Politécnico de Coimbra

Escola Superior de Tecnologia e Gestão

de Oliveira do Hospital

Licenciatura em Engenharia Informática

- Base de Dados II - 2023/24 -

TRABALHO PRÁTICO N.º 2

PROJETO BASE DE DADOS

APLICAÇÃO GESTÃO DE EVENTOS

Discentes:

Guilherme Gonçalves n.º 2022156457

Hugo Pais n.º 2022129956

Docente:

Prof. Doutor Gonçalo Marques

Elaborado em: 08 de maio de 2024

Índice

Lista de Figuras

1. Introdução

As bases de dados têm um papel crucial nos dias de hoje e são necessárias para armazenar dados e permitir às organizações a gestão eficiente de grandes volumes de informação. Entre os diversos sistemas de gestão de bases de dados estão o MySQL e o PostgreSQL escolhas populares entre os utilizadores por oferecerem robustez, desempenho e flexibilidade (e.g., Veloso, 2023). Neste projeto iremos recorrer ao PostgreSQL, sistema open-source gratuito, que oferece suporte a extensões de várias linguagens, incluindo Python. Pretende-se simplificar a escrita de funções e procedimentos considerados em Python, recorrendo diretamente ao PostgreSQL e aproveitando o seu poder e a flexibilidade desta linguagem. O suporte nativo para o tipo de dados JSON pelo PostgreSQL, torna-o muito útil para o desenvolvimento de aplicações web onde se insira manipulação de dados.

Este documento pretende explorar uma aplicação prática dos conceitos fundamentais de bases de dados, aplicando os conteúdos programáticos da UC Bases de Dados I e II. Através deste projeto será desenvolvido um sistema de gestão de eventos, com possibilidade de venda de ingressos. A utilização da linguagem SQL para criar, modificar e consultar bases de dados é crucial para construir um sistema de informação eficiente e funcional.

Este trabalho é uma continuação do trabalho prático número um, onde foram apresentados os diagramas concetuais e físicos, o script SQL relativo à criação da base de dados, um dicionário de dados e a definição das principais operações, transações a serem desenvolvidas para concretização da aplicação. Neste projeto apresentar-se-ão as views, rules, cursores, exceções, procedimentos, funções e triggers, entre outras funcionalidades, necessárias para concretização do enunciado. Além disso criar-se-á a API REST, com recurso ao Flask, que será a porta de entrada para o acesso a determinados serviços disponibilizados pela plataforma , nomeadamente interação com a base de dados através de pedidos HTTP.

Ao longo deste relatório, serão exploradas implicações e aplicações do PostgreSQL na criação e gestão de bases de dados e analisar-se-ão as vantagens e a utilidade da plataforma desenvolvida, destacando como as tabelas modeladas podem contribuir para a organização e recuperação eficiente de informação no contexto de uma empresa promotora de eventos.

A aplicação específica foca-se em requisitos que vão desde a gestão de utilizadores, eventos, inscrições até à regulação de vendas e subscrição de mensagens/notificações. A abordagem centrada nas tabelas e nas transações proporciona uma visão holística da funcionalidade do sistema, demonstrando a capacidade do PostgreSQL em lidar com as complexidades do mundo real.

A linguagem SQL, por ser uma ferramenta padrão na gestão de bases de dados relacionais, desempenha um papel vital neste projeto. Desde a criação das tabelas até à realização de consultas complexas, o PostgreSQL oferece uma sintaxe poderosa e eficiente para manipular dados.

Ao final deste documento, espera-se que o leitor não só compreenda os aspetos técnicos da linguagem utilizada e da modelagem de bases de dados, mas também a aplicação prática destes conhecimentos.

2. Métodos

Este projeto experimental insere-se no segundo trabalho prático a desenvolver na Unidade Curricular (UC) de Base de Dados II da Escola Superior Tecnologia e Gestão de Oliveira do Hospital (ESTGOH), unidade orgânica, do Instituto Politécnico de Coimbra. O objetivo central do projeto visa proporcionar uma experiência no desenvolvimento de sistemas de base de dados considerando uma situação comum que pressupõe a concretização das principais etapas associadas ao desenvolvimento de software (Marques, 2024). Para o efeito foi necessário compreender como um projeto de desenvolvimento de aplicação de base de dados é organizado, projetado e executado. Nesta segunda fase do projeto serão criadas as views, rules, cursores, exceções, procedimentos, funções e triggers necessários para o funcionamento da aplicação. Além disso criar-se-á a API REST que possibilitará a interação com a base de dados. A otimização do sistema, a segurança nos acessos, as distinção entre perfis de utilizador no acesso a dados são alguns dos pontos tidos em atenção para atingir os objetivos propostos. A metodologia adotada segue o enunciado proposto e os requisitos definidos para o projeto.

2.1. Instrumentos / Recursos

O projeto recorreu a ferramentas e tecnologias essenciais para a criação e gestão eficiente da base de dados.

A plataforma online Onda (https://onda.dei.uc.pt/v4/) foi um dos recursos utilizados para o desenho dos modelos concetuais e consequente criação do diagrama físico e script gerador da base de dados. Esta aplicação facilitou a representação visual das entidades, atributos e relacionamentos, fornecendo uma visão clara e organizada da estrutura da base de dados.

O Sitema de Gestão de Base de Dados (SGBD) PostgreSQL foi a escolha central para o desenvolvimento deste trabalho por proporcionar uma estrutura robusta e amplamente utilizada no armazenamento e recuperação de dados relacionais. A interação com o SGBD PostgreSQL foi realizada através da linguagem SQL. Esta linguagem permitiu a execução de comandos de pesquisa, criação e modificação da base de dados, demonstrando a versatilidade e eficiência do PostgreSQL. Por sua vez o pgAdmin foi o software de administração e desenvolvimento de base de dados para PostgreSQL utilizado.

No que se refere ao desenvolvimento da API, o Postman foi a plataforma de colaboração que permitiu a criação e testagem dos serviços disponibilizados através da implementação de “rotas”

executadas de forma rápida e fácil. Com ele foi possível enviar solicitações HTTP para API RESTful e observar as respostas obtidas.

Por último a necessidade de ter um IDE (ambiente de desenvolvimento integrado) que permitisse a codificação, depuração, testagem de código levou à utilização do Visual Studio Code um editor de código leve e personalizável desenvolvido pela Microsoft, bastante popular e já utilizado pelos desenvolvedores em outros projetos.

2.2. Planeamento

O planeamento estruturado foi essencial para o sucesso do projeto, tanto na primeira fase como na segunda. A distribuição eficiente das tarefas ao longo do tempo, dada a complexidade, abrangência e recursos necessários neste trabalho permitiu o desenvolvimento articulado e segmentado entre os desenvolvedores facilitando a concretização das tarefas e o alcance do objetivo final.

No âmbito do trabalho em equipa foram atribuídas tarefas a cada membro definindo-o como responsável pela correta execução. Não obstante, todas as opções tomadas, código criado, funcionalidades implementadas foram amplamente discutidas em equipa e revistas pelos dois elementos do grupo. Esta abordagem permitiu uma colaboração eficaz, otimizando os recursos e competências individuais. Paralelamente, a revisão do trabalho desenvolvido por ambos os elementos permitiu encontrar erros e problemas que de outra forma só seriam detetados na fase de testagem.

Em suma, o planeamento detalhado proporcionou uma gestão eficiente do tempo, permitindo a conclusão do projeto dentro dos prazos estabelecidos. No entanto dado o impacto de outras Unidades Curriculares no tempo disponível houve algumas etapas que viram o seu prazo derrapar um pouco. As fases do projeto e os prazos estabelecidos são apresentados em seguida na Figura 2-1. O alicerce para este planeamento integra o briefing dado pelo enunciado e as funcionalidades que devem estar garantidas.

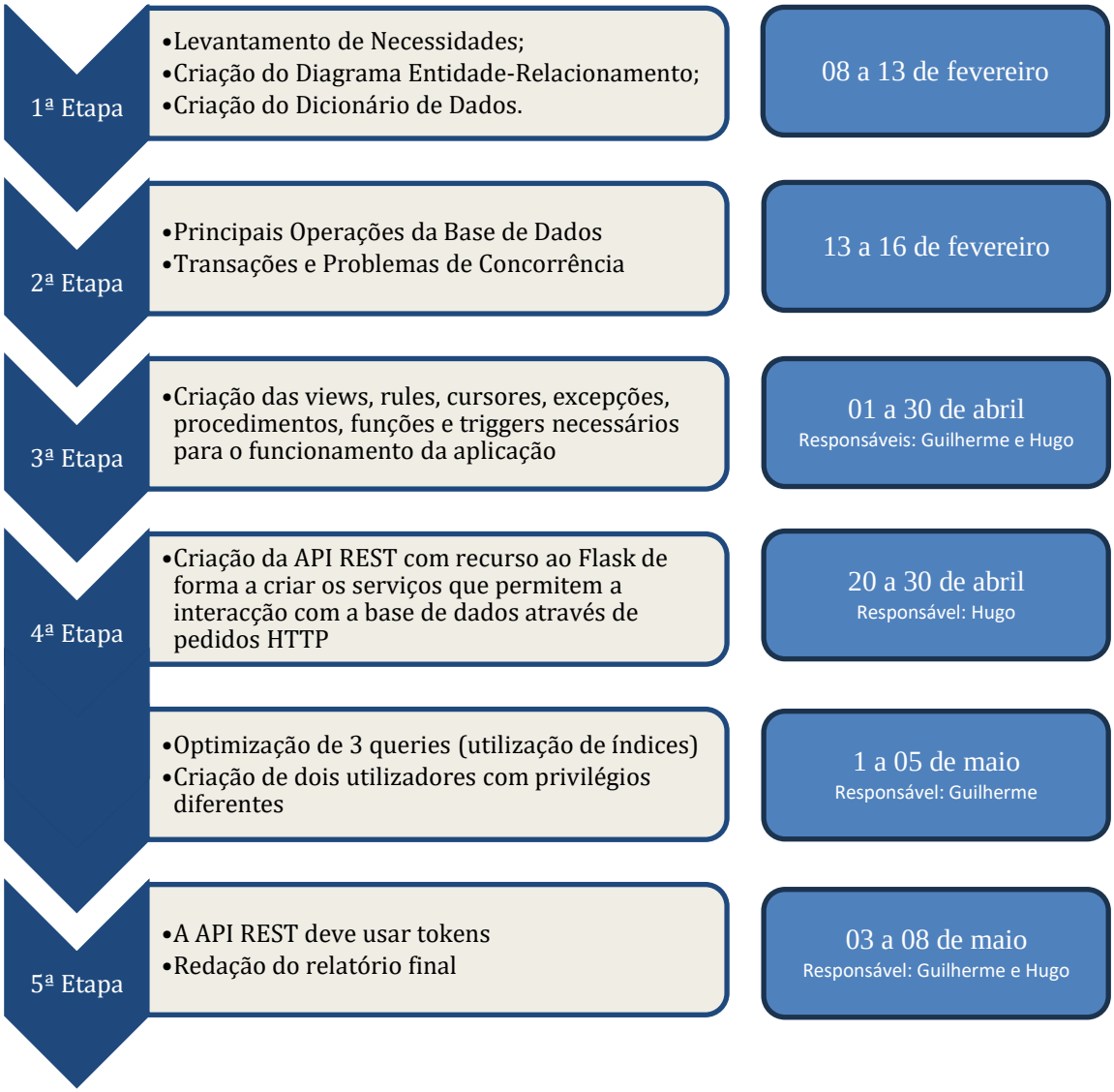

Figura 2-1 Fases do projeto e prazos estabelecidos.

3. Resultados

Ao longo deste capítulo destacaremos os diversos elementos que permitiram criar a aplicação para gestão de eventos. Isso inclui uma representação da base de dados através dos diagramas configurados (i.e., conceptual e físico), um dicionário de dados para compreensão do significado dos vários elementos e o script de criação da base de dados, gerado após os diagramas. Apresentar-se-á o código-fonte, scripts, ficheiros executáveis e bibliotecas necessárias para compilar e executar o software, bem como as definições de tabelas, restrições, sequências, utilizadores, funções, permissões, triggers e procedimentos necessários à correta operacionalização do sistema.

De seguida, expõem-se e descrevem-se os diferentes elementos, de forma a ser possível perceber o projeto geral.

3.1. Modelo Conceptual

Após terem sido levantados os requisitos para realização do projeto deu-se seguimento à criação do modelo conceptual (que também se pode designar de diagrama Entidade-Relacionamento). Para a criação do diagrama foi necessário ter como base a inserção de entidades (i.e., as tabelas) e posteriormente fazer o relacionamento entre estas.

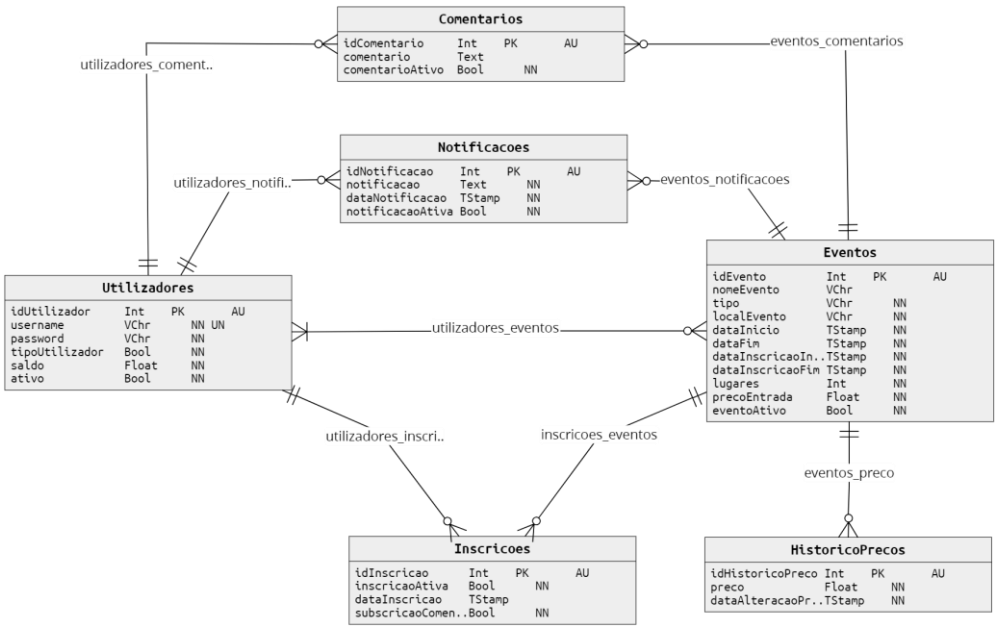

Figura 3-1 – Diagrama Conceptual.

3.2. Modelo Físico

Partindo do diagrama conceptual, estruturado na secção anterior foi gerado o diagrama físico, onde se encontram as tabelas e chaves forasteiras criadas pelo SGBD, como resultado da aplicação das regras de relacionamentos binários entre entidades. O modelo físico, gerado a partir do modelo conceptual é importante na implementação de bases de dados pois fornece detalhadamente as relações entre as diversas entidades. É possível observar a criação de mais tabelas que representam as relações entre as entidades, e também se pode verificar que foram introduzidas chaves primárias e forasteiras resultando dessas relações, garantindo que as relações entre entidades estão feitas corretamente e organizadas.

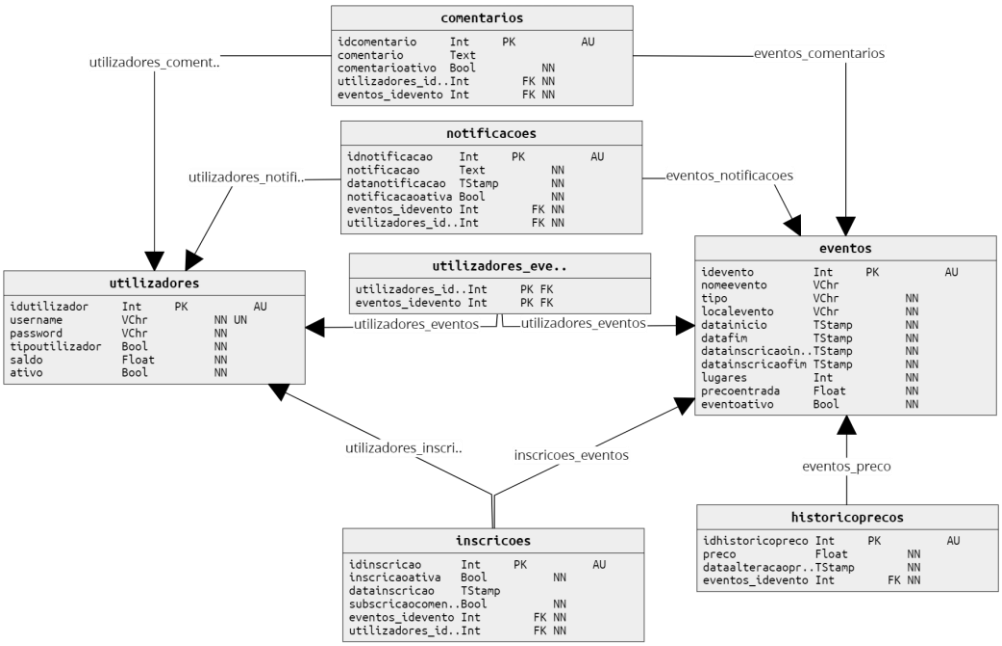

Figura 3-2 – Diagrama Físico.

3.3. Dicionário de Dados

O dicionário de dados apresenta e descreve as entidades e atributos essenciais para o funcionamento do sistema de gestão de eventos projetado. Considera-se uma ferramenta crucial para o correto funcionamento da aplicação uma vez que fornece uma visão detalhada das estruturas de dados necessárias e, por sua vez, utilizadas para armazenar e manipular as informações. As tabelas expostas em seguida representam as entidades e estruturas relevantes à implementação das funcionalidades especificadas pelo cliente (e.g., Utilizadores, Eventos, Inscricoes, Notificacoes). Paralelamente, as características detalhadas em cada tabela, incluindo os atributos, tipos de dados, restrições facilitam e são a base que irá alicerçar o desenvolvimento, implementação e manutenção do sistema de gestão de eventos.

Tabela 1 – Tabela que armazena os utilizadores.

Tabela	Utilizadores
Descrição	Armazena as informações relativas aos utilizadores

Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
idUtilizador	Código identificação utilizador	Int	7	PK / Auto Increment
username	Nome do utilizador	Varchar	60	Not Null / Unique
password	Código do utilizador	Varchar	256	Not Null
tipoUtilizador	Segmenta administrador de utilizador	Boolean		Not Null
saldo	Saldo disponível do utilizador	Float	9	Not Null
ativo	Utilizador ativo / inativo	Boolean		Not Null

Tabela 2 – Tabela que armazena os eventos.

Tabela	Eventos
Descrição	Armazena as informações relativas aos eventos

Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
idEvento	Código identificação evento	Int	7	PK / Auto Increment
nomeEvento	Nome do evento	Varchar	60	Not Null
tipo	Tipo do evento	Varchar	60	Not Null
localEvento	Local onde se realiza o evento	Varchar	60	Not Null
dataInicio	Data de início do evento	TimeStamp		Not Null
dataFim	Data de fim do evento	TimeStamp		Not Null
dataInscricaoIncio	Data de início das inscrições evento	TimeStamp		Not Null
dataInscricaoFim	Data de fim das inscrições evento	TimeStamp		Not Null
lugares	Lugares disponíveis para o evento	Int	7	Not Null
precoEntrada	Preço de entrada estipulado ao evento	Float	9	Not Null
eventoAtivo	Evento ativo / inativo	Boolean		Not Null

Tabela 3 – Tabela que armazena as inscrições e subscrições.

Tabela	Inscricoes
Descrição	Armazena as inscrições/subscrições em eventos e comentários

Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
idInscricao	Código identificação inscrição	Int	7	PK / Auto Increment
inscricaoAtiva	Inscrição ativa / inativa	Boolean		Not Null
dataRegisto	Data de inscrição	TimeStamp		
subscricaoComentario	Subscrição comentários ativa / inativa	Boolean		Not Null
eventos_idEvento	Chave Forasteira com eventos	Int	7	FK / Not Null
utilizadores_idUtilizador	Chave Forasteira com utilizadores	Int	7	FK / Not Null

Tabela 4 – Tabela que armazena o histórico de preços.

Tabela	HistoricoPrecos
Descrição	Armazena os diferentes preços associados a um evento

Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
idHistoricoPreco	Código de identificação do preço	Int	7	PK / Auto Increment
preco	Preço associado ao evento	Float	9	Not Null
dataAlteracaoPreco	Data associada à alteração do preço	TimeStamp		Not Null
eventos_idEventos	Chave Forasteira com eventos	Int	7	FK / Not Null

Tabela 5 – Tabela que armazena os comentários.

Tabela	Comentarios			
Descrição	Armazena e associa os comentários dos diferentes utilizadores aos eventos			
Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
idComentario	Código de identificação do comentário	Int	7	PK / Auto Increment
comentário	Mensagem / Comentário	Text	512	
comentarioAtivo	Comentário visível / não visível	Boolean		Not Null
utilizadores_idUtilizador	Chave Forasteira com utilizadores	Int	7	FK / Not Null
eventos_idEvento	Chave Forasteira com eventos	Int	7	FK / Not Null

Tabela 6 – Tabela que armazena as notificações.

Tabela	Notificacoes			
Descrição	Armazena e associa as notificações remetidas pelos eventos aos utilizadores			
Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
idNotificacao	Código de identificação do comentário	Int	7	PK / Auto Increment
comentário	Mensagem / Notificação	Text	512	Not Null
dataNotificacao	Data da notificação	TimeStamp		
notificacaoAtiva	Notificação lida / não lida	Boolean		Not Null
idNotificacao	Código de identificação do comentário	Int	7	PK / Auto Increment
utilizadores_idUtilizador	Chave Forasteira com utilizadores	Int	7	FK / Not Null
eventos_idEvento	Chave Forasteira com eventos	Int	7	FK / Not Null

Tabela 7 – Tabela que armazena as relações entre as tabelas utilizadores e eventos.

Tabela	Utilizadores_Eventos			
Descrição	Armazena as associações entre eventos e utilizadores			
Variáveis	**Descrição**	**Tipo de Dados**	**Tamanho**	**Restrições Domínio**
utilizadores_idUtilizador	Chave Forasteira com utilizadores	Int	7	FK / Not Null
eventos_idEvento	Chave Forasteira com eventos	Int	7	FK / Not Null

3.4. Código de Criação das Tabelas

Depois de se gerar os modelos conceptual e físico, gerou-se o código SQL que permite criar a base de dados e as respetivas tabelas. De seguida encontra-se esse código SQL:

```
CREATE TABLE eventos (
     idevento                SERIAL,
     nomeevento              VARCHAR(60),
     tipo                    VARCHAR(60) NOT NULL,
     localevento             VARCHAR(60) NOT NULL,
     datainicio              TIMESTAMP NOT NULL,
     datafim                 TIMESTAMP NOT NULL,
     datainscricaoinicio     TIMESTAMP NOT NULL,
     datainscricaofim        TIMESTAMP NOT NULL,
     lugares                 INTEGER NOT NULL,
```

```
	precoentrada		FLOAT(9) NOT NULL,
	eventoativo		BOOL NOT NULL,
	PRIMARY KEY(idevento)
);

CREATE TABLE utilizadores (
	idutilizador		SERIAL,
	username		VARCHAR(60) NOT NULL,
	password		VARCHAR(256) NOT NULL,
	tipoutilizador		BOOL NOT NULL,
	saldo			FLOAT(9) NOT NULL,
	ativo			BOOL NOT NULL,
	PRIMARY KEY(idutilizador)
);

CREATE TABLE inscricoes (
	idinscricao		SERIAL,
	inscricaoativa		BOOL NOT NULL,
	datainscricao		TIMESTAMP,
	subscricaocomentarios	BOOL NOT NULL,
	eventos_idevento	INTEGER NOT NULL,
	utilizadores_idutilizador INTEGER NOT NULL,
	PRIMARY KEY(idinscricao)
);

CREATE TABLE historicoprecos (
	idhistoricopreco	SERIAL,
	preco			FLOAT(9) NOT NULL,
	dataalteracaopreco	TIMESTAMP NOT NULL,
	eventos_idevento	INTEGER NOT NULL,
	PRIMARY KEY(idhistoricopreco)
);

CREATE TABLE comentarios (
	idcomentario		SERIAL,
	comentario		TEXT,
	comentarioativo		BOOL NOT NULL,
	utilizadores_idutilizador INTEGER NOT NULL,
	eventos_idevento	INTEGER NOT NULL,
	PRIMARY KEY(idcomentario)
);

CREATE TABLE notificacoes (
	idnotificacao		SERIAL,
	notificacao		TEXT NOT NULL,
	datanotificacao		TIMESTAMP NOT NULL,
	notificacaoativa	BOOL NOT NULL,
	eventos_idevento	INTEGER NOT NULL,
	utilizadores_idutilizador INTEGER NOT NULL,
	PRIMARY KEY(idnotificacao)
);

CREATE TABLE utilizadores_eventos (
	utilizadores_idutilizador INTEGER,
	eventos_idevento	 INTEGER,
	PRIMARY KEY(utilizadores_idutilizador,eventos_idevento)
);
```

```
ALTER TABLE utilizadores ADD UNIQUE (username);
ALTER TABLE inscricoes ADD CONSTRAINT inscricoes_fk1 FOREIGN KEY
(eventos_idevento) REFERENCES eventos(idevento);
ALTER TABLE inscricoes ADD CONSTRAINT inscricoes_fk2 FOREIGN KEY
(utilizadores_idutilizador) REFERENCES utilizadores(idutilizador);
ALTER TABLE historicoprecos ADD CONSTRAINT historicoprecos_fk1 FOREIGN
KEY (eventos_idevento) REFERENCES eventos(idevento);
ALTER TABLE comentarios ADD CONSTRAINT comentarios_fk1 FOREIGN KEY
(utilizadores_idutilizador) REFERENCES utilizadores(idutilizador);
ALTER TABLE comentarios ADD CONSTRAINT comentarios_fk2 FOREIGN KEY
(eventos_idevento) REFERENCES eventos(idevento);
ALTER TABLE notificacoes ADD CONSTRAINT notificacoes_fk1 FOREIGN KEY
(eventos_idevento) REFERENCES eventos(idevento);
ALTER TABLE notificacoes ADD CONSTRAINT notificacoes_fk2 FOREIGN KEY
(utilizadores_idutilizador) REFERENCES utilizadores(idutilizador);
ALTER TABLE utilizadores_eventos ADD CONSTRAINT utilizadores_eventos_fk1
FOREIGN KEY (utilizadores_idutilizador) REFERENCES
utilizadores(idutilizador);
ALTER TABLE utilizadores_eventos ADD CONSTRAINT utilizadores_eventos_fk2
FOREIGN KEY (eventos_idevento) REFERENCES eventos(idevento);
```

3.5. Outros Elementos Implementados

O SGBD utilizado foi o PostgreSQL e após a criação da Base de Dados houve a implementação de algumas funções, procedimentos, triggers, rules, exceptions para que a aplicação fosse ao encontro dos requisitos iniciais. A esse nível foram criados os seguintes elementos através do código apresentado:

- FUNÇÕES

```
-- FUNÇÃO PARA REEMBOLSAR PREÇO DO BILHETE
CREATE OR REPLACE FUNCTION verificaAlteraPrecoReembolso(IN aIdEvento integer)
RETURNS void
LANGUAGE plpgsql
AS $$
DECLARE
  precoReembolso float;
  aIdUtilizador integer;
BEGIN
  IF eventoExecutado(aIdEvento) IS NOT TRUE THEN
    FOR aIdUtilizador IN (SELECT idutilizador FROM
      utilizadores_inscritos_para_evento(aIdEvento))
    LOOP
      SELECT DISTINCT(precoentrada) INTO precoReembolso
      FROM utilizadores, inscricoes i, eventos, historicoprecos h
      WHERE idutilizador = i.utilizadores_idutilizador
       AND i.eventos_idevento = idevento
       AND i.eventos_idevento = h.eventos_idevento
       AND datainscricao >= (SELECT MAX(dataalteracaopreco) FROM historicoprecos
       WHERE eventos_idevento = aIdEvento)
       AND idevento = aIdEvento
       AND idutilizador = aIdUtilizador;
```

```
            IF precoReembolso IS NOT NULL THEN
              UPDATE utilizadores SET saldo = saldo + precoReembolso WHERE idutilizador
             = aIdUtilizador;
            ELSE
              SELECT preco INTO precoReembolso
              FROM utilizadores, inscricoes i, historicoprecos h
              WHERE idutilizador = i.utilizadores_idutilizador
               AND i.eventos_idevento = h.eventos_idevento
               AND datainscricao < dataalteracaopreco
               AND h.eventos_idevento = aIdEvento
               AND idutilizador = aIdUtilizador
               LIMIT 1;
              UPDATE utilizadores SET saldo = saldo + precoReembolso WHERE idutilizador
             = aIdUtilizador;
            END IF;
          END LOOP;
        END IF;
      END;
      $$;
```

```
-- FUNÇÃO PARA CANCELAR UM EVENTO QUANDO O SEU CRIADOR É BANIDO
      CREATE OR REPLACE FUNCTION cancelaEvento(IN aIdUtilizador integer)
      RETURNS void
      LANGUAGE plpgsql
      AS $$
      BEGIN
        UPDATE eventos SET eventoativo = false WHERE idevento = (SELECT evento_idevento
             FROM utilizadores_evento WHERE utilizadores_idutilizador = aIdUtilizador);
      END;
      $$;
```

```
-- FUNÇÃO PARA CANCELAR UM EVENTO
      CREATE OR REPLACE FUNCTION cancelaEventoId(IN aIdEvento integer)
      RETURNS void
      LANGUAGE plpgsql
      AS $$
      BEGIN
        UPDATE eventos SET eventoativo = false WHERE idevento = aIdEvento;
      END;
      $$;
```

```
-- FUNÇÃO PARA VER SE UM EVENTO CANCELADO JÁ PASSOU A DATA DE FIM DO EVENTO
      CREATE OR REPLACE FUNCTION eventoExecutado(IN aIdEvento integer)
      RETURNS boolean
      LANGUAGE plpgsql
      AS $$
      BEGIN
        SELECT datafim FROM eventos WHERE idevento = aIdEvento
        IF CURRENT_TIMESTAMP > datafim THEN
          RETURN true
        END IF;
      END;
      $$;
```

```
-- FUNÇÃO PARA OBTER OS TODOS UTILIZADORES DE UM DETERMINADO EVENTO
CREATE OR REPLACE FUNCTION utilizadoresInscritosDeterminadoEvento(IN aIdEvento
      integer)
RETURNS TABLE (idutilizador integer)
LANGUAGE plpgsql
AS $$
BEGIN
    RETURN QUERY
    SELECT idutilizador
    FROM eventos e, inscricoes i, utilizadores u
    WHERE e.idevento = i.eventos_idevento
      AND i.utilizadores_idutilizador = u.idutilizador
      AND e.idevento = aIdEvento
END;
$$;
```

```
-- FUNÇÃO PARA OBTER TODOS OS EVENTOS QUE A FASE DE INSCRICOES EXPIROU
CREATE OR REPLACE FUNCTION eventos_fase_inscricoes_expiradas()
RETURNS TABLE (idevento integer)
LANGUAGE plpgsql
AS $$
BEGIN
  RETURN query
  SELECT idevento
  FROM eventos
  WHERE datafiminscricao < CURRENT_TIMESTAMP
    AND datafim > CURRENT_TIMESTAMP;
  exception
    when no_data_found then
    raise exception 'Nenhum evento foi encontrado';
END;
$$;
```

```
-- FUNÇÃO PARA OBTER PREÇO ENTRADA
CREATE OR REPLACE FUNCTION getPrecoEntrada(IN aIdEvento integer)
RETURNS float
LANGUAGE plpgsql
AS $$
DECLARE
  precoOriginal float;
BEGIN
  SELECT precoentrada INTO precoOriginal
  FROM eventos
  WHERE idevento = aIdEvento;
  EXCEPTION
    when no_data_found then
    raise exception 'Preço não encontrado';
  RETURN precoOriginal;
END;
$$;
```

```
-- FUNÇÃO PARA OBTER SALDO UTILIZADOR
CREATE OR REPLACE FUNCTION getSaldo(IN aIdUtilizador integer)
RETURNS float
```

```
LANGUAGE plpgsql
AS $$
DECLARE
  saldoUtilizador float;
BEGIN
  SELECT saldo INTO saldoUtilizador
  FROM utilizadores
  WHERE idutilizador = aIdUtilizador;
  RETURN saldoUtilizador;
  exception
  when no_data_found then
    raise exception 'Saldo não encontrado';
END;
$$;
```

```
-- FUNÇÃO PARA OBTER NUMEROS DE LUGARES OCUPADOS
CREATE OR REPLACE FUNCTION getNumeroLugares(IN aIdEvento integer)
RETURNS integer
LANGUAGE plpgsql
AS $$
DECLARE
  lugares int;
BEGIN
  SELECT COUNT(*) INTO lugares
  FROM inscricoes
  WHERE eventos_ideventos = aIdEvento
   AND inscricaoativa = true;
  RETURN lugares;
END;
$$;
```

```
-- FUNÇÃO PARA ADICIONAR UTILIZADOR
CREATE OR REPLACE FUNCTION registarUtilizador (aUsername varchar, aPassword
      varchar, aTipoUtilizador boolean)
RETURNS void
LANGUAGE plpgsql
AS $$
BEGIN
    INSERT INTO utilizadores (username, password, tipoutilizador, saldo, ativo)
    VALUES (aUsername, aPassword, aTipoUtilizador, 0, true);
END;
$$;
```

- PROCEDIMENTOS

```
-- PROCEDIMENTO PARA GUARDAR O PREÇO ANTIGO
CREATE PROCEDURE inserirAlteracaoPreco(precoAntigo float, idEvento integer)
LANGUAGE SQL
AS $$
  INSERT INTO historicoprecos (preco, dataalteracaopreco, eventos_idevento)
      VALUES (precoAntigo, CURRENT_TIMESTAMP, idEvento);
$$;
```

```
-- CRIAR NOTIFICAÇÃO QUANDO FASE DE INSCRIÇÕES TERMINA
CREATE PROCEDURE notificacao_fim_fase_inscricoes ()
LANGUAGE plpgsql
AS $$
DECLARE
  aIdevento integer;
  mensagem varchar;
BEGIN
  FOR aIdevento IN (select idevento FROM eventos_fase_inscricoes_expiradas())
      LOOP
    mensagem = 'Fase de inscricao no evento ' || (SELECT nomeevento FROM eventos
      WHERE idevento = aIdevento) || ' terminou';
    IF (mensagem, aIdevento) NOT IN (SELECT notificacao, eventos_idevento FROM
      notificacoes) THEN
    INSERT INTO notificacoes (notificacao, datanotificacao, notificacaoativa,
      eventos_idevento, utilizadores_idutilizador)
      VALUES (mensagem, CURRENT_TIMESTAMP, true, aIdevento, NULL);
    END IF;
  END LOOP;
END;
$$;
```

```
-- INSCRIÇÃO NUM EVENTO
CREATE PROCEDURE inscricao_evento (aIdUtilizador integer, aIdEvento integer,
      precoEntrada float)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO inscricoes (inscricaoativa, datainscricao, subscricaocomentarios,
      eventos_idevento, utilizadores_idutilizador) VALUES (1, CURRENT_DATE, 0,
      aIdEvento, aIdUtilizador);
  UPDATE utilizadores SET saldo = (saldo - precoEntrada) WHERE idUtilizador =
      aIdUtilizador;
END;
$$;
```

```
-- UPDATE DO SALDO
CREATE PROCEDURE atualizarSaldo (aIdUtilizador integer, aSaldo float)
LANGUAGE sql
AS $$
BEGIN
  UPDATE utilizadores SET saldo = saldo + aSaldo WHERE idUtilizador =
      aIdUtilizador;
END;
$$;
```

```
-- CRIAR EVENTO
CREATE PROCEDURE criarEvento (aNomeEvento varchar, aTipo varchar, aLocalEvento
      varchar, aDataInicio timestamp, aDataFim timestamp, aDataInscricaoInicio
      timestamp, aDataInscricaoFim timestamp, aLugares integer, aPrecoEntrada
      float, aEventoAtivo boolean, aIdUtilizador integer))
LANGUAGE plpgsql
AS $$
DECLARE
```

```
  aIdEvento integer;
BEGIN
  INSERT INTO eventos (nomeevento, tipo, localevento, datainicio, datafim,
      datainscricaoinicio, datainscricaofim, lugares, precoentrada, eventoativo)
      VALUES (aNomeEvento, aTipo, aLocalEvento, aDataInicio, aDataFim,
      aDataInscricaoInicio, aDataInscricaoFim, aLugares, aPrecoEntrada,
      aEventoAtivo);
  aIdEvento = (SELECT idevento FROM eventos WHERE nomeevento = aNomeEvento);
  INSERT INTO utilizadores_eventos (utilizadores_idutilizador, eventos_idevento)
      VALUES (aIdUtilizador, aIdEvento);
  INSERT INTO notificacoes (notificacao, datanotificacao, notificacaoativa,
      eventos_idevento, utilizadores_idutilizador) VALUES ('O evento ' ||
      aNomeEvento || ' foi criado', CURRENT_TIMESTAMP, true, aIdEvento,
      aIdUtilizador);
END;
$$;
```

```
-- ATUALIZAR EVENTO
CREATE PROCEDURE atualizarEvento(aIdEvento integer, aNomeEvento varchar, aTipo
      varchar, aLocalEvento varchar, aDataInicio timestamp, aDataFim timestamp,
      aDataInscricaoInicio timestamp, aDataInscricaoFim timestamp, aLugares
      integer, aPrecoEntrada float, aIdUtilizador integer)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE eventos SET nomeevento = aNomeEvento, tipo = aTipo, localevento =
      aLocalEvento, datainicio = aDataInicio, datafim = aDataFim,
      datainscricaoinicio = aDataInscricaoInicio, datainscricaofim =
      aDataInscricaoFim, lugares = aLugares, precoentrada = aPrecoEntrada,
      eventoativo = True WHERE idevento = aIdEvento;
  INSERT INTO notificacoes (notificacao, datanotificacao, notificacaoativa,
      eventos_idevento, utilizadores_idutilizador) VALUES ('O evento ' ||
      aNomeEvento || ' foi atualizado', CURRENT_TIMESTAMP, true, aIdEvento,
      aIdUtilizador);
END;
$$;
```

```
-- INSERIR COMENTÁRIO
CREATE PROCEDURE inserirComentario (aComentario varchar, aIdUtilizador integer,
      aIdEvento integer)
LANGUAGE plpgsql
AS $$
BEGIN
  INSERT INTO comentarios (comentario, comentarioativo,
      utilizadores_idutilizador, eventos_idevento) VALUES (aComentario, True,
      aIdUtilizador, aIdEvento)
END;
$$;
```

```
-- SUBSCREVER COMENTÁRIOS
CREATE PROCEDURE subscreverComentario (aTipo boolean, aIdUtilizador integer,
      aIdEvento integer)
LANGUAGE plpgsql
```

```
AS $$
BEGIN
  UPDATE inscricoes SET subscricaocomentarios = aTipo WHERE eventos_idevento =
      aIdEvento AND utilizadores_idutilizador = aIdUtilizador;
END;
$$;
```

```
-- ATIVAR / DESATIVAR UTILIZADOR
CREATE PROCEDURE ativaDesativaUtilizador (aAtivo boolean, aIdUtilizador integer)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE utilizadores SET utilizadorativo = (not aAtivo) WHERE idutilizador =
      aIdUtilizador;
END;
$$;
```

```
-- CANCELAR EVENTO
CREATE PROCEDURE cancelaEventoUtilizador (aIdEvento integer)
LANGUAGE plpgsql
AS $$
BEGIN
  UPDATE eventos SET eventoativo = false WHERE idevento = aIdEvento;
END;
$$;
```

- RULES

```
-- REEMBOLSAR UTILIZADORES QUANDO EVENTO FOR CANCELADO
CREATE RULE reembolsoUtilizadores AS ON UPDATE TO eventos
WHERE (OLD.eventoativo = true AND NEW.eventoativo = false)
DO ALSO
EXECUTE FUNCTION verificaAlteraPrecoReembolso(NEW.idevento)
```

```
-- REEMBOLSAR UTILIZADORES QUANDO UTILIZADOR CRIADOR DO EVENTO FOR BANIDO
CREATE OR REPLACE RULE inativarEvento AS ON UPDATE TO utilizadores
WHERE (OLD.ativo = true AND NEW.ativo = false)
DO ALSO
EXECUTE FUNCTION cancelaEvento(OLD.idUtilizador)
```

- TRIGGERS E RESPETIVAS FUNÇÕES

```
-- CRIAR NOTIFICAÇÃO QUANDO HOUVER ALTERAÇÕES NOS PREÇOS
CREATE OR REPLACE FUNCTION funcao_historico_precos_eventos() RETURNS TRIGGER AS
$$
DECLARE
 aIdUtilizador integer;
BEGIN
    SELECT utilizadores_idutilizador INTO aIdUtilizador FROM utilizadores_eventos
WHERE eventos_idevento = NEW.eventos_idevento;
```

```
        INSERT INTO notificacoes (notificacao, datanotificacao, notificacaoativa,
    eventos_idevento, utilizadores_idutilizador)
        VALUES ('O preço de entrada de um evento foi alterado', CURRENT_TIMESTAMP,
    true, NEW.eventos_idevento, aIdUtilizador);
        RETURN NEW;
    END;
    $$ LANGUAGE plpgsql;

    CREATE TRIGGER historico_precos_eventos
    AFTER INSERT ON historicoprecos
    FOR EACH ROW
    EXECUTE FUNCTION funcao_historico_precos_eventos();
```

```
-- CRIAR NOTIFICAÇÃO QUANDO HOUVER INSCRIÇÕES
    CREATE OR REPLACE FUNCTION funcao_inscricao_eventos()
    RETURNS TRIGGER
    LANGUAGE plpgsql
    AS $$
    declare
      aIdUtilizador integer;
      aNomeEvento varchar;
    BEGIN
        SELECT utilizadores_idutilizador INTO aIdUtilizador FROM utilizadores_eventos
    WHERE eventos_idevento = NEW.eventos_idevento;
        SELECT nomeevento INTO aNomeEvento FROM eventos WHERE idevento =
    NEW.eventos_idevento;
        INSERT INTO notificacoes (notificacao, datanotificacao, notificacaoativa,
    eventos_idevento, utilizadores_idutilizador)
        VALUES ('Nova inscrição no evento "' || aIdEvento || '"', CURRENT_TIMESTAMP,
    true, NEW.eventos_idevento, aIdUtilizador);
        RETURN NEW;
    END;
    $$

    CREATE TRIGGER inscricao_eventos
    AFTER INSERT ON inscricoes
    FOR EACH ROW
    EXECUTE FUNCTION funcao_inscricao_eventos();
```

```
-- CRIAR NOTIFICAÇÃO QUANDO EVENTO FOR CANCELADO
    CREATE OR REPLACE FUNCTION funcao_cancelamento_eventos()
    RETURNS TRIGGER
    LANGUAGE plpgsql
    AS $$
    declare
      aIdUtilizador integer;
    BEGIN
        IF (OLD.eventoativo = true AND NEW.eventoativo = false) THEN
            SELECT utilizadores_idutilizador INTO aIdUtilizador FROM
    utilizadores_eventos WHERE eventos_idevento = NEW.idevento;
            INSERT INTO notificacoes (notificacao, datanotificacao, notificacaoativa,
    eventos_idevento, utilizadores_idutilizador)
            VALUES ('O evento "' || NEW.nomeevento || '" foi cancelado',
    CURRENT_TIMESTAMP, true, NEW.idevento, aIdUtilizador);
        END IF;
```

```
        RETURN NEW;
    END;
    $$

    CREATE TRIGGER cancelamento_eventos
    AFTER UPDATE OF eventoativo ON eventos
    FOR EACH ROW
    EXECUTE FUNCTION funcao_cancelamento_eventos();
```

```
-- ATUALIZAR VIEW MATERIALIZADA QUANDO HÁ NOVA NOTIFICAÇÃO
    CREATE OR REPLACE FUNCTION funcao_atualiza_view()
    RETURNS TRIGGER
    LANGUAGE plpgsql
    AS $$
    BEGIN
        REFRESH MATERIALIZED VIEW eventosNotificacoes;
        RETURN NULL;
    END;
    $$

    CREATE TRIGGER atualiza_view
    AFTER INSERT ON notificacoes
    FOR EACH ROW
    EXECUTE FUNCTION funcao_atualiza_view();
```

- VIEWS

```
-- VIEW UTILIZADORES POR EVENTO
    CREATE VIEW utilizadoresEvento AS
    SELECT e.idevento, e.nomeevento, u.idutilizador, u.username
    FROM eventos e, inscricoes i, utilizadores u
    WHERE e.idevento = i.eventos_idevento
        AND i.utilizadores_idutilizador = u.idutilizador;
```

```
-- VIEW TRÊS EVENTOS COM MAIS INSCRIÇÕES
    CREATE OR REPLACE VIEW topEventos AS
    SELECT e.idevento AS idEvento, e.nomeevento AS nomeEvento, COUNT(i.idinscricao)
    AS totalInscricoes
    FROM eventos e, inscricoes i
    WHERE e.idevento = i.eventos_idevento
    GROUP BY e.idevento, e.nomeevento
    ORDER BY totalInscricoes DESC
    LIMIT 3;
```

```
-- VIEW MATERIALIZADA NOTIFICAÇÕES POR UTILIZADOR E EVENTO
    CREATE MATERIALIZED VIEW eventosNotificacoes AS
    SELECT u.idutilizador AS idu,
        u.username AS nomeu,
        e.idevento AS ide,
        e.nomeevento AS nomee,
        n.idnotificacao AS idn,
        n.notificacao AS mensagemn,
```

```
        n.datanotificacao AS datan,
        i.datainscricao AS datai
        true AS ativan
    FROM utilizadores u
        JOIN inscricoes i ON u.idutilizador = i.utilizadores_idutilizador
        JOIN eventos e ON i.eventos_idevento = e.idevento
        LEFT JOIN notificacoes n ON e.idevento = n.eventos_idevento
    WHERE n.idnotificacao IS NOT NULL
    ORDER BY u.idutilizador, e.idevento, n.datanotificacao;
```

- INDEXES E CURSORES

A articulação e otimização dos recursos, bem como a forma utilizada para percorrer a base de dados, foi alavancada na utilização de cursores implícitos e de indexes. Neste projeto, consideraram-se este tipo de cursores os mais apropriados dada a sua implementação ser simplificada e cumprirem com o preceituado. A criação de indexes, por exemplo, na tabela eventos para a coluna idevento e tipo (i.e., ind_idevento, ind_tipo) na tabela notificações e na utilizadores para a coluna idnotificacao e idutilizador, respetivamente (i.e., ind_idnotificacao, ind_idutilizador) foram modos de agilizar as pesquisas feitas nestas tabela.

3.6. Bibliotecas e Ficheiros (Rotas, Pedidos à Base de Dados e Json)

O funcionamento da API REST só será pleno se forem considerados três ficheiros que seguirão anexos ao presente relatório, nomeadamente, o index.py, o bd.py e o Trabalho Final - BDII.postman_collection.json. No ficheiro .json encontrar-se-á o "menu" (i.e., as chamadas para as rotas) com envio de informações que permitem a interação com a base de dados. No ficheiro index.py estão as rotas propriamente ditas, com as mensagens que serão devolvidas e as chamadas para funções que se irão conectar à base de dados. No ficheiro bd.py encontram-se as conexões e pedidos desencadeados mediante a rota selecionada que levam à execução de ações na base de dados.

Importa salientar que há bibliotecas que devem ser importadas e outras extensões instaladas. No início tanto do ficheiro bd.py, como do index.py encontram-se os imports que devem ser tidos em consideração (e.g., import os | from re import I | import jwt | import psycopg2 | from flask import Flask, jsonify, request).

4. Discussão

Os resultados obtidos foram congruentes com os pressupostos iniciais e a aplicação funcionou corretamente, conforme expectável. Tal é observável através do manual do utilizador onde os exemplos do sistema em funcionamento demonstram a concretização das ações sem erros ou disparidades face ao pretendido. Não obstante importa referir que as informações por vezes ambíguas em requisitos definidos para o sistema levaram a considerações muito particulares baseadas na interpretação dos desenvolvedores. Neste sentido reforça-se que após o cliente tomar contacto com a aplicação dispõe de um período de testes e de revisão onde é permitida a inclusão ou aprimoramento de funcionalidades. As testagens não evidenciaram erros e os requisitos enunciados consideram-se totalmente implementados. Não houve deteção de disparidades ou inconformidades face ao funcionamento expectável do sistema de gestão de eventos.

No entender dos desenvolvedores a aplicação está operacional, ficando alguns pormenores por serem aprimorados no desenvolvimento Frontend. Concretamente, os menus presentes na aplicação e o acesso aos serviços da plataforma devem ser disponibilizados mediante o tipo de utilizador autenticado.

5. Conclusão

Este trabalho prático, tendo como objetivo central a criação de uma base de dados com implementação de funcionalidades associadas, permitiu a construção de uma aplicação para gestão de eventos com recurso a uma API REST, usando o software POSTMAN e a consolidação de conhecimentos e competências adquiridas nas Unidades Curriculares de Bases de Dados I e II.

Para este projeto, foi necessário rever conteúdos adquiridos anteriormente, aprender a manusear a plataforma online Onda, o software POSTMAN e o pgAdmin. Todo o trabalho desenvolvido na primeira fase do projeto foi aproveitado, sendo que não houve qualquer alteração de fundos nos diagramas, scripts e planos transacionais criados. Referir que o único ponto alterado do primeiro projeto para esta entrega final foi a inclusão de um atributo na tabela eventos. Já no que respeita à fase final foram implementados na base de dados rules, triggers, funções, views, exceções, indexes, procedimentos, cursores, encriptações que otimizaram o funcionamento da base de dados, garantiram uma interação mais segura pela API, tornando-a uma ferramenta com potencial para implementação no mercado.

Em síntese, este projeto tendo chegado à fase final e cumprindo com excelência os objetivos iniciais continuará em contínuo desenvolvimento para melhorar não só os recursos oferecidos aos clientes como para possibilitar uma articulação cada vez mais simplificada da informação e voltando a resposta da plataforma para as necessidades do mercado.

Forças

As principais forças deste trabalho que o distingue das outras abordagens, é de já existir um bom conhecimento deste tema, com uma estruturação articulada com o conhecimento científico mais atualizado e a operacionalização da base de dados conforme os pressupostos, que fazem da proposta apresentada uma opção devidamente consolidada que irá garantir ao projeto mais eficácia e viabilidade.

Limitações

As principais limitações encontradas no trabalho foram associadas ao tempo escasso para a sua execução. Não obstante a dedicação nesta fase final, as horas que foram dispensadas por dia para garantir que estava tudo conforme, foram a salvaguarda necessária para não ter havido prejuízo na trabalho desenvolvido, nem no funcionamento da aplicação.

Tabela 8 – Tempo Utilizado com a Unidade Curricular de Tecnologias e Arquitetura de Computadores.

Tempo Utilizado com a UC por Aluno				
Aulas	**Relatório TP2**	**TP 2**	**Estudo**	**Total**
5h / semana	± 7h / relatório	35h	1h / semana	68h

6. Referências

Marques, G. (2024). *TPBD2 - Enunciado* [Documentos de Aula]. IPC: ESTGOH.

Veloso, M. (2023). Slides e Materiais de Apoio às Aulas. [PDF de apoio à UC de Bases de Dados, lecionada na ESTGOH - IPC].