

# Evention

Mestrado em Engenharia Informática

Pedro Miguel Gomes Martins

Luís Miguel Da Costa Anjo

Diogo Gomes Silva

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

PROJETO DE COMPUTAÇÃO NA CLOUD

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
<b>2</b>	<b>Objetivos</b>	<b>2</b>
2.1	Público-alvo . . . . .	2
2.2	Utilizadores . . . . .	2
2.3	Requisitos funcionais . . . . .	3
2.4	Requisitos não funcionais . . . . .	3
<b>3</b>	<b>Identificação e Análise do Problema</b>	<b>4</b>
3.1	Identificação do Problema . . . . .	4
3.2	Análise do Problema . . . . .	4
<b>4</b>	<b>Processo de Negócio</b>	<b>5</b>
4.1	Microserviço User . . . . .	5
4.1.1	Funcionalidades . . . . .	5
4.1.2	Estrutura . . . . .	6
4.1.3	Diagrama BPMN . . . . .	6
4.2	Microserviço Event . . . . .	7
4.2.1	Funcionalidades . . . . .	7
4.2.2	Estrutura . . . . .	8
4.2.3	Diagrama BPMN . . . . .	8
4.3	Microserviço UserInEvent . . . . .	9
4.3.1	Funcionalidades . . . . .	9
4.3.2	Estrutura . . . . .	9
4.3.3	Diagrama BPMN . . . . .	10
4.4	Microserviço Payment . . . . .	10
4.4.1	Funcionalidades . . . . .	10
4.4.2	Estrutura . . . . .	11

4.4.3	Diagrama BPMN . . . . .	11
4.5	Microserviço Location . . . . .	11
4.5.1	Funcionalidades . . . . .	11
4.5.2	Estrutura . . . . .	12
4.5.3	Diagrama BPMN . . . . .	12
<b>5</b>	<b>Arquitetura</b>	<b>13</b>
5.1	Modelo Arquitetural . . . . .	13
5.2	Ambiente Kubernetes . . . . .	14
5.3	Tecnologias . . . . .	15
5.4	Endpoints . . . . .	16
5.4.1	Autenticação . . . . .	16
5.4.2	Utilizadores . . . . .	16
5.4.3	Eventos . . . . .	16
5.4.4	Feedbacks . . . . .	17
5.4.5	Pagamentos . . . . .	17
5.4.6	Bilhetes . . . . .	17
<b>6</b>	<b>Implementação</b>	<b>18</b>
6.1	Estrutura . . . . .	18
6.2	Autenticação . . . . .	18
6.2.1	Utilização de JWT para Gestão de Tokens . . . . .	18
6.2.2	Encriptação de passwords com Bcrypt . . . . .	19
6.2.3	Redefinição de Senha com Nodemailer . . . . .	19
6.2.4	Logout com Redis . . . . .	19
6.3	Segurança de rotas . . . . .	19
6.3.1	Mecanismo de Verificação de Tokens . . . . .	19
6.3.2	Gestão de Permissões com Middlewares Específicos . . . . .	20
6.4	Documentação . . . . .	21
<b>7</b>	<b>Melhorias realizadas</b>	<b>22</b>
7.1	Documentação aprimorada do swagger . . . . .	22
7.2	Documentação aprimorada do código . . . . .	23
7.3	Controlo de erros <i>multilanguage</i> . . . . .	25
7.4	Implementação da <i>API Gateway</i> . . . . .	26
7.5	Implementação de comunicação <i>HTTPS</i> . . . . .	28
7.6	Registo e Login com Google . . . . .	29

7.7	Notificações com RabbitMQ . . . . .	29
7.8	Arquitetura Atual . . . . .	31
<b>8</b>	<b>Análise de Resultados</b>	<b>32</b>
8.1	Testes HTTP vs HTTPS . . . . .	32
8.2	Impacto do protocolo HTTPS nos Serviços . . . . .	33
8.3	Análise Final dos Resultados . . . . .	36
<b>9</b>	<b>Futuras implementações</b>	<b>37</b>
9.1	Integração com uma API de Previsão Meteorológica . . . . .	37
9.2	Implementação de um Chat para Eventos . . . . .	37
9.3	Implementação de uma Super App . . . . .	37
9.4	Migração do Projeto para a Cloud . . . . .	38
9.5	Implementação de Bases de Dados Partilhadas . . . . .	38
<b>10</b>	<b>Conclusão</b>	<b>39</b>
<b>11</b>	<b>Link do Repositório</b>	<b>40</b>
<b>12</b>	<b>Bibliografia</b>	<b>41</b>

# Lista de Figuras

4.1	Microserviço User . . . . .	6
4.2	Diagrama BPMN User . . . . .	6
4.3	Subprocesso Recuperação de password . . . . .	7
4.4	Microserviço Event . . . . .	8
4.5	Diagrama BPMN Event . . . . .	8
4.6	Subprocesso cancelar evento . . . . .	9
4.7	Microserviço UserInEvent . . . . .	9
4.8	Diagrama BPMN UserInEvent . . . . .	10
4.9	Microserviço Payment . . . . .	11
4.10	Diagrama BPMN Payment . . . . .	11
4.11	Microserviço Location . . . . .	12
4.12	Diagrama BPMN Location . . . . .	12
5.1	Modelo arquitetural . . . . .	13
5.2	Visualização dos pods . . . . .	15
6.1	Exemplo de rotas no swagger . . . . .	21
7.1	Exemplo de uma rota detalhada no swagger . . . . .	23
7.2	Modelo Arquitetural utilizando API Gateway . . . . .	26
7.3	Modelo Arquitetural com API Gateway utilizando HTTPS . . . . .	28
7.4	Rotas Swagger para Registo e Login com Google . . . . .	29
7.5	Portal do RabbitMQ . . . . .	30
7.6	Visualização das notificações recebidas pelo worker . . . . .	30
7.7	Modelo Arquitetural Atualizado . . . . .	31
8.1	Teste com API Gateway utilizando HTTP . . . . .	32
8.2	Teste com API Gateway utilizando HTTPS . . . . .	33
8.3	Métricas iniciais dos pods com API Gateway utilizando HTTP . . . . .	34
8.4	Teste aos pods com API Gateway utilizando HTTP . . . . .	34

8.5	Métricas iniciais dos pods com API Gateway utilizando HTTPS . . . . .	35
8.6	Teste aos pods com API Gateway utilizando HTTPS . . . . .	35

# Lista de Tabelas

2.1	Requisitos funcionais . . . . .	3
2.2	Requisitos não funcionais . . . . .	3
5.1	Endpoints da autenticação . . . . .	16
5.2	Endpoints dos utilizadores . . . . .	16
5.3	Endpoints dos eventos . . . . .	16
5.4	Endpoints dos feedbacks . . . . .	17
5.5	Endpoints dos pagamentos . . . . .	17
5.6	Endpoints dos bilhetes . . . . .	17

# 1. Introdução

No âmbito das unidades curriculares de Projeto de Computação na Cloud, 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.



## 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.

### 2.1 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 procuram 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.

### 2.2 Utilizadores

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

## 2.3 Requisitos funcionais

ID	Utilizador	Requisito	MoSCoW
1.1	Utilizador / Eventor	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	Eventor	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 2.1: Requisitos funcionais

## 2.4 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 para que o utilizador possa pesquisar rapidamente os seus desejos de evento.	Should
4	Compatibilidade	A nossa aplicação apenas será suportada em telemóveis a partir da versão 7.0.	Should

Tabela 2.2: Requisitos não funcionais

## 3. Identificação e Análise do Problema

### 3.1 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.

### 3.2 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.

## 4. 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.

### 4.1 Microsserviço User

#### 4.1.1 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.

### 4.1.2 Estrutura

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

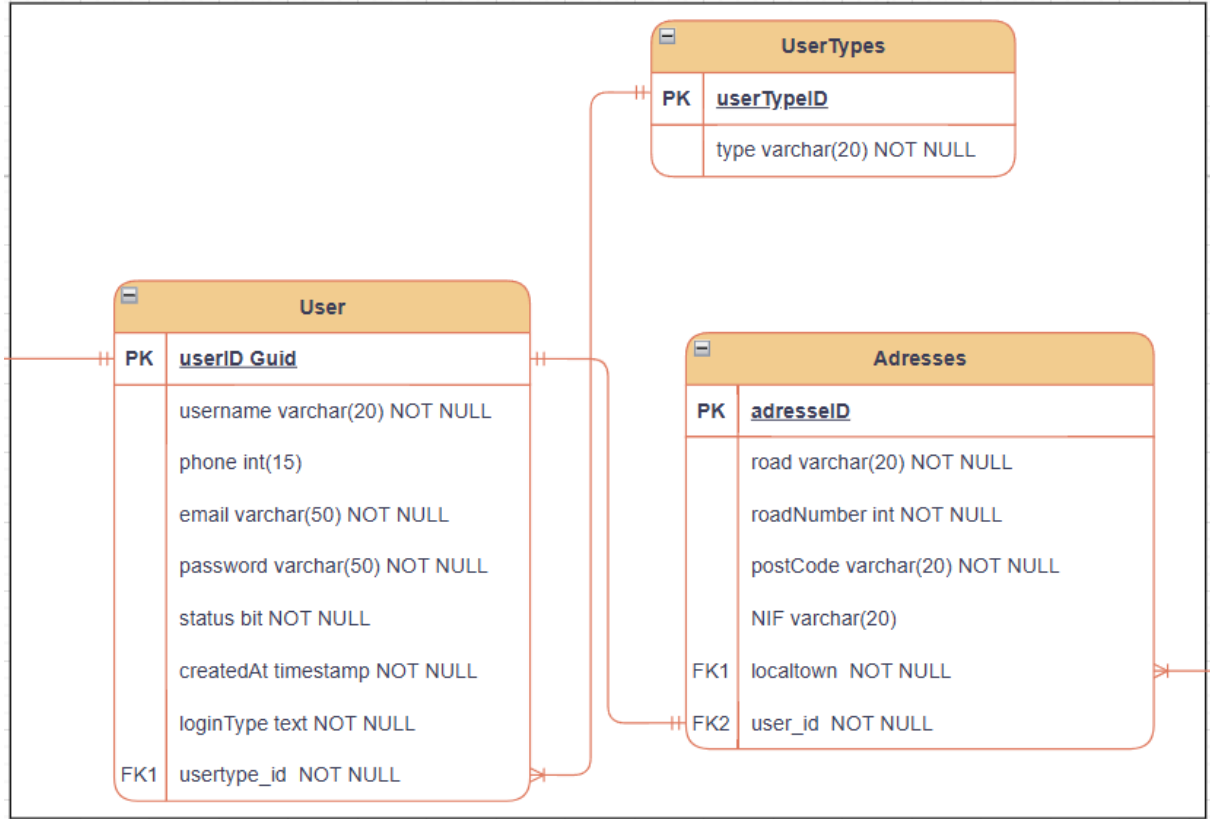


Figura 4.1: Microserviço User

### 4.1.3 Diagrama BPMN

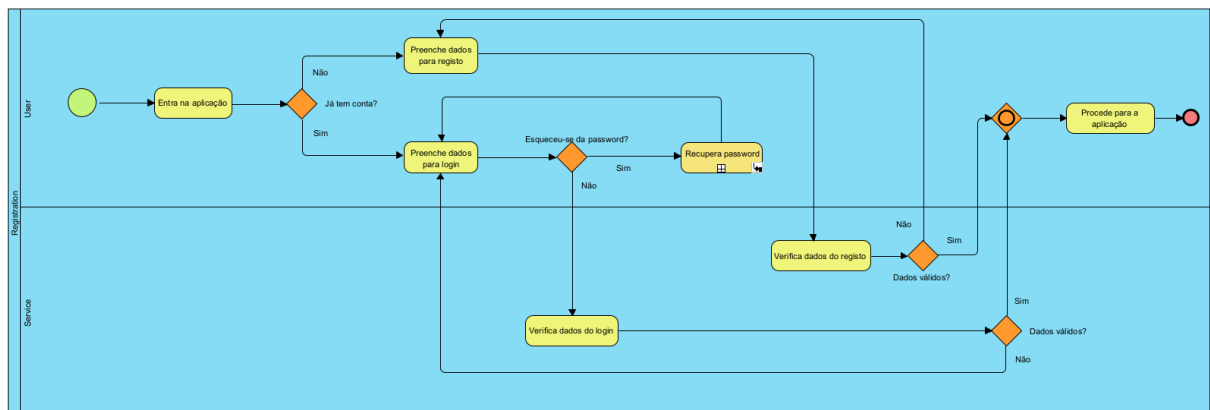


Figura 4.2: Diagrama BPMN User

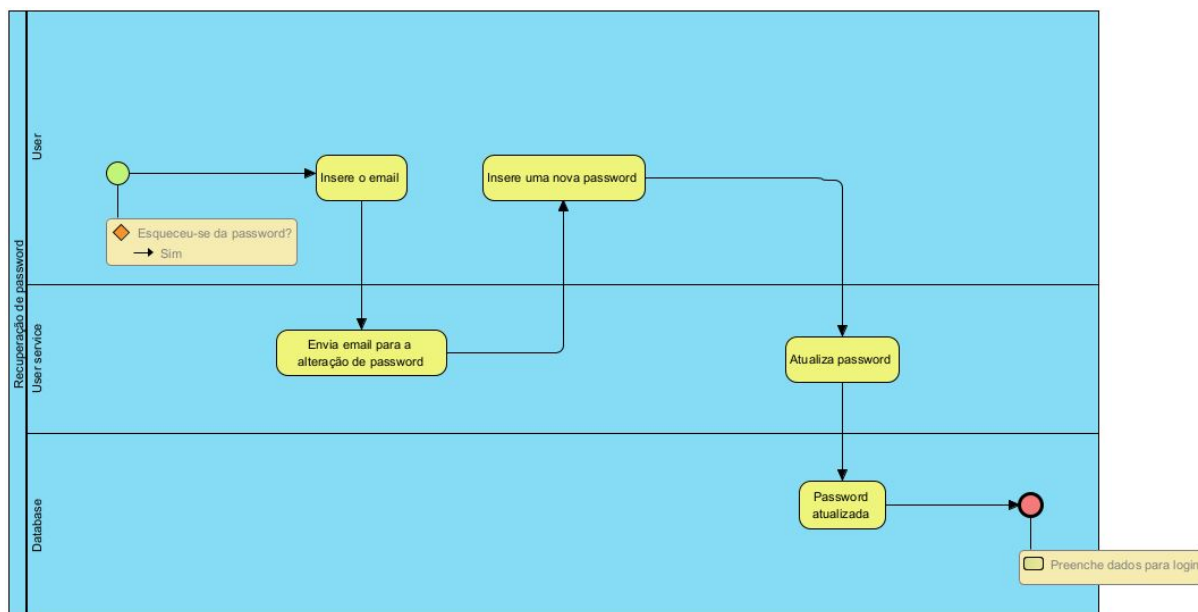


Figura 4.3: Subprocesso Recuperação de password

## 4.2 Microserviço Event

### 4.2.1 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.

## 4.2.2 Estrutura

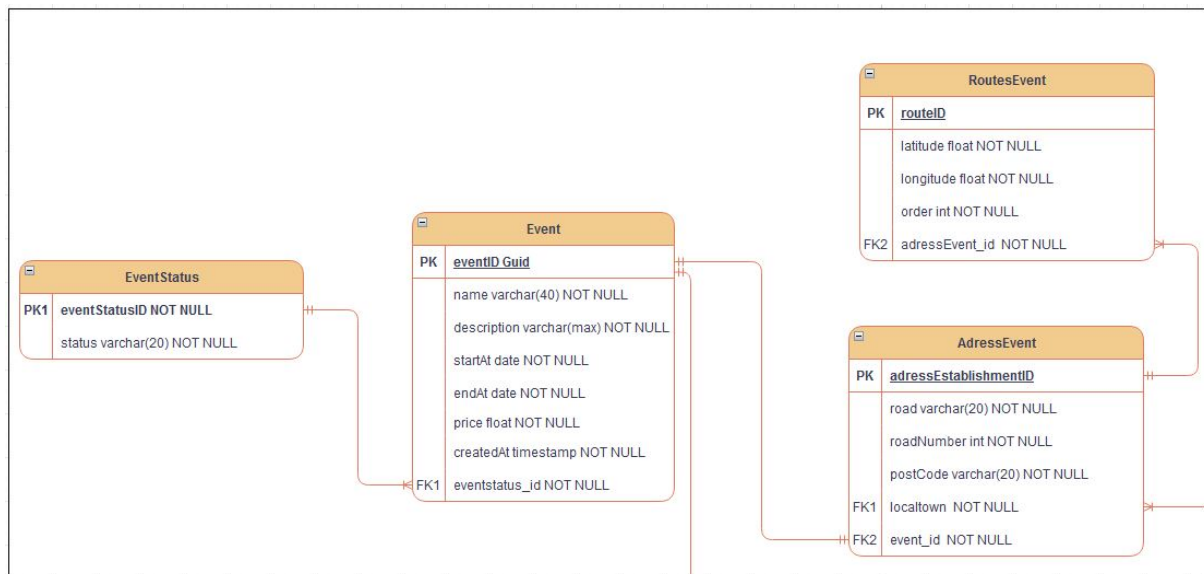


Figura 4.4: Microserviço Event

## 4.2.3 Diagrama BPMN

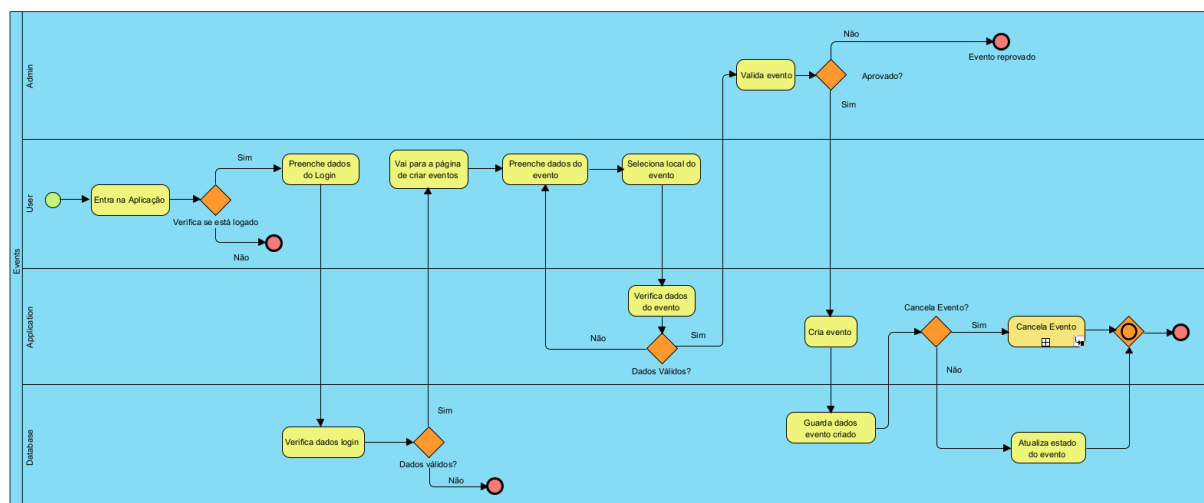


Figura 4.5: Diagrama BPMN Event

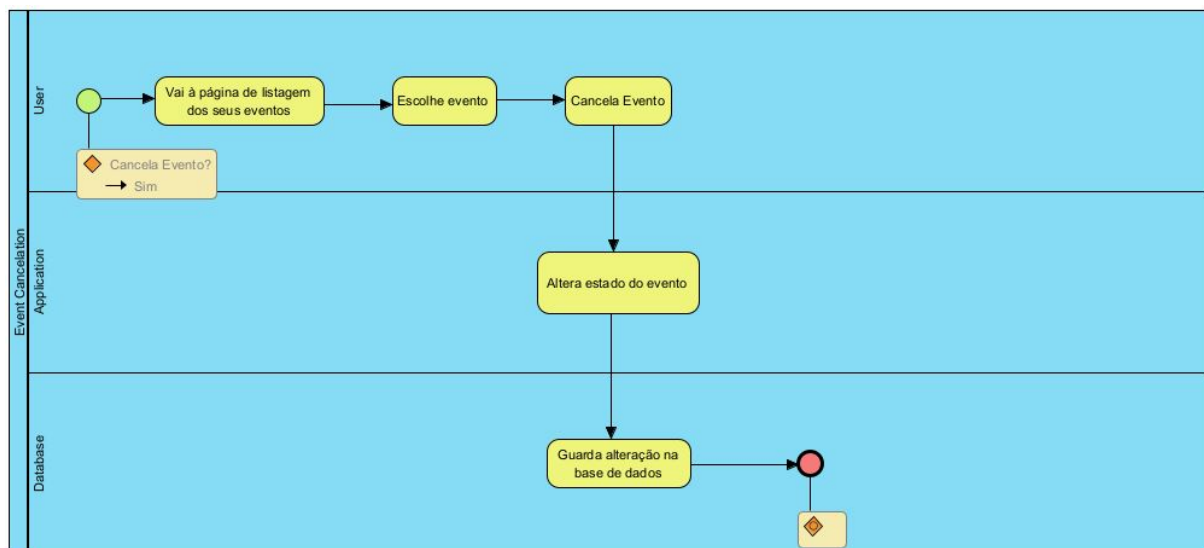


Figura 4.6: Subprocesso cancelar evento

## 4.3 Microserviço UserInEvent

### 4.3.1 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.

### 4.3.2 Estrutura

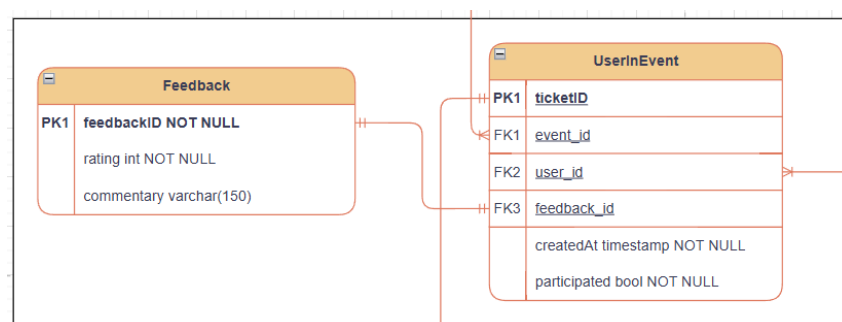


Figura 4.7: Microserviço UserInEvent



### 4.3.3 Diagrama BPMN

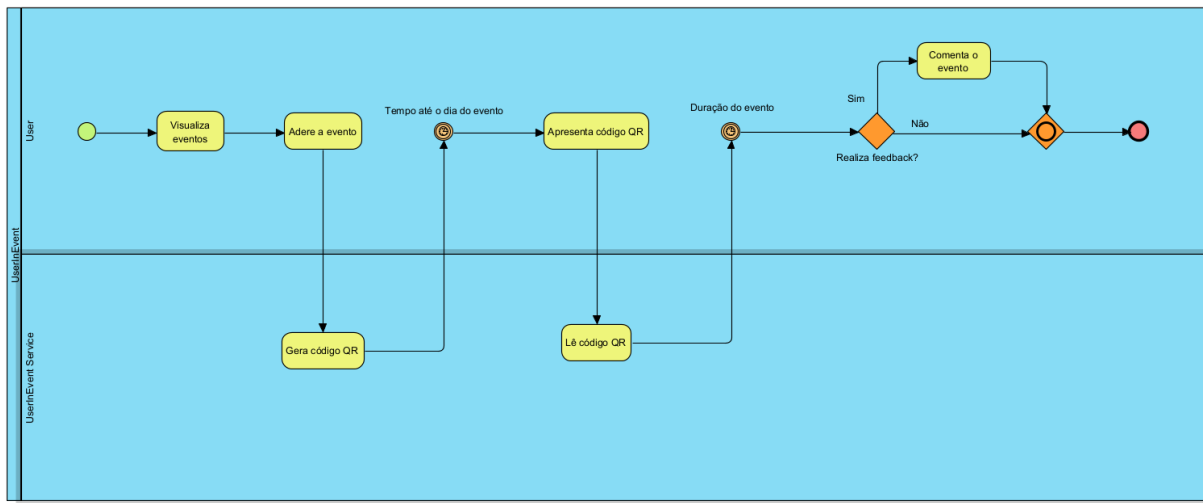


Figura 4.8: Diagrama BPMN UserInEvent

## 4.4 Microserviço Payment

### 4.4.1 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.

## 4.4.2 Estrutura

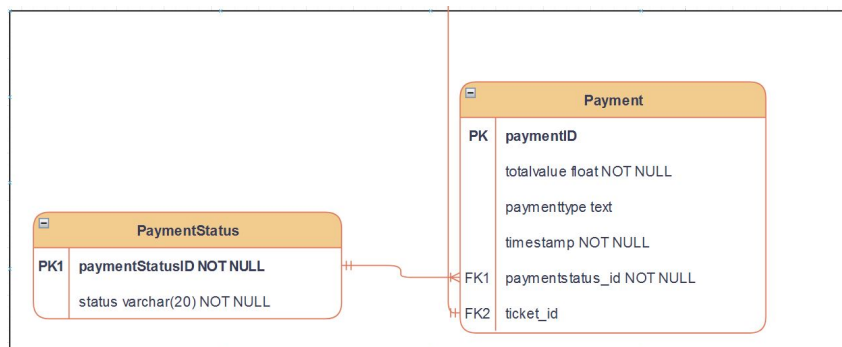


Figura 4.9: Microserviço Payment

## 4.4.3 Diagrama BPMN

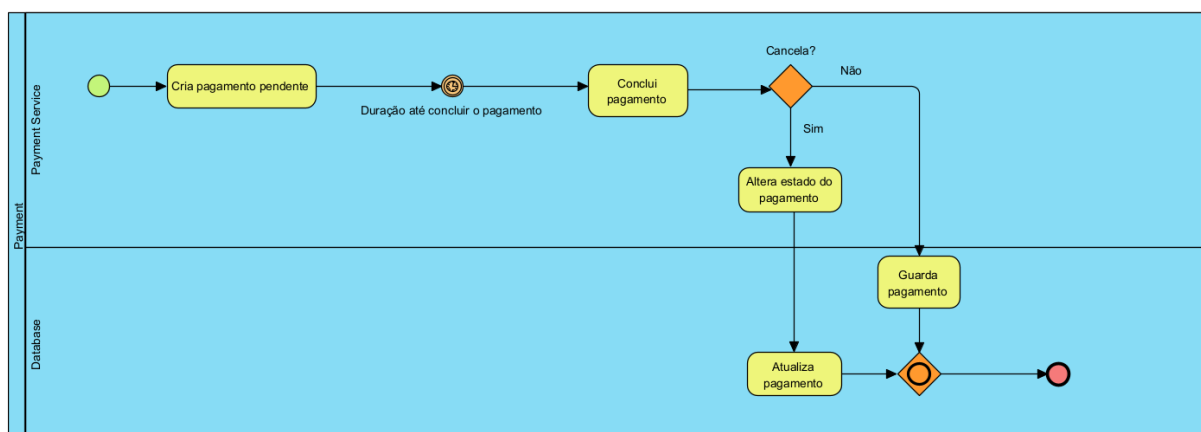


Figura 4.10: Diagrama BPMN Payment

## 4.5 Microserviço Location

### 4.5.1 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.

## 4.5.2 Estrutura



Figura 4.11: Microserviço Location

## 4.5.3 Diagrama BPMN

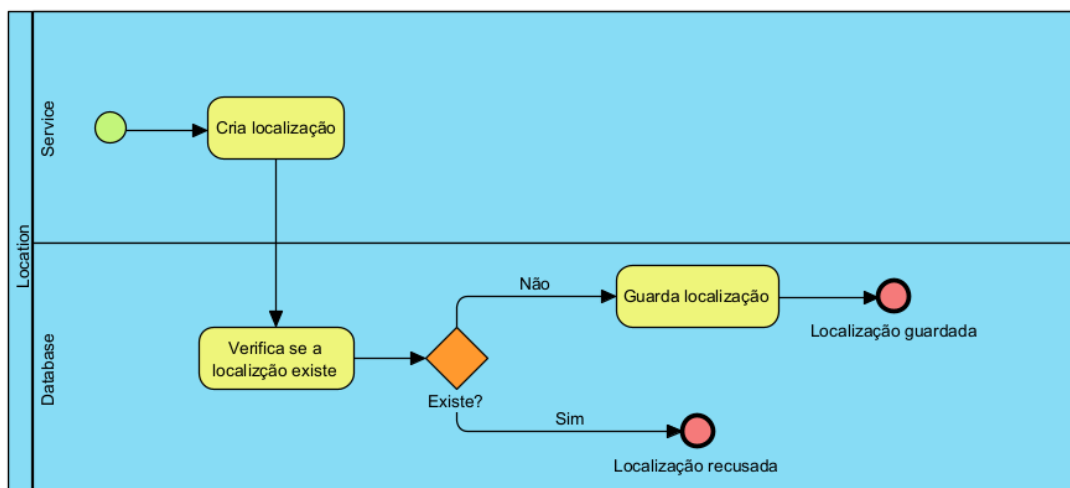


Figura 4.12: Diagrama BPMN Location

## 5. Arquitetura

### 5.1 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 diretamente através do Cluster Kubernetes.

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*.

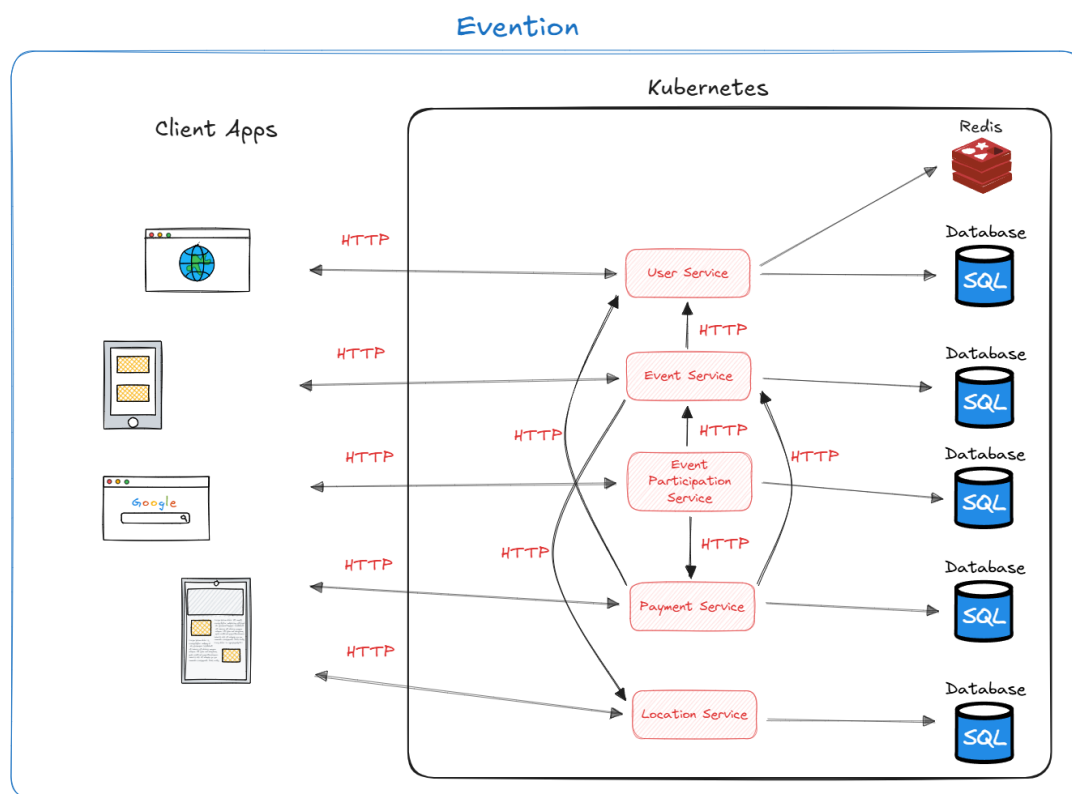


Figura 5.1: Modelo arquitetural

## 5.2 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
payment-service-db-0	1/1	Running	3 (3h19m ago)	28h
payment-service-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 5.2: Visualização dos pods

## 5.3 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*.

## 5.4 Endpoints

### 5.4.1 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 5.1: Endpoints da autenticação

### 5.4.2 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 5.2: Endpoints dos utilizadores

### 5.4.3 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 5.3: Endpoints dos eventos

#### 5.4.4 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 5.4: Endpoints dos feedbacks

#### 5.4.5 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 5.5: Endpoints dos pagamentos

#### 5.4.6 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 5.6: Endpoints dos bilhetes



## 6. Implementação

### 6.1 Estrutura

Antes de implementar as funcionalidades relacionadas com a autenticação, o projeto foi estruturado seguindo o padrão arquitetural MVC (*Model-View-Controller*). 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.

### 6.2 Autenticação

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

#### 6.2.1 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.

### 6.2.2 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.

### 6.2.3 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.

### 6.2.4 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.

## 6.3 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.

### 6.3.1 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.

### 6.3.2 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.

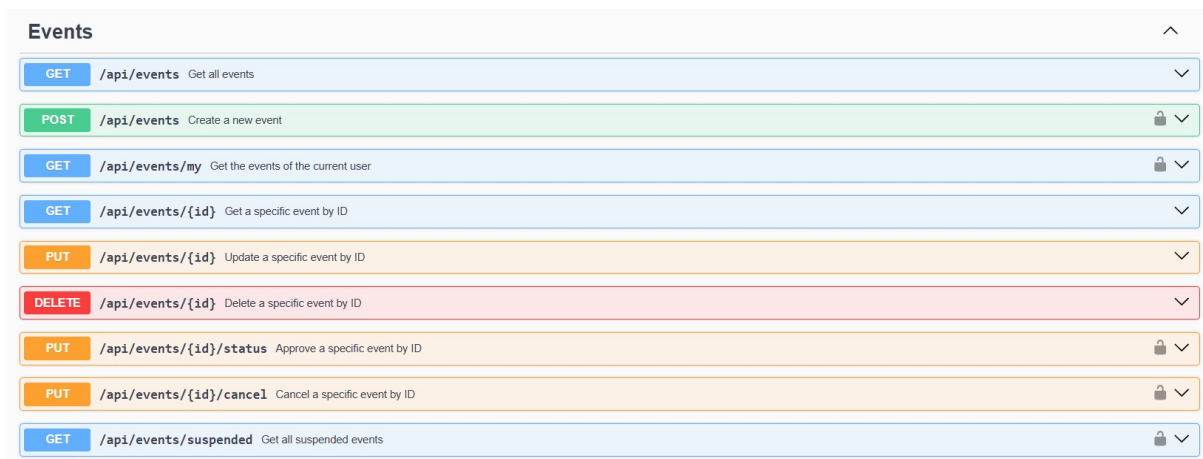
Por fim, os *middlewares* serão integrados nas rotas para impor a autenticação e as permissões adequadas da seguinte forma:

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

## 6.4 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 6.1: Exemplo de rotas no swagger

## 7. Melhorias realizadas

Neste capítulo, apresentamos as melhorias implementadas no âmbito da unidade curricular *Projeto de Computação na Cloud*, com o objetivo de otimizar o desempenho, a segurança e a usabilidade do sistema. As melhorias abrangem vários aspetos do projeto, incluindo a documentação, a comunicação entre microsserviços e a experiência do utilizador na interação com erros.

Entre as principais otimizações realizadas, destacamos o aperfeiçoamento da documentação com o Swagger, facilitando a compreensão e utilização do serviço "Event" da API. Além disso, melhorámos a comunicação entre microsserviços para garantir maior eficiência e fiabilidade no processamento de dados. Outra melhoria significativa foi a implementação da deteção de idioma, permitindo que os erros de backend sejam apresentados na língua do utilizador, proporcionando uma experiência mais intuitiva. De forma a reforçar também a segurança das comunicações, passámos a utilizar pedidos *HTTPS* em vez de *HTTP*. Por fim, foi implementado um sistema de notificações utilizando o RabbitMQ.

Nos próximos tópicos, detalhamos cada uma destas melhorias e o respetivo impacto no projeto.

### 7.1 Documentação aprimorada do swagger

Para melhorar a usabilidade e a compreensão da API, aprimorámos a documentação com o Swagger, fornecendo descrições mais detalhadas para cada rota e respetivo funcionamento. A nova documentação inclui explicações claras sobre o propósito de cada endpoint, os parâmetros necessários, os diferentes códigos de resposta e exemplos de erros, facilitando a utilização da API pelos programadores e garantindo maior previsibilidade no desenvolvimento.

Por exemplo, a rota `GET /events`, responsável por recuperar todos os eventos disponíveis, agora apresenta:

1. **Resumo e descrição detalhada:** Explica o objetivo da rota, indicando que retorna uma lista de eventos ou um erro 404 caso não existam eventos disponíveis.
2. **Códigos de resposta documentados:** Especificação clara das respostas possíveis, como 200 OK para sucesso, 404 Not Found para ausência de eventos e 500 Internal Server Error para falhas inesperadas no servidor.
3. **Exemplos estruturados:** Os exemplos de resposta foram melhorados para incluir mensagens descritivas, como "message": "No events found", permitindo aos programadores compreender rapidamente o formato esperado.

A seguinte imagem ilustra a nova documentação do Swagger, destacando a organização das informações e a clareza das descrições para cada rota.

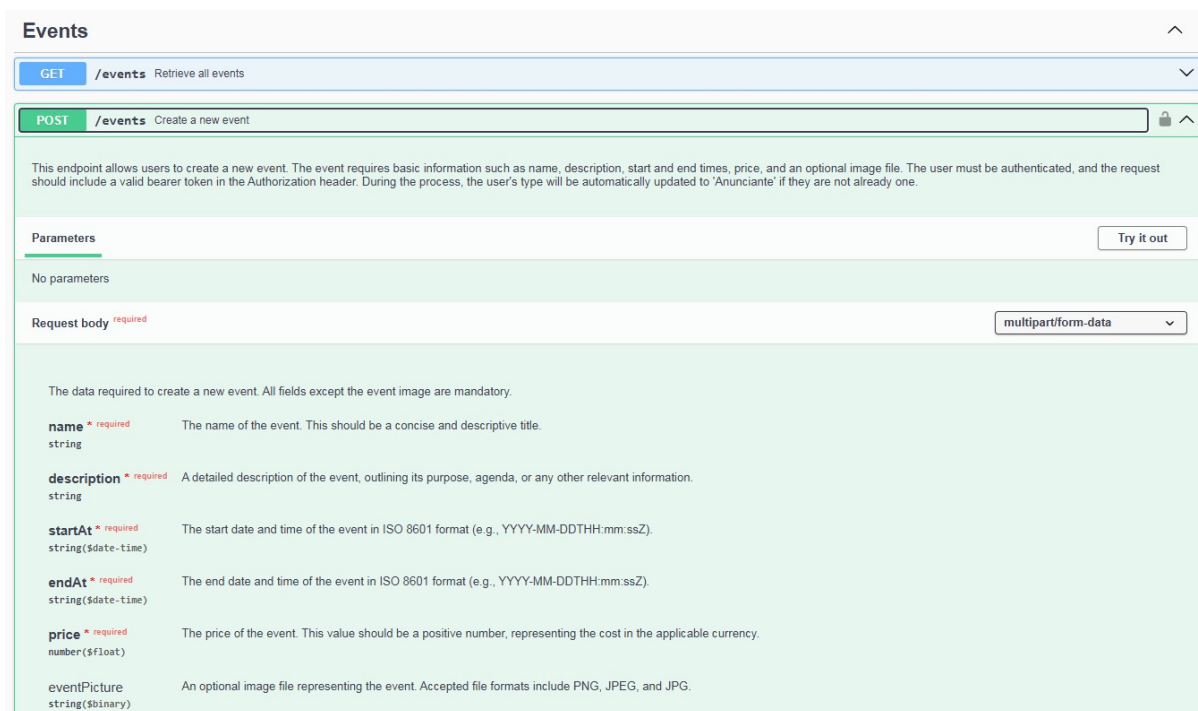


Figura 7.1: Exemplo de uma rota detalhada no swagger

## 7.2 Documentação aprimorada do código

Com o objetivo de melhorar a clareza e a manutenção do código, aprimorámos a documentação do controller responsável pela gestão de eventos. Foram adicionados comentários detalhados no formato JSDoc, assegurando que cada função do controller esteja devidamente documentada.

A nova documentação inclui a descrição de cada rota, os parâmetros esperados (incluindo cabeçalhos e corpo da requisição) e os possíveis retornos. Isto facilita a compreensão do fluxo de dados, melhora a integração com outros serviços e auxilia futuros programadores na manutenção e evolução do código.

Para ilustrar as melhorias na documentação do código, apresentamos abaixo um exemplo de como a documentação de uma das funções do controller foi aprimorada:

```
/**
 * Get all events
 * @route {GET} /events
 * @returns {Array} List of events
 * @description Fetches all events from the database.
 * If no events are found, it returns a 404 response.
 */
async getAllEvents(req, res) {
  const lang = req.headers['accept-language'] || 'en';
  const errorMessages = loadErrorMessages(lang);
  try {
    const events = await eventService.getAllEvents();

    if (events == null || events.length === 0) {
      return res.status(404).json({ message: errorMessages.NO_EVENTS_FOUND });
    }

    res.status(200).json(events);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: errorMessages.ERROR_FETCHING_EVENTS });
  }
},
```

## 7.3 Controlo de erros *multilanguage*

Para melhorar a experiência do utilizador e tornar a aplicação mais acessível a uma audiência global, implementámos uma funcionalidade de controlo de erros multilíngue. Esta funcionalidade permite que as mensagens de erro sejam devolvidas no idioma preferido pelo utilizador, com base no cabeçalho Accept-Language da requisição. Com o uso de um ficheiro JSON contendo traduções para diferentes idiomas, conseguimos fornecer respostas adequadas em várias línguas, como Português, Inglês e Espanhol, dependendo da configuração de linguagem do utilizador.

A seguir, apresentamos exemplos do formato do ficheiro de mensagens e como este foi integrado no código do controller.

```
{
  "pt": {
    "NO_EVENTS_FOUND": "Nenhum evento encontrado",
    "ERROR_FETCHING_EVENTS": "Erro ao buscar eventos",
    "NO_SUSPENDED_EVENTS_FOUND": "Nenhum evento suspenso encontrado",
  },
  "en": {
    "NO_EVENTS_FOUND": "No events found",
    "ERROR_FETCHING_EVENTS": "Error fetching events",
    "NO_SUSPENDED_EVENTS_FOUND": "No suspended events found",
  },
  "es": {
    "NO_EVENTS_FOUND": "No se encontraron eventos",
    "ERROR_FETCHING_EVENTS": "Error al obtener eventos",
    "NO_SUSPENDED_EVENTS_FOUND": "No se encontraron eventos suspendidos",
  }
}

async getAllEvents(req, res) {
  const lang = req.headers['accept-language'] || 'en';
  const errorMessages = loadErrorMessages(lang);
  try {
    const events = await eventService.getAllEvents();

    if (events == null || events.length === 0) {
      return res.status(404).json({ message: errorMessages.NO_EVENTS_FOUND });
    }

    res.status(200).json(events);
  } catch (error) {
    console.error(error);
    res.status(500).json({ message: errorMessages.ERROR_FETCHING_EVENTS });
  }
}
```



## 7.4 Implementação da *API Gateway*

Como foi explicado anteriormente no capítulo "Modelo Arquitetural", a nossa aplicação utilizava uma arquitetura de microsserviços, na qual estes serviços comunicavam entre si e diretamente com o exterior, utilizando chamadas *HTTP*.

De forma a melhorar e tornar as comunicações mais seguras e eficientes, foi implementado o padrão de *API Gateway*. Esse padrão consiste na utilização de um serviço que fornece um ponto de entrada único para determinados grupos de microsserviços, que no nosso caso, serão todos os microsserviços à exceção do próprio *gateway*.

A *API Gateway*, fica entre as aplicações cliente e os microsserviços. Assim, este irá lidar com as chamadas efetuadas de clientes para serviços, e no nosso caso, também ficará responsável pela gestão das chamadas entre os serviços, redirecionando os pedidos do serviço que originou a chamada, para o serviço destinatário.

Com base na implementação efetuada, foi elaborado o seguinte esquema atualizado com o padrão de *API Gateway* devidamente implementado, no qual a comunicação entre o cliente e a *API Gateway* e a comunicação entre este e os restantes serviços é efetuada utilizando chamadas *HTTP*, como podemos visualizar na imagem que se segue.

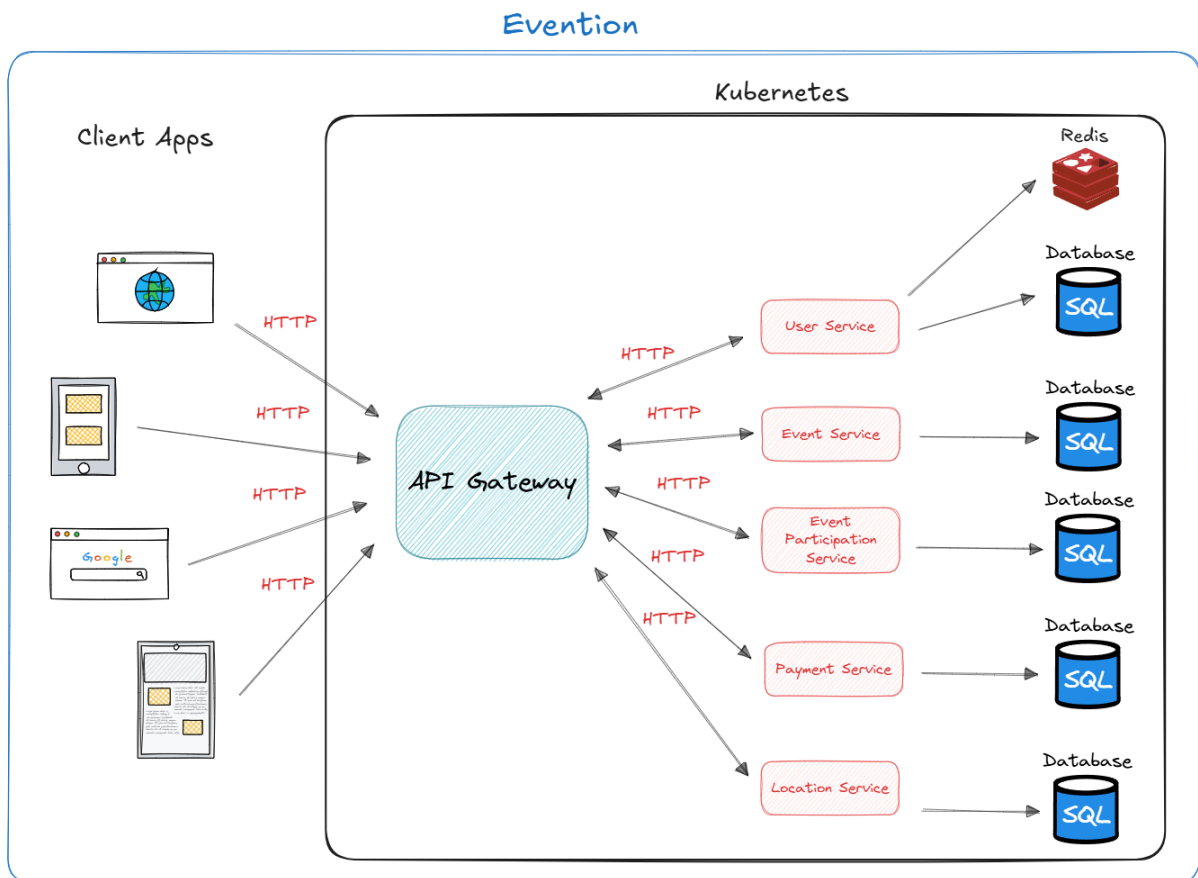


Figura 7.2: Modelo Arquitetural utilizando API Gateway

De forma a implementar a API Gateway, foi utilizado o NGINX e foram criados quatro ficheiros YAML, de forma a configurar a mesma.

Os ficheiros criados:

- **nginx-configmap:** Este arquivo define um ConfigMap que contém a configuração específica da API Gateway. Define as rotas e encaminhamento para serviços internos;
- **nginx-deployment:** Este arquivo define o Deployment do Kubernetes para implantar o NGINX como uma API Gateway. Implanta o NGINX como um pod no Kubernetes, usando as configurações dos ConfigMaps;
- **ngin-service:** Este arquivo define um serviço do Kubernetes para expor o NGINX como uma API Gateway. Utiliza um Service do tipo LoadBalancer na porta 5010;
- **nginx-main-config:** Este arquivo define um ConfigMap que contém a configuração principal do NGINX, e inclui a limitação de taxa de carga.

Para os limites de carga foram testados vários valores, ficando o valor de um request por segundo definido no final, com a possibilidade de ser alterado no futuro consoante as necessidades da aplicação.

## 7.5 Implementação de comunicação *HTTPS*

De forma a criar uma camada de segurança adicional, foi estudada a implementação de chamadas utilizando o protocolo *HTTPS* (*Hyper Text Transfer Protocol Secure*), que consiste na implementação do protocolo *HTTP* sobre uma camada adicional de segurança que utiliza o protocolo *TLS/SSL*. Essa camada adicional permite que os dados sejam transmitidos por meio de uma conexão criptografada e que se verifique a autenticidade do servidor e do cliente por meio de certificados digitais.

Com a utilização deste protocolo, foi pretendido garantir a veracidade e consistência dos dados, uma vez que estes dados podem ser interceptados no caso dos serviços da aplicação estarem alocados em diferentes servidores e em diferentes partes do planeta, assim conseguimos garantir que os dados transferidos são mais difíceis de acessar, tornando as comunicações entre os serviços e a *API Gateway* mais seguras.

Com isto, é possível visualizar na imagem seguinte a arquitetura final, com a *API Gateway* implementada anteriormente, e com as comunicações entre os clientes e a *Gateway*, e as comunicações entre a *Gateway* e os Serviços todas utilizando *HTTPS*, garantido assim uma camada adicional de segurança na nossa aplicação.

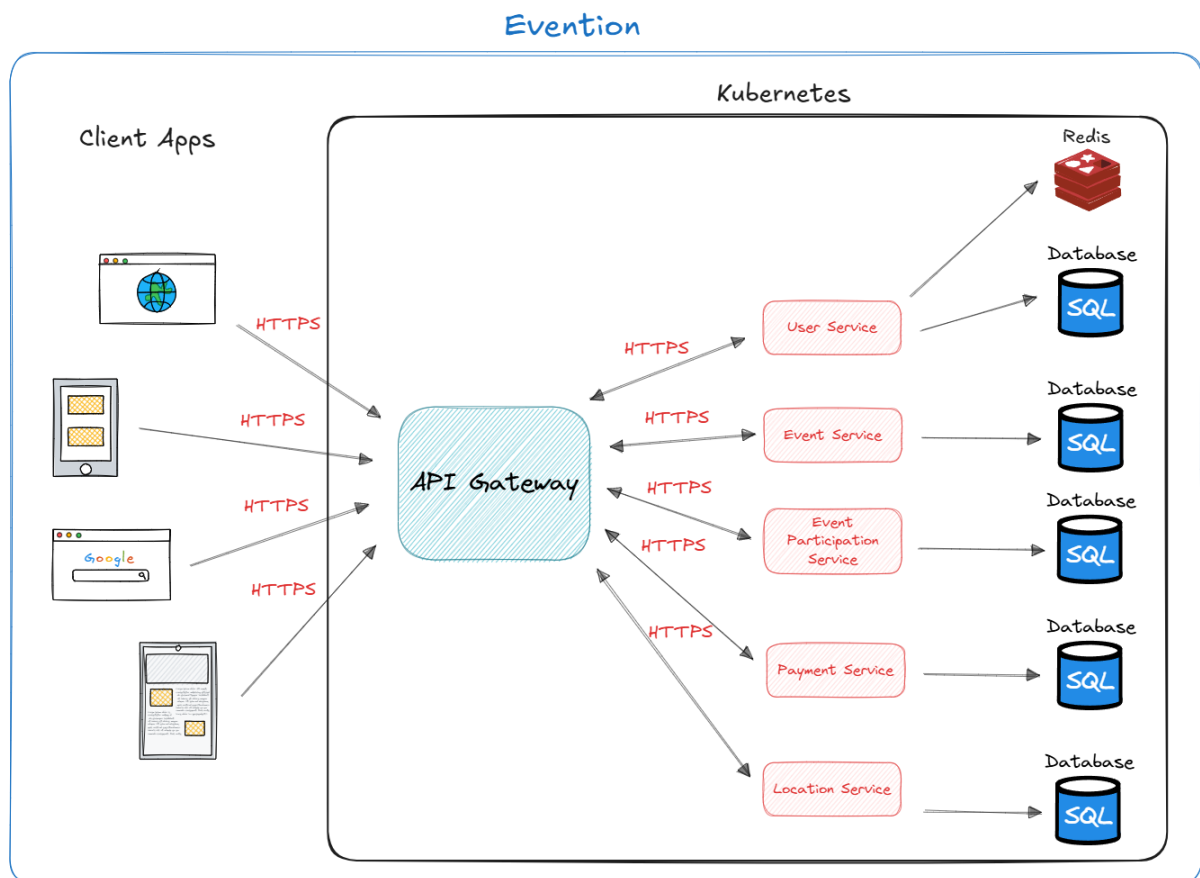


Figura 7.3: Modelo Arquitetural com API Gateway utilizando HTTPS

## 7.6 Registo e Login com Google

Outra melhoria implementada na nossa aplicação, foi a implementação do Registo e do Login utilizando o serviço da Google, assim, os utilizadores podem entrar na aplicação diretamente utilizando a sua conta Google, ligando diretamente a mesma à nossa aplicação.

Na seguinte imagem é possível visualizar as rotas adicionadas no Swagger, para criar uma conta com uma conta Google e para efetuar o login com uma conta Google.

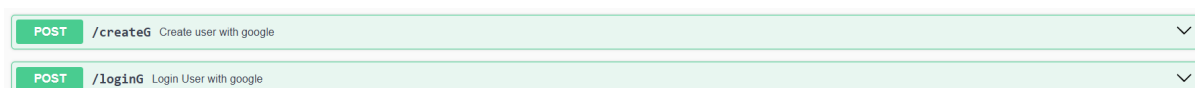


Figura 7.4: Rotas Swagger para Registo e Login com Google

## 7.7 Notificações com RabbitMQ

Para as notificações, utilizamos um message broker e optámos pelo RabbitMQ por ser uma solução robusta e eficiente para uma comunicação assíncrona em tempo real. No nosso caso, quando um utilizador adere a um evento, uma notificação é enviada para a fila de mensagens do broker. Em seguida, um worker fica em constante escuta dessa fila, processando e encaminhando as notificações conforme necessário.

No portal do RabbitMQ, podemos visualizar a fila de mensagens pendentes e monitorizar o funcionamento do worker na consola. Isso permite-nos garantir que as mensagens estão a ser corretamente enfileiradas e consumidas.

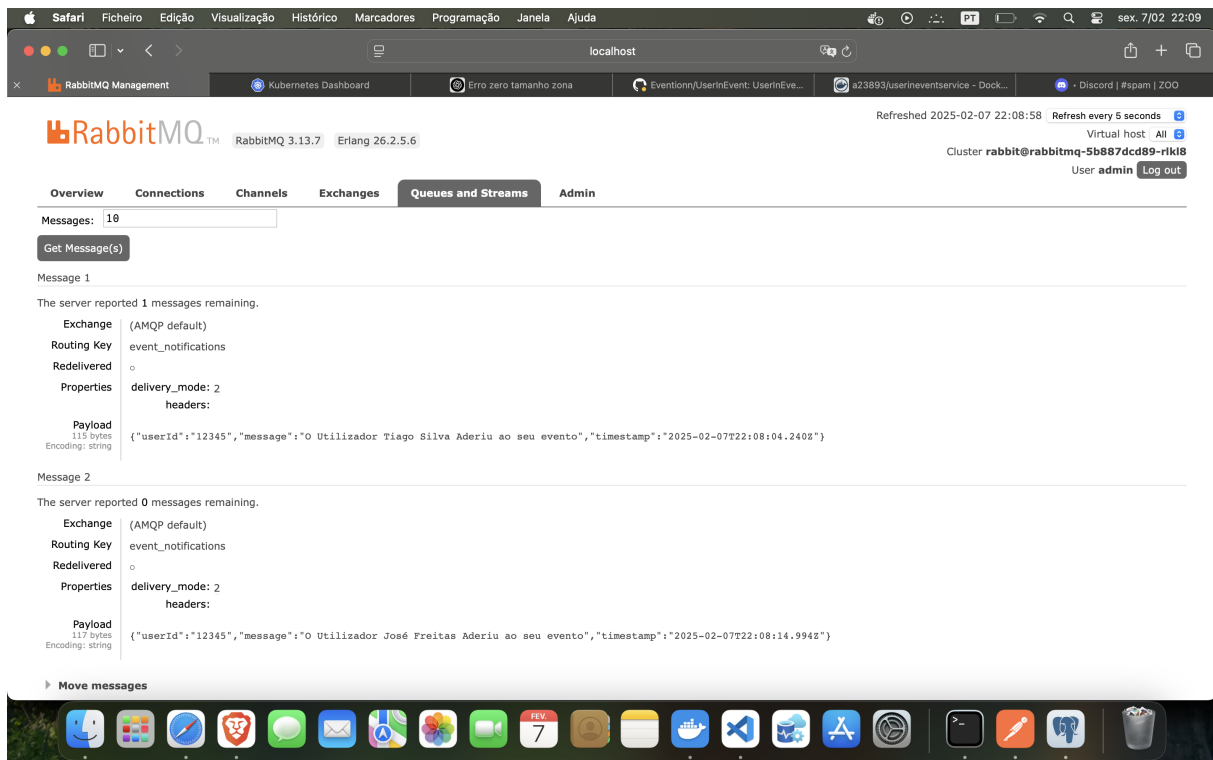


Figura 7.5: Portal do RabbitMQ

Na nossa implementação final, enfrentámos desafios ao integrar o RabbitMQ e o worker dentro do cluster Kubernetes. No entanto, conseguimos realizar testes localmente para validar o fluxo de funcionamento e garantir que as notificações são entregues corretamente.

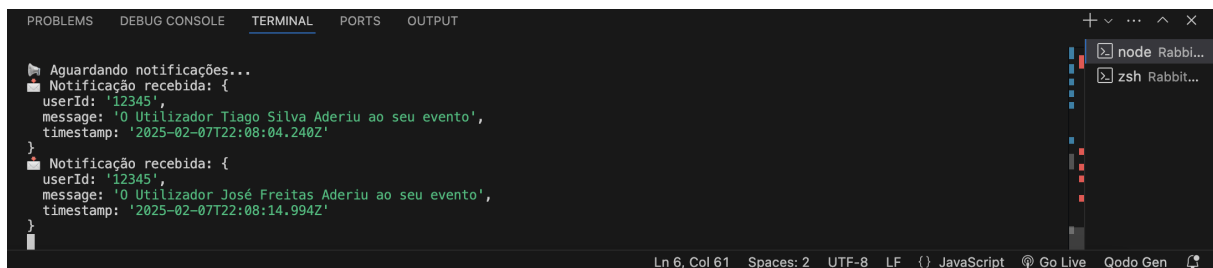


Figura 7.6: Visualização das notificações recebidas pelo worker

## 7.8 Arquitetura Atual

Com base nas alterações efetuadas à arquitetura, o modelo arquitetural definido anteriormente ficou obsoleto, sendo assim necessária a demonstração do modelo arquitetural atualizado e que se encontra implementado atualmente.

Na imagem seguinte, é possível visualizar o modelo arquitetural devidamente atualizado, contendo todas as alterações efetuadas, como o padrão *API Gateway*, a comunicação utilizando o protocolo *HTTPS*, o sistema de notificações utilizando o RabbitMQ e a ligação à API externa da Google que permite o registo e login com conta Google.

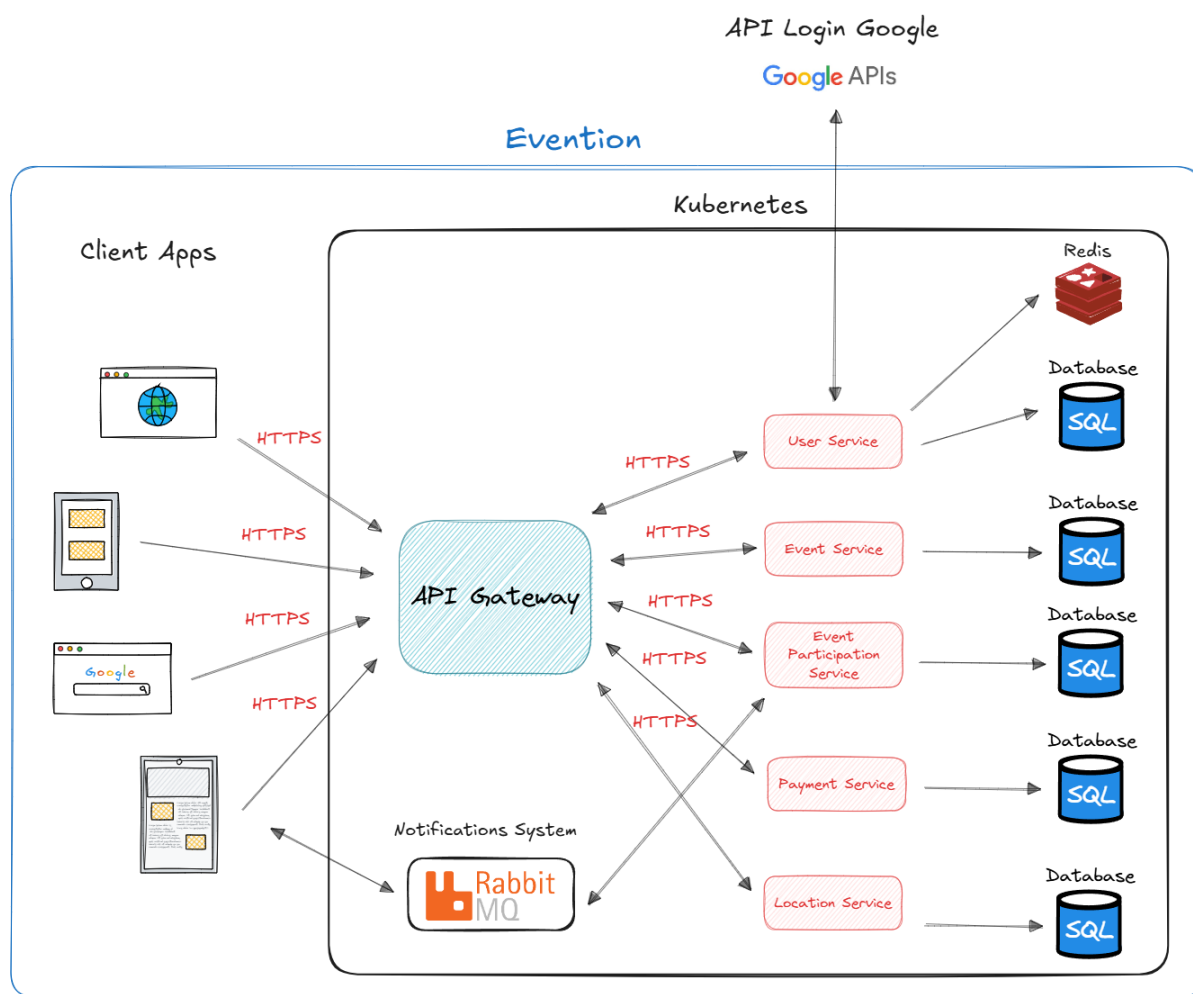


Figura 7.7: Modelo Arquitetural Atualizado

## 8. Análise de Resultados

De forma a analisar os resultados obtidos com as melhorias efetuadas na nossa estrutura, foi testada a performance das requisições à *API Gateway*, de forma a comparar a performance das requisições utilizando o protocolo *HTTP* e utilizando o protocolo *HTTPS*.

Para isto, foi criado um *script* em Python de forma a executar 200 chamadas *HTTP* ou *HTTPS*, e assim comparar o tempo das respostas. Para isto, utilizamos a rota do serviço dos bilhetes, para obter os bilhetes de um evento, que faz uma chamada ao serviço dos eventos para verificar se existe algum evento com o id fornecido.

### 8.1 Testes HTTP vs HTTPS

Começamos então por executar o script de chamada com a *API Gateway* utilizando *HTTP*, para desta forma obter os resultados iniciais para realizar a comparação.

```
{
  "total_requests": 200,
  "successful_requests": 200,
  "failed_requests": 0,
  "avg_response_time": 0.0213601553440094,
  "min_response_time": 0.008420705795288086,
  "max_response_time": 0.06243705749511719
}
```

Figura 8.1: Teste com API Gateway utilizando HTTP

Ao executar o script, obtivemos então um tempo médio de aproximadamente 0.021 segundos nas 200 requisições à rota utilizada na API Gateway com chamadas HTTP, como podemos ver na linha *"avg\_response\_time"*.

Depois de fazer as alterações na API Gateway e nos Serviços para estes passarem a utilizar o protocolo HTTPS, voltamos então a executar o script com as devidas alterações para o teste e obtivemos o seguinte resultado:

```
{
  "total_requests": 200,
  "successful_requests": 200,
  "failed_requests": 0,
  "avg_response_time": 0.07298123121261596,
  "min_response_time": 0.03399205207824707,
  "max_response_time": 0.09033942222595215
}
```

Figura 8.2: Teste com API Gateway utilizando HTTPS

Com este resultado, foi possível verificar que o tempo médio das 200 requisições aumentou consideravelmente, passando de aproximadamente 0.021 segundos para aproximadamente 0.073 segundos, que seria o expectável, uma vez que o protocolo HTTPS utiliza chamadas encriptadas, que aumenta o tempo de processo das requisições. Também é possível verificar que o tempo mínimo das respostas aumentou também, passando de aproximadamente 0.008 segundos para aproximadamente 0.034 segundos e o tempo máximo também aumentou, passando de aproximadamente 0.062 segundos para aproximadamente 0.090 segundos.

## 8.2 Impacto do protocolo HTTPS nos Serviços

De forma a testar também o impacto da performance que o protocolo HTTPS teria nos serviços da aplicação alojados no cluster de kubernetes, o script de requisições foi assim modificado de forma a mostrar também o impacto que esta alteração teria no pods do cluster. Para isto, foi necessário instalar o *plugin Metrics Server*, que permite ver algumas métricas de performance dos pods do cluster, utilizando o comando "kubectl top pods".



Começamos então pela API Gateway utilizando HTTP, começando por verificar os pods antes de correr o script, como podemos ver na imagem seguinte:

```
C:\Users\Diogo>kubectl top pods
```

NAME	CPU(cores)	MEMORY(bytes)
eventservice-db-0	1m	23Mi
eventservice-deployment-84878f6598-v6jfc	1m	69Mi
locationservice-db-0	1m	17Mi
locationservice-deployment-5945bc4fcd-74xv2	1m	53Mi
nginx-api-gateway-deployment-6cd66754d6-dlgcr	0m	10Mi
paymentservice-db-0	1m	20Mi
paymentservice-deployment-f5b8754c7-nsrdz	1m	68Mi
userineventservice-db-0	1m	21Mi
userineventservice-deployment-764588f6b6-n42pc	1m	61Mi
userservice-db-0	1m	42Mi
userservice-deployment-7865bf9b7f-27jwq	1m	55Mi
userservice-redis-0	4m	15Mi

Figura 8.3: Métricas iniciais dos pods com API Gateway utilizando HTTP

Assim, após executar o script é mostrado as métricas de utilização dos pods do cluster, e podemos ver os resultados na seguinte imagem:

```
{
  "total_requests": 200,
  "successful_requests": 200,
  "failed_requests": 0,
  "avg_response_time": 0.022643789052963256,
  "min_response_time": 0.008968830108642578,
  "max_response_time": 0.13539338111877441
}
```

-----

Coletando métricas de memória dos pods...

Métricas de uso de memória para o serviço UserInEvent

NAME	CPU(cores)	MEMORY(bytes)
eventservice-db-0	1m	23Mi
eventservice-deployment-84878f6598-v6jfc	1m	69Mi
locationservice-db-0	1m	17Mi
locationservice-deployment-5945bc4fcd-74xv2	1m	53Mi
nginx-api-gateway-deployment-6cd66754d6-dlgcr	0m	10Mi
paymentservice-db-0	1m	20Mi
paymentservice-deployment-f5b8754c7-nsrdz	1m	68Mi
userineventservice-db-0	1m	21Mi
userineventservice-deployment-764588f6b6-n42pc	1m	61Mi
userservice-db-0	1m	42Mi
userservice-deployment-7865bf9b7f-27jwq	1m	55Mi
userservice-redis-0	4m	15Mi

Figura 8.4: Teste aos pods com API Gateway utilizando HTTP

Com estes resultados obtidos, foi possível verificar que não existiu nenhum aumento dos cores no CPU nem de memória no pod "userineventservice-deployment", que é o pod onde está alojado o serviço dos bilhetes, que é o serviço responsável pela rota que estamos a testar, como foi explicado anteriormente.

Depois, fizemos o mesmo teste para a versão da aplicação que utiliza a API Gateway com comunicações, utilizando o protocolo HTTPS. Começamos então por executar o comando "Kubectl top pods" para obter as métricas iniciais dos pods no cluster de kubernetes:

```
C:\Users\Diogo>kubectl top pods
NAME                                CPU(cores)   MEMORY(bytes)
eventservice-db-0                   1m           25Mi
eventservice-deployment-84878f6598-nrr67  1m           83Mi
locationservice-db-0               1m           17Mi
locationservice-deployment-5945bc4fcd-74xv2  1m           52Mi
nginx-api-gateway-deployment-8445687f89-9jcrm  0m           14Mi
paymentservice-db-0                1m           20Mi
paymentservice-deployment-f5b8754c7-nsrdz    1m           67Mi
userineventservice-db-0            1m           21Mi
userineventservice-deployment-764588f6b6-9kgd5  1m           60Mi
userservice-db-0                   1m           42Mi
userservice-deployment-7865bf9b7f-gb6cx      1m           59Mi
userservice-redis-0                4m           15Mi
```

Figura 8.5: Métricas iniciais dos pods com API Gateway utilizando HTTPS

E por fim, executamos o script onde é mostrado as métricas de utilização dos pods do cluster, e podemos ver os resultados na seguinte imagem:

```
{
  "total_requests": 200,
  "successful_requests": 200,
  "failed_requests": 0,
  "avg_response_time": 0.07454877614974975,
  "min_response_time": 0.0677785873413086,
  "max_response_time": 0.13126063346862793
}
-----
Coletando métricas de memória dos pods...
Métricas de uso de memória para o serviço UserInEvent
NAME                                CPU(cores)   MEMORY(bytes)
eventservice-db-0                   1m           25Mi
eventservice-deployment-84878f6598-nrr67  2m           83Mi
locationservice-db-0               1m           17Mi
locationservice-deployment-5945bc4fcd-74xv2  1m           52Mi
nginx-api-gateway-deployment-8445687f89-9jcrm  2m           14Mi
paymentservice-db-0                1m           20Mi
paymentservice-deployment-f5b8754c7-nsrdz    1m           67Mi
userineventservice-db-0            1m           23Mi
userineventservice-deployment-764588f6b6-9kgd5  5m           65Mi
userservice-db-0                   1m           42Mi
userservice-deployment-7865bf9b7f-gb6cx      1m           59Mi
userservice-redis-0                4m           15Mi
-----
```

Figura 8.6: Teste aos pods com API Gateway utilizando HTTPS

Podemos assim verificar que utilizando chamadas HTTPS, as métricas mostram um aumento de cores do CPU e um aumento de memória na execução do pod "userineventservice", que não foi verificada no teste anterior, mostrando assim que a utilização de HTTPS prejudica diretamente a performance dos serviços em questão.

## **8.3 Análise Final dos Resultados**

Com estes testes realizados, foi possível concluir que a mudança para a utilização do protocolo HTTPS para as requisições, afeta negativamente a performance da arquitetura definida, mesmo que seja apenas em centésimas de segundo, este déficit pode ser prejudicial em requisições mais complexas, mas por outro lado cria uma camada adicional de segurança na aplicação, e com base nisto foi decidido que o protocolo HTTPS seria necessário na nossa aplicação, mesmo tendo em conta o impacto de performance obtido, já que este aumenta consideravelmente a segurança das comunicações, que era um requisito importante, uma vez que os serviços poderão ser alojados em ambientes cloud, na qual os servidores podem estar em diversas partes diferentes do planeta e pode haver complicações com informações sensíveis.

## 9. Futuras implementações

Para além das melhorias já implementadas na estrutura, temos planeadas algumas implementações futuras com o objetivo de otimizar o desempenho e expandir as funcionalidades da estrutura existente. Estas melhorias visam não só aumentar a eficiência do sistema, mas também proporcionar uma melhor experiência ao utilizador, tornando a aplicação mais intuitiva.

De seguida, apresentamos algumas das principais atualizações previstas.

### 9.1 Integração com uma API de Previsão Meteorológica

Uma das melhorias previstas para a aplicação consiste na integração com uma API externa de previsão meteorológica. Esta funcionalidade permitirá obter informações em tempo real sobre a temperatura e outras condições climáticas no local do evento, tendo em conta a hora a que este irá decorrer.

Com esta implementação, os utilizadores poderão consultar antecipadamente as condições meteorológicas esperadas, permitindo-lhes uma melhor preparação para o evento. Esta funcionalidade será particularmente útil para eventos ao ar livre, onde as condições do tempo podem ter um impacto significativo na experiência dos participantes.

### 9.2 Implementação de um Chat para Eventos

Outra melhoria prevista para a aplicação é a implementação de um sistema de chat dedicado a cada evento. Esta funcionalidade permitirá que o anunciante e os participantes comuniquem diretamente dentro da plataforma, facilitando a partilha de informações importantes e o esclarecimento de dúvidas.

### 9.3 Implementação de uma Super App

A implementação de uma Super App seria também uma outra implementação futura na aplicação. Um dos principais objetivos desta funcionalidade será a sincronização automática dos eventos com o calendário do telemóvel do utilizador.

Com esta implementação, sempre que um utilizador aderir a um evento, este será

automaticamente adicionado ao seu calendário, garantindo que não se esquece da data e hora. Esta funcionalidade não só facilita a organização pessoal, como também melhora a experiência do utilizador, evitando a necessidade de inserção manual dos eventos no calendário.

## **9.4 Migração do Projeto para a Cloud**

Dado que este projeto está inserido na unidade curricular de "Projeto de Computação na Cloud", uma das principais melhorias previstas é a migração da aplicação para a cloud. Esta implementação permitirá aumentar a escalabilidade, a fiabilidade e o desempenho da aplicação, garantindo um acesso mais rápido e eficiente para todos os utilizadores.

Com a infraestrutura baseada na cloud, será possível otimizar o armazenamento de dados, melhorar a gestão de recursos e garantir maior disponibilidade do sistema.

## **9.5 Implementação de Bases de Dados Partilhadas**

Outra possível melhoria para a aplicação consiste na implementação de bases de dados partilhadas em alguns serviços estratégicos, utilizando o padrão Saga. Esta abordagem permitirá reduzir a necessidade de comunicações excessivas entre serviços, promovendo uma maior eficiência e desempenho do sistema.

Ao utilizar bases de dados partilhadas, os serviços poderão aceder diretamente a dados estratégicos sem depender constantemente de chamadas entre microsserviços. O padrão Saga garantirá a consistência dos dados distribuídos, gerindo transações complexas de forma assíncrona e minimizando problemas como falhas parciais ou inconsistências nos registos.

## 10. Conclusão

O trabalho desenvolvido neste projeto foi fundamental para melhorar a estrutura, segurança e escalabilidade do sistema. A adição do registo e login com conta Google permite aos utilizadores uma forma simples e rápida de criar conta e assim aceder facilmente à plataforma, a implementação de uma *API Gateway* permitiu uma maior eficiência na comunicação entre os microsserviços e uma experiência de utilizador mais segura e fluída, a utilização do protocolo *HTTPS* nas comunicações permitiu adicionar uma camada de segurança que não existia anteriormente, a clareza na documentação também facilita a manutenção e futuras expansões do sistema, tornando-o mais acessível para outras equipas de desenvolvimento e por fim, o sistema de notificações utilizando o RabbitMQ permite aos utilizadores receberem notificações do sistema de uma forma mais precisa e eficiente.

As melhorias já implementadas juntamente com possíveis alterações futuras, garantirão que o sistema esteja cada vez mais preparado para acompanhar as necessidades tecnológicas e do mercado, proporcionando soluções inovadoras e úteis para os utilizadores.

## 11. Link do Repositório

O trabalho desenvolvido pode ser consultado através do seguinte link do Github:

[`https://github.com/Eventionn`](https://github.com/Eventionn)

Link do Repositório do Github: Evention Team - Repositório Figma.

## 12. Bibliografia

- 1 - <https://nodejs.org/docs/latest/api/>
- 2 - <https://www.w3schools.com/nodejs/>
- 3 - <https://jwt.io/introduction>
- 4 - <https://medium.com/cloud-native-daily/how-to-deploy-a-containerized-node-js-api->
- 5 - <https://kubernetes.io/docs/tasks/run-application/horizontal-pod-autoscale-walkth>
- 6 - <https://medium.com/@ravisharma23523/sending-notifications-to-mobile-devices-with>
- 7 - <https://www.prisma.io/docs>
- 8 - <https://learn.microsoft.com/en-us/azure/architecture/microservices/design/gateway>
- 9 - <https://www.geeksforgeeks.org/explain-working-of-https/>
- 10 - <https://learn.microsoft.com/pt-br/azure/architecture/patterns/saga>
- 11 - <https://developers.google.com/identity/sign-in/web/sign-in?hl=pt-pt>