

# Evention

Mestrado em Engenharia Informática

Pedro Miguel Gomes Martins

Luís Miguel Da Costa Anjo

Diogo Gomes Silva

(nrº 23527, 23528, 23893, regime pós-laboral)

ARQUITETURAS E INTEGRAÇÃO DE SISTEMAS  
ESCOLA SUPERIOR DE TECNOLOGIA  
INSTITUTO POLITÉCNICO DO CÁVADO E DO AVE

# Conteúdo

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Descrição . . . . .	1
1.2	Objetivos . . . . .	2
1.3	Público-alvo . . . . .	2
<b>2</b>	<b>Identificação e Análise do Problema</b>	<b>3</b>
2.1	Identificação do Problema . . . . .	3
2.2	Análise do Problema . . . . .	3
<b>3</b>	<b>Processo de Negócio</b>	<b>4</b>
3.1	Microserviço User . . . . .	4
3.1.1	Funcionalidades . . . . .	4
3.1.2	Estrutura . . . . .	5
3.1.3	Diagrama BPMN . . . . .	5
3.2	Microserviço Event . . . . .	6
3.2.1	Funcionalidades . . . . .	6
3.2.2	Estrutura . . . . .	7
3.2.3	Diagrama BPMN . . . . .	7
3.3	Microserviço UserInEvent . . . . .	8
3.3.1	Funcionalidades . . . . .	8
3.3.2	Estrutura . . . . .	8
3.3.3	Diagrama BPMN . . . . .	9
3.4	Microserviço Payment . . . . .	9
3.4.1	Funcionalidades . . . . .	9
3.4.2	Estrutura . . . . .	10
3.4.3	Diagrama BPMN . . . . .	10
3.5	Microserviço Location . . . . .	10
3.5.1	Funcionalidades . . . . .	10

3.5.2	Estrutura . . . . .	11
3.5.3	Diagrama BPMN . . . . .	11
<b>4</b>	<b>Arquitetura</b>	<b>12</b>
4.1	Modelo Arquitetural . . . . .	12
4.2	Ambiente Kubernetes . . . . .	13
4.3	Tipos de Utilizador . . . . .	14
4.4	Requisitos funcionais . . . . .	14
4.5	Requisitos não funcionais . . . . .	15
4.6	Tecnologias . . . . .	15
4.7	Endpoints . . . . .	16
4.7.1	Autenticação . . . . .	16
4.7.2	Utilizadores . . . . .	16
4.7.3	Eventos . . . . .	16
4.7.4	Feedbacks . . . . .	17
4.7.5	Pagamentos . . . . .	17
4.7.6	Bilhetes . . . . .	17
<b>5</b>	<b>Desenvolvimento</b>	<b>18</b>
5.1	Estrutura . . . . .	18
5.2	Autenticação . . . . .	18
5.2.1	Utilização de JWT para Gestão de Tokens . . . . .	18
5.2.2	Encriptação de passwords com Bcrypt . . . . .	19
5.2.3	Redefinição de Senha com Nodemailer . . . . .	19
5.2.4	Logout com Redis . . . . .	19
5.3	Segurança de rotas . . . . .	19
5.3.1	Mecanismo de Verificação de Tokens . . . . .	19
5.3.2	Gestão de Permissões com Middlewares Específicos . . . . .	20
5.4	Notificações PUSH . . . . .	20
5.4.1	Integração Futura com Aplicação Móvel . . . . .	21
5.5	Bilhetes com QR Code . . . . .	21
5.6	Documentação . . . . .	21
<b>6</b>	<b>Dificuldades</b>	<b>22</b>
<b>7</b>	<b>Conclusão</b>	<b>23</b>
<b>8</b>	<b>Bibliografia</b>	<b>24</b>

# Lista de Figuras

3.1	Microserviço User . . . . .	5
3.2	Diagrama BPMN User . . . . .	5
3.3	Subprocesso Recuperação de password . . . . .	6
3.4	Microserviço Event . . . . .	7
3.5	Diagrama BPMN Event . . . . .	7
3.6	Subprocesso cancelar evento . . . . .	8
3.7	Microserviço UserInEvent . . . . .	8
3.8	Diagrama BPMN UserInEvent . . . . .	9
3.9	Microserviço Payment . . . . .	10
3.10	Diagrama BPMN Payment . . . . .	10
3.11	Microserviço Location . . . . .	11
3.12	Diagrama BPMN Location . . . . .	11
4.1	Modelo arquitetural . . . . .	12
4.2	Visualização dos pods . . . . .	14
5.1	Exemplo de rotas no swagger . . . . .	21

# Lista de Tabelas

4.1	Requisitos funcionais . . . . .	14
4.2	Requisitos não funcionais . . . . .	15
4.3	Endpoints da autenticação . . . . .	16
4.4	Endpoints dos utilizadores . . . . .	16
4.5	Endpoints dos eventos . . . . .	16
4.6	Endpoints dos feedbacks . . . . .	17
4.7	Endpoints dos pagamentos . . . . .	17
4.8	Endpoints dos bilhetes . . . . .	17

# 1. Introdução

No âmbito das unidades curriculares de Desenvolvimento de Interfaces Aplicacionais, Arquiteturas e Integração de Sistemas, Sistemas de Computação na Cloud e Bases de Dados Avançadas foi proposto o desenvolvimento de uma Aplicação Móvel.

A aplicação foi nomeada de ***Evention*** e consistirá numa plataforma na qual será possível a criação de eventos e, consequentemente, a disponibilização destes para futuras adesões dos utilizadores.

## 1.1 Descrição

A plataforma permitirá aos utilizadores a criação de eventos de diferentes tipos, sejam eles gratuitos ou pagos, fornecendo informações detalhadas como a data, hora, localização, descrição e número de participantes.

Ao criar um evento, o utilizador terá a opção de definir para o mesmo, um local ou percurso, que será apresentado num mapa interativo, permitindo aos participantes visualizar a localização ou trajeto sugerido e obter direções detalhadas antes da participação no evento. Para além disso, os utilizadores poderão pesquisar e encontrar eventos disponíveis, quer através de um mapa interativo que mostrará eventos próximos do local onde o utilizador se encontra atualmente, ou pela pesquisa direta de uma localização específica, facilitando assim ao utilizador encontrar eventos relevantes que sejam do seu interesse.

Após a confirmação da adesão de um utilizador em um evento, o criador do mesmo receberá uma notificação automática contendo todas as informações relevantes sobre o novo participante e este receberá um código qr que será usado para confirmar a compra do bilhete no local do evento.

Por fim, o utilizador terá a oportunidade de avaliar o evento, de forma a proporcionar um feedback detalhado sobre a sua experiência no evento que participou.

## 1.2 Objetivos

O principal objetivo deste projeto é desenvolver uma aplicação móvel para a criação, gestão e participação em eventos. A aplicação pretende focar-se em:

- **Otimização:** Otimizar a participação em eventos, proporcionando aos utilizadores uma maneira fácil de descobrir eventos, registar-se, e adquirir bilhetes de forma segura;
- **Eficiência:** Garantir um processo de entrada eficiente, através da geração de QR Codes únicos, assegurando uma verificação rápida e automatizada dos participantes nos eventos;
- **Avaliação de eventos:** Incentivar a avaliação dos eventos, dando a possibilidade aos participantes a realização de um comentário relacionado com o evento em questão;
- **Notificações:** Manter os utilizadores informados através de um sistema de notificações em caso de entrada em eventos.

## 1.3 Público-alvo

O público-alvo da aplicação será bastante diversificado, abrangendo tanto organizadores como participantes de eventos, com perfis variados. A plataforma destina-se a:

- **Organizadores de eventos:** Pessoas ou empresas que organizam eventos;
- Pequenas empresas, freelancers, associações ou até promotores de eventos que procurem uma solução integrada para gerir bilhetes, participantes e pagamentos de forma fácil e eficiente;
- **Pessoas em busca de experiências sociais:** Pessoas que desejam encontrar eventos em locais próximos para fazerem novas conexões.

## 2. Identificação e Análise do Problema

### 2.1 Identificação do Problema

Na sociedade atual, os eventos desempenham um papel fundamental para promover a interação social, cultural e profissional. No entanto, indivíduos e pequenas organizações enfrentam dificuldades em organizar eventos devido à falta de recursos e de conhecimentos específicos em gestão, como:

- Gestão dos participantes;
- Pagamentos e Processamento de inscrições;
- Verificação e Controlo de Entradas.

### 2.2 Análise do Problema

Para responder a estas necessidades, a aplicação *Evention* propõe uma solução com funcionalidades específicas que facilitam a gestão dos eventos, permitindo aos organizadores o foco no sucesso do evento e na experiência dos participantes, como:

- **Gestão dos participantes:** A aplicação *Evention* permitirá o registo e a monitorização dos participantes, com acesso a informações detalhadas de cada um;
- **Atualizações de adesões:** A aplicação enviará notificações automáticas para informar o organizador em tempo real sobre as novas adesões ou cancelamentos, garantindo que este tenha uma visão atualizada da lista de participantes do seu evento;
- **Processamento de pagamentos:** *Evention* integra uma plataforma de pagamentos segura que facilita a gestão das inscrições pagas. Esta funcionalidade reduz a carga administrativa, diminui o risco de erros e garante um processamento imediato de inscrições;
- **Controlo eficiente de entradas:** Com um sistema de verificação de entradas que utiliza códigos QR, a aplicação permitirá um controlo rápido e seguro dos acessos ao evento.



## 3. Processo de Negócio

O projeto foi dividido em 5 microsserviços, sendo estes *User*, *Event*, *UserInEvent*, *Payment* e *Location*. Cada um destes microsserviços é responsável por uma parte específica da aplicação, promovendo uma arquitetura escalável e mais fácil de manter.

### 3.1 Microsserviço User

#### 3.1.1 Funcionalidades

O microsserviço *User* foi desenvolvido para gerir todos os aspetos relacionados com a autenticação e a gestão de utilizadores no sistema, sendo responsável pelas seguintes funcionalidades:

- **Registo:** Permitirá a criação de novos utilizadores na plataforma;
- **Login:** Realizará a autenticação dos utilizadores com a verificação das credenciais do utilizador;
- **Recuperação de password:** Permitirá ao utilizador a redefinição da sua *password* caso a tenha esquecido;
- **Desativar utilizador:** Permitirá ao utilizador inativar a sua conta, impedindo o login;
- **Tipos de utilizadores:** Possibilita a existência de vários tipos de utilizador, que neste caso serão (Utilizador, *Eventor* e *Admin*);
- **Gestão de endereços:** Permitirá o armazenamento de dados detalhados do endereço numa tabela separada, facilitando o encapsulamento e a organização das informações de localização do utilizador.

### 3.1.2 Estrutura

Na base de dados, o microserviço será constituído por 3 tabelas (*User*, *UserTypes* e *Adresses*):

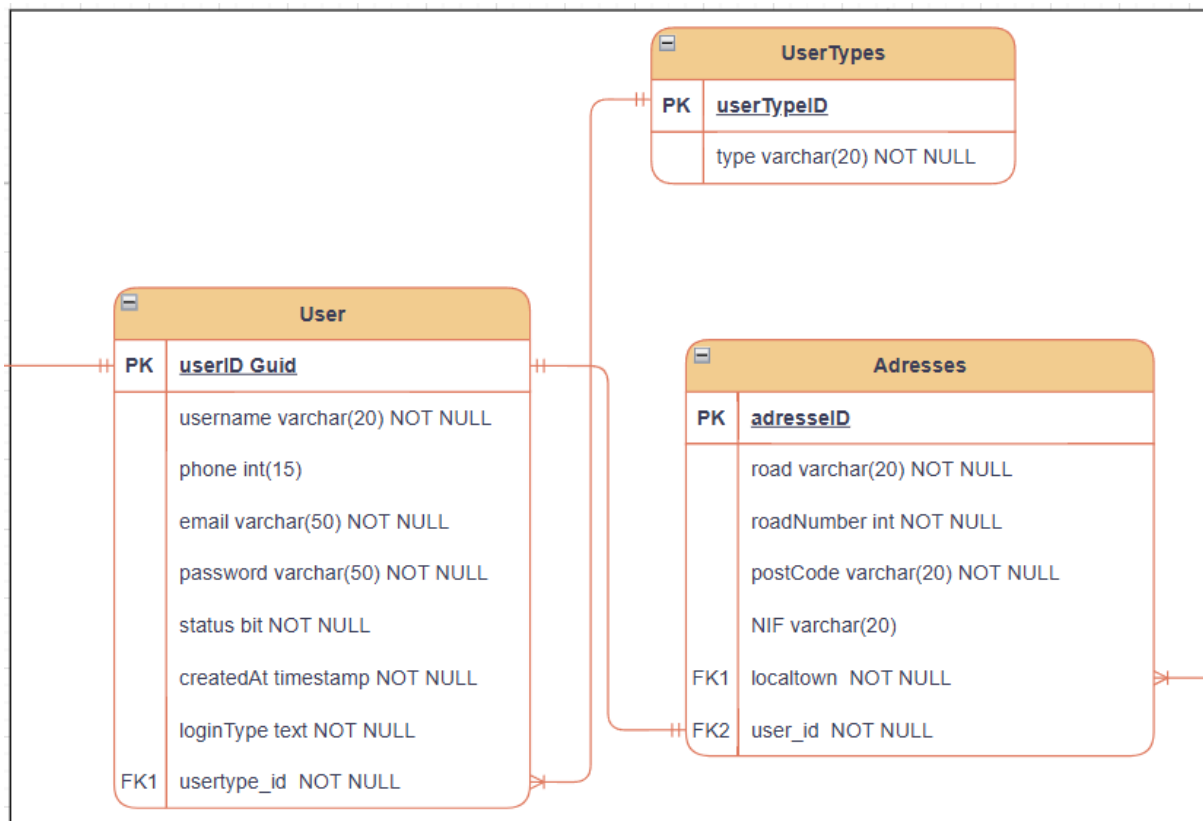


Figura 3.1: Microserviço User

### 3.1.3 Diagrama BPMN

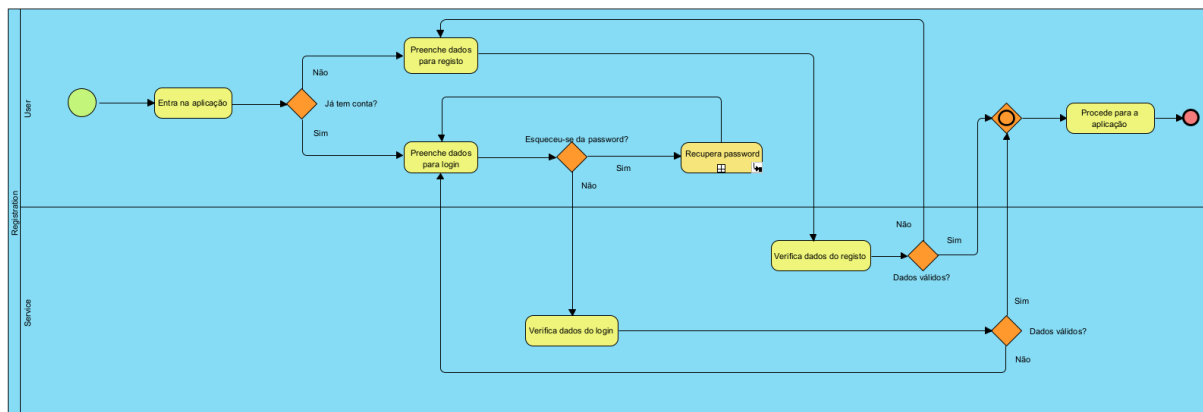


Figura 3.2: Diagrama BPMN User

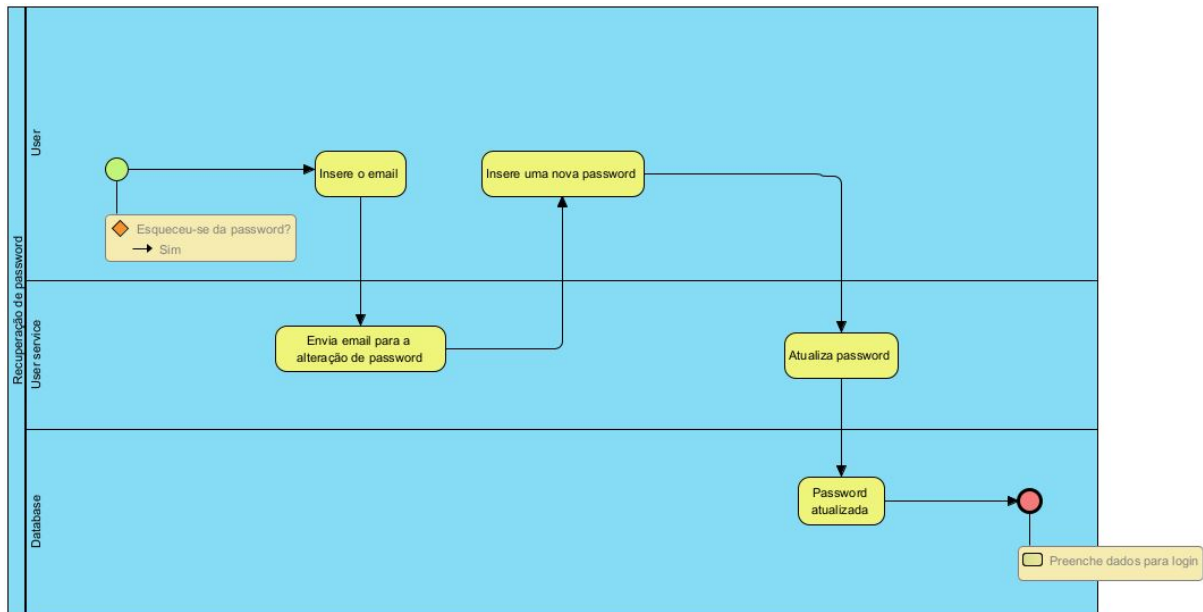


Figura 3.3: Subprocesso Recuperação de password

## 3.2 Microserviço Event

### 3.2.1 Funcionalidades

O microserviço *Event* é responsável pela criação e gestão de eventos na aplicação, incluindo informações relacionadas à localização e ao estado de cada evento. Este terá as seguintes funcionalidades:

- **Criação de Evento:** Permitirá a criação de novos eventos, armazenando informações principais como nome, descrição, data e outros detalhes relevantes para o evento;
- **Gestão de Endereço do Evento:** Possibilitará a definição e armazenamento do endereço onde o evento terá lugar, utilizando uma tabela específica para os dados da localização;
- **Criação de Rotas para o Evento:** Permite a criação de uma rota associada ao evento, caso este inclua um percurso específico (por exemplo, para eventos de caminhada, maratonas, etc.);
- **Gestão de Estado do Evento:** Controla o estado de cada evento ao longo do seu ciclo de vida, incluindo os seguintes estados:
  - **Por Aprovar:** Estado inicial do evento, a aguardar aprovação do administrador para publicação;
  - **Aprovado:** Evento aprovado e visível para os utilizadores;
  - **Cancelado:** Evento que foi cancelado, tornando-se indisponível para os participantes;
  - **Concluído:** Evento realizado e encerrado com sucesso.

### 3.2.2 Estrutura

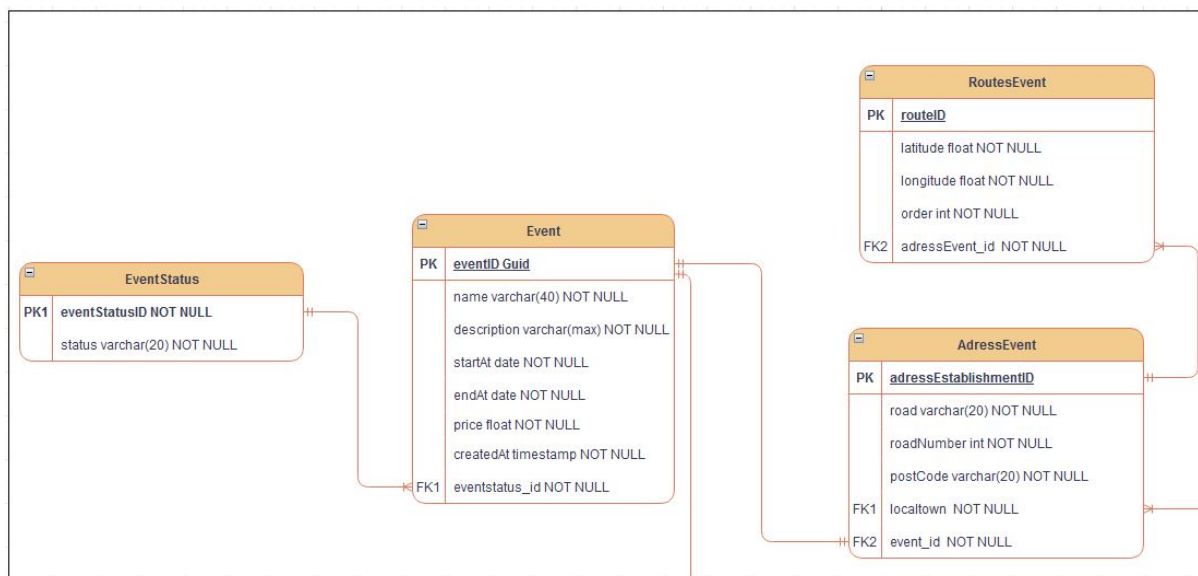


Figura 3.4: Microserviço Event

### 3.2.3 Diagrama BPMN

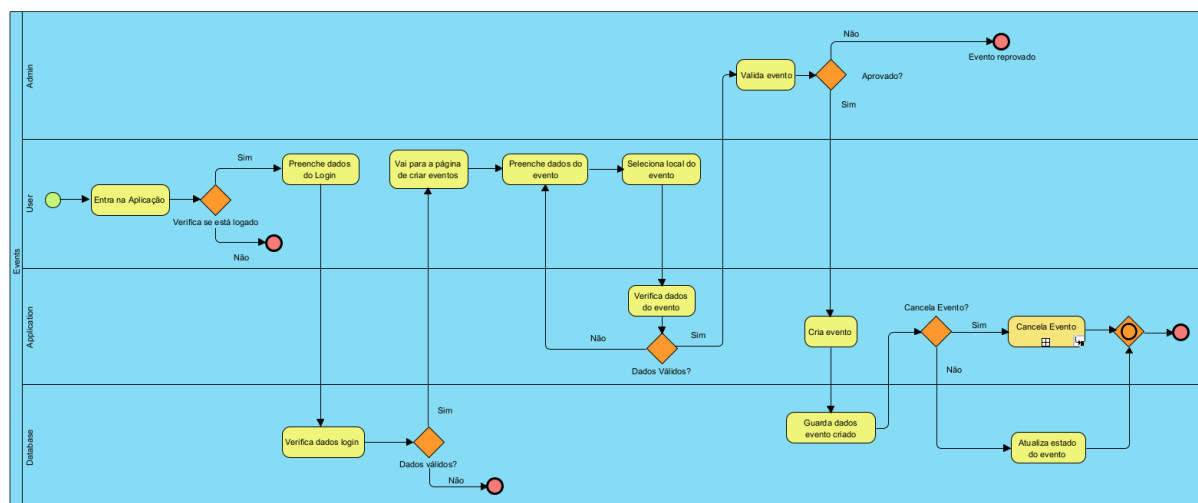


Figura 3.5: Diagrama BPMN Event

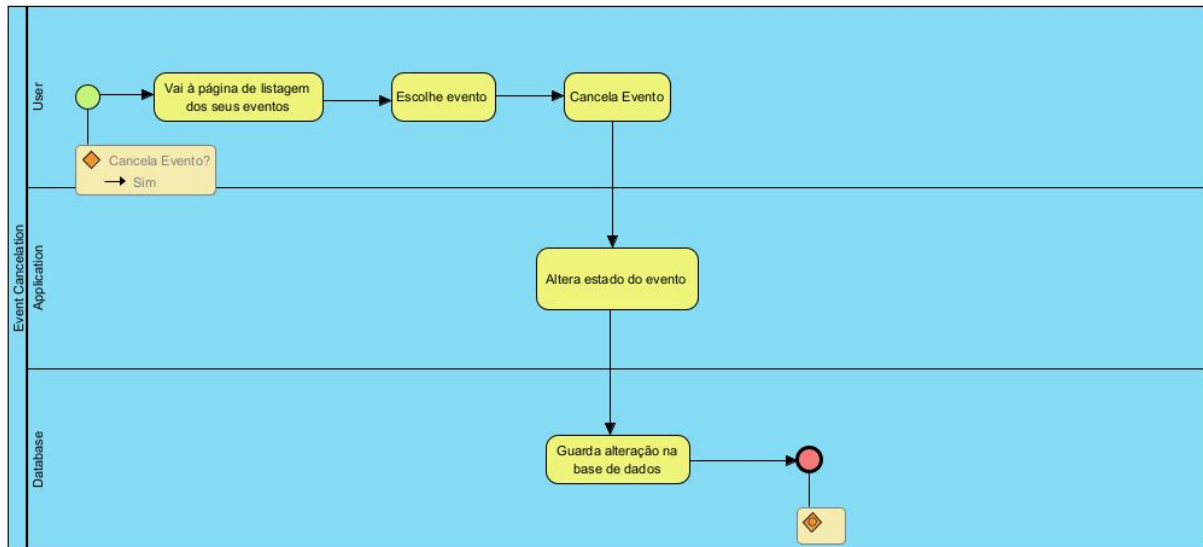


Figura 3.6: Subprocesso cancelar evento

### 3.3 Microserviço UserInEvent

#### 3.3.1 Funcionalidades

O microserviço *UserInEvent* é responsável por armazenar os participantes de cada evento, bem como o respetivo feedback.

- **Registo de Participantes:** Armazena a lista de participantes para cada evento, permitindo que o sistema mantenha registo de todos os utilizadores inscritos;
- **Gestão de Feedback dos Participantes:** Permite que os participantes deixem *feedback* sobre o evento em que participaram. Este *feedback* é guardado no sistema e pode ser utilizado para avaliar a qualidade dos eventos e realizar melhorias futuras.

#### 3.3.2 Estrutura

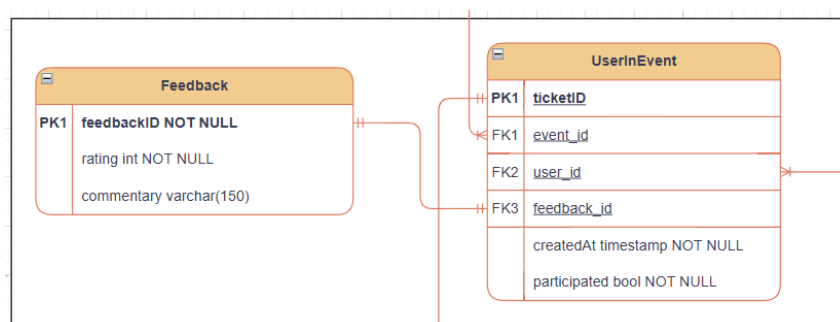


Figura 3.7: Microserviço UserInEvent

### 3.3.3 Diagrama BPMN

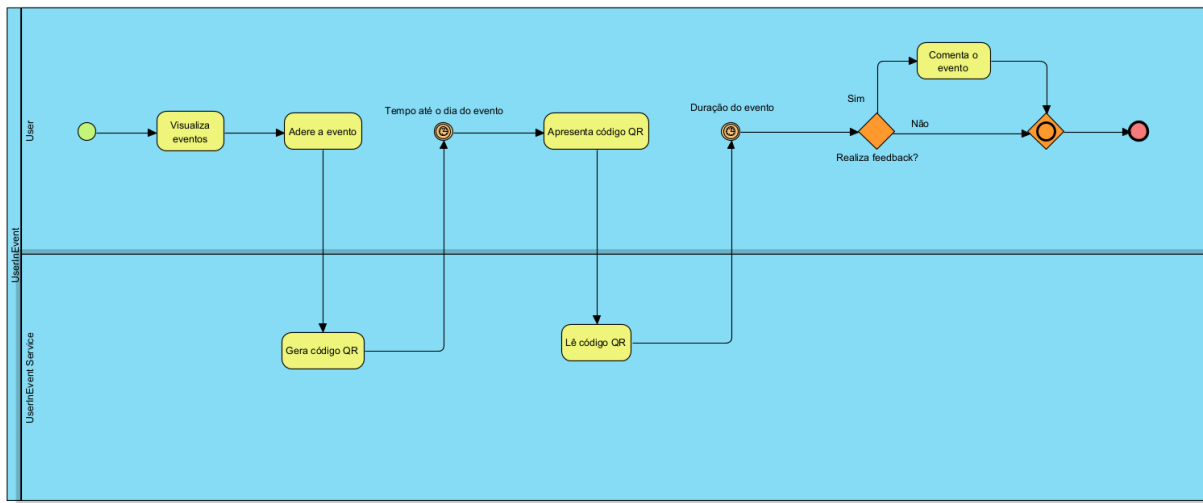


Figura 3.8: Diagrama BPMN UserInEvent

## 3.4 Microserviço Payment

### 3.4.1 Funcionalidades

O microserviço *Payment* é responsável por gerir todos os pagamentos realizados na plataforma, assegurando um registo estruturado e seguro de todas as transações financeiras, além de permitir o controlo do estado de cada pagamento.

Este microserviço contém as seguintes funcionalidades:

- **Registo de Pagamentos:** Armazena todos os pagamentos efetuados na plataforma, incluindo informações relevantes como o valor, a data e o utilizador associado ao pagamento, permitindo uma visão detalhada de todas as transações;
- **Gestão de Estado do Pagamento:** Inclui uma tabela específica para o estado do pagamento, o que permite monitorizar o ciclo de vida de cada transação com os seguintes estados:
  - **Pendente:** Pagamento iniciado mas ainda não concluído;
  - **Concluído:** Pagamento realizado com sucesso;
  - **Cancelado:** Pagamento que foi cancelado.

### 3.4.2 Estrutura

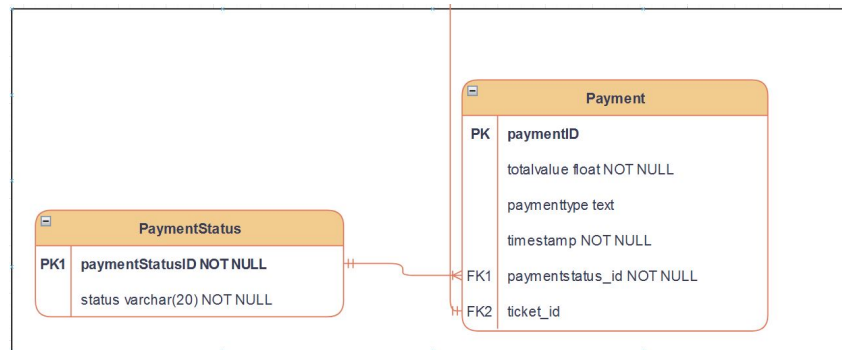


Figura 3.9: Microserviço Payment

### 3.4.3 Diagrama BPMN

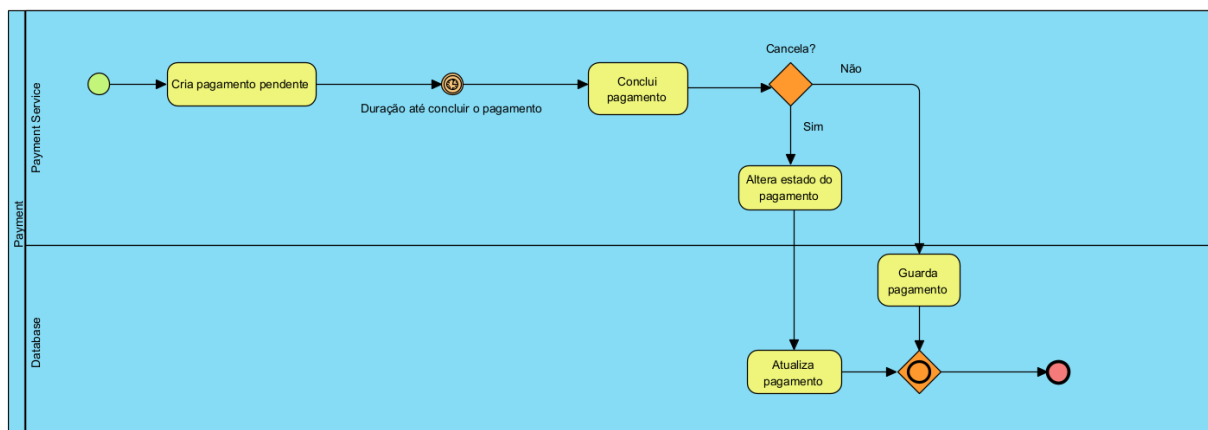


Figura 3.10: Diagrama BPMN Payment

## 3.5 Microserviço Location

### 3.5.1 Funcionalidades

O microserviço *Location* é responsável por armazenar e gerir informações da localização que serão utilizadas em várias tabelas da plataforma. Este microserviço centraliza os dados de cidades e outras localizações relevantes, evitando duplicação de informações. Este contém as seguintes funcionalidades:

- **Armazenamento de localizações:** Contém uma tabela dedicada às localizações, com informações sobre as cidades, permitindo que várias tabelas no sistema referenciem essas informações de forma unificada.

### 3.5.2 Estrutura

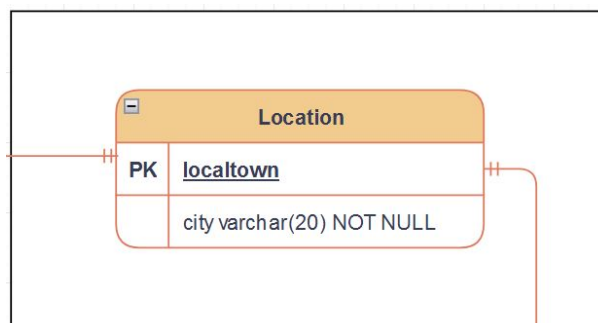


Figura 3.11: Microserviço Location

### 3.5.3 Diagrama BPMN

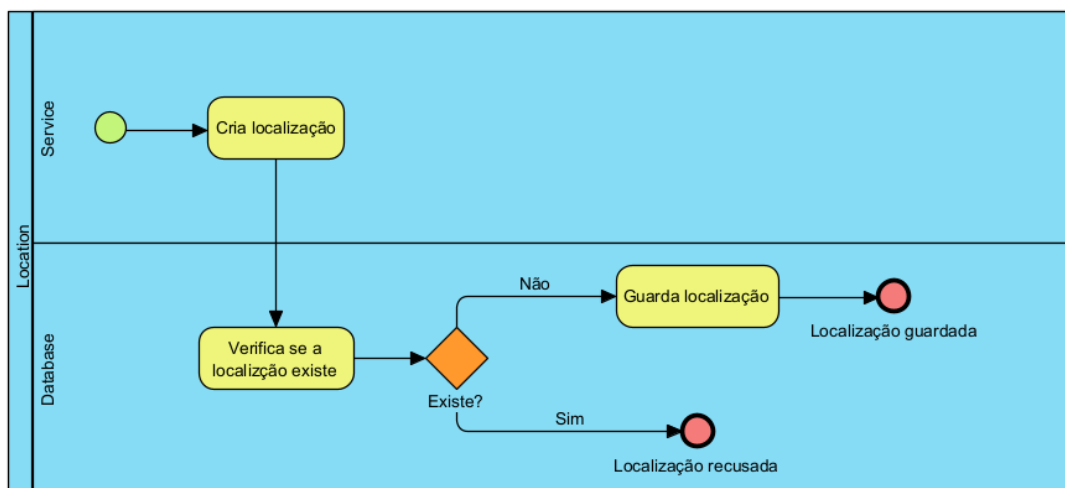


Figura 3.12: Diagrama BPMN Location



## 4. Arquitetura

### 4.1 Modelo Arquitetural

A arquitetura da nossa aplicação segue o modelo de microsserviços, onde cada funcionalidade é implementada de forma independente, garantindo escalabilidade. Os microsserviços comunicam entre si através do Kubernetes, que gere os serviços e encaminha as requisições de forma eficiente.

Cada microsserviço tem a sua própria base de dados, permitindo uma gestão autónoma e evitando dependências entre os serviços.

Adicionalmente, o microsserviço *User Service* integra-se com o *Redis* para armazenamento em *cache*.

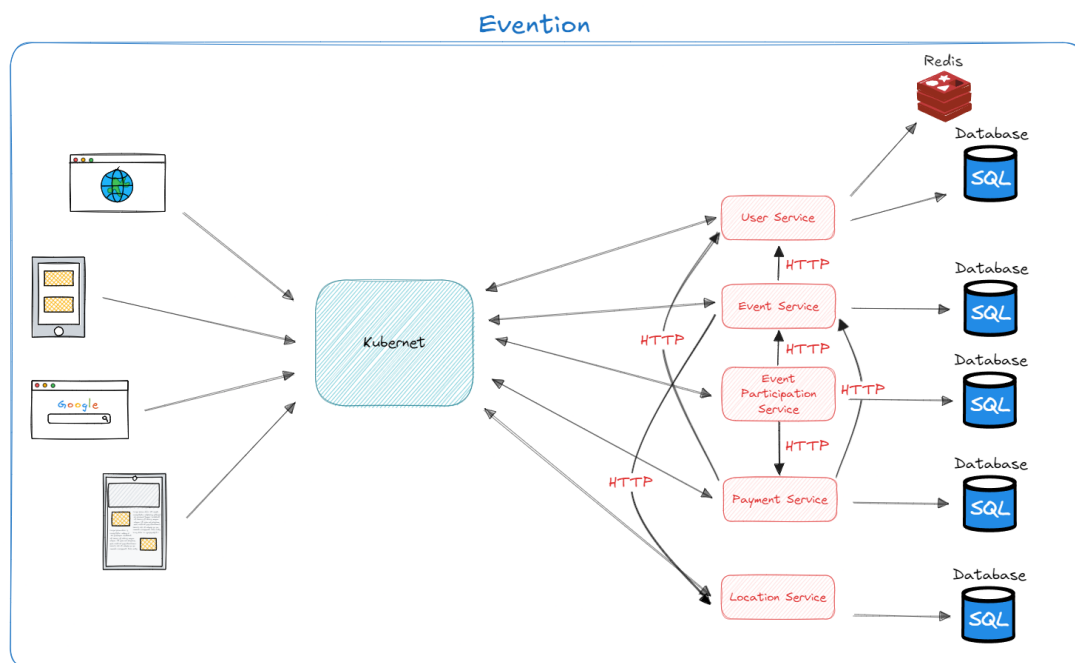


Figura 4.1: Modelo arquitetural

## 4.2 Ambiente Kubernetes

Para o desenvolvimento da nossa arquitetura, utilizamos um ambiente Kubernetes com Minikube, permitindo criar um cluster local onde cada microserviço é executado em contêineres isolados.

Também integramos o GitHub Actions ao processo, automatizando a criação e publicação das imagens Docker no Docker Hub. Essas imagens são posteriormente utilizadas para configurar os ficheiros de deployment de cada microserviço no cluster.

Usamos um Horizontal Pod Autoscaler (HPA) no microserviço do user só para demonstrar que é possível escalar o número de pods de um microserviço conforme a sua utilização e fazendo então que o mesmo serviço seja replicado em até 10 vezes.

```
apiVersion: autoscaling/v2
kind: HorizontalPodAutoscaler
metadata:
  name: userservice-hpa
spec:
  scaleTargetRef:
    apiVersion: apps/v1
    kind: Deployment
    name: userservice-deployment
  minReplicas: 1
  maxReplicas: 10
  metrics:
  - type: Resource
    resource:
      name: cpu
      target:
        type: Utilization
        averageUtilization: 80
  - type: Resource
    resource:
      name: memory
      target:
        type: Utilization
        averageUtilization: 80
```

Podemos ver também na figura abaixo todos os pods existentes no nosso cluster

```
PS C:\Users\pedro> kubectl get pods
```

NAME	READY	STATUS	RESTARTS	AGE
eventservice-db-0	1/1	Running	12 (3h19m ago)	14d
eventservice-deployment-84878f6598-g2r7d	1/1	Running	0	3h1m
locationservice-db-0	1/1	Running	12 (3h19m ago)	14d
locationservice-deployment-5945bc4fcd-kpn4s	1/1	Running	3 (3h19m ago)	28h
paymentservice-db-0	1/1	Running	3 (3h19m ago)	28h
paymentservice-deployment-f5b8754c7-kngng	1/1	Running	1 (3h19m ago)	5h8m
userineventservice-db-0	1/1	Running	3 (3h19m ago)	28h
userineventservice-deployment-764588f6b6-cr5vr	1/1	Running	1 (3h19m ago)	4h13m
userservice-db-0	1/1	Running	5 (3h19m ago)	3d4h
userservice-deployment-7865bf9b7f-5wk2q	1/1	Running	0	3h15m
userservice-redis-0	1/1	Running	3 (3h19m ago)	29h

Figura 4.2: Visualização dos pods

### 4.3 Tipos de Utilizador

- **Utilizador** – Representa os utilizadores que pretendem aderir a eventos;
- **Anunciante** – Representa os utilizadores que pretendem criar eventos;
- **Administrador** – Órgão que gere a aplicação.

### 4.4 Requisitos funcionais

ID	Utilizador	Requisito	MoSCoW
1.1	Utilizador / Anunciante	Registo	Must
1.2		Login	Must
1.3		Edição de perfil	Should
1.4		Pesquisa de eventos	Must
1.5		Adesão em eventos	Must
1.6		Cancelar a adesão do evento	Must
1.7		Consultar eventos aderidos	Should
1.8		Consultar dados do evento	Must
1.9		Avaliar evento	Could
1.10		Efetuar pagamento	Must
1.11		Criar evento	Must
2.1	Admin Anuncia.	Apagar evento	Should
2.2		Consultar adesões do evento	Must
2.3		Cancelar adesões	Should
3.1	Admin	Aprovar eventos	Must
3.2		Consultar eventos existentes	Must
3.3		Consultar utilizadores registados	Must

Tabela 4.1: Requisitos funcionais

## 4.5 Requisitos não funcionais

ID	Requisito	Descrição	MoSCoW
1	Desempenho	A aplicação deverá ser rápida.	Must
2	Segurança	Encriptação dos dados do utilizador.	Must
3	Usabilidade	Após a entrada na aplicação deverá ser imediatamente apresentado os eventos principais, como também filtros de pesquisa.	Should
4	Compatibilidade	A nossa aplicação apenas será suportada em telemóveis a partir da versão 7.0.	Should

Tabela 4.2: Requisitos não funcionais

## 4.6 Tecnologias

- **Mockups:** Utilizaremos a plataforma Figma para criar os *mockups*.
- **API:** A *API* da plataforma será desenvolvida na linguagem Node.js.
- **Base de Dados:** Os dados da plataforma serão armazenados em uma base de dados no *PostgreSQL*.
- **Docker:** Utilizaremos o Docker para dividir a aplicação em containers, garantindo a consistência do ambiente de desenvolvimento. Também será utilizado o *DockerHub* para guardar e publicar as imagens geradas.
- **GitHub:** Utilizaremos o Github para o devido controlo de versões do código desenvolvido e utilizaremos as Github Actions de forma a automatizar a publicação das imagens e dos testes.
- **MiniKube:** Utilizaremos a plataforma *Minikube* para montar um ambiente de *cluster kubernetes* local na nossa máquina.
- **ORM:** Utilizaremos o Prisma como ORM para gerir a comunicação entre a aplicação e a base de dados.
- **Testes Unitários:** Para realizar os testes unitários utilizaremos a framework Jest.
- **Testes de Integração:** Para realizar os testes de integração utilizaremos o *Postman*.

## 4.7 Endpoints

### 4.7.1 Autenticação

Request	Rota	Descrição
POST	api/auth/login	Login do utilizador.
POST	api/auth/logout	Logout do utilizador.
POST	api/auth/reset-password	Faz reset da password do utilizador.
POST	api/auth/send-reset-token	Envia token para email do utilizador.

Tabela 4.3: Endpoints da autenticação

### 4.7.2 Utilizadores

Request	Rota	Descrição
POST	api/users/create	Registar um utilizador.
POST	api/users/validateEmail	Verifica se Email já existe.
UPDATE	api/users/userID	Desativar um utilizador.
UPDATE	api/users/change-password	Altera Password do utilizador.
DELETE	api/users/userID	Apagar um utilizador.
GET	api/users	Listar todos os utilizadores.
GET	api/users/byemail/email	Procura utilizadores por email.

Tabela 4.4: Endpoints dos utilizadores

### 4.7.3 Eventos

Request	Rota	Descrição
GET	api/events	Listar todos os eventos.
POST	api/events	Criar um evento.
GET	api/events/my	Listar os eventos do utilizador.
PUT	api/events/id	Atualizar um evento específico.
DELETE	api/events/my	Apagar um evento específico.
PUT	api/events/id/status	Aprovar um evento pendente.
PUT	api/events/id/cancel	Cancelar um evento em específico.
GET	api/events/suspended	Listar todos eventos suspensos.

Tabela 4.5: Endpoints dos eventos

#### 4.7.4 Feedbacks

Request	Rota	Descrição
POST	api/feedbacks/ticketID	Criar feedback para um bilhete.
UPDATE	api/feedbacks/feedbackID	Editar um feedback.
DELETE	api/feedbacks/feedbackID	Apagar um feedback.
GET	api/feedbacks	Listar todos os feedbacks.
GET	api/feedbacks/feedbackID	Obter um feedback.
GET	api/feedbacks/event/eventID	Listar feedbacks de um evento.

Tabela 4.6: Endpoints dos feedbacks

#### 4.7.5 Pagamentos

Request	Rota	Descrição
POST	api/payments	Criar um pagamento.
GET	api/payments/my	Listar os meus pagamentos.
GET	api/payments	Listar todos os pagamentos.
GET	api/payments/paymentID	Listar um pagamento.
GET	api/payments/ticket/ticketID	Listar os pagamentos do bilhete.
UPDATE	api/payments/paymentID	Alterar um pagamento.
UPDATE	api/payments/paymentID/cancel	Cancelar um pagamento.
DELETE	api/payments/paymentID	Apagar um pagamento.

Tabela 4.7: Endpoints dos pagamentos

#### 4.7.6 Bilhetes

Request	Rota	Descrição
POST	api/tickets	Criar um bilhete.
DELETE	api/tickets/ticketID	Apagar um bilhete.
GET	api/tickets	Listar todos os bilhetes.
GET	api/tickets/ticketID	Obter um bilhete pelo ID.
GET	api/tickets/event/eventID	Obter bilhetes de um evento.
GET	api/tickets/my	Devolve os tickets do utilizador.
UPDATE	api/tickets/ticketID	Alterar um bilhete.
POST	api/tickets/qrcode/ticketID	Gerar QR Code do bilhete.
UPDATE	api/tickets/qrcode/read/ticketID	Ler QR Code do bilhete.

Tabela 4.8: Endpoints dos bilhetes

## 5. Desenvolvimento

### 5.1 Estrutura

Antes de implementar as funcionalidades relacionadas com a autenticação, o projeto foi estruturado seguindo o padrão arquitetural MVC. Esta abordagem separa as responsabilidades do sistema em três camadas principais:

- **Controllers:** responsáveis por receber as requisições, delegar a lógica para os serviços e devolver as respostas apropriadas.
- **Services:** contêm a lógica de negócio e interagem com a base de dados para processar as informações necessárias.
- **Routes:** definem os endpoints da API e direcionam as requisições para os controladores correspondentes.
- **Models:** representam a estrutura dos dados e facilitam a interação com a base de dados.

### 5.2 Autenticação

A autenticação foi desenvolvida utilizando uma combinação de tecnologias modernas e práticas recomendadas para garantir segurança e usabilidade. Foram implementados os seguintes componentes:

#### 5.2.1 Utilização de JWT para Gestão de Tokens

Utilizamos JSON Web Tokens para gerir os tokens dos utilizadores. No nosso sistema, os tokens JWT são gerados durante o processo de login, contendo informações essenciais como o identificador do utilizador, o nome de utilizador, o tipo de utilizador e o e-mail. Estes tokens são assinados com uma chave secreta, armazenada de forma segura no ficheiro de ambiente (`.env`), garantindo que apenas o servidor pode gerar e validar os tokens.

Os tokens terão uma validade de 24 horas, sendo incluídos no cabeçalho das requisições subsequentes. No lado do servidor, validamos cada requisição para garantir que o token não está expirado e que é legítimo.

### 5.2.2 Encriptação de passwords com Bcrypt

Para proteger as passwords dos utilizadores, utilizamos a biblioteca `bcryptjs` para encriptação. Durante o registo ou redefinição de senha, transformamos a senha fornecida pelo utilizador num hash antes de a armazenar na base de dados. No login, comparamos a senha introduzida pelo utilizador com o hash armazenado.

### 5.2.3 Redefinição de Senha com Nodemailer

Para proporcionar uma funcionalidade de redefinição de password, implementámos um sistema que utiliza a biblioteca `nodemailer`. Este sistema funciona da seguinte forma:

1. O utilizador solicita a redefinição de password, fornecendo o seu endereço de e-mail.
2. Verificamos a existência do utilizador e geramos um token JWT temporário, com validade de 1 hora.
3. Enviamos este token para o e-mail do utilizador, acompanhado de instruções para redefinir a password.
4. O utilizador utiliza o token para definir uma nova password, que encriptamos antes de armazenar na base de dados.

### 5.2.4 Logout com Redis

Para garantir que os tokens JWT não são reutilizados após o logout, implementámos um mecanismo de *blacklist* utilizando o Redis. Durante o logout, adicionamos o token do utilizador a um conjunto denominado `blacklistedTokens`. Todas as requisições subsequentes verificam se o token está na lista negra antes de prosseguir.

Com este sistema de autenticação, conseguimos garantir um elevado nível de segurança e simplicidade para os utilizadores.

## 5.3 Segurança de rotas

No nosso sistema, implementámos uma camada de segurança nas rotas para garantir que apenas utilizadores com as permissões adequadas podem aceder a determinados endpoints. Esta abordagem foi realizada utilizando *middlewares*, que verificam a autenticação e as permissões antes de processar qualquer requisição.

### 5.3.1 Mecanismo de Verificação de Tokens

Utilizamos tokens JWT para autenticação, sendo que o *middleware* `authMiddleware` é responsável por validar o token recebido por parâmetro. Este *middleware* segue os seguintes passos:

1. Verificamos se o token foi incluído no cabeçalho da requisição.



2. Consultamos o Redis para verificar se o token está na lista negra, garantindo que tokens revogados não possam ser reutilizados.
3. Decodificamos e validamos o token utilizando a chave secreta armazenada no ambiente.
4. Incluímos os dados decodificados do utilizador no objeto da requisição para uso posterior.

### 5.3.2 Gestão de Permissões com Middlewares Específicos

Para diferenciar os níveis de acesso entre tipos de utilizadores, desenvolvemos *middlewares* adicionais para validar as permissões específicas:

#### Middleware de Anunciante

O `advertiserMiddleware` permite que apenas utilizadores do tipo anunciante possam aceder a endpoints específicos. Da mesma forma, verificamos o tipo de utilizador e rejeitamos a requisição caso não possua as permissões adequadas.

#### Middleware de Administrador

O `adminMiddleware` garante que apenas utilizadores com o tipo de utilizador correspondente a administrador possam aceder a determinadas rotas. O *middleware* verifica o campo `userType` presente no token decodificado e bloqueia a requisição se o utilizador não for administrador.

Por fim, os *middlewares* serão integrados nas rotas para impor a autenticação e as permissões adequadas da seguinte forma:

```
router.post('/events', verifyToken, eventController.createEvent);
router.get('/events/my', verifyToken, verifyAdvertiser,
  eventController.getUserEvents);
```

## 5.4 Notificações PUSH

No sistema, implementámos o código necessário para lidar com notificações push utilizando a biblioteca `firebase-admin`. Esta funcionalidade será responsável por enviar notificações aos utilizadores através de tokens de registo gerados pela aplicação móvel, garantindo uma comunicação eficiente e em tempo real.

A lógica das notificações push foi implementada em duas camadas principais: *controller* e *service*. No *controller*, os dados da notificação, como título e corpo, são configurados e enviados ao *service*, que é responsável por construir e enviar a mensagem através do Firebase.

### 5.4.1 Integração Futura com Aplicação Móvel

É importante salientar que o token utilizado neste momento é fictício, uma vez que a aplicação móvel ainda está em desenvolvimento. Assim, a funcionalidade de envio de notificações push encontra-se implementada, mas será totalmente integrada apenas quando a aplicação móvel estiver concluída e os tokens reais forem disponibilizados.

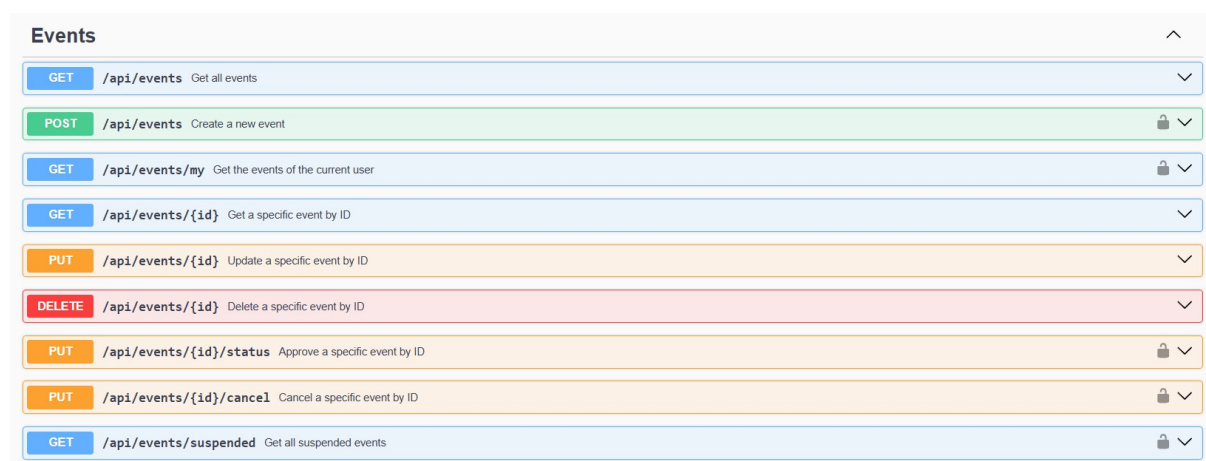
## 5.5 Bilhetes com QR Code

De forma a possibilitar a marcação de presenças dos participantes por parte do anunciante do evento, foi definida a criação de um QR Code único que irá conter o id do respetivo bilhete. Este QR Code será lido na aplicação e será enviado o id do bilhete para a API, de forma a marcar a participação do participante.

## 5.6 Documentação

Para facilitar a comunicação e a utilização da nossa API, implementámos uma documentação utilizando o Swagger para cada microserviço. O Swagger permitiu que a documentação da API fosse gerada automaticamente com base nos endpoints definidos no código.

Abaixo, apresentamos um exemplo de rotas, onde é possível visualizar os endpoints documentados de forma clara e interativa:



Events		
GET	/api/events	Get all events
POST	/api/events	Create a new event
GET	/api/events/my	Get the events of the current user
GET	/api/events/{id}	Get a specific event by ID
PUT	/api/events/{id}	Update a specific event by ID
DELETE	/api/events/{id}	Delete a specific event by ID
PUT	/api/events/{id}/status	Approve a specific event by ID
PUT	/api/events/{id}/cancel	Cancel a specific event by ID
GET	/api/events/suspended	Get all suspended events

Figura 5.1: Exemplo de rotas no swagger

## 6. Dificuldades

Durante o desenvolvimento e a implementação do nosso sistema, enfrentámos várias dificuldades que exigiram estudo e experimentação para serem superadas. Entre os principais desafios, destacamos o uso de Kubernetes para orquestração de containers.

A adoção de Kubernetes revelou-se um desafio significativo no início do projeto. Tivemos de aprender a configurar os *manifests* corretamente, criando *Deployments* e *Services* para cada microserviço para suportar a nossa aplicação. Um dos pontos críticos foi compreender como os *pods* interagem entre si e como gerir o escalonamento manual das instâncias.

## 7. Conclusão

Ao longo deste trabalho, explorámos a arquitetura de uma API de microsserviços utilizando Kubernetes, destacando a sua aplicação no contexto do nosso projeto e as principais vantagens, desafios e os elementos críticos para uma implementação bem-sucedida.

Contudo, a implementação de uma API baseada em microsserviços requer um planeamento detalhado, incluindo a definição clara das responsabilidades de cada serviço, a escolha de padrões de comunicação e a escolha dos dados armazenados pelo mesmo.

Neste trabalho foi possível colocar em prática os conhecimentos obtidos na unidade curricular de Arquiteturas e Integração de Sistemas, de forma a construir a API utilizando a arquitetura de microsserviços.

## 8. Bibliografia

<https://nodejs.org/docs/latest/api/>

<https://www.w3schools.com/nodejs/>

<https://jwt.io/introduction>

<https://www.ibm.com/docs/pt-pt>

<https://medium.com/cloud-native-daily/how-to-deploy-a-containerized-node-js-api-using-kubernetes-8192c54ec465>

<https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkthrough/>

<https://medium.com/@ravisharma23523/sending-notifications-to-mobile-devices-with-firebase-cloud-messaging-fcm-in-node-js-8fe3faead58b>

<https://www.prisma.io/docs>