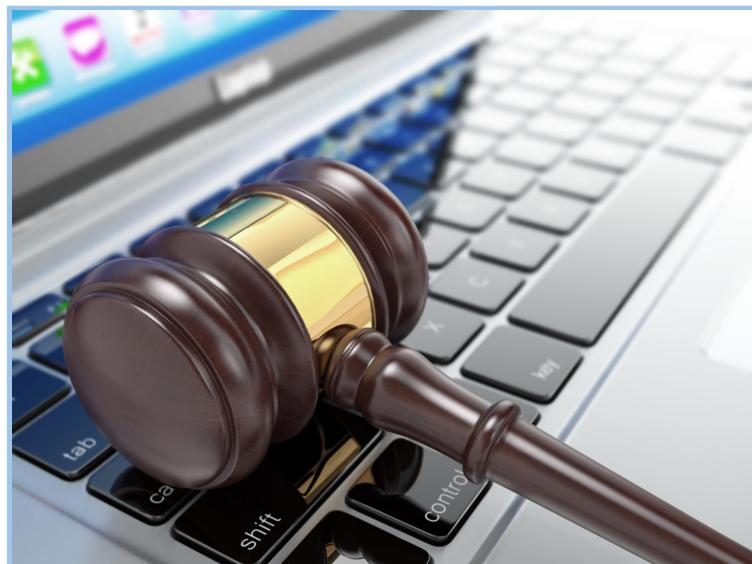


Base de Dados

31 de Maio 2021

Relatório

Leilões Online



Ana Luís da Rocha Alves Rainha Coelho, 2015231777, uc2015231777@student.uc.pt
João Francisco Santos Ferreira, 2015257786, jfsf@student.dei.uc.pt
Sofia Meireles Fonseca Costa, 2018296218, sofiacosta@student.dei.uc.pt

Manual de instalação:

1. Instalar PgAdmin ou psql (shell client) para criar a base de dados.
2. Criar uma base de dados com o nome 'project'.
3. Correr os seguintes comandos no psql.
 - a. CREATE EXTENSION pgcrypto;
 - b. CREATE EXTENSION "uuid-ossp"
4. Correr o script tabelas.sql (i tabelas.sql) no psql.
5. Usar pip para instalar o psycopg2, flask e dotenv (Alternativamente instalar a partir das binaries disponíveis nos sites das frameworks).
6. Instalar Postman.

Manual de utilizador:

Para correr a aplicação, não foi usada uma interface gráfica. Utilizamos o Postman para testar a API, este vai enviar as informações como um utilizador.

Nota: Verificar a conexão à base de dados.

exemplo:

```
598
599     def db_connection():
600         db = psycopg2.connect(user="postgres",
601                               password="1379",
602                               host="localhost",
603                               port="5432",
604                               database="meta2")
605
606
```

Sendo o user o username do perfil do psql ou PgAdmin, e a database o nome da nossa base de dados.

1. Correr a aplicação que vai iniciar o servidor. (python app.py, ie.)

```
DEBUG CONSOLE PROBLEMS ② OUTPUT TERMINAL
anita@iMac-de-Ana ~ % /opt/anaconda3/bin/python /Users/anita/Desktop/BD_meta2/app.py
* Serving Flask app "app" (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
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
* Restarting with fsevents reloader
* Debugger is active!
* Debugger PIN: 245-052-860
```

2. Fazer pedidos com o Postman.

PUT

The screenshot shows the Postman interface with a PUT request to `localhost:5000/dbproj/user`. The body is set to `JSON` and contains the following JSON payload:

```

1
2   "username" : "anita",
3   "password" : "admin"
4

```

The response status is `200 OK`.

POST

The screenshot shows the Postman interface with a POST request to `localhost:5000/post_auction/`. The body is set to `JSON` and contains the following JSON payload:

```

1
2   "minimumprice" : 10,
3   "begindate" : "2023-01-19 09:26:03.478839",
4   "enddate" : "2018-05-31 14:59:03.478839",
5   "title" : "LEILAO 10",
6   "description" : "Saqueando melhor leilão do país",
7   "person_userid" : "Stafabdf89-278d-455d-bf6e-c118088476da",
8   "admin_person_userid" : "Stafabdf89-278d-455d-bf6e-c118088476da",
9   "description_unit" : "DS masticadas pelo Harry Potter",
10  "token" : "1fc90178-f2ac-4d5b-b9ed-8772bf604e40"
11

```

The response status is `200 OK`.

GET

The screenshot shows the Postman interface with a GET request to `localhost:5000/get_auction/`. The body is set to `JSON` and contains the following JSON payload:

```

1
2   "token" : "1fc90178-f2ac-4d5b-b9ed-8772bf604e40"
3

```

The response status is `200 OK`.

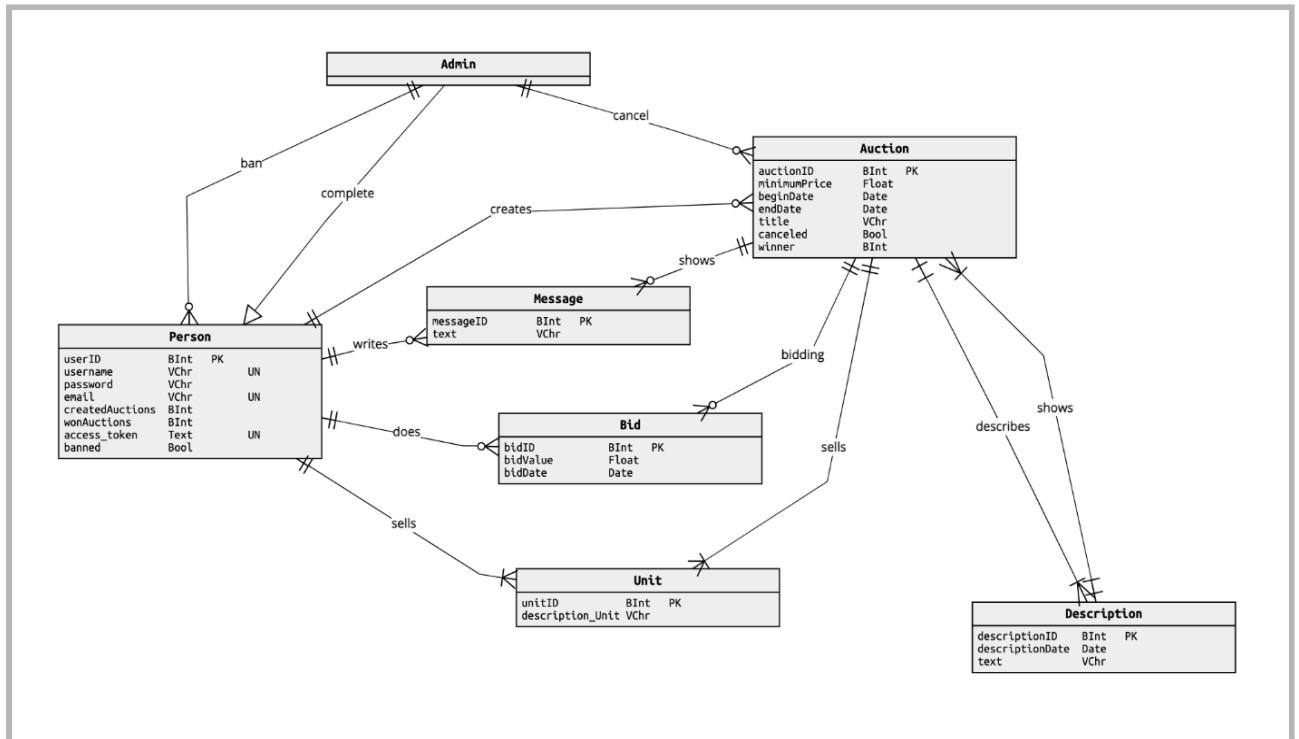
Os pedidos ao servidor podem ser POST, GET e PUT. Para isso o utilizador tem de colocar o URL que vai com o `localhost`, o `port` e o nome da ação (caso seja necessário passar um parâmetro no URL, também terá de ser introduzido pelo utilizador). No body o utilizador carrega no botão `[raw]` e altera o tipo de package a ser enviado através do pedido HTTP para `JSON`. De seguida, vai escrever o nome das colunas das tabelas guardadas nas base de dados com os dados a inserir (`"username" : "anita"`, ie.). É necessário separar os atributos por vírgula. Depois é só carregar no botão de `Send` e o pedido é enviado para o servidor.

Os pedidos são tratados nas ações que se encontram implementadas na `app.py`.

As routes são as seguintes:

Create User - localhost:5000/dbproj/user [POST]
Login - localhost:5000/dbproj/user [PUT]
Create Auction - localhost:5000/post_auction/ [POST]
Get Auction - localhost:5000/get_auction/ [GET]
Search Auction - localhost:5000/search_auction/ [GET]
Get Description - localhost:5000/get_description/ [GET]
Update Auction - localhost:5000/dbproj/leilao [PUT]
Post Message - localhost:5000/post_mesg/ [POST]
Get Mural - localhost:5000/dbproj/mural/<auction_id> [GET]
Cancel Auction - localhost:5000/dbproj/cancel/<auction_id> [POST]
List Auctions - localhost:5000/dbproj/auctions [GET]
Add Bid - localhost:5000/dbproj/bid/<auction_id>/<bid_value> [POST]
Get Auction Details - localhost:5000/dbproj/auction/details/<auctionid> [GET]
Get Active User Auctions - localhost:5000/dbproj/active/<userid> [GET]
(Admin only) Get Stats - localhost:5000/dbproj/stats [GET]
(Admin only) Cancel Auction - localhost:5000/dbproj/cancel/<auction_id> [POST]
(Admin only) Ban User - localhost:5000/dbproj/ban/<user_id> [POST]

Diagrama Entidade Relacionamento (ER)



Relational data model (Diagrama Físico):

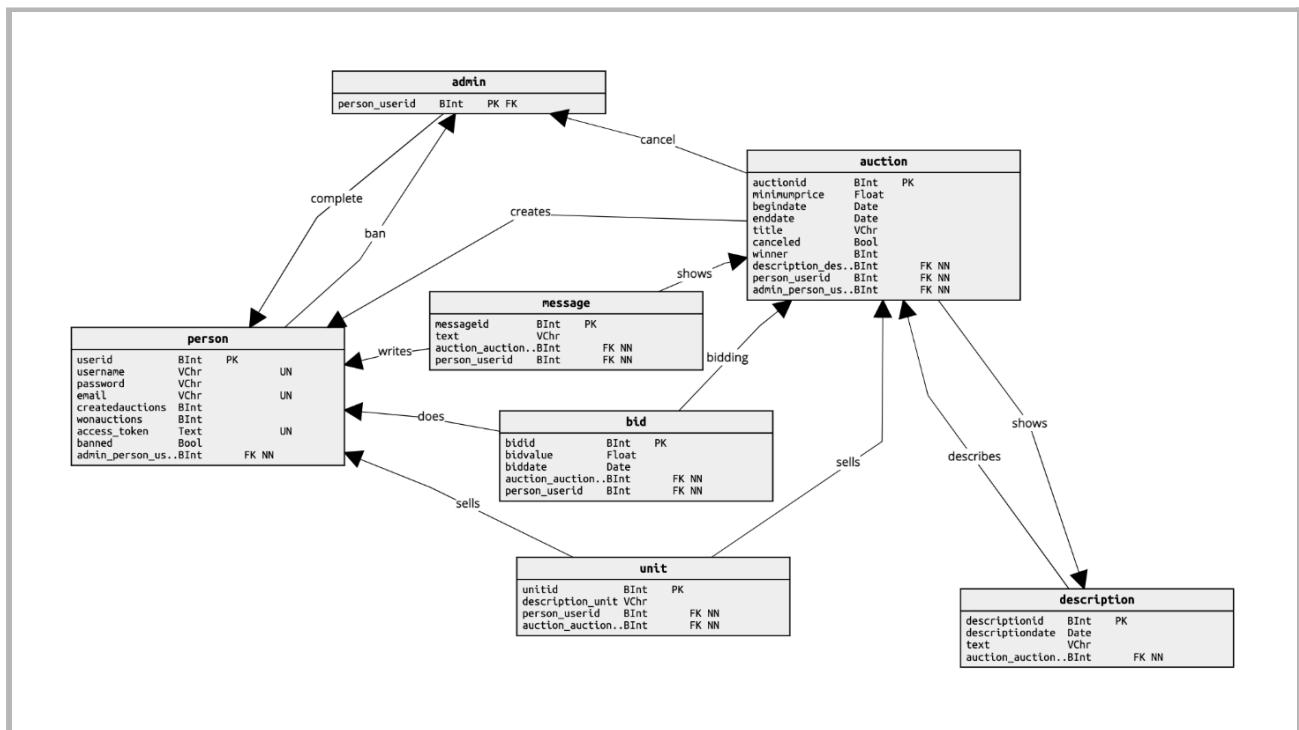


Tabela de tarefas:

O projecto demorou aproximadamente 60 horas a desenvolver. E foi implementado em conjunto pelos três elementos do grupo. A divisão de tarefas foi igualmente dividida.

Requisitos funcionais	Pass/Fail
Registo de utilizadores	Pass
Autenticação de utilizadores	Pass
Criar um novo leilão	Pass
Listar todos os leilões existentes	Pass
Pesquisar leilões existentes	Pass
Consultar detalhes de um leilão	Pass
Efetuar uma licitação num leilão	Pass
Editar propriedades de um leilão	Pass
Término do leilão na data, hora e minuto marcados	Pass
Escrever mensagem no mural de um leilão	Pass
Um administrador pode cancelar um leilão	Pass
Um administrador pode obter estatísticas de atividade na aplicação	Pass
Listar todos os leilões em que o utilizador tenha atividade	Pass
Um administrador pode banir permanentemente um utilizador	Pass
Não implementados	
Entrega imediata de notificações a utilizadores	
Notificação de licitação ultrapassada	