FCTUC FACULDADE DE CIÊNCIAS
E TECNOLOGIA
UNIVERSIDADE DE COIMBRA

# Leilões online

Base de Dados

Licenciatura em Engenharia Informática

2020/2021

Duarte Emanuel Ramos Meneses – 2019216949 – duartemeneses@student.dei.uc.pt

Inês Martins Marçal – 2019215917 – inesmarcal@student.dei.uc.pt

Patrícia Beatriz Silva Costa – 2019213995 – patriciacosta@student.dei.uc.pt

# Índice

# Introdução

Este projeto tinha como intuito que os alunos que o realizassem tivessem um contacto mais próximo com a realidade de trabalhar na área das bases de dados. Deste modo, foi pedido aos alunos que desenvolvessem um sistema de gestão de base de dados que suportasse um sistema típico de leilão.

Para permitir ao utilizador aceder ao sistema, foi requerido que este fosse disponibilizado através de uma API REST. Isto permite ao utilizador aceder ao sistema através de pedidos HTTP. Cada leilão é iniciado por um utilizador que define um artigo, o preço mínimo que está disposto a receber e o momento em que o leilão vai terminar. Os compradores fazem licitações que vão sucessivamente aumentando o preço até ao término do leilão (ganha o utilizador que licitar o valor mais alto). Existem ainda administradores que podem cancelar leilões, banir utilizadores e obter estatísticas da aplicação.

A arquitetura da "comunicação" "utilizador-base de dados" segue o funcionamento da figura seguinte:

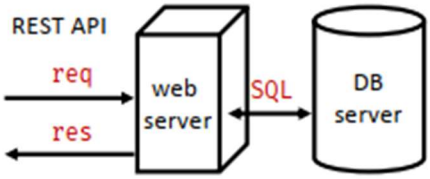


*Figura 1 – Arquitetura da "comunicação" " utilizador-base de dados"*

Pela figura acima percebemos que o utilizador comunica com o servidor web através de pedidos e respostas (*request*/*response*) e este, por sua vez, comunica com a base de dados através de uma interface SQL.

Tendo em conta o que era pedido, optamos por utilizar Python como linguagem de programação da API REST (logo utilizamos a interface Psycopg para efetuar a comunicação entre servidor e base de dados). Utilizamos o Postman como cliente REST e a plataforma pgAdmin para nos auxiliar na visualização das operações que íamos realizando.

# Manual de utilizador

Para interagir com a nossa aplicação basta ter instalado um cliente REST (por exemplo, o Postman) que vai conseguir testar a API REST por nós desenvolvida. As figuras 4, 5 e 6 são dessa plataforma.

Para ligar a API REST, o utilizador deve abrir na sua linha de comandos a diretoria onde o script da aplicação se encontra. Para o fazer, supondo que o script se encontra na pasta exemplo do Ambiente de Trabalho do utilizador, basta (em *Windows*):

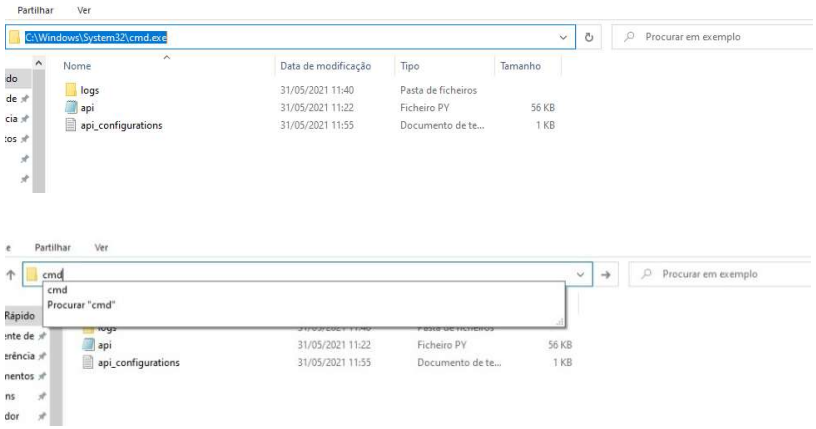


*Figura 2 – Forma de abrir a diretoria no terminal (em Windows)*

Clicando no *enter*, a linha de comandos na diretoria pretendida irá abrir. Depois disso, basta correr o código através do comando "*python3 api.py*" e a interface deverá apresentar o seguinte:

```
C:\Users\Utilizador\Desktop\exemplo>python3 api.py
13:37:59 [INFO]:
----------------------------------------------------------------
API v1.0 online: http://localhost:8080/dbproj/


 * Serving Flask app 'api' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: on
13:38:04 [INFO]:
----------------------------------------------------------------
API v1.0 online: http://localhost:8080/dbproj/
```

*Figura 3 – Forma de inicializar a API REST e layout inicial*

Tal como a figura 2 mostra, é importante que na mesma diretoria do script exista uma pasta intitulada *logs* e o script de configurações *api_configuration*s. Sem estes ficheiros a API nunca iniciará.

A nossa aplicação permite realizar várias ações. Para cada uma é necessário comunicar com a API REST através de pedidos HTTP. No Postman, é necessário abrir uma nova *request*, inserir o endereço para o qual a ação está pré-definida e, caso seja necessário, inserir os dados no *form-data* do *Body* (se esses dados não forem incluídos no endereço). Por exemplo, para adicionar um utilizador, supondo que se quer adicionar o utilizador com o *username* chico, password fininho e email chiquinho@gmail.com, basta clicar onde a seta aponta:

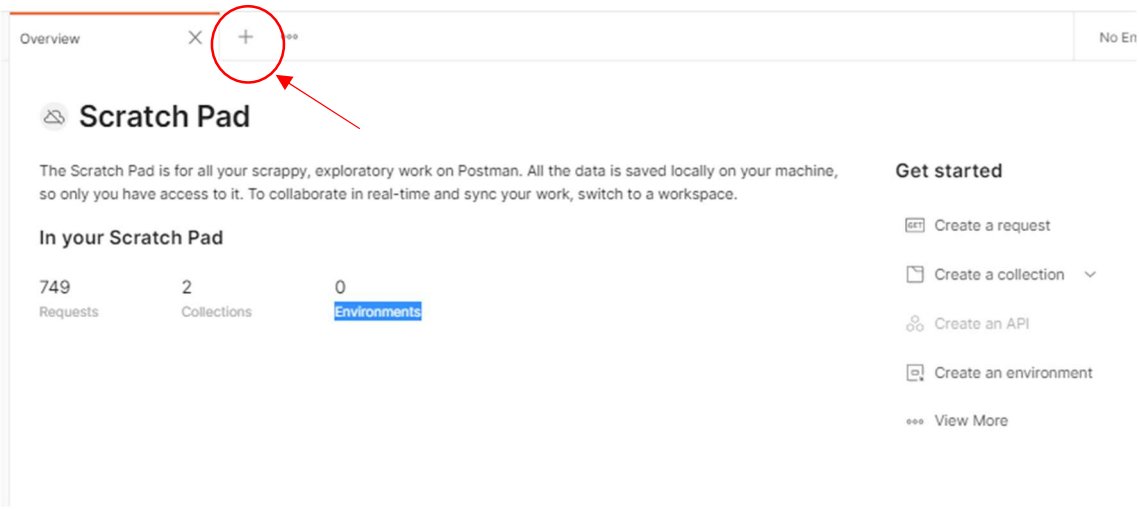


*Figura 4 – Como criar uma request*

Fazendo isso, já se tem tudo para fazer um pedido à API REST.

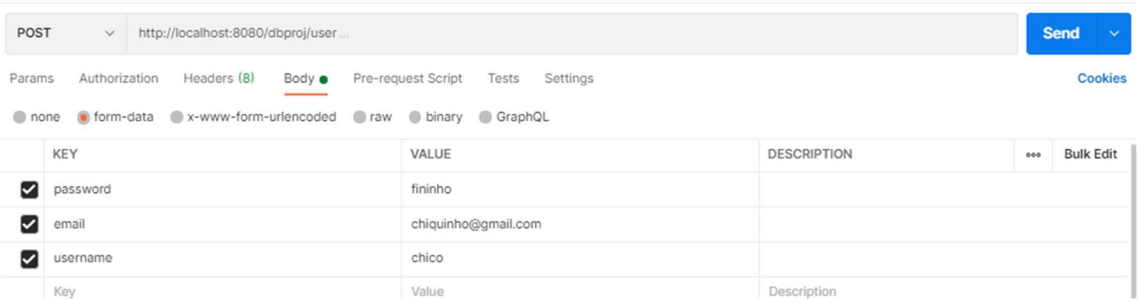


*Figura 5 – Exemplo de como inserir os dados para fazer um pedido à API REST (novo utilizador neste caso)*

Preenchendo os dados como está exemplificado acima (de acordo com o exemplo referido anteriormente), como o endereço HTTP onde a ação de adicionar um utilizador se encontra é http://localhost:8080/dbproj/user e esta está configurada para ser do tipo POST, basta clicar no *Send*. Isto irá fazer um pedido (*request*) à API REST que posteriormente, se tudo correr bem, irá introduzir na base de dados este utilizador e responder a mensagem correspondente ao sucesso.

Tal como referido acima, é possível que os dados em vez de serem introduzidos no *Body* sejam inseridos no endereço em algumas ações. Exemplo disso é a ação de listar os detalhes de um leilão em que o ID desse leilão é inserido no endereço HTTP:

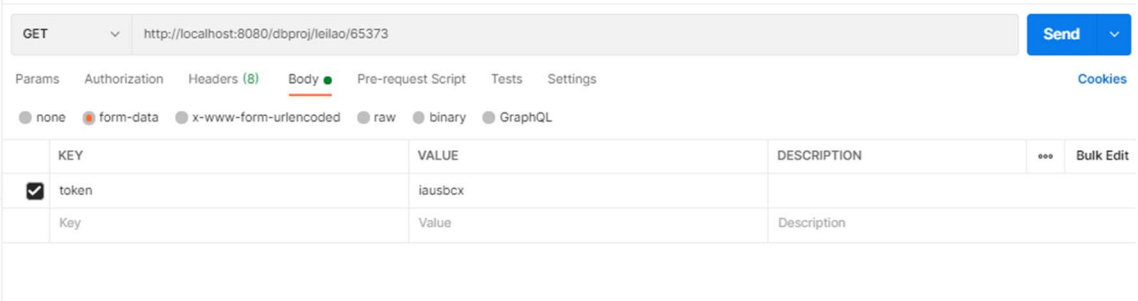


*Figura 6 – Exemplo de como inserir os dados caso a informação tenha de ir no endereço (consultar detalhes de um leilão neste caso)*

Na figura anterior, o utilizador com o *token "iausbcx"* está a pedir à aplicação que devolva os detalhes do leilão com o ID 65373.

Para estar conectado à aplicação, cada utilizador deve estar registado no sistema. Caso esteja, deve-se autenticar e é-lhe atribuído um *token* de autenticação. Esse *token* é válido durante 20 minutos. Após esse tempo, o *token* fica inválido e o utilizador deve-se autenticar novamente.

De um modo geral, as ações disponíveis e a forma de as operar são as seguintes:

- **Registo de um utilizador –** [POST] http://localhost:8080/dbproj/user - inserindo *username*, *password* e *email* no *Body* (sendo as *keys* exatamente as palavras usadas nesta descrição);
- **Autenticar um utilizador –** [PUT] http://localhost:8080/dbproj/user - inserindo *username* e *password* no *Body* (sendo as *keys* exatamente as palavras usadas nesta descrição);
- **Criar um leilão –** [POST] http://localhost:8080/dbproj/leilao - inserindo o *token* de autenticação (devolvido aquando da autenticação se for bem-sucedida – para se saber que utilizador o está a fazer), código de artigo, título do leilão, data e hora de término, possíveis detalhes adicionais (não obrigatório), preço mínimo de licitação e descrição no *Body* (utilizando para isso as *keys*: token, código_artigo, titulo, data_hora_fim, detalhes_adicionais, preco_minimo, descrição_atual);
- **Listar todos os leilões existentes –** [GET] http://localhost:8080/dbproj/leiloes - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token);
- **Pesquisar leilões existentes –** [GET] http://localhost:8080/dbproj/leiloes/<keyword> - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token) e substituindo <keyword> no endereço pelo ID do leilão ou pela descrição do mesmo que se pretende obter os detalhes;
- **Consultar detalhes de um leilão –** [GET] http://localhost:8080/dbproj/leilao/<leilaoID> - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token) e substituindo <leilaoID> no endereço pelo ID do leilão que se pretende obter os detalhes
- **Listar todos os leilões em que um utilizador tenha alguma atividade –** [GET] http://localhost:8080/dbproj/leiloes_user- inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token)*;*
- **Efetuar licitação num leilão –** [POST] http://localhost:8080/dbproj/licitar/<leilaoID>/<valor> - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token) e substituindo <leilaoID> no endereço pelo ID do leilão que se pretende licitar bem como substituindo <valor> pelo valor da licitação;
- **Editar propriedades de um leilão –** [PUT] http://localhost:8080/dbproj/leilao/<leilaoID> - inserindo o *token* de autenticação, o parâmetro que se quer editar bem como a nova descrição desse parâmetro no *Body* (utilizando para isso as *keys:* token, parâmetro, nova_desc) e substituindo <leilaoID> no endereço pelo ID do leilão que se pretende editar. Apenas são editáveis o título, descrição e detalhes adicionais (titulo, descrição_atual, detalhes_adicionais, respetivamente, como parâmetro);
- **Escrever mensagens no mural de um leilão –** [POST] http://localhost:8080/dbproj/mensagem_leilao/<leilaoID> - inserindo o *token* de autenticação bem como a mensagem que se deseja publicar no *Body* (utilizando para

isso as *keys:* token, mensagem) e substituindo <leilaoID> no endereço pelo ID do leilão que se pretende escrever no mural;

- **Receber mensagens –** [GET] http://localhost:8080/dbproj/recebermens - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token).

Estas são as ações que qualquer utilizador não banido pode realizar. No entanto, se o utilizador for administrador, pode ainda:

- **Cancelar leilão –** [PUT] http://localhost:8080/dbproj/cancelarleilao/<leilaoID> - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token) e substituindo <leilaoID> no endereço pelo ID do leilão que se pretende cancelar;
- **Banir utilizador –** [PUT] http://localhost:8080/dbproj/banuser/<username> - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token) e substituindo <userID> no endereço pelo username do utilizador que se pretende banir;
- **Obter estatísticas da atividade da aplicação –** [GET] http://localhost:8080/dbproj/estatisticasapp - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token).

Existe ainda outra ação que qualquer pessoa pode realizar. Esta é a de verificar se há leilões que já podem ser encerrados. Se essa condição se verificar, o leilão encerra. Nem é necessário ser-se utilizador para realizar esta ação.

- **Encerrar leilões caso já tenham passado a data de término –** [PUT] http://localhost:8080/dbproj/endleilao

Embora não fosse explicitamente pedido, como no enunciado constava que o histórico de um leilão podia ser acedido a qualquer altura, decidimos criar um endpoint onde o histórico de detalhes de um leilão seja consultado.

- **Obter histórico de leilão –** [GET] http://localhost:8080/dbproj/historico/<leilaoID> - inserindo o *token* de autenticação no *Body* (utilizando para isso a *key* token) e substituindo <leilaoID> no endereço pelo ID do leilão que se pretende obter o histórico.

# Manual de instalação

Para utilizar a nossa aplicação é necessário instalar alguns packages. Tendo sido a API REST desenvolvida em Python, é necessário ter o Python3 instalado. Além disso, é preciso instalar os packages correspondentes aos módulos que importamos no código. Alguns já vêm com o Python3, porém, se não vierem, é necessário instalar. Os comandos para tal (em Windows) são os seguintes:

- flask - pip install flask
- logging - pip install logging
- psycopg2 - pip install psycopg2
- time - pip install time
- datetime – pip install datetime

Utilizamos ainda módulos específicos do flask como Flask, jsonify e request. Em princípio, ao instalar o flask estes vêm incorporados. Porém, se não vierem, é necessário instalá-los através de:

- Flask - pip install Flask
- jsonify - pip install jsonify
- request - pip install request

Tendo isto tudo instalado, a API REST está pronta a ser utilizada.

Quanto à base de dados, é necessário criar uma com o nome *projeto* num servidor chamado *localhost*. Esse servidor deve estar conectado ao porto 5432, sendo o seu utilizador intitulado de *postgres* com a password *postgres*. No desenvolvimento da nossa aplicação, recorremos ao *pgAdmin* para nos auxiliar. Deste modo, este processo resumiu-se ao seguinte:

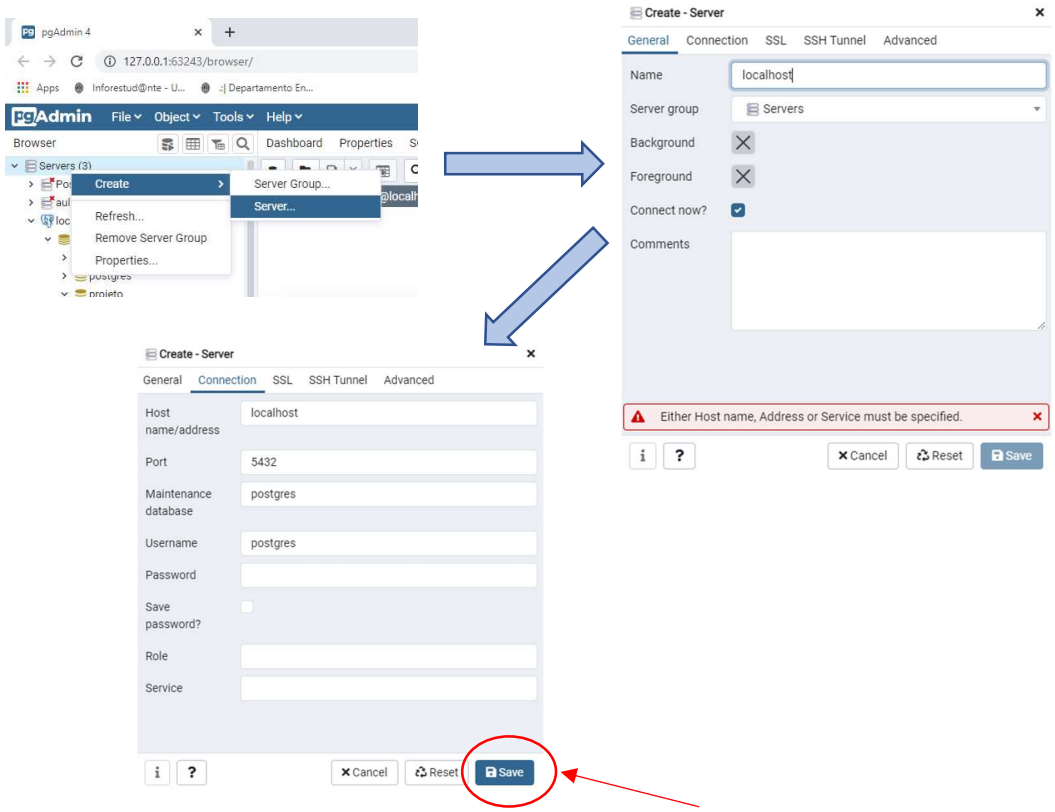


*Figura 7 – Processo de criação do servidor no pgAdmin*

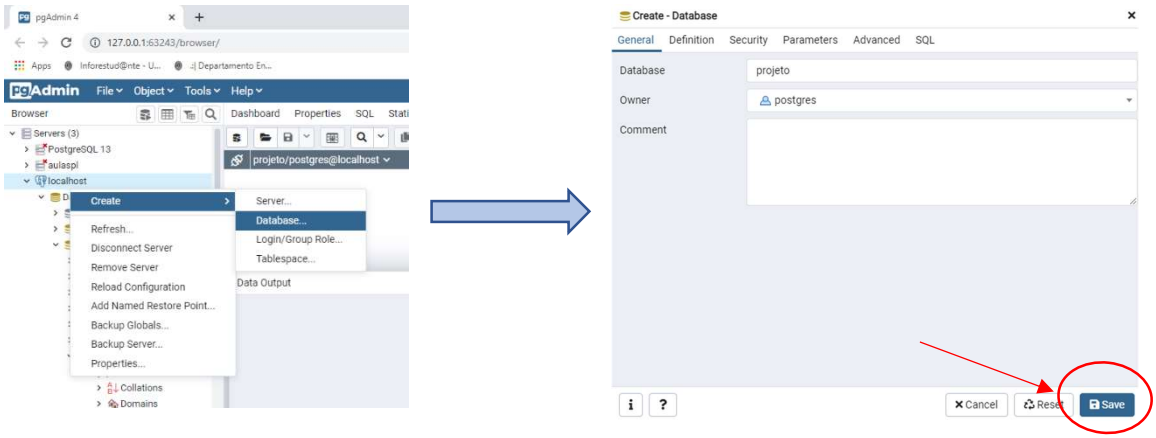


*Figura 8 – Processo de criação da base de dados no pgAdmin*

Qualquer alteração efetuada a estes parâmetros deve ser igualmente alterada no ficheiro de configurações da API REST (*api_configurations.txt*).

Tendo a base de dados operacional, basta criar as tabelas. Para isso, é necessário correr o script da sua criação (*cria_tabelas.txt*). Para tal, bastou criar uma *Query Tool* na base de dados em questão, colar o código que consta no *script cria_tabelas.txt* e executar da seguinte maneira:

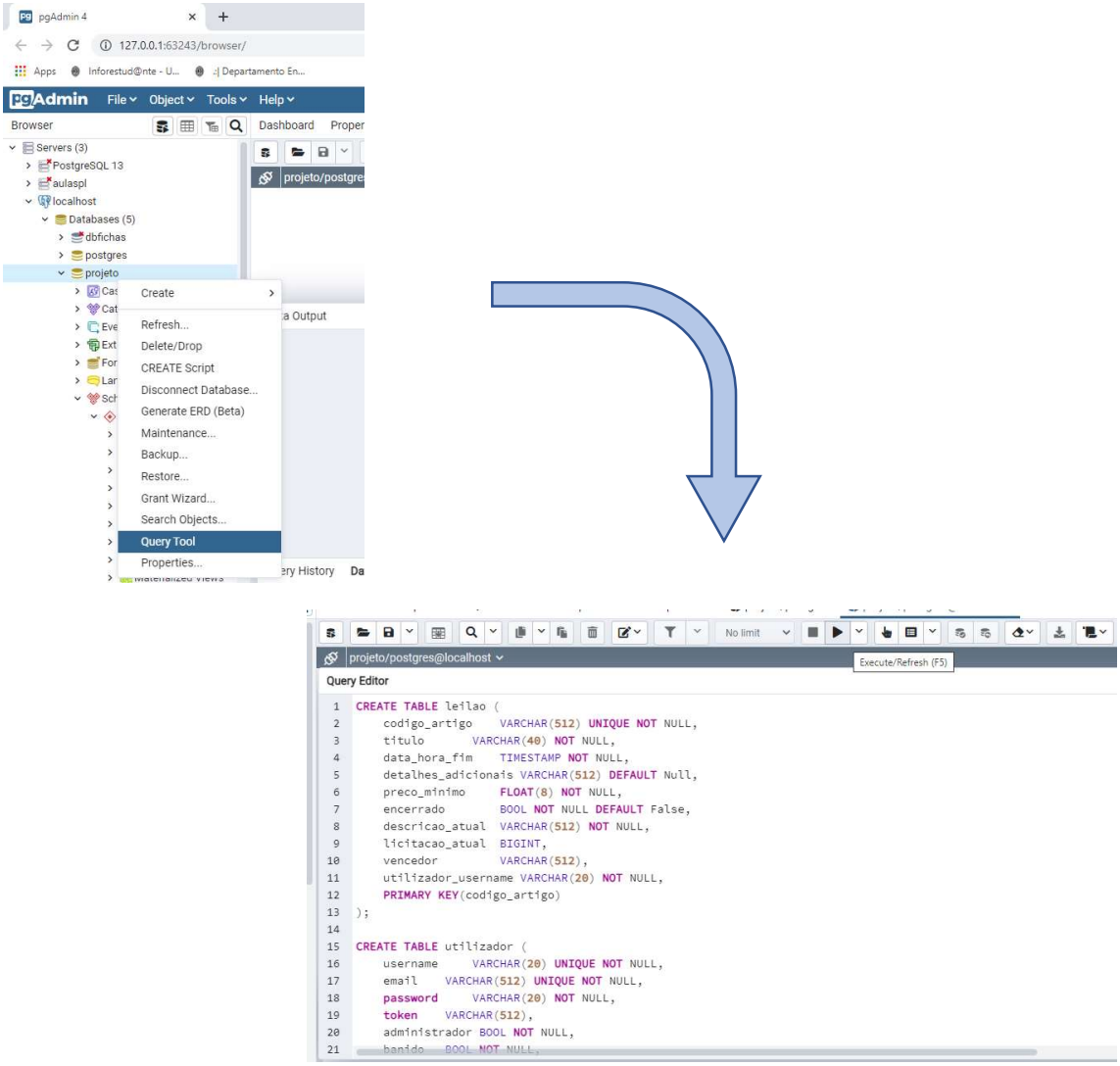


*Figura 9 – Processo de criação das tabelas no pgAdmin*

## Diagrama ER e modelos de dados relacionais

- Modelo conceptual:

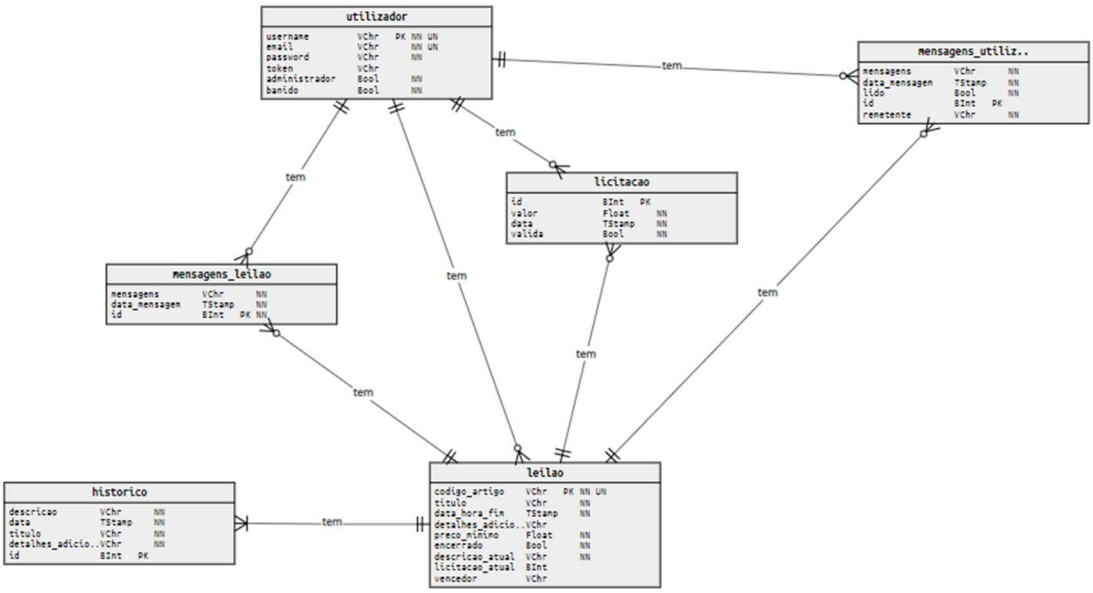


*Figura 10 – Modelo conceptual do diagrama ER*

- Modelo físico:

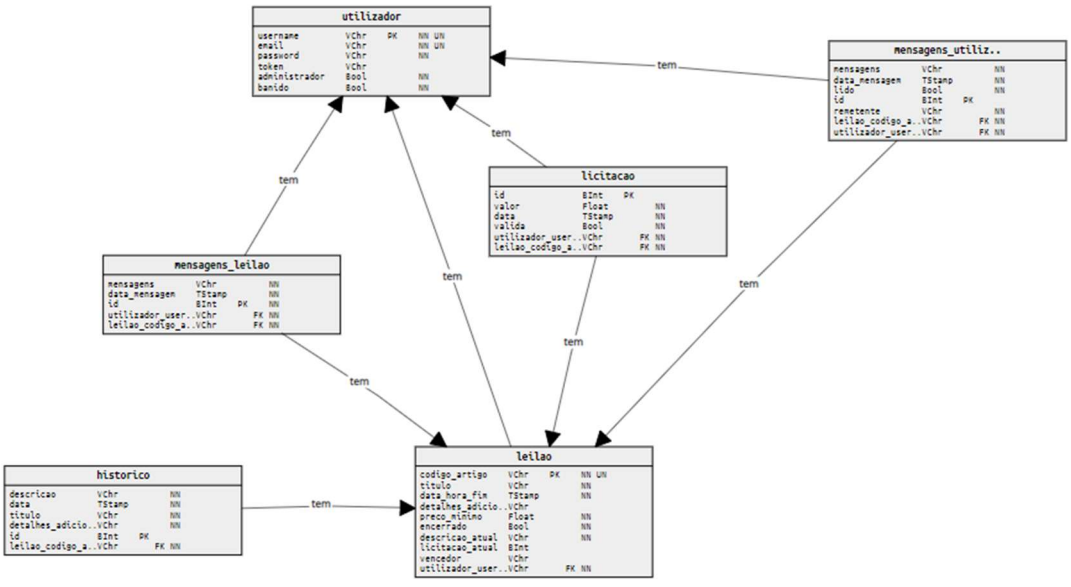


*Figura 11 – Modelo físico do diagrama ER*

## Divisão de tarefas

| Tarefa | Responsável | Data de início | Data de fim |
|---|---|---|---|
| **Arquitetura da base de dados** | Todos | 24/03/2021 | 13/04/2021 |
| **Instalação dos packages e programas necessários** | Todos | 05/05/2021 | 08/05/2021 |
| **Criação do servidor e da base de dados** | Todos | 06/05/2021 | 06/05/2021 |
| **Desenvolvimento da API REST** | Todos | 23/05/2021 | 31/05/2021 |
| **Relatório** | Todos | 30/05/2021 | 31/05/2021 |

## Esforço horário

Para a elaboração deste projeto, o número de horas despendidas por cada elemento do grupo foram as seguintes:

- Duarte Meneses – 51 horas
- Inês Marçal – 51 horas
- Patrícia Costa - 51 horas

# Manual de programador

No processo de registo de um utilizador decidimos que, para evitar que a base de dados devolvesse um erro caso se inserisse um utilizador com o mesmo *username* ou *email* de outro já existente, devíamos antes de realizar a transação, verificar se já existe algum utilizador que verifica a condição descrita. Se sim, o utilizador que estava a tentar entrar no sistema é recusado aparecendo a mensagem de erro correspondente. O sistema não aceita mais do que 1 utilizador com o mesmo username ou email pois o username é chave primária da tabela utilizador e porque definimos o email como único, isto é, apenas pode existir 1 utilizador com o mesmo email.

Já no processo de autenticação de utilizadores, optamos por gerar um token de autenticação de acordo com o username do utilizador em causa. Sempre que o utilizador se autentica, o prazo da sessão aumenta. Após a autenticação, cada utilizador dispõe de 20 minutos até necessitar de nova autenticação. Caso se tente autenticar com um username ou uma password não existentes no sistema, a aplicação devolve o erro correspondente.

À semelhança do que fazemos ao registar um utilizador, ao criar um leilão verificamos se já existe algum leilão criado com o mesmo código de artigo. Fazemos isto para evitar que a base de dados devolva erro por não poder existir mais que um leilão com o mesmo código de artigo (chave primária da tabela leilão). Nesta ação, verificamos ainda se o preço mínimo que o vendedor está disposto a receber é válido (ou seja, se é superior a 0 euros). Como é pedido que se guarde o histórico dos detalhes de cada leilão, quando criamos um automaticamente adicionamos à tabela histórico os detalhes dele.

Nas ações relativas a listar os detalhes de leilões, o mecanismo é quase sempre o mesmo. As únicas coisas que mudam são o critério de pesquisa e os detalhes a apresentar. Na ação de listar todos os leilões existentes, percorremos todos e selecionamos apenas os que ainda estão ativos (ou seja, na nossa tabela leilao os que têm o valor False no atributo encerrado). Já na ação de pesquisar os leilões existentes através do ID do leilão (codigo_artigo) ou da descrição (descricao_atual), selecionamos os leilões que verificam a condição anterior e que ainda não encerraram. Nestas duas ações, antes de efetuar a seleção é chamada a função que vai encerrar leilões caso estes já tenham ultrapassado a sua data de término. Isto leva a que nunca apareçam leilões que já deviam ter encerrado como ativos.

No que toca a listar os detalhes de um leilão, decidimos que a melhor maneira a fazer seria realizar três select (1 para ver se o leilão existe, outro para selecionar as suas mensagens e outro para selecionar as suas licitações). Já para listar os leilões em que um utilizador tem atividade, selecionamos aqueles em que ele interveio como criador (ou seja, está como atributo na coluna do utilizador_username da tabela leilao) e aqueles em que ele licitou (isto é, está como atributo na coluna utilizador_username na tabela licitacao).

Passemos agora para a ação de licitar. Quando um utilizador tenta licitar um leilão, a nossa aplicação vai verificar se o leilão ainda aceita licitações (se o atributo encerrado na tabela leilao está a False) e se o valor que se pretende licitar é válido. O valor é válido se for superior ao valor pedido pelo vendedor (se ainda não existirem licitações para esse leilão) ou se for superior ao da última licitação. Se for válido, é acrescentada essa licitação à tabela licitação e o valor da licitacao_atual é atualizado na tabela leilão. Em seguida, são enviadas mensagens a quem licitou a informar que alguém subiu a parada no leilão. É importante realçar que o criador não pode licitar o próprio leilão e que um utilizador pode licitar cobrindo uma licitação feita por

si anteriormente. Antes de se licitar é chamada a função que vai encerrar leilões caso estes já tenham ultrapassado a sua data de término. Isto leva a que não se consiga licitar um leilão após a data do seu término.

Na edição de propriedades de leilão, a nossa aplicação pede 2 valores. Um deles diz respeito ao parâmetro que se quer alterar (sendo a designação exatamente igual à que consta na tabela leilao) e o outro diz respeito à nova descrição que constará nesse atributo. Como era pedido que apenas fosse possível editar as propriedades textuais, apenas permitimos 3 tipos de parâmetros: titulo, descricao_atual e detalhes_adicionais. Ao alterar uma descrição textual, guardamos os detalhes do leilão no histórico tal como fizemos aquando da sua criação. É importante realçar que apenas o criador do leilão o pode alterar.

Para entregar de forma imediata as mensagens a um utilizador, decidimos colocar um atributo booleano na tabela mensagens_utilizador chamado lido. Criamos um endpoint para que cada utilizador consiga receber as suas mensagens. Funciona do seguinte modo: se um utilizador se conectar a esse endpoint, a aplicação irá selecionar as mensagens cujo destinatário (utilizador_username) é o utilizador em causa e que o atributo lido esteja a False. A partir do momento que essas mensagens são selecionadas, são enviadas como resposta ao utilizador e o booleano lido passa a True. Deste modo, um utilizador nunca vê a mesma mensagem mais do que uma vez.

Para o término dos leilões decidimos criar um endpoint em que qualquer pessoa pode aceder. Ao fazê-lo a aplicação vai verificar se existem leilões cuja data de encerramento já passou. Se isso acontecer, o atributo encerrado do leilão passa a True e é determinado o vencedor (a quem pertence a licitação atual). Esta determinação de vencedor é feita através da comparação do valor da licitação atual com as licitações na tabela correspondente cujos ID de leilão e valor são os mesmos da licitação atual na tabela leilao. Ao encontrar uma licitação que verifique esta condição e que esteja válida, obtém-se quem a fez e consequentemente quem é o vencedor. Por outro lado, se o leilão encerrar sem licitação atual não existe vencedor.

Caso seja necessário, um administrador pode cancelar um leilão. Se isso acontecer, o atributo na tabela leilao encerrado passa a True mas não se determina o vencedor. São ainda enviadas mensagens a informar do sucedido ao criador, a quem licitou e a quem escreveu no mural. Caso alguém tenha realizado mais que uma ação das mencionadas anteriormente, irá receber apenas uma mensagem.

Um administrador pode ainda banir um utilizador (desde que não seja ele próprio). Se isso acontecer, o atributo banido na tabela do utilizador para esse utilizador em questão passa a True. Em seguida, são selecionados os leilões em que ele era o criador para estes serem cancelados, ou seja, o atributo encerrado passar a True sem que seja determinado o vencedor. Logicamente que os leilões que já estavam encerrados não irão sofrer alterações, uma vez que não fazia sentido tirar a vitória a um utilizador depois de este ganhar. Em seguida, também as licitações realizadas pelo utilizador banido são invalidadas, ou seja, o atributo valido na tabela licitacao passa a False. Isto tem impacto nas licitações deste leilão. Se esta for a licitacao_atual da tabela leilao na altura do banimento, a licitacao_atual passa a ser a maior das restantes (se existirem). Se não existirem mais licitações, o leilão deixa de ter licitacao_atual. Porém, se existirem licitações mais elevadas, estas são invalidadas à exceção da mais elevada que se mantem como licitacao_atual passando a ter o valor da mais alta do utilizador banido para o leilão em questão.

Para finalizar as tarefas específicas dos administradores, estes podem obter as estatísticas da atividade da aplicação (top10 de utilizadores que mais leilões criaram, top10 de utilizadores que mais leilões venceram e número total de leilões criados nos últimos 10 dias). Para o top10 de criadores decidimos que a melhor maneira seria selecionar da tabela leilao o nome de utilizador e um count de códigos de artigo agrupados por nome de utilizador limitando o resultado a 10 respostas. De igual modo, para o top10 de vencedores utilizamos o mesmo raciocínio só que em vez de selecionar e agrupar pelo nome de utilizador decidimos fazê-lo pelo atributo vencedor. Neste atributo consta o nome de utilizador do vencedor de determinado leilão. No entanto, o resultado do código descrito pode conter valores NULL uma vez que o vencedor inicialmente não está determinado nem é garantido que haja vencedor. Deste modo, no nosso código em Python tivemos de implementar uma condição que apenas irá considerar os valores diferentes de NULL (None neste caso por ser Python).

Já para obter o número total de leilões criados nos últimos 10 dias decidimos que a melhor maneira de o fazer era comparar a data atual de quem obtém as estatísticas com a data de criação do leilão (presente na tabela historico com o id mais baixo para o mesmo ID de leilão). Deste modo, sempre que a data proveniente dessa tabela e a data atual de quem obtém estatísticas tenham uma diferença não superior a 10 dias, um contador incrementa.

É importante realçar que os administradores da aplicação estão já pré-definidos. Não é dada a possibilidade de criar mais. Todos os utilizadores adicionados ao sistema entram com o atributo administrador a False e com o banido a False. De igual modo, quando um leilão é criado o seu atributo encerrado é False (se a data de término já tiver passado entra com o atributo a False mas depois mal se tente licitar ou tente fazer algo que precise que o leilão esteja ativo, através da nossa implementação de código, o atributo passa automaticamente a True pelo que não deixa fazer nada). Também uma licitação entra sempre com o seu atributo valida a True.

Para evitar falhas de informação não permitimos que sejam criados utilizadores e leilões em que as suas informações sejam strings vazias (à exceção dos detalhes adicionais do leilão que não é obrigatório ter). Já para evitar que utilizadores façam coisas que não estão autorizados, em todas as ações que é preciso estar autenticado verificamos se o utilizador não está banido e nas ações específicas de administrador verificamos sempre se o utilizador é administrador.

É também relevante dizer que consideramos que o ID do leilão corresponde ao código do artigo a ser leiloado (codigo_artigo na tabela leilao) e que o userID de um utilizador corresponde ao seu username (nome de utilizador) na tabela de utilizador.

Para evitar conflitos de concorrência decidimos que bloquear as tabelas utilizadas em cada transação onde constam operações de INSERT ou UPDATE em modo exclusivo seria a melhor opção. Assim, ninguém interfere com as alterações de dados nas tabelas enquanto uma transação não se realizar (evitando que dados se corrompam ou se percam).

Embora não fosse explicitamente pedido, como no enunciado constava que o histórico podia ser acedido a qualquer altura, decidimos criar também um endpoint onde o histórico de detalhes de um leilão pudesse ser consultado. Essa função limita-se a percorrer a tabela historico e selecionar a data, o título, a descrição e os detalhes adicionais das linhas cujo ID de leilão seja o mesmo que se quer ver o histórico.

Para não ter os dados do servidor e da base de dados à vista no código, decidimos colocá-los num ficheiro de texto e acedê-lo sempre que pretendemos aceder à base de dados.

# Conclusão

Este projeto tinha como objetivo permitir que quem o elaborasse ficasse mais ciente da realidade que é trabalhar com bases de dados. Neste ponto de vista, pensamos que fomos bem-sucedidos visto que aprendemos a efetuar certas ações que até então não tínhamos tido oportunidade. Com este trabalho conseguimos colocar em prática os conceitos abordados nas aulas teóricas, teórico práticas e laboratoriais de Base de Dados e aplicar outros tantos.

Sendo nós estudantes de Engenharia Informática e sendo este trabalho referente à cadeira de Base de Dados, uma área tão abrangente e tão presente na sociedade atual, consideramos que este projeto nos foi útil. Consideramos isso não só por nos ter ajudado a entender os conceitos teóricos (uma vez que os tivemos de colocar em prática) mas também por nos ter dado bases para o nosso futuro como eventuais profissionais na área.

Em suma, consideramos que realizamos tudo o que era pedido no enunciado e que conseguimos ser bem-sucedidos no que toca aos objetivos gerais deste projeto.

# Bibliografia


- Materiais disponibilizados pelos docentes da cadeira [31 de maio de 2021]