Base de Dados

Projeto 2020/2021

Trabalho realizado por:

- João Carlos Borges Silva nº2019216753 - j0a0carll21@gmail.com
- Sofia Santos Neves nº2019220082 - sofianeves@student.dei.uc.pt
- Tatiana Silva Almeida nº2019219581 - tatianasilvaalmeida24@gmail.com

Professores orientadores:

- Marco Paulo Amorim Vieira - mvieira@dei.uc.pt
- Nuno Manuel dos Santos Antunes - nmsa@dei.uc.pt

31 de Maio de 2021

Índice

Introdução

Este projeto foi criado no âmbito da disciplina de Base de Dados, com o objetivo de desenvolver uma API Rest que fará a gestão de leilões e da sua respectiva base de dados.

Um leilão é iniciado por um utilizador que define um artigo que pretende leiloar, indica o preço mínimo que está disposto a receber, e decide o momento em que o leilão vai terminar. Indica também a condição do produto (novo, usado, não especificado) e adiciona uma breve descrição sobre o mesmo. Todos os leilões têm um mural, onde os utilizadores podem deixar a sua mensagem/comentário.

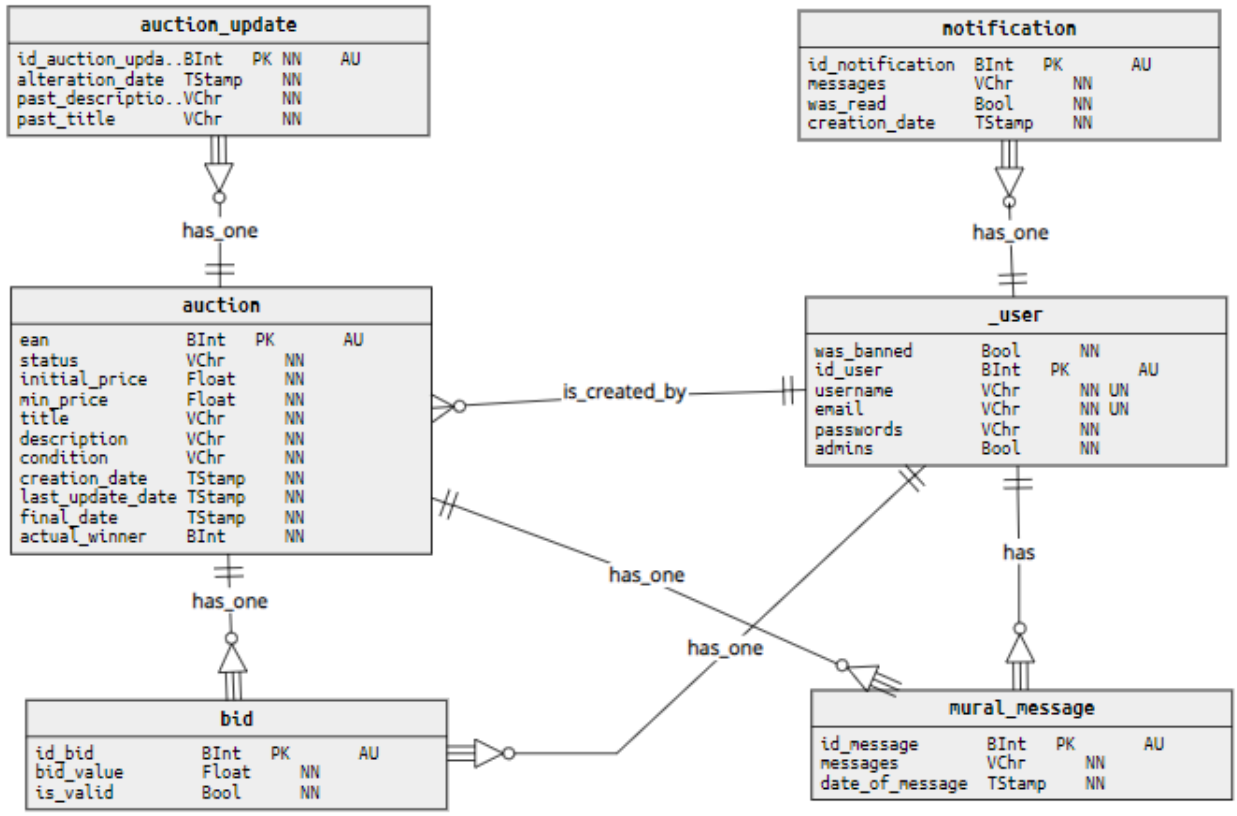
Os utilizadores, que pretendam adquirir o artigo, fazem licitações que vão sucessivamente aumentando o preço, podendo fazê-lo até à data de término do leilão. Ganha o utilizador que licitar o valor mais alto.

Todos os utilizadores que façam licitações em um leilão ou que escrevam uma mensagem no mural do mesmo ficam, automaticamente, associados a ele e sujeitos à receção de notificações que envolvam esse mesmo leilão.

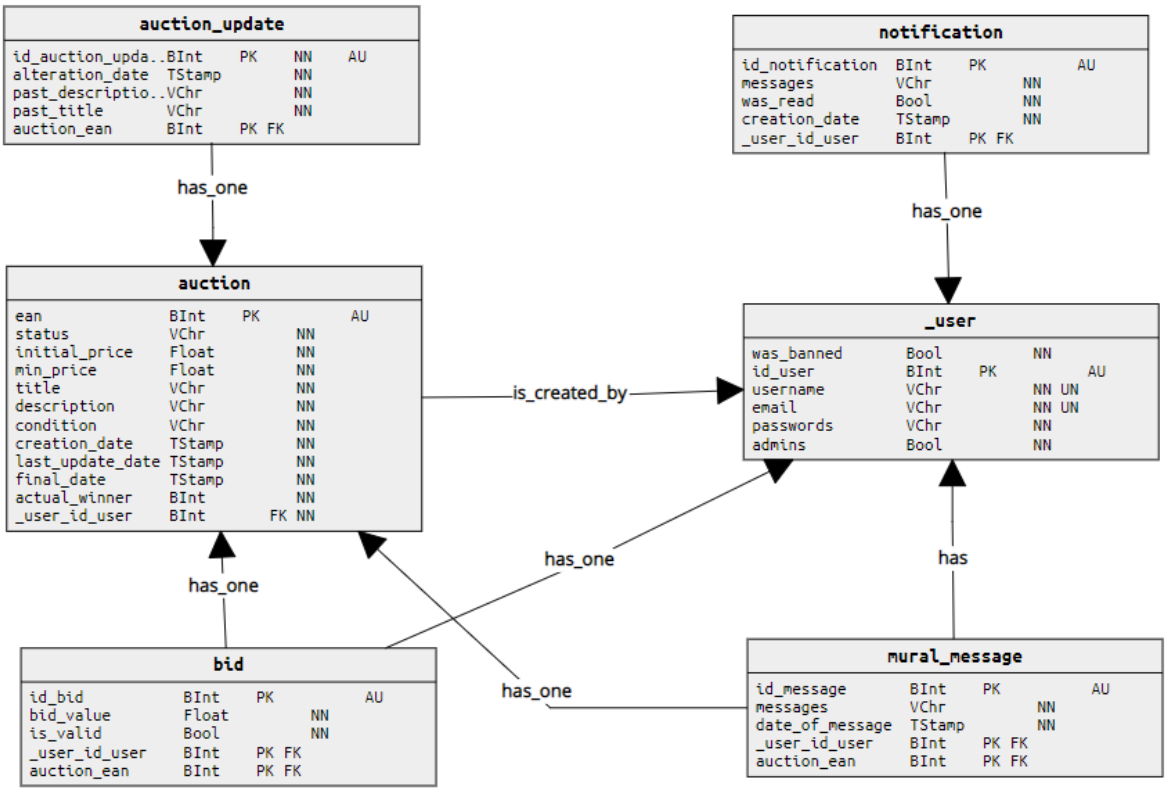
As notificações são enviadas caso haja uma nova mensagem no mural , quando um utilizador associado é banido, ou quando o leilão é cancelado. Sempre que há uma licitação é também enviada uma notificação para o utilizador que se encontrava posteriormente com a licitação mais elevada desse leilão.

Existe também um tipo diferente de utilizadores, administradores. Estes têm permissões diferentes que lhes permitem cancelar um leilão ou banir um utilizador.

Modelo Conceptual ER



Modelo Físico ER



Manual de instalação

Para a instalação do software criado basta seguir os seguintes passos:

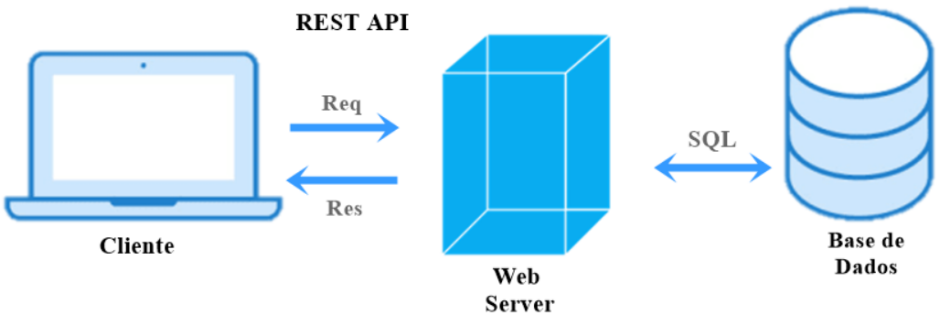
1. Fazer download do ficheiro **Auctions.py** a partir do InforEstudante.
2. Certificar-se que tem instalada a versão mais recente de Python3, nomeadamente, a partir da versão Python 3.9.5 para cima.
3. Certificar-se que tem instalada a versão mais recente do pip, nomeadamente, a partir da versão 21.1.2 para cima.
4. Instalar as bibliotecas Flask, PyJWT, logging, psycopg2, time e DateTime usando o comando: pip install <nome_da_biblioteca>.
5. Instalar o programa Postman a partir do site: https://www.postman.com/downloads/ para poder fazer pedidos e obter respostas do servidor web.

De seguida, basta seguir os passos descritos no manual do utilizador.

Manual do utilizador

O utilizador interage com o web server através da troca de request/response REST e, por sua vez, o web server interage com o servidor de base de dados através de uma interface SQL.

A nossa REST API permite ao utilizador aceder ao sistema através de pedidos HTTP. Existem várias funcionalidades que foram implementadas e todas elas estão disponíveis através de um endpoint diferente.



Configuração do Web Server e Base de Dados:

Para executar o software é necessário ter em atenção o modo como estão definidos os parâmetros nas funções **db_connection()** e **app.run()**.

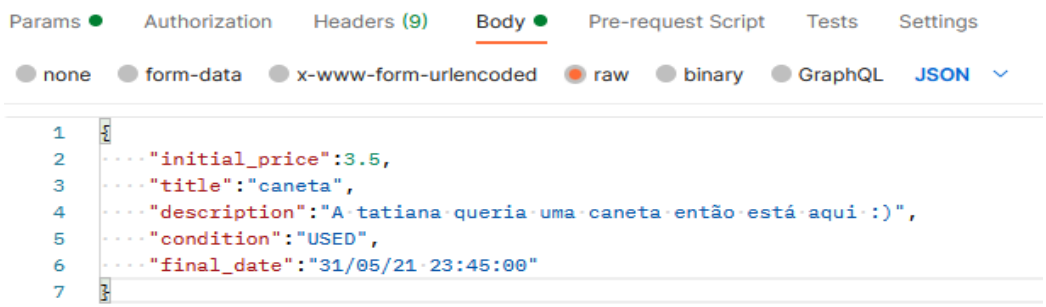
1. Na função **db_connection()** do script Auctions.py definir os parâmetros da conexão com a base de dados PostgreSQL:
    a. **user:** Nome do utilizador usado para a autenticação
    b. **password:** Password usada na autenticação
    c. **host:** O endereço da base de dados do servidor, pode ser o localhost ou um endereço de IP
    d. **port:** O número da porta que por definição é 5432 se não for dada outra
    e. **database:** O nome da base de dados à qual se quer conectar

2. Na main - if __name__ == “__main__” - definir na função **app.run()** os seguintes parâmetros:
    a. **host:** O nome do host onde se vai ficar à escuta
    b. **port:** O número da porta do servidor web

Modo de envio dos pedidos ao Servidor Web:

Para conseguir fazer os pedidos (Req) e respostas (Res) ao servidor web é usado o programa Postman. De modo a enviar um pedido para o servidor é necessário seguir os seguintes passos:

1. Criar uma nova coleção na aba *Collections* e introduzir um nome.
2. Clicar com o botão direito do rato na nova coleção criada e adicionar um novo pedido através da seleção *Add Request.*
3. Escolher o modo HTTP de comunicação entre o servidor e cliente. Há vários mas os mais usados durante o trabalho são o GET, POST e PUT.
4. Introduzir o URL do pedido com o host e port definido no **app.run()** juntamente com o decorador definido em **app.route()** na função a utilizar do programa , e.g: **http://localhost:8080/dbproj/auctions** .
5. Introduzir o corpo do pedido na opção *Body* → *raw* e alterar o tipo de texto para JSON. E introduzir os parâmetros que se quer enviar. Por exemplo:



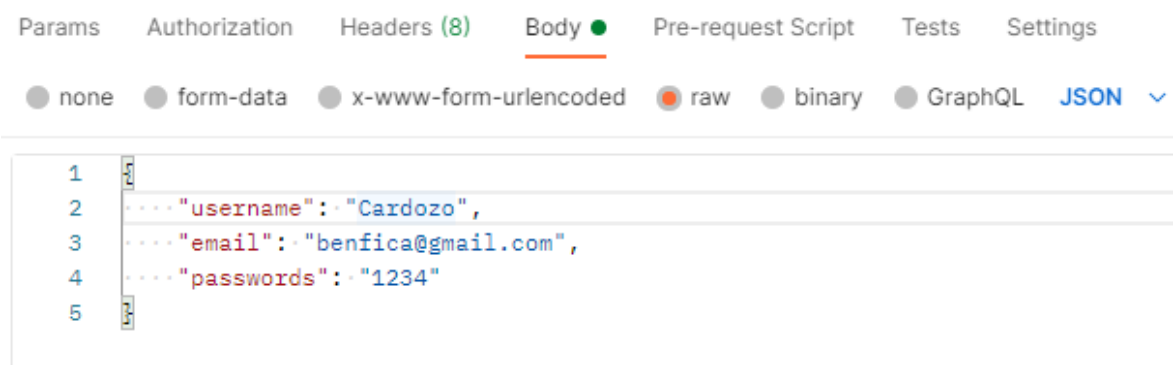
Repetir os passos de 2 a 5 para cada pedido diferente.

Funcionalidades Desenvolvidas no trabalho

1. **Registo de um utilizador -** Criar um novo utilizador.

    req POST http://localhost:8080/dbproj/user
    {“username”: username, “email”: email, “passwords”:password}
    ---
    res { “UserId”: UserId} or {“Error”: "codigoErro"}

- Para a criação de um novo utilizador é necessária a inserção de um username, um email e uma password. É importante referir que o username e o email tem que ser únicos dentro da base de dados.



Assim que a base de dados for criada não existirá qualquer utilizador com cargo de administrador (qualquer utilizador que se registre nunca será administrador), assim para poder ser dado o PRIMEIRO cargo como administrador é preciso fazer um UPDATE fora da API Rest da coluna **admins** = TRUE.

2. **Autenticação de utilizadores -** Login com username e password, recebendo um token de autenticação em caso de sucesso, token esse que deve ser incluído nas chamadas subsequentes.

req PUT http://localhost:8080/dbproj/user
{"username": username, "passwords": password}

---

res { AuthToken: "AuthToken"} or {Error: "Failed to authenticate user!"} or {Error: "Password does not match!"}

- Na autenticação de um utilizador é necessário introduzir um username já registado e uma password que lhe corresponda.
- É devolvido um token único, que diz respeito apenas àquele utilizador, e deve ser usado nas chamadas subsequentes por ele realizadas. O token mantém-se ativo durante 30 minutos.

Para a implementação do token foram criadas duas funções, uma para conseguir codificar o token e outra para decodificar:

1. **encode(user) -** Função que permite codificar o nome do utilizador juntamente com a data de codificação e de expiração do token. A data de expiração do token é 30 min após a criação do mesmo. A codificação é feita utilizando o algoritmo "HS256".
2. **decode(token)** - Função que permite decodificar o token com o algoritmo "HS256". Caso o token já tenha expirado, devolve um código de erro.

Todas as funcionalidades que se seguem exigem a presença do token de autenticação que representa o utilizador que está a exercer a funcionalidade. Por exemplo, http://localhost:8080/dbproj/<funcionalidade>?token=<AuthToken> . Note que é necessário colocar no URL sempre que for necessário o token de autenticação: "**?token=<AuthToken>**" cujo <AuthToken> é dado pela resposta do "**2.Autenticação de utilizadores**".

Todas as funcionalidades que se seguem também verificam se o utilizador a fazer o pedido URL já foi banido. Se sim, então é apresentada uma mensagem de erro.

3. **Criar um novo leilão -** Cria-se um leilão começando por identificar o artigo que se pretende comprar. Para simplificar, considera-se que cada artigo tem um código EAN que o identifica univocamente. Cada leilão deve igualmente ter um título, uma descrição. Para criar o leilão, o vendedor indica o preço mínimo que está disposto a receber, bem como a data, hora e minuto em que o leilão termina e o estado de condição do produto ("new", "used", "not specified").

req POST http://localhost:8080/dbproj/auction
{"initial_price": initial_price,"title": title,"description": description, "condition":condition, "final_date": final_date}

---

res { Ean: ean} or {Error:Failed to create the auction!} or {Error:" Please, insert a date after the current time"}

4. **Listar todos os leilões existentes**. - Listar todos os leilões ativos e as suas respectivas descrições

req GET http://localhost:8080/dbproj/auctions

---

res {Ean1 : “ean1” , Description1: “description1”
Ean2 : “ean2” , Description2: “description2”
Ean3 : “ean3” , Description3: “description3”}

5. **Pesquisar leilões existentes -** Listar os leilões que estão a decorrer, pesquisando pelo código EAN ou pela descrição do artigo.

req GET http://localhost:8080/dbproj/auctions/{keyword}

---

res {Ean1 : “ean1” , Description1: “description1”
Ean2 : “ean2” , Description2: “description2”}

- Para a pesquisa de um leilão ou de um certo tipo de leilões a keyword representa a sequência de caracteres pela qual se pretende pesquisar.

6. **Consultar detalhes de um leilão-** Para qualquer leilão escolhido, obter todos os detalhes relativos à descrição do artigo, ao término do leilão, às mensagens escritas no seu mural e ao histórico de licitações efetuadas nesse mesmo leilão.

req GET http://localhost:8080/dbproj/auction/<ean>

---

res {Ean1 : “ean1” , Description1: “description1”, /**restantes detalhes e mensagens}

7. **Listar todos os leilões em que o utilizador tenha atividade.** Um utilizador deve poder listar os leilões nos quais tem ou teve alguma atividade, seja como criador do leilão seja como licitador. Esta listagem contém os detalhes de cada leilão.

req GET http://localhost:8080/dbproj/user/auctions_activity

---

res {Ean1 : “ean1” , Description1: “description1”, /**restantes detalhes e mensagens} ou {Error: “códigoErro”}

8. **Efetuar uma licitação num leilão**. Um comprador pode licitar com um preço mais alto num determinado leilão, desde que o leilão não tenha terminado e que não haja uma sua licitação mais alta do que a que está a fazer e seja, pelo menos, superior ao preço mínimo.

req GET http://localhost:8080/dbproj/bid/<ean>/<bidding>

---

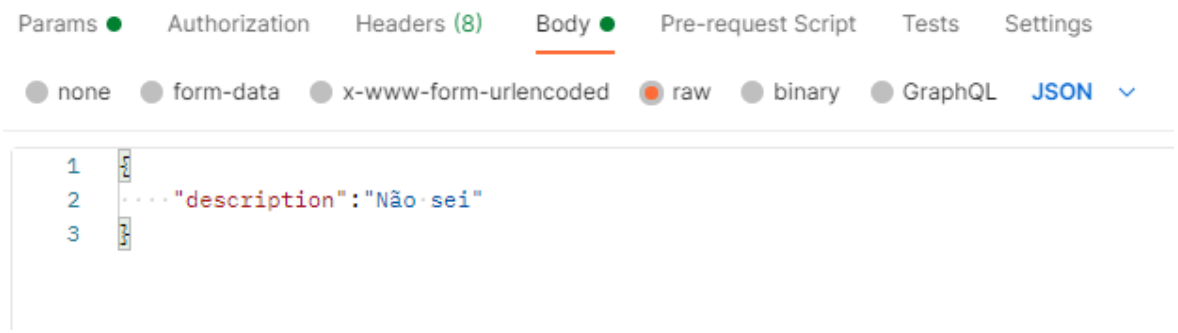
res { "Bidding done!"} ou {Error: "codigoErro"}

9. **Editar propriedades de um leilão.** O vendedor pode ajustar todas as descrições textuais relativas a um leilão seu, sendo que todas as versões anteriores ficam guardadas e podem ser consultadas posteriormente para referência.

req PUT http://localhost:8080/dbproj/auction/<ean>
{/**informação a ser alterada**/ description= "descrição_atualizada", (...)}

---

res {/**informação sobre o leilão **/} ou {Error: "codigoErro"}



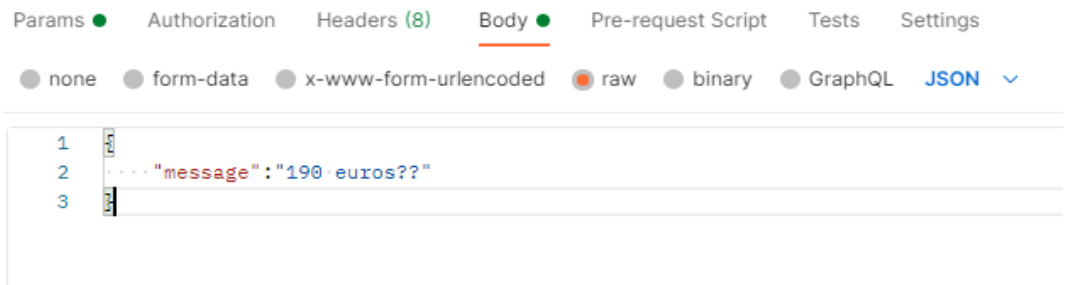
Para editar as propriedades do leilão pode ser alterada a coluna **description**, **title** ou **description** e **title.**

10. **Escrever mensagem no mural de um leilão.** Cada leilão deve ter um "mural" onde poderão ser escritos comentários, questões e esclarecimentos relativos ao leilão.

req POST http://localhost:8080/dbproj/<ean>/mural_message
{message : "mensagem"}

---

res {"Message has been written"} ou {Error: "codigoErro"}



11. **Entrega imediata de notificações a utilizadores.** Os utilizadores recebem imediatamente na sua caixa de entrada as notificações acerca das mensagens publicadas, e deverão estar disponíveis no endpoint correspondente. O criador de um leilão é notificado de todas as mensagens relativas a esse leilão. Todos os utilizadores que tiverem escrito num mural passam a ser notificados acerca de mensagens escritas nesse mesmo mural.

    req GET http://localhost:8080/dbproj/notification_box/<view>

    ---

    res {/**informação sobre o leilão **/} ou {Error: "codigoErro"}

- Existem 2 opções que podem ser colocadas no parâmetro ''view'' e que vão retornar coisas diferentes.

    "all" - retorna todas as notificações do utilizador

    "not_read" - retorna apenas as notificações ainda não lidas, quando é lida troca a coluna was_read para **True.**

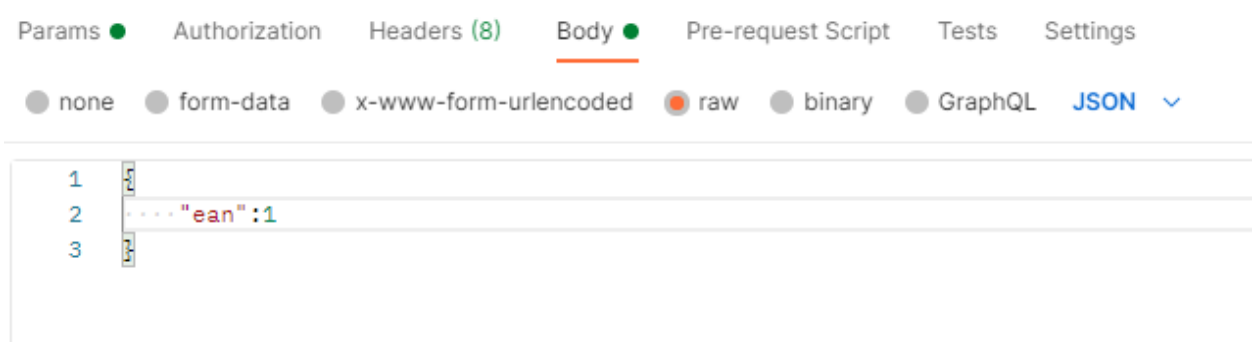
12. **Notificação de licitação ultrapassada.** Um comprador que tenha feito uma licitação num leilão recebe uma mensagem na sua caixa de mensagens sempre que houver outra licitação melhor que a sua.

13. **Término do leilão na data, hora e minuto marcados.** No momento indicado pelo vendedor (data, hora e minuto) o leilão termina. Determina-se aí o vencedor e fecha-se a possibilidade de realizar mais licitações. Os detalhes deste leilão são atualizados e podem ser consultados posteriormente.

    req PUT http://localhost:8080/dbproj/<ean>/terminate_auction

    ---

    res {"Action <ean> has finished!"} ou {"Auction still has some time left to finish"}{Error: "codigoErro"}



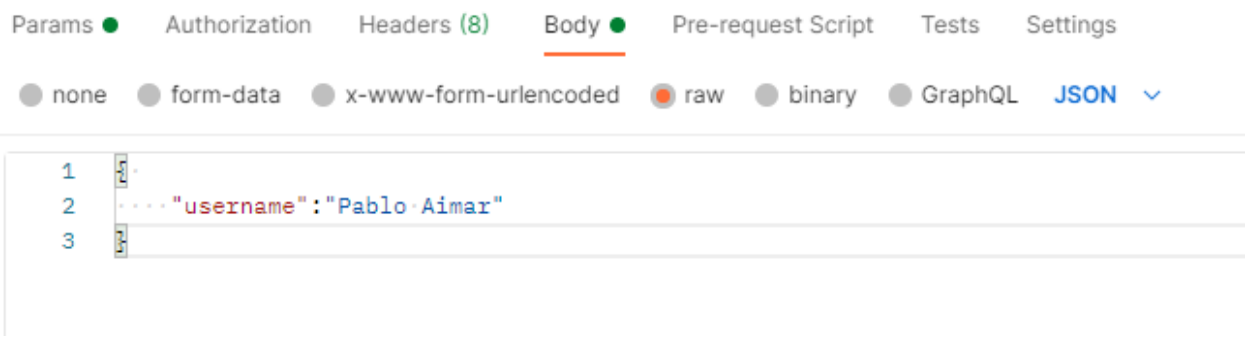
Todas as funcionalidades que se seguem só podem ser utilizadas recorrendo a um token de autenticação que pertence a um administrador. Para a atribuição do cargo de administrador criamos um novo endpoint para dar permissões de administrador a utilizadores normais.

14. **Dar cargo de administrador.** Um administrador consegue atribuir o cargo de administrador a outro utilizador.

    req PUT http://localhost:8080/dbproj/give_permissions
    {username = "username"}

    ---

    res {"User <username> was promoted to admin with success!"} ou {Error: "codigoErro"}

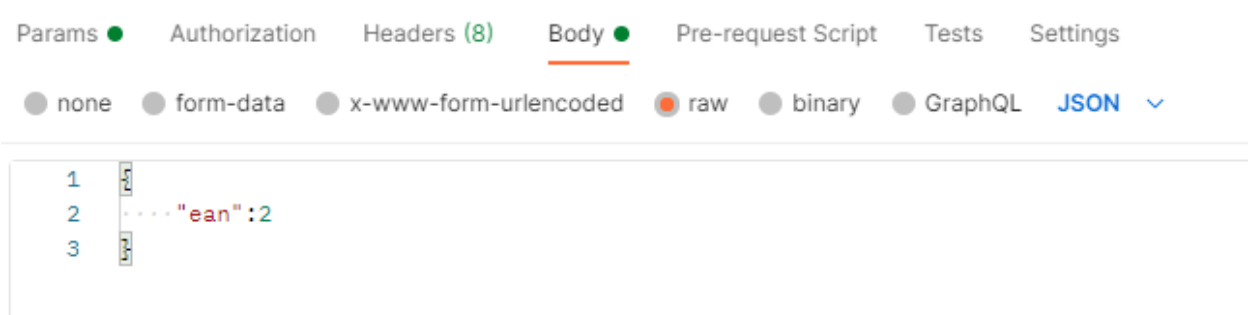


15. **Um administrador pode cancelar um leilão.** Um administrador pode cancelar um leilão se tal for necessário. O leilão continua a ser consultado pelos utilizadores, mas está dado como cancelado e não podem ser feitas licitações. Todos os utilizadores interessados recebem uma notificação.

    req PUT http://localhost:8080/dbproj/auction_cancellation
    {ean : "ean"}

    ---

    res {"Action <ean> was cancelled with success!"} ou {Error: "codigoErro"}

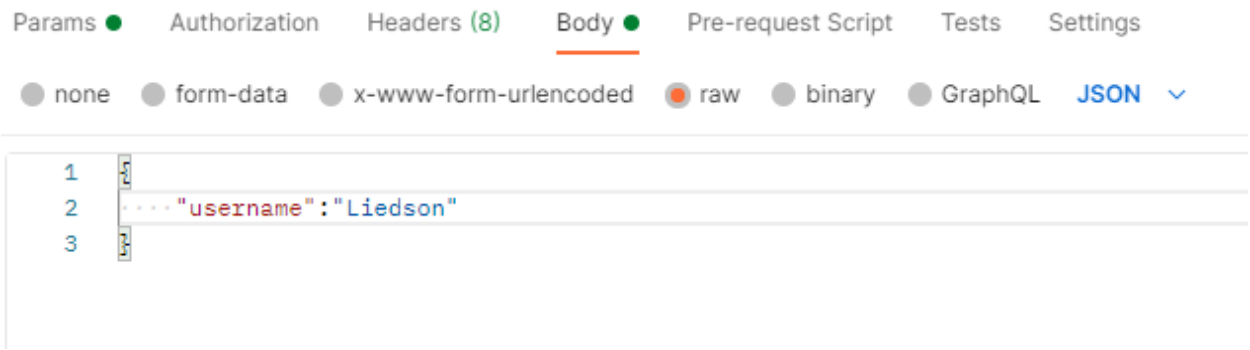


16. **Um administrador pode banir permanentemente um utilizador.** Um administrador pode banir um utilizador se tal for necessário. Todos os leilões criados por esse utilizador são cancelados. Todas as licitações efetuadas por esse utilizador são invalidadas (ainda que mantidas nos registos). Ao invalidar uma licitação num leilão, quaisquer licitações superiores a essa são igualmente invalidadas exceto a melhor delas, cujo valor se torna igual ao valor da que for invalidada. Automaticamente é criada uma mensagem no mural dos leilões afetados lamentando o incômodo e todos os utilizadores envolvidos recebem uma notificação.

req PUT http://localhost:8080/dbproj/ban_user
{username : "username_to_ban"}

---

res {"User <username> was successfully banned"} ou {Error: "codigoErro"}



Para verificar nas diversas funcionalidades se o utilizador já foi banido, foi implementada a função **was_banned(user_id)** que a partir do id do utilizador verifica se o mesmo já foi ou não banido. Se tiver sido então devolve um código de erro.

17. **Um administrador pode obter estatísticas de atividade na aplicação.** Um administrador pode consultar estatísticas da utilização da aplicação: top 10 utilizadores com mais leilões criados, top 10 utilizadores que mais leilões venceram, número total de leilões nos últimos 10 dias

req GET http://localhost:8080/dbproj/app_stats

---

res {/** informação sobre a aplicação**/} ou {Error: "codigoErro"}

Plano de desenvolvimento do trabalho

Foi-nos proposto apresentar um plano de desenvolvimento do trabalho, especificando por tarefas o número de horas investido por cada um. Em seguida encontra-se esse mesmo plano:

Tarefa	João Silva	Sofia Neves	Tatiana Almeida
Diagrama ER	5h	5h	5h
Setup do ambiente de desenvolvimento	2h	7h	5h
Criação da Base de Dados	3h	3h	3h
Gestão da Base de Dados	24h	1h	2h
Implementação das funcionalidades	40h	40h	40h
Relatório	1h	5h	7h