Uma imagem com texto

Descrição gerada automaticamente

***Projeto de Base de Dados  
Leilões Online***

Diogo Jordão Filipe, José Miguel Silva Gomes, Pedro Tiago dos Santos Marques  
FCTUC, Departamento de Engenharia Informática,  
Universidade de Coimbra, Portugal

{uc2018288391, uc2018286225, uc2018285632}@student.uc.pt

31, maio 2021

Índice

[1 Manual do Utilizador 3](#_Toc73372302)

[2 Endpoints Implementados 6](#_Toc73372303)

[2.1 Registo de Utilizadores 6](#_Toc73372304)

[2.2 Autenticação de utilizadores 6](#_Toc73372305)

[2.3 Registo do perfil Vendedor 6](#_Toc73372306)

[2.4 Registo do perfil Comprador 6](#_Toc73372307)

[2.5 Registo do perfil Administrador 7](#_Toc73372308)

[2.6 Criar um novo leilão 7](#_Toc73372309)

[2.7 Listar todos os leilões existentes 7](#_Toc73372310)

[2.8 Pesquisar leilões existentes 8](#_Toc73372311)

[2.9 Consultar detalhes de um leilão 8](#_Toc73372312)

[2.10 Listar todos os leilões em que o utilizador tenha atividade 9](#_Toc73372313)

[2.11 Editar propriedades de um leilão 9](#_Toc73372314)

[2.12 Consultar versões anteriores de um leilão 9](#_Toc73372315)

[2.13 Escrever mensagem no mural do leilão 10](#_Toc73372316)

[2.14 Consultar caixa de entrada 10](#_Toc73372317)

[2.15 Termino do leilão na data, hora e minuto marcados 11](#_Toc73372318)

[2.16 Administrador cancelar leilão 11](#_Toc73372319)

[2.17 Administrador banir utilizador 11](#_Toc73372320)

[2.18 Administrador obtêm estatísticas de atividade da aplicação 11](#_Toc73372321)

[3 Lista de Erros 13](#_Toc73372322)

[4 Manual de Instalação 14](#_Toc73372323)

[5 Artefactos 15](#_Toc73372324)

[5.1 Diagrama ER 15](#_Toc73372325)

[5.2 Tabelas 15](#_Toc73372326)

[5.3 Análise da evolução do diagrama ER 16](#_Toc73372327)

[6 Construção da aplicação 17](#_Toc73372328)

[6.1 Triggers Implementados 18](#_Toc73372329)

[7 Plano de Desenvolvimento 19](#_Toc73372330)

# Manual do Utilizador

Após a instalação correta do software fornecido, o utilizador tem a liberdade de escolher qualquer um dos *endpoints* que importou para o *Postman*. Estes estão organizados de maneira sugestiva, seguindo uma ordem lógica de utilização.

Uma imagem com texto

Descrição gerada automaticamente

Figura 1 Lista de endpoints disponíveis

Para utilização dos endpoints basta seguir a documentação explícita na secção 2. Tomemos como exemplo o **“Registo de utilizador”:**

**req** **POST** <http://localhost:8080/dbproj/user>

Body:

{“username”: “user1”,

“email”: “user1@email.pt”,

“password": “123”}

**res** Em caso de sucesso {“userId”: novoUserId}

Ou em caso de erro {“erro”: codigoErro}

Para tal instrução basta inserir a seguinte informação no *Postman*:

Uma imagem com texto, captura de ecrã, monitor, preto

Descrição gerada automaticamente

Figura 2 Exemplo utilização do Postman

Neste caso, após pressionar o botão “*SEND*”, obtemos erro “001” o que implica que estamos a tentar inserir um utilizador cujo *username* ou *email* já está registado na base de dados. Corrigindo o erro obtemos a seguinte mensagem:

{“userId”: idUtilizadorInserido}

Note-se que todos os endpoints debaixo de “Autenticação de utilizador” exigem que um código de autenticação seja enviado no *header* do Postman. Este token é fornecido pela API quando um utilizador se autentica.

Uma imagem com texto

Descrição gerada automaticamente

Figura 3 Exemplo de como enviar um token de autenticação no Postman

Finalmente, chamamos a atenção para as seguintes restrições:

1. Todos os utilizadores podem criar perfil de vendedor e/ou perfil de comprador. Estes permitem criar leilões e licitar respetivamente;
2. Apenas donos de leilões podem realizar o término do seu leilão ou alterar propriedades do mesmo;
3. O *endpoint* “Consultar caixa de entrada” diz respeito às mensagens e notificações correspondentes de um utilizador (cujo *authToken* foi fornecido);
4. Só utilizadores do tipo administrador são capazes de executar os três últimos endpoints;
5. Todos os erros se encontram documentados na secção 3 deste documento;
6. É possível consultar versões anteriores de um leilão através do seguinte endpoint: “Consultar versões anteriores de um leilão”;

# Endpoints Implementados

## Registo de Utilizadores

**req** **POST** <http://localhost:8080/dbproj/user>

Body: { “username”: “user1”,

“email”: “user1@email.pt”,

“password”: “123”

}

**res** Em caso de sucesso {“userId”: novoUserId}

Ou em caso de erro {“erro”: codigoErro}

## Autenticação de utilizadores

**req** **PUT** <http://localhost:8080/dbproj/user>

Body: { “username”: “user1”,

“password”: “123”

}

**res** Em caso de sucesso {“authToken”: authToken}

Ou em caso de erro {“erro”: AuthError}

## Registo do perfil Vendedor

**req** **POST** <http://localhost:8080/dbproj/vendedor>

Enviar authToken no **header**

****

Body: {

“moradaEnvio”: “RuaNr1”

}

**res** Em caso de sucesso {“vendedorId”: novoVendedorId}

Ou em caso de erro {“erro”: codigoErro

## Registo do perfil Comprador

**req** **POST** <http://localhost:8080/dbproj/comprador>

Enviar authToken no **header**

****

Body: {

“moradaRececao”: “RuaNr99”

}

**res** Em caso de sucesso {“compradorId”: novoCompradorId}

Ou em caso de erro {“erro”: codigoErro}

## Registo do perfil Administrador

**req** **POST** <http://localhost:8080/dbproj/admin>

Enviar authToken no **header**

****

**res** Em caso de sucesso {“adminId”: novoAdminId}

Ou em caso de erro {“erro”: codigoErro}

## Criar um novo leilão

**req** **POST** <http://localhost:8080/dbproj/leilao>

Enviar authToken no **header**

****

Body: {

“leilaoPrecoMinimo”: “500”,

“leilaoTitulo”: “Titulo Leilao”,

“leilaoDescricao”: “Descricao PT”,

“leilaoDataFim”: “2021-05-31 23:59”,

“artigoId”: “1112223334”,

“nomeArtigo”: “ArtigoPT”

}

**res** Em caso de sucesso {“leilaoId”: novoLeilaoId}

Ou em caso de erro {“erro”: codigoErro}

## Listar todos os leilões existentes

**req** **GET** <http://localhost:8080/dbproj/leiloes>

**res** [

{“leilaoId”: leilaoId1, “descricao”: “Descrição do Leilão 1”},

{“leilaoId”: leilaoId2, “descricao”: “Descrição do Leilão 2”},

{“leilaoId”: leilaoId3, “descricao”: “Descrição do Leilão 3”}

]

## Pesquisar leilões existentes

**req** **GET** [http://localhost:8080/dbproj/leiloes/{keyword}](http://localhost:8080/dbproj/leiloes/%7bkeyword%7d)

**res** [

{“leilaoId”: leilaoId1, “descricao”: “Descrição do Leilão 1”},

{“leilaoId”: leilaoId3, “descricao”: “Descrição do Leilão 3”}

]

## Consultar detalhes de um leilão

**req** **GET** [http://localhost:8080/dbproj/leilao/{leilao\_id}](http://localhost:8080/dbproj/leilao/%7bleilao_id%7d)

Enviar authToken no **header**

****

**res** Em caso de sucesso

[

{

“DETALHES DO LEILAO”: “1”

},

{“leilaoid”: 1,

“titulo”: “NovoTituloLeilaoNovo”,

“descricao”: “Nova DEscricaoNovo”,

“datafim”: “Tue, 01 Jun 2021 20:00:00 GMT”,

“artigoid”: “1234654345”,

“nomeartigo”: “Artigo 3”,

“maiorlicitcao”: 700,

“username”: “user5”},

{

“DETALHES DO MURAL LEILAO”: “1”

},

{“id”: 1,

“comentario”: “\"Gostei!\"”,

“momento”: “Mon, 31 May 2021 16:05:49 GMT”,

“username”: “user3”},

{

“DETALHES DAS LICITACOES LEILAO”: “1”

},

{“id”: 5,

“valor”: 700,

“valida”: true,

“username”: “user2”}

]

Ou em caso de erro {“erro”: codigoErro}

## Listar todos os leilões em que o utilizador tenha atividade

**req** **GET** <http://localhost:8080/dbproj/leiloesAtividade>

Enviar authToken no **header**

****

**res** [

{“leilaoId”: leilaoId1, “descricao”: “Descrição do Leilão 1”, /\*\* OS RESTANTES DETALHES e MENSAGENS\*\*/},

{“leilaoId”: leilaoId2, “descricao”: “Descrição do Leilão 2”, /\*\* OS RESTANTES DETALHES e MENSAGENS\*\*/}

]

## Editar propriedades de um leilão

**req** **PUT** [http://localhost:8080/dbproj/leilao/{leilaoId}](http://localhost:8080/dbproj/leilao/%7bleilaoId%7d)

Enviar authToken no **header**

****

Body: {

“novoTitulo”: “TituloLeilaoNew1”,

“novaDescricao”: “DescricaoNew1”

} /\*Ou apenas um dos campos desejados\*/

**res** Em caso de sucesso {“leilaoId”: novoLeilaoId, /\*\* informação completa do leilao\*\*/}

Ou em caso de erro {“erro”: codigoErro}

## Consultar versões anteriores de um leilão

**req** **GET** [http://localhost:8080/dbproj/versoes/{leilaoId}](http://localhost:8080/dbproj/versoes/%7bleilaoId%7d)

**res** [

{“versaoId”: 1,

“titulo”: “NovoTituloLeilaoNovo”,

“descricao”: “Nova DEscricaoNovo” },

{“versaoId”: 2,

“titulo”: “TituloLeilaoPT9”,

“descricao”: “DescricaoPT9” }

]

## Escrever mensagem no mural do leilão

**req** **POST** [http://localhost:8080/dbproj/msgMural/{leilaoId}](http://localhost:8080/dbproj/msgMural/%7bleilaoId%7d)

Enviar authToken no **header**

****

Body: { "mensagem": "Mensagem de teste 1"}

**res** Em caso de sucesso{“mensagemId”: novaMensagemId}

Ou em caso de erro {“erro”: codigoErro}

## Consultar caixa de entrada

**req** **GET** <http://localhost:8080/dbproj/caixaEntrada>

Enviar authToken no **header**

****

**res** Em caso de sucesso

[

[

“Notificações”,

{“LeilaoId”: 7, “Aviso”: “Licitação ultrapassada.”, “Momento”: “2021-05-27 19:13:49”},

{“LeilaoId”: 2, “Aviso”: “Licitação ultrapassada.”, "Momento": "2021-05-27 19:13:49"}

],

[

“Mensagens”,

{“Username”: “User 1”, “LeilaoId”: 2, “Momento”: “2021-05-27 11:17:54”, “Comentario”: “oi” },

{“Username”: “User 2”, "LeilaoId": 1, “Momento”: “2021-05-27 11:18:51”, “Comentario”: “ola”}

]

]

## Término do leilão na data, hora e minuto marcados

**req** **PUT** <http://localhost:8080/dbproj/terminarLeiloes>

Enviar authToken no **header**

****

Body: {“idLeilao”: idLeilao }

**res** Em caso de sucesso

{“leilaoId”: idLeilao, “vencedor”: “username1”} ou

{“leilaoId”: idLeilao, “cancelado”: AdminId} ou

{“leilaoId”: idLeilao, “aviso”: “nenhum vencedor”}

// Envia também uma notificação ao vencedor e outros licitadores

Ou em caso de erro {“erro”: codigoErro}

## Administrador cancelar leilão

**req** **PUT** [http://localhost:8080/dbproj/cancelarLeilao/{idLeilao}](http://localhost:8080/dbproj/cancelarLeilao/%7bidLeilao%7d)

Enviar authToken no **header**

****

**res** {“idLeilao”: idLeilao} Ou em caso de erro {“erro”: codigoErro}

## Administrador banir utilizador

**req** **PUT** <http://localhost:8080/dbproj/leilao/ban/>

Enviar authToken no **header**

****

Body: {“userID”: idUser}

**res** Em caso de sucesso {“userID”: idUser}

Ou em caso de erro {“erro”: codigoErro}

## Administrador obtêm estatísticas de atividade da aplicação

**req** **GET** <http://localhost:8080/dbproj/adminStats>

Enviar authToken no **header**

****

**res** [

[

“Top 2 Utilizadores com mais leiloes criados”,

{“Leiloes Criados”: 2, “Username”: “User2”, “userId”: 2},

{“Leiloes Criados”: 1, “Username”: “User1”, “userId”: 1}

],

[

“Top 2 Utilizadores que mais leiloes venceram”,

{“Leiloes Vencidos”: 2, “Username”: “User1”, “userId”: 1},

{“Leiloes Vencidos”: 1, “Username”: “User2”, “userId”:}

],

{“Numero total de leiloes nos ultimos 10 dias”: 3}

] /\*Top 2 é figurativo, pode ir até 10\*/

# Lista de Erros

- 001 → Utilizador Duplicado;

- 002 → Input Inválido (tamanho das variáveis);

- 003 → Payload incorreto (nome das variáveis);

- 004 → Credenciais de login incorretas/user inválido;

- 005 → Token de autenticação não corresponde a nenhum utilizador;

- 006 → Vendedor Inválido;

- 007 → Leilão inativo/inexistente;

- 008 → Vendedor não tem leilões a decorrer;

- 009 → Vendedor inexistente;

- 010 → Utilizador inexistente;

- 011 → Utilizador não é um admin/não existe;

- 012 → Licitação menor/igual que a atual ou não excede preço mínimo;

- 013 → Comprador inválido;

- 014 → Utilizador não é um comprador/não existe;

- 015 → Um vendedor não pode licitar no próprio leilão;

- 016 → Vendedor Duplicado;

- 017 → Comprador Duplicado;

- 018 → Administrador Duplicado;

- 019 → Comprador sem licitações;

- 020 → Licitação única não ultrapassada;

- 021 → Vendedor não é dono desse leilão;

- 022 → Utilizador não é um vendedor/não existe;

- 023 → Leilão ainda não terminou;

- 024 → Criar leilão com um artigo presente noutro leilão ativo;

- 025 → Utilizador Inexistente/já banido;

- 999 → Outro erro.

# Manual de Instalação

Antes de interagir com a aplicação, o utilizador tem de instalar alguns programas no seu computador:

* Docker
  + <https://www.docker.com/>
* PostgreSQL no Docker
  + <https://hub.docker.com/_/postgres>

Após a instalação destas ferramentas, é necessário abrir o *Docker* e executar o script incluído “docker-compose-python-psql.sh”. Este vai criar dois *containers* no Docker, um que corresponde à API (“api”) e outro que é a base de dados (“db”).

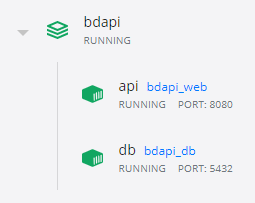


Figura 4 Containers criados no Docker

Para aceder aos *endpoints* desenvolvidos na API pode-se utilizar um *browser*, como por exemplo o *Google* *Chrome*. Todavia, recomendamos a utilização do *Postman* que é uma ferramenta muito prática no contexto do trabalho.

* *Postman*
  + <https://www.postman.com/downloads/>

A última coisa a fazer é importar para o Postman o ficheiro “ProjetoBDendpoints.json” que representa uma coleção de todos os *endpoints* desenvolvidos.

# Artefactos

## Diagrama ER

## Tabelas

## Análise da evolução do diagrama ER

Para concretizar este trabalho aplicámos várias alterações ao diagrama ER. Note-se a inserção da tabela “notificação” que aliada à tabela “mensagem” (que representa o mural de um leilão) permite constituir uma caixa de entrada personalizada para cada utilizador. Além disso, eliminámos a tabela “artigo” visto que, na nossa perspetiva, não faz sentido cada utilizador registar os seus produtos na base de dados (para posterior escolha de qual deles quer leiloar) porque não se adequa à realidade. Ao criar um leilão, o vendedor já deve ter em mente qual artigo deseja vender. Por causa disso, integrámos na tabela “leilão” todas as propriedades pertinentes de um artigo, tal como o seu identificador e nome.

Como constituímos um grupo de 3 optámos por tornar a tabela “administrador” uma especialização de “utilizador”, lembrando que tanto “vendedor” e “comprador” são entidades fracas de “utilizador” que podem criar leilões e licitar respetivamente.

Finalmente, realçamos a tabela “versao” que agora é uma entidade fraca de leilão e permite guardar edições textuais antigas de um registo da tabela “leilão”.

# Construção da aplicação

Tal como foi dito anteriormente, a aplicação corre num *container* da aplicação *Docker* e de forma que seja possível executar o programa é necessário estabelecer ligação da API à base de dados criada para o efeito. Esta conexão servirá também para verificar o funcionamento de todas os *endpoints* implementados e simular um cliente. Para tal, utilizámos a plataforma *Postman* que nos permite verificar todas as interações *request*/*response* que um utilizador teria ao utilizar a aplicação.

Para este projeto, foi escolhida a linguagem de programação *Python* e, para fazer a integração com a base de dados criada, foi utilizado o adaptador Psycopg2 que nos permite fazer todas as operações pretendidas para o nosso DBMS.

Utilizámos também a biblioteca *CryptContext* de forma a encriptar a palavra-passe do utilizador na base de dados e, desencriptá-la quando for necessário. Esta biblioteca usa funções *hash* criptográficas SHA-256 que, também é usada para encriptar a palavra-passe da conexão à base de dados. Para além disso, note-se que temos a password de acesso à base de dados encriptada no ficheiro de texto “dbPass.txt” que é desencriptada na lógica da API.

Por fim, utilizamos a biblioteca jwt (JSON Web Token) que nos permite gerar um *token* diferente a cada vez que é criado um utilizador. Este token é único para cada pessoa e é usado como forma de autenticação.

Para evitar erros de concorrência em cenários que existam vários utilizadores, foram feitos *locks* explícitos, usando o comando “SELECT FOR UPDATE”, em funções que achámos pertinente. Por exemplo, para fazer uma licitação, o programa começa por obter da base de dados o leilão em que o utilizador pretende efetuar uma licitação, para se poder verificar que a é superior à maior atual do leilão. Com o comando acima referido é possível dar *lock* ao leilão fazendo com que outra transação que queira obter esse mesmo leilão para efetuar uma licitação, tenha que aguardar que o lock seja removido, evitando assim que a maior licitação do leilão seja alterada depois de uma outra transação a ler, ou que uma das licitações leve overwrite, mesmo sendo superior (caso obtenham o leilão simultaneamente).

Também foi tido o cuidado de dar *commit* no final correto de uma transação e *rollback* em caso de erro, de modo a manter os dados corretos na base de dados, e de libertar qualquer lock feito, explícito ou implícito.

Finalmente, aplicámos várias maneiras de controlar o fluxo da aplicação. Nos *endpoints* que dizem respeito a um leilão (Editar propriedades de um leilão, Término do leilão na data, hora e minuto marcados) é necessário introduzir um authToken correspondente ao dono deste leilão. Nas funcionalidades referentes aos administradores, só esse tipo de utilizador é que pode executar essas funcionalidades. Por outro lado, não se pode licitar em leilões onde se é dono, ou fazer licitações abaixo do valor mínimo (ou que não supere o maior valor atual). Ora este controlo do código está associado aos erros que estão descritos no ponto 3 onde o grupo cobriu todas as possibilidades para garantir o bom funcionamento da aplicação.

## Triggers Implementados

Os *triggers* demonstraram ser uma funcionalidade avançada que permitiram computar automaticamente operações complexas. No projeto de Base de Dados desenvolvemos 3 triggers:

* **triggerVersion** é executado sempre que as propriedades textuais de um leilão são atualizadas. Este guarda na tabela “versao” a edição antiga do leilão, antes de ser alterado.
* **tLeilaoCancelado** é concretizado sempre que um administrador cancela um leilão. Este envia uma notificação a informar o sucedido a todos os que licitaram no leilão, informando que o evento foi cancelado e não haverá nenhum vencedor.
* **tLicitacaoUltrapassada** é efetivado assim que algum comprador licitar por cima de outro licitador. Este notifica todos os interessados que a sua licitação foi ultrapassada por outro membro e por que valor.

# Plano de Desenvolvimento

|  |  |  |
| --- | --- | --- |
| **Nome da Tarefa** | **Pessoa Responsável** | **Duração Estimada** |
| Criação do ER | Todos | 2-3 horas |
| Registo de Utilizadores | Pedro Marques | 30 minutos |
| Autenticação de Utilizadores | Diogo Filipe | 1:15 horas |
| Criação de um novo leilão | José Gomes | 40 minutos |
| Listar todos os leilões existentes | Pedro Marques | 30 minutos |
| Pesquisar leilões existentes | Diogo Filipe | 40 minutos |
| Consultar detalhes de um leilão | José Gomes | 35 minutos |
| Listar leilões em que o utilizador tenha atividade | Pedro Marques | 1 hora |
| Efetuar uma licitação | Diogo Filipe | 1 hora |
| Editar as propriedades textuais de um leilão | José Gomes | 1 hora |
| Escrever no mural de um leilão | Pedro Marques | 1 hora |
| Entrega de notificações a utilizadores | Diogo Filipe | 30 minutos |
| Notificação de licitação ultrapassada | José Gomes/Pedro Marques | 1:30 horas |
| Término do leilão no tempo marcado | Pedro Marques | 2 horas |
| Cancelamento de um leilão | Diogo Filipe | 40 minutos |
| Banir um utilizador | José Gomes | 3 horas |
| Estatísticas da aplicação | Pedro Marques | 1:30 horas |
| Testagem da aplicação e das diversas funcionalidades | Todos | 8 horas |
| Realização do documento final | Todos | 4 horas |