  - Criar álbuns, adicionar fotografias a eventos, áreas ou álbuns específicos.
  - Obter fotos por ID de álbum ou evento.

### 7.5.6. /api/rating

- **Rotas associadas:** ratingRoutes
- **Função:** Avaliação de conteúdo.
  - Inserir avaliações (pontuação) para posts e eventos.

---

### 7.5.7. /api/administration

- **Rotas associadas:** adminRoutes
- **Função:** Funcionalidades administrativas.
  - Validação ou rejeição de conteúdos (posts, fóruns, eventos).
  - Gestão de centros e avisos.
  - Gestão de usuários (ativar/desativar).
  - Gestão de denúncias e métrica de *engagement* de usuários.

### 7.5.8. /api/form

- **Rotas associadas:** formsRoutes
- **Função:** Gestão de formulários relacionados a eventos.
  - Criar formulários para eventos.
  - Adicionar, editar ou remover campos de formulários.
  - Inserir e consultar respostas de formulários.

### 7.5.9. /api/comment

- **Rotas associadas:** commentsRoutes
- **Função:** Gestão de comentários.
  - Adicionar, editar, remover e visualizar a árvore de comentários.
  - Dar gosto em comentários e denunciá-los.

### 7.5.10. /api/user

- **Rotas associadas:** userRoutes
- **Função:** Gestão de usuários.
  - Gerir *bookmarks*, preferências e perfis de utilizadores.
  - Atualização de status da conta.
  - Consultar conteúdo e eventos registados por utilizador.

### 7.5.11. /api/dynamic

- **Rotas associadas:** dynamicRoutes
- **Função:** Manipulação de conteúdos dinâmicos (posts, fóruns, eventos).



- 
- Obter conteúdo (posts, eventos, fóruns) por cidade.
  - Obter posts, fóruns e eventos por ID e data.
  - Atualizar escritório dos utilizadores.

#### 7.5.12. /api/notification

- **Rotas associadas:** notificationsRoutes
- **Função:** Gestão de notificações em tempo real.
  - Envio e receção de notificações para eventos ou atualizações relevantes, comentários, entre outros.

#### 7.5.13. /api/auth

- **Rotas associadas:** authRoutes
- **Função:** Autenticação e gestão de *tokens*.
  - Registo de novos utilizadores, *login* e alteração de credenciais.
  - Recuperação de credenciais e geração de novos *tokens* JWT.

#### 7.5.14. /api/dashboard

- **Rotas associadas:** dashboardRoutes
- **Função:** Gestão de *dashboards* e visualização de métricas.
  - Exibir conteúdos a serem validados e já validados.
  - Exibir posts, eventos e discussões (fóruns) por cidade.

#### 7.5.15. /api/upload

- **Rotas associadas:** uploadRoute
- **Função:** Gestão de uploads.
  - Upload de ficheiros de imagens associadas ao conteúdo.

#### 7.5.16. /api/uploads

- **Rotas associadas:** express.static(path.join(\_\_dirname, 'uploads'))
- **Função:** Servir ficheiros estáticos (uploads de imagens).
- Servir ficheiros estáticos diretamente do diretório “uploads”, como enviadas via upload.

---

Esta estrutura de *endpoints* é bem modular e organizada, seguindo princípios de separação de responsabilidades e boas práticas REST. Cada *endpoint* está associado a uma área específica da aplicação (autenticação, gestão de imagens, eventos, comentários, etc.), garantindo uma manutenção e escalabilidade simplificadas.

## 7.6. Conclusão *backend*

A arquitetura do *backend* desempenha um papel crucial na estruturação e operacionalização da aplicação, proporcionando uma base robusta para o processamento de dados, validação, segurança e comunicação em tempo real. Através da organização dos controladores, rotas e utilitários, o *backend* mantém-se modular, escalável e eficiente, assegurando a integridade e consistência dos dados.

A implementação das APIs segue os princípios RESTful, permitindo que as diferentes partes da aplicação interajam de forma harmoniosa e eficaz. Com a utilização de *middleware* para autenticação via JWT, o sistema oferece uma camada adicional de segurança, protegendo recursos sensíveis e garantindo que apenas utilizadores autorizados tenham acesso às funcionalidades adequadas.

Ferramentas utilitárias, como a validação de dados de entrada, notificações em tempo real, e o uso de bibliotecas como o “Nodemailer” para o envio de emails, oferecem flexibilidade e melhoram a experiência do utilizador. A integração com o PostgreSQL, utilizando notificações via “LISTEN/NOTIFY”, exemplifica a capacidade do *backend* de gerir eventos em tempo real, o que é fundamental para manter uma interação contínua e comunicação instantânea entre os utilizadores.

Em resumo, o *backend* desta aplicação foi projetado para ser altamente funcional, seguro e responsivo às necessidades dinâmicas da aplicação. É capaz de lidar com grandes volumes de dados, realizar autenticações complexas e, simultaneamente, proporcionar uma experiência fluida aos utilizadores finais. Ao manter-se modular e extensível, o *backend* está preparado para suportar expansões futuras e novas funcionalidades de forma contínua, sem comprometer o desempenho do sistema.

---

## 8. Integração com Base de Dados

### 8.1. Estrutura da Base de Dados

De forma a compartimentalizar tanto as tabelas como as diversas funções, procedimentos armazenados e *triggers*, procedeu-se a criação dos seguintes ‘Schemas’:

- SECURITY
- USER\_INTERACTIONS
- HR
- CONTROL
- CENTERS
- STATIC\_CONTENT
- DYNAMIC\_CONTENT
- COMMUNICATION
- FORMS
- ADMIN

### 8.2. Modelagem de Dados

Os modelos da base de dados deste projeto são definidos utilizando o “Sequelize”, um ORM (*Object-Relational Mapping*) que facilita a interação com a base de dados, permitindo a criação e gestão de tabelas e associações sem a necessidade de escrever consultas SQL manualmente. A seguir, são descritos alguns dos principais modelos, cada um representando uma tabela e as suas relações no sistema.

#### 8.2.1. ActiveDiscussions

Este modelo é responsável por armazenar discussões ativas em fóruns, associando-as à tabela de fóruns e gerindo informações como a última data de atividade e o número de participantes ativos.

- **Chaves:** “discussion\_id”, “forum\_id”
- **Associações:** Relaciona-se com “Forums”, utilizando a chave estrangeira “forum\_id”.

#### 8.2.2. ContentValidationStatus

Este modelo armazena o estado de validação de diversos tipos de conteúdo. Cada conteúdo pode estar validado, pendente ou rejeitado, e a tabela regista o ID do validador e a data de validação.

- **Chaves:** “content\_id”, “content\_real\_id”
- **Associações:** Relaciona-se com “Users” pela chave “validator\_id”, armazenando o utilizador que efetuou a validação.

#### 8.2.3. Offices

Este modelo representa os escritórios registados no sistema, contendo informações como a cidade e a imagem do escritório.

- 
- **Chaves:** “office\_id”
  - **Associações:** Relaciona-se com “OfficeAdmins” e “OfficeWorkers”, gerindo os administradores e trabalhadores de cada escritório. Relaciona-se também com conteúdos dinâmicos como “Posts”, “Forums” e “Events”, que podem estar associados a um escritório.

#### 8.2.4. OfficeWorkers

Este modelo mantém o relacionamento entre os trabalhadores e os respetivos escritórios.

- **Chaves:** “office\_id”, “user\_id”
- **Associações:** Relaciona-se com “Offices” e “Users”, permitindo que um trabalhador seja associado a um escritório.

#### 8.2.5. Comments

Representa os comentários feitos em fóruns ou publicações. Armazena o conteúdo, a data de criação, o número de "gostos" e a associação com o fórum ou publicação em questão.

- **Chaves:** “comment\_id”
- **Associações:** Relaciona-se com “Users”, “Posts” e “Forums” para vincular o autor do comentário e o conteúdo relacionado.

#### 8.2.6. Likes

Armazena os "gostos" que os utilizadores atribuem aos comentários.

- **Chaves:** “like\_id”
- **Associações:** Relaciona-se com “Users” e “Comments” para gerir quem fez o "gosto" e em qual comentário.

#### 8.2.7. Participation

Este modelo regista a participação dos utilizadores em eventos, armazenando o ID do utilizador, o ID do evento e a data de inscrição.

- **Chaves:** “user\_id”, “event\_id”
- **Associações:** Relaciona-se com “Users” e “Events”, vinculando a participação dos utilizadores aos eventos.

---

### 8.2.8. Reports

Armazena denúncias relacionadas a comentários. Cada denúncia é associado a um comentário específico e a um utilizador que fez a denúncia.

- **Chaves:** “report\_id”
- **Associações:** Relaciona-se com “Users” e “Comments” para associar o autor da denúncia e o comentário denunciado.

### 8.2.9. Warnings

Representa avisos emitidos por administradores a utilizadores ou escritórios.

- **Chaves:** “warning\_id”
- **Associações:** Relaciona-se com “Users” e “OfficeAdmins”, armazenando informações sobre o administrador que emitiu o aviso e o escritório associado.

### 8.2.10. Events

Modelo que gere os eventos criados na plataforma, incluindo detalhes como data, local, número de participantes e estado de validação.

- **Chaves:** “event\_id”
- **Associações:** Relaciona-se com “Users”, “Offices”, “SubAreas” e “Scores” para gerir os participantes, organizadores e pontuações dos eventos.

### 8.2.11. Forums

Gere os fóruns criados dentro de subáreas e escritórios. Inclui informações como título, conteúdo e estado de validação.

- **Chaves:** “forum\_id”
- **Associações:** Relaciona-se com “Users”, “Offices”, “SubAreas” e “Events”, armazenando quem criou o fórum e o contexto em que foi criado.

### 8.2.12. Posts

Modelo que representa publicações feitas em subáreas e escritórios, com dados como título, conteúdo, preço, localização e estado de validação.

- **Chaves:** “post\_id”
- **Associações:** Relaciona-se com “Users”, “Offices”, “SubAreas” e “Scores”.

---

### 8.2.13. Albuns e Photographs

Gere os álbuns de fotos criados para eventos ou áreas e as respectivas fotografias.

- **Chaves:** “album\_id”, “photo\_id”
- **Associações:** “Albuns” relacionam-se com “Events” e “SubAreas”, enquanto “Photographs” estão associados a “Albuns” e “Users”.

### 8.2.14. Scores e Ratings

Gere as pontuações de eventos e publicações. “Scores” armazena a média de avaliações, enquanto “Ratings” regista as avaliações individuais de cada utilizador.

- **Chaves:** “avg\_rating\_id”, “rating\_id”
- **Associações:** “Scores” relaciona-se com “Posts” e “Events”, enquanto “Ratings” relaciona-se com “Users”.

### 8.2.15. Users

Modelo principal que gere os utilizadores da plataforma, com informações pessoais e de autenticação, como nome, e-mail e permissões.

- **Chaves:** “user\_id”
- **Associações:** Relaciona-se com uma ampla gama de modelos, incluindo “Posts”, “Events”, “Forums”, “Comments”, entre outros, permitindo que um utilizador participe ativamente em todas as funcionalidades do sistema.

Estes modelos, juntamente com as suas associações, formam a base para a gestão de toda a interação na plataforma, assegurando que todas as funcionalidades, desde discussões em fóruns até avaliações de eventos, sejam registadas e geridas de forma organizada e eficiente.

## 8.3. Triggers

Os *triggers* desempenham um papel fundamental no *backend* deste sistema, automatizando diversas operações e garantindo a integridade e eficiência dos dados. A seguir, detalhamos os principais *triggers* implementados e as suas respectivas funções.

### 8.3.1. Trigger para Criar Álbum Após a Criação de Área

Este *trigger* cria automaticamente um álbum associado a uma área recém-criada, garantindo que cada área tenha um álbum correspondente para armazenar fotos e outros conteúdos multimédia.

- 
- **Função:** `dynamic_content.create_album_for_area()`
  - **Evento:** Disparado após a inserção de uma nova área na tabela “`static_content.area`”.
  - **Uso:** Insere um álbum na tabela “`dynamic_content.albums`” com base no título e ID da nova área.

### 8.3.2. Trigger para Incrementar e Decrementar Contagem de Curtidas em Comentários

Os *triggers* “`increment_like_count`” e “`decrement_like_count`” são responsáveis por atualizar a contagem de "gostos" dos comentários na tabela “`comments`” após a inserção ou exclusão de uma "curtida".

- **Função:** “`communication.increment_like_count()`”, “`communication.decrement_like_count()`”
- **Evento:** Disparado após a inserção ou exclusão de um "gosto" na tabela “`communication.likes`”.
- **Uso:** Incrementa ou decrementa a contagem de "gostos" num comentário específico.

### 8.3.3. Trigger para Moderação de Conteúdo de Evento

Quando um evento é criado, este *trigger* modera o conteúdo e insere uma entrada na tabela de estado de validação de conteúdo. Caso o evento seja aprovado, também insere registros nas tabelas de participação e pontuação.

- **Função:** `dynamic_content.trg_moderate_event_content()`
- **Evento:** Disparado após a inserção de um novo evento na tabela “`dynamic_content.events`”.
- **Uso:** Modera o conteúdo e insere dados relevantes nas tabelas de validação e participação.

### 8.3.4. Trigger para Validação de Conteúdo

Quando o estado de validação de qualquer conteúdo (publicação, evento, fórum) é alterado para "Aprovado", o sistema automaticamente notifica o sistema de notificações e atualiza as tabelas correspondentes.

- **Função:** `admin.trg_content_status_approved()`
- **Evento:** Disparado após a inserção ou atualização na tabela “`admin.content_validation_status`”.
- **Uso:** Atualiza o estado do conteúdo e insere dados nas tabelas de participação e pontuação.

---

### 8.3.5. Trigger para Criação de Álbum Após Validação de Evento

Após a validação de um evento, este *trigger* cria automaticamente um álbum para armazenar fotos relacionadas ao evento.

- **Função:** `dynamic_content.create_album_for_validated_event()`
- **Evento:** Disparado após a inserção ou atualização de um evento na tabela “`dynamic_content.events`”, quando o evento é validado.
- **Uso:** Garante que todo evento validado tenha um álbum correspondente na tabela “`álbuns`”.

### 8.3.6. Trigger para Atualização da Pontuação Média de Publicações e Eventos

Este *trigger* calcula e atualiza a pontuação média de publicações e eventos após a inserção ou atualização de uma nova avaliação (“`rating`”). Utiliza a função “`fn_reverse_rating`” para recalcular a média com base nas novas avaliações.

- **Função:** `dynamic_content.trg_update_average_score()`
- **Evento:** Disparado após a inserção ou atualização de uma nova avaliação na tabela “`ratings`”.
- **Uso:** Atualiza a pontuação média e o número de avaliações nas tabelas de pontuação.

### 8.3.7. Trigger para Gestão de Participação em Eventos

Quando um utilizador se inscreve ou é removido de um evento, este *trigger* atualiza a contagem de participantes no evento. Além disso, há uma função dedicada para limpar a participação de utilizadores inativos.

- **Função:** `control.trg_event_participation_count()`
- **Evento:** Disparado após a inserção ou exclusão de registos na tabela “`control.participation`”.
- **Uso:** Atualiza o número de participantes no evento na tabela “`events`”.

### 8.3.8. Triggers para Notificações

Quando o conteúdo é validado, um *trigger* envia notificações em tempo real através do sistema PostgreSQL “`LISTEN/NOTIFY`” para o servidor, de forma a este notificar os utilizadores inscritos em um certo tópico sobre novos conteúdos disponíveis.

- **Função:** `admin.notify_content_validated()`



- 
- **Evento:** Disparado após a inserção ou atualização de conteúdo na tabela “admin.content\_validation\_status”.
  - **Uso:** Envia notificações *push* para os utilizadores inscritos.

Estes *triggers* são parte integrante do sistema, automatizando tarefas essenciais, como a validação de conteúdo, a gestão de "gostos", a participação em eventos e o envio de notificações *push*. Eles garantem que o sistema funcione de forma eficiente e responsiva, otimizando a experiência do utilizador e mantendo a integridade dos dados.

## **8.4. Procedimentos armazenados**

Os procedimentos armazenados foram reimplementados como funções em JavaScript, utilizando consultas SQL diretas (*raw queries*), após a migração da base de dados de Microsoft SQL Server para PostgreSQL. Esta abordagem foi adotada para garantir uma maior flexibilidade na execução de operações específicas, uma vez que o PostgreSQL utiliza um modelo ligeiramente diferente para o tratamento de procedimentos armazenados em comparação com o SQL Server. A utilização de *raw queries* permitiu a adaptação eficiente das funcionalidades existentes, mantendo o desempenho e a integridade das operações da base de dados, enquanto se beneficiava das capacidades avançadas do PostgreSQL, como o suporte a eventos e *triggers*.

---

## 9. Avaliação dos Objetivos

A avaliação dos objetivos do projeto reflete o cumprimento integral das funcionalidades propostas e sua implementação eficaz no sistema. O projeto atingiu com sucesso os requisitos funcionais e as especificações previstas, conforme descrito nos objetivos iniciais. A seguir, destacam-se os principais resultados alcançados:

**Criação e Gerenciamento de Utilizadores:** O sistema permite a criação de novos utilizadores, ativação e desativação de contas, além da atribuição de permissões específicas e associação de utilizadores a centros. Foi implementada a obrigatoriedade de atualização da senha no primeiro login (para contas geradas no *backoffice*), proporcionando um nível adicional de segurança, além do envio automático de emails de confirmação após o registo, garantindo a comunicação eficaz com os novos utilizadores.

**Estrutura Modular de Áreas e Subáreas:** A possibilidade de criação de áreas e subáreas foi implementada com sucesso. Os administradores podem gerir categorias como desporto e formação, criando fóruns de discussão e álbuns associados a cada área, ampliando a flexibilidade e dinamismo do sistema.

**Gerenciamento de Fóruns e Conteúdos:** O sistema permite a criação e moderação de fóruns por categoria e atividade. Os administradores podem editar conteúdos submetidos pelos utilizadores, validá-los e publicá-los em tempo real. Adicionalmente, há *dashboards* que oferecem uma visão geral das atividades mais comentadas, tópicos abertos e conteúdos a serem validados.

**Funcionalidades Avançadas para Eventos:** A criação e gestão de eventos está bem suportada, com a possibilidade de inscrição, apresentação de calendário e notificações automáticas de alterações e novas interações nos eventos. O sistema também permite a adição de geolocalização e a partilha de eventos em redes sociais, promovendo a interação dos utilizadores.

**Funcionalidades de Interação Social:** O sistema possibilita a criação de recomendações, comentários, avaliações e fóruns, com a adição de funcionalidades para curtir e denunciar conteúdos. As notificações em tempo real mantêm os utilizadores informados sobre novas interações em suas publicações e eventos.

**Customização e Personalização:** A plataforma oferece uma área pessoal para os utilizadores, onde podem definir preferências de conteúdo, notificações e áreas de interesse. Essa personalização é refletida na receção de notificações automáticas sempre que há novos conteúdos publicados nas áreas de interesse do utilizador.

**Gestão Centralizada para Administradores:** Administradores podem moderar conteúdos, administrar eventos, recomendações e fóruns, e validar tudo diretamente no *backoffice*, restrito ao centro que administram (com exceção do “*Server Admin*”). Esta funcionalidade permite maior controle e descentralização das operações, assegurando a consistência e conformidade dos conteúdos.

**Notificações e Comunicação:** O sistema de notificações foi implementado com sucesso, com alertas automáticos sobre eventos, comentários e atualizações de fóruns. Além disso, foi

---

configurado um sistema de saudação dinâmica com base no horário de acesso, melhorando a experiência de utilizador.

Com a implementação de todos esses pontos, os objetivos propostos no início do projeto foram cumpridos com excelência, o que demonstra a robustez e a abrangência do sistema em termos de funcionalidade, usabilidade e segurança.

---

## 10. Observações

### 10.1. Dificuldades

Durante a realização deste projeto, enfrentámos diversas dificuldades, particularmente na utilização do **Sequelize** e na compreensão do seu funcionamento adequado. A integração com o **Multer** também apresentou desafios, dado que, por vezes, não operou como esperado. Adicionalmente, a compreensão dos **JWT** e a configuração correta do **serviço de email SMTP** exigiram esforços significativos. No que se refere ao conteúdo, a edição seletiva de registos da base de dados, como Posts, Fóruns e Eventos, revelou-se uma tarefa complexa, assim como a implementação e gestão de **formulários dinâmicos**, que se mostraram difíceis de manipular. Outro ponto desafiador foi o processo de implementação do **SSO** bem como a sua integração funcional com a API. Por fim, a configuração e operação correta das **notificações push** também representaram um obstáculo significativo ao longo do desenvolvimento.

### 10.2. Pontos Fortes e Fracos

#### 10.2.1. Pontos Fortes

- Uso de cursores nos *triggers* de forma a assegurar que o trigger é sempre executado no caso de haver muitas inserções na mesma tabela no mesmo segundo;
- Utilização de *Schemas* de forma a assegurar uma maior compartimentalização tanto das tabelas como dos objetos lógicos;
- Controlo de transações e registo dos erros ocorridos durante a execução de algum objeto lógico.
- Uso de encriptação dos *tokens* para evitar que pessoas não autorizadas acedam à informação.
- Diferentes *middlewares* para a validação de *tokens* para uso da API (uma para os utilizadores todos, uma para operações apenas disponíveis para os administradores e uma para as operações que apenas o “*Server Admin*” pode executar).

#### 10.2.2. Pontos Fracos

- Falta de controlo de concorrências;
- Falta de índices de forma a melhorar a performance no caso de existirem muitos registos na base de dados.
- Falta de otimização da aplicação móvel de forma a esta executar sem falhas e sem problemas.

---

## 11. Conclusão

O projeto desenvolvido apresenta uma solução inovadora e robusta para a gestão de eventos, fóruns, recomendações e outros conteúdos, com um forte enfoque na segurança e integração de tecnologias modernas. Com a implementação de mecanismos como o **JWT** para autenticação e autorização segura, a **criptação de tokens** e a **imposição de políticas de reutilização de senhas**, o sistema assegura que os dados e interações dos utilizadores sejam protegidos contra acessos não autorizados. Além disso, a segregação de permissões de acesso, por meio de *middlewares* específicos para administradores e “*Server Admin*”, garante um controlo refinado sobre quem pode executar determinadas ações críticas.

Outro ponto forte do sistema é a implementação de **notificações push**, que mantêm os utilizadores atualizados sobre atividades relevantes, como interações em eventos e fóruns, em tempo real. O uso de **Single Sign-On (SSO)** com Google e Facebook acrescenta conveniência e segurança ao simplificar o processo de login. Adicionalmente, a criação de conteúdos dinâmicos, como posts e eventos, e a **integração com o Google Maps** para associar geolocalização a atividades e recomendações proporcionam uma experiência completa e interativa aos utilizadores.

Contudo, o desenvolvimento do projeto não foi isento de desafios. A utilização de tecnologias como **Sequelize**, **Multer** e a **configuração de serviços de email SMTP** exigiram um esforço considerável para garantir uma integração adequada. A complexidade de lidar com **formulários dinâmicos** e a **implementação do SSO** também foram dificuldades significativas. Além disso, a implementação de **notificações push** exigiu ajustes para funcionar corretamente, especialmente devido às especificidades da infraestrutura.

A segurança foi um dos principais pilares deste projeto, com o uso de **triggers** para monitorar transações e alterações na base de dados, evitando inserções incorretas, além do registo detalhado de erros no sistema. A divisão de dados por **schemas** trouxe maior organização e eficiência ao sistema, tornando a manutenção mais ágil e o sistema mais escalável.

Apesar de algumas limitações, como a falta de controlo de concorrência e a necessidade de otimizar o desempenho da aplicação móvel, o sistema, na sua forma atual, cumpre os requisitos funcionais estabelecidos. Apresenta um conjunto sólido de funcionalidades tanto para os utilizadores finais quanto para os administradores, com uma arquitetura bem planeada e voltada para futuras expansões.

O “The SoftShares” tem um grande potencial para transformar a forma como eventos, fóruns e recomendações são geridos e consumidos pelos utilizadores. Embora ainda haja espaço para melhorias, particularmente em termos de desempenho e otimização da aplicação móvel, os avanços já implementados colocam a aplicação num patamar elevado em termos de usabilidade, segurança e inovação tecnológica.

---

## 12. Bibliografia

- Spainhour, Stephen, & Eckstein, Robert. (1999) Webmaster in a nutshell (2ª ed.). O'Reilly.
- Queiros, Ricardo, & Portela, Filipe. (2018) Introdução ao Desenvolvimento para a Web (1ª ed.). FCA.
- Queiros, Ricardo, & Portela, Filipe. (2020) Desenvolvimento Avançado para Web (1ª ed.). FCA.
- Richardson, Leonard & Amundsen, Mike. (2015) RESTful Web APIs (2ª ed.). O'Reilly.
- Krause, Martin. (2024) The Complete Developer (1ª ed.). Nostarch Press.
- Haverbeke, Marijn. (2018) Eloquent Javascript (3ª ed.). Nostarch Press.
- Ramakrishnan, Raghu, & Gehrke, Johannes. (2000) Database Management Systems (2ª ed.). McGraw-Hill International Editions – Computer Science Series.
- Karwin, Bill. (2022) SQL Antipatterns, Volume 1 Avoiding the Pitfalls of Database Programming. The Pragmatic Bookshelf.
-

---

## 13. Referências

- kelektiv. (n.d.). node.bcrypt.js: Usage. GitHub. Disponível em: <https://github.com/kelektiv/node.bcrypt.js#usage>
- npmjs. (n.d.). bcryptjs. npm. Disponível em: <https://www.npmjs.com/package/bcryptjs>
- Node.js. (n.d.). crypto.ECDH.generateKeys(encoding, format). Node.js API. Disponível em: <https://nodejs.org/api/crypto.html#ecdhgeneratekeysencoding-format>
- Node.js. (n.d.). API Documentation. Node.js. Disponível em: <https://nodejs.org/docs/latest/api/>
- colorlibhq. (n.d.). AdminLTE. GitHub. Disponível em: <https://github.com/colorlibhq/AdminLTE>
- npmjs. (n.d.). morgan. npm. Disponível em: <https://www.npmjs.com/package/morgan>
- Maison Moa. (2019). Using Bcrypt to Hash and Compare Passwords with Node.js and MongoDB. LevelUp. Disponível em: <https://levelup.gitconnected.com/using-bcrypt-to-hash-and-compare-passwords-with-nodejs-and-mongodb-366ff80138b7>
- Moa, M. (2019). Using JWT (JSON Web Tokens) to Authorize Users and Protect API Routes. Medium. Disponível em: <https://medium.com/@maison.moa/using-jwt-json-web-tokens-to-authorize-users-and-protect-api-routes-3e04a1453c3e>
- Bitsrc.io. (2018). Understanding JSON Web Token Authentication. Medium. Disponível em: <https://blog.bitsrc.io/understanding-json-web-token-authentication-a1febf0e15>
- JWT.io. (n.d.). JSON Web Tokens - Introduction. Disponível em: <https://jwt.io/>
- W3Schools. (n.d.). React useEffect. Disponível em: [https://www.w3schools.com/react/react\\_useeffect.asp](https://www.w3schools.com/react/react_useeffect.asp)
- Medium. (2019). How to Upload Image Using Multer in Node.js. Disponível em: <https://medium.com/swlh/how-to-upload-image-using-multer-in-node-js-f3aeffb90657>
- SweetAlert2. (n.d.). SweetAlert2 Documentation. Disponível em: <https://sweetalert2.github.io/>
- Sequelize. (n.d.). Sequelize Validator API v7. Disponível em: [https://sequelize.org/api/v7/modules/\\_sequelize\\_validator\\_js.html](https://sequelize.org/api/v7/modules/_sequelize_validator_js.html)
- Sequelize. (n.d.). QueryInterface.bulkInsert API v6. Disponível em: <https://sequelize.org/api/v6/class/src/dialects/abstract/query-interface.js~queryinterface#instance-method-bulkInsert>
- Carmine Zacc. (2020). Securely Storing JWTs in Flutter Web Apps. Dev.to. Disponível em: <https://dev.to/carminezacc/securely-storing-jwts-in-flutter-web-apps-2nal>
- Carmine Zacc. (2020). User Authentication & JWT Authorization with Flutter and Node.js. Dev.to. Disponível em: <https://dev.to/carminezacc/user-authentication-jwt-authorization-with-flutter-and-node-176l>
- PostgreSQL Tutorial. (n.d.). PostgreSQL Triggers. Disponível em: <https://www.postgresqltutorial.com/postgresql-triggers/>
- PostgreSQL Tutorial. (n.d.). PostgreSQL After Insert Trigger. Disponível em: <https://www.postgresqltutorial.com/postgresql-triggers/postgresql-after-insert-trigger/>
- Axios. (n.d.). Axios Introduction. Disponível em: <https://axios-http.com/docs/intro>

- 
- Singh, V. (2021). Creating CRUD APIs with Node.js and Sequelize CLI. Medium. Disponible en: <https://medium.com/@vinayak-singh/creating-crud-apis-with-node-js-and-sequelize-cli-8b90e8784422>
  - Medium. (2019). How to Upload Image Using Multer in Node.js. Disponible en: <https://medium.com/swlh/how-to-upload-image-using-multer-in-node-js-f3aeffb90657>
  - Mail.ru. (n.d.). E-mail. Disponible en: <https://e.mail.ru/>
  - EmailJS. (n.d.). EmailJS Documentation. Disponible en: <https://www.emailjs.com/>
  - Android Developers. (n.d.). Android Manifest Introduction. Disponible en: <https://developer.android.com/guide/topics/manifest/manifest-intro>
  - XML Validation. (n.d.). XML Validation Service. Disponible en: <https://www.xmlvalidation.com/>
  - Firebase. (n.d.). Firebase Setup for Flutter. Disponible en: <https://firebase.google.com/docs/flutter/setup>
  - GPT. (2021). Flutter and Google Sign-In Plugin. YouTube. Disponible en: <https://www.youtube.com/watch?v=gptBM2CPMQs>
  - Stack Overflow. (2019). Flutter and Google Sign-In Plugin Error: PlatformException(sign\_in\_failed). Disponible en: <https://stackoverflow.com/questions/54557479/flutter-and-google-sign-in-plugin-platformexceptionsign-in-failed-com-google>
  - Google Developers. (n.d.). Google OAuth2 Authentication Guide. Disponible en: <https://developers.google.com/identity/protocols/oauth2>
  - Facebook Developers. (n.d.). Facebook Android Quickstart Guide. Disponible en: <https://developers.facebook.com/quickstarts/1204186127679127/?platform=android>
  - Facebook Developers. (n.d.). Getting Started with Facebook Android SDK. Disponible en: <https://developers.facebook.com/docs/android/getting-started/>
  - Facebook Developers. (n.d.). Facebook Login for Android. Disponible en: <https://developers.facebook.com/docs/facebook-login/android>
  - Ably. (2021). Firebase vs WebSocket: Which to Use When Building Realtime Features. Dev.to. Disponible en: <https://dev.to/ably/firebase-vs-websocket-which-to-use-when-building-realtime-features-5gem>
  - Ably. (n.d.). Are Web Sockets Faster Than HTTP?. Disponible en: <https://ably.com/topic/websockets#are-web-sockets-faster-than-http>
  - Ably. (2020). Building a Realtime Chat App with WebSockets, Angular, and Firebase. Disponible en: <https://ably.com/blog/realtime-chat-app-websockets-angular-firebase>
  - Codesphere. (2020). Getting Started with Web Sockets in Node.js. Dev.to. Disponible en: <https://dev.to/codesphere/getting-started-with-web-sockets-in-nodejs-49n0>
  - GeeksforGeeks. (2021). Web Socket in Node.js. Disponible en: <https://www.geeksforgeeks.org/web-socket-in-node-js/>
  - Hypirion. (2021). Implementing System-Versioned Tables in Postgres. Disponible en: <https://hypirion.com/musings/implementing-system-versioned-tables-in-postgres>
  - MSSQLTips. (2008). Using Table-Valued Parameters (TVP) in SQL Server. Disponible en: <https://www.mssqltips.com/sqlservertip/1483/using-table-valued-parameters-tvp-insqlserver/#:~:text=The%20benefit%20of%20the%20TVP,to%201%2C000%20or%20so%20rows.>
-



- 
- Microsoft. (2021). Use Table-Valued Parameters in SQL Server. Disponível em: <https://learn.microsoft.com/en-us/sql/relational-databases/tables/use-table-valuedparameters-database-engine?view=sql-server-ver16>
  - Red Gate. (2019). SQL Server Closure Tables. Simple Talk. Disponível em: <https://www.red-gate.com/simple-talk/databases/sql-server/t-sql-programming-sqlserver/sql-server-closure-tables/>
  - Google Developers. (n.d.). Client Authentication Guide. Disponível em: <https://developers.google.com/android/guides/client-auth>
  - Facebook Developers. (n.d.). Guia de início rápido para Android. Disponível em: <https://developers.facebook.com/quickstarts/1204186127679127/?platform=android>
  - Android Developers. (n.d.). Android Notification Channels. Disponível em: <https://developer.android.com/develop/ui/views/notifications/channels>
  - Firebase. (n.d.). Firebase Cloud Messaging (JS) - Client Setup. Disponível em: <https://firebase.google.com/docs/cloud-messaging/js/client>
  - Firebase. (n.d.). Firebase Cloud Messaging (JS) - Send Multiple Messages. Disponível em: <https://firebase.google.com/docs/cloud-messaging/js/send-multiple>
  - Firebase. (n.d.). Firebase Cloud Messaging for Flutter - Receiving Messages. Disponível em: <https://firebase.google.com/docs/cloud-messaging/flutter/receive>
  - Firebase. (n.d.). Firebase Cloud Messaging for Flutter - First Message. Disponível em: <https://firebase.google.com/docs/cloud-messaging/flutter/first-message>
  - Klein, K. (2020). Push Notification Using Firebase, Node.js, Flutter, Dart. Medium. Disponível em: <https://medium.com/@kayoneklein/push-notification-using-firebase-node-js-flutter-dart-f87e6a0567d>
  - LogRocket. (2021). Add Flutter Push Notifications with Firebase Cloud Messaging. LogRocket. Disponível em: <https://blog.logrocket.com/add-flutter-push-notifications-firebase-cloud-messaging/>
  - King Rittik. (2020). Flutter Notifications Made Easy Using Firebase Cloud Messaging (FCM). Medium. Disponível em: <https://medium.com/@kingrittik/flutter-notifications-made-easy-using-firebase-cloud-messaging-fcm-eada3f88af77>
  - Ravi Sharma. (2021). Sending Notifications to Mobile Devices with Firebase Cloud Messaging (FCM) in Node.js. Medium. Disponível em: <https://medium.com/@ravisharma23523/sending-notifications-to-mobile-devices-with-firebase-cloud-messaging-fcm-in-node-js-8fe3faead58b>
  - Firebase. (n.d.). Firebase Cloud Messaging (JS) - Device Groups. Disponível em: <https://firebase.google.com/docs/cloud-messaging/js/device-group>
  - Nybles. (2020). Sending Push Notifications by Using Firebase Cloud Messaging. Medium. Disponível em: <https://medium.com/nybles/sending-push-notifications-by-using-firebase-cloud-messaging-249aa34f4f4c>