BD-Project

About the project

Project for the Databases subject of the Informatics Engineering course @University of Coimbra

The aim of this project is to provide students with experience in developing database systems. The project is oriented by the industry's best practices for software development; thus, students will experience all the main stages of a common software development project, from the very beginning to delivery.

This project consists of developing a typical auction system, supported by a database management system. An auction is initiated by a seller who defines an item, indicates the minimum price you are willing to receive, and decides when that the auction will end (date, hour and minute). Buyers place bids that successively increase the price until the end of the auction. Wins the buyer who bids the highest amount.

The system must be made available through a REST API that allows the user to access the system through HTTP requests (when content is needed, JSON should be used). The figure represents a simplified view of the system to be developed. As we can see, the user interacts with the web server through the exchange of REST request / response and in turn the web server interacts with the database through an SQL interface.

Index

- BD-Project
  - About the project
  - Index
  - Technologies used
  - Project structure
  - Native installation
    - Tools installation
    - Database setup
    - Running the application
  - Docker support
    - Docker and Docker-Compose installation
    - Image installation and container creation
      - Using docker-compose
      - Manually
    - Interacting with the REST API and database containers
      - Database Container
      - REST API
    - Useful docker commands
  - Dragon server
  - Project features

        * REST API specification
- Collaborators

Technologies used

1. Programming Languages
    - Python
    - SQL and PL/pgSQL
2. Database Management System
    - PostgreSQL
3. Python Libraries
    - Flask
    - Psycopg2
4. Other Technologies
    - Curl
    - Onda
    - Docker
    - Postman

Project structure

```
.
  docs
    deliverables
      meta1.zip
    db_project.pdf
  python
    app
      API.py
    Dockerfile
  scripts
    compose.sh
    compose.yml
  src
    app
      logs
        log_file.log
      api.py
      Dockerfile
    db
      data.sql
      Dockerfile
      schema.sql
  .gitignore
  LICENSE
  README.md
  REQUIREMENTS.txt
```

`TODO: Add Project Structure`

Native installation

If you choose to install and run the application and all its tools natively please follow these instructions.

`NOTE:` The native installation allows you to have more control over the application and database configurations but if you are looking to try the application and you don't want to worry about all the small configuration details, this project has docker support allowing you build pre-configured images and create containers that will run this application components with little or no configuration. To know how you use them check the Docker Support chapter.

Tools installation

In order to run this project it is required to have an installation of a python interpreter `python3` or `pypy3` the python package installer `pip` and the `curl` or `postman` tool which provides the same functionality as curl but is being more GUI oriented. In our project we are opting to use the python3 interpreter and the curl tool. These programs can be installed using your operating system package manager as shown bellow for some Linux distributions and MacOS.

```
# MacOS:
brew install python3 curl

# CentOS:
sudo yum install python3 python3-pip curl

# Ubuntu/Debian
sudo apt install python3 python3-pip curl

# Fedora:
sudo dnf install python3 python3-pip curl

# OpenSUSE:
sudo zypper install python3 python3-pip curl

# Arch Linux:
sudo pacman -S python python-pip curl
```

In case of Windows/MacOS machines it is possible to install the software via a graphical installer. For more information about the download installation of python3, pip, curl and postman check the following websites:

- Python
- Pip

- Curl
- Postman

To store information this project uses the PostgreSQL Database management system. This DBMS can be installed using your operating system package manager as shown bellow for a debian based Linux distributions.

```
# Create the file repository configuration:
sudo sh -c 'echo "deb http://apt.postgresql.org/pub/repos/apt $(lsb_release -cs)-pgdg main"

# Import the repository signing key:
wget --quiet -O - https://www.postgresql.org/media/keys/ACCC4CF8.asc | sudo apt-key add -

# Update the package lists:
sudo apt-get update

# Install the latest version of PostgreSQL.
# If you want a specific version, use 'postgresql-12' or similar instead of 'postgresql':

# In this case we are installing it on a debian based distro,
sudo apt-get -y install postgresql psql
```

If you have a machine running any other linux distributions, check the postgreSQL website to see how you can install this software in your machine. In the case MacOS or Windows you can find graphical based installers. To to know more about the installation process and options of this DBMS this link will point you to the postgreSQL official installation page.

- PostgreSQL

In the project implementation a few python libraries are used, namely: Flask to set up a REST web service and Psycogp2 in order to interact with a postgreSQL database. To see more about them check the links bellow:

- Flask
- Psycogp2

To install all the python libraries described above run the following command on the REQUIREMENTS.txt file:

```
python3 -m pip install -r REQUIREMENTS.txt
```

Database setup

After the native installation of all the tools you are almost ready to begin using the application's REST API. Before that you will need to create and setup your native postgreSQL installation and the web server where the REST service will run. To setup de database you need to access your postgreSQL DMBS with a client like `psql` or `pgadmin4` in order to create a new user and database to host all the application's data. The following instructions show you how you can do this using the `psql` client (that was installed along with postgreSQL).

```
# Use psql to login to the default superuser account on your DBMS.
# This information can be changed after should you wish to make
# the native installation more secure

# Default superuser account information
# Username: postgres
# Password: postgres

psql -h localhost -p 5432 -U postgres

-- Create a database to be used by the application
-- As an example we are going to create a database called dbauction

CREATE DATABASE dbauction;

-- We will also create a example user with regular privileges so we
-- don't have to access our database with only the root user.

CREATE USER example PASSWORD '123';
EXIT

# Now to configure the database just access it with your recently created user.

# Database: dbauction

# Example user account information
# Username: example
# Password: 123

psql -h localhost -p 5432 -U example -d dbauction

\i src/db/schema.sql       # The database table schema
\i src/db/data.sql         # Some example data

IMPORTANT: Add all the SQL files that we are still going to make
as the project grows bigger
```

Running the application

After all the configuration of the database is time to run our web server making the REST API accessible to the outside world. To do that just run the api.py file providing it the required arguments .The arguments consist of a database name, host and port we want to connect to, where we can store and retrieve information and a the user credentials (username and password) necessary to access it. To see which arguments to pass check the help text of the program by executing it with no arguments.

```
# To see the usage help text

python3 api.py

# Example usage (using the database and user created for demonstration)

# Username: example
# Password: 123
# Database Name: dbauction
# Machine Hostname: localhost
# Database Port: 5432

python3 api.py -u example -p 123 -D dbauction -H localhost -P 5432
```

Once we execute the api.py after all this setup the web server that will provide the REST service will be started and will run in the host machine until the program is stopped. By default a flask web server will run on port 5000, in order to interact with it we can use a web browser or make curl/postman calls to the following URL or other URLs derived from this one and described in the REST API endpoint specification.

```
http://localhost:5000/
```

Docker support

Docker is an open platform for developing, shipping, and running applications. Docker enables you to separate your applications from your infrastructure so you can deliver software quickly. With Docker, you can manage your infrastructure in the same ways you manage your applications. By taking advantage of Docker's methodologies for shipping, testing, and deploying code quickly, you can significantly reduce the delay between writing code and running it in production.

The components of this project such as the PostgreSQL database and REST API service (that will be provided via flask web server/application) can be ran in docker containers instead natively in your machine. The docker images are already pre-configured saving you the trouble of installing all the software natively in your machine.

Docker and Docker-Compose installation

The docker and docker-compose can be installed using your operating system package manager as shown bellow for a debian based Linux distribution (for other distributions the process is similar).

```
## INSTALL DOCKER (debian based distro)
```

```
sudo apt-get update

sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg-agent \
    software-properties-common

curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
sudo apt-key fingerprint 0EBFCD88

sudo apt-get update
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose

sudo add-apt-repository \
   "deb [arch=amd64] https://download.docker.com/linux/ubuntu \
   $(lsb_release -cs) \
   stable"
```

In case of Windows/MacOS machines it is possible to install the software via a graphical installer. For more information about the download installation of docker and docker-compose tools check the following links.

- docker
- docker-compose

Image installation and container creation

In order to install the images, create containers and run them in your machine you have multiple ways to do it:

Using docker-compose

You can use the docker-compose tool to automate the build process of docker images. This docker file follows a list of recipes specified in .yml file and that tells docker how should the images be built which images depend on which, the volumes to mount, the ports to expose, etc… For this project a docker-compose configuration file (compose.yml) has already been made. You can add more information to it should you wish to configure it even more so it fits your needs. The following examples show example usage of the script

In order to build this particular project you can issue the following commands.

```
mkdir -p ../src/app/logs
docker-compose -f compose.yml up --build
```

These commands will create a “logs” file where the REST API server logs will be placed and will assemble the app and db images (downloading the necessary components) create containers for both of them and run them (in foreground) until you issue a SIGINT and stop the process. For ease of use we added the script compose.sh which allows you to execute these commands and add more options

```
# Give the user permission to execute this script
chmod u+x compose.sh

 # Executes the commands shown above (default behaviour)
./compose.sh

# Use the --no-start option to compose the images only
# This command installs the images only
./compose.sh --no-start


# Use the -d option to compose the images and run
# containers from those images in the background
./compose.sh -d
```

`NOTE:` After you run the docker-compose command or script the db and auction-web-app images and will be installed in your system and the containers from those images will be created too. The next time you want to run a container from those images please consider using `docker start {container_name}` to avoid the creation of a new image/container equal to the already installed/created. Just keep in mind that the database container must be started before the auction-rest-api. Use `docker stop {container_name}` to stop the containers.

Manually

You can build the images and run the containers manually should you although it might be error prone since something can go wrong with the configuration. We highly discourage this usage since its safer to build the images and run containers by following the instructions in the docker compose file. The examples bellow show a the same configuration that you get when using docker-compose but executed manually.

```
# Build docker images
docker build -t  auction-rest-api:latest  /src/app
docker build -t  db:latest  /src/db

# Create a logs folder
mkdir -p logs

# Run docker images
docker run --name db -p 6000:5432 db:latest
docker run --name auction-rest-api -p 8080:5000
```

```
\ -v logs:/app/logs auction-rest-api:latest
```

Interacting with the REST API and database containers

Once you get the containers running the REST API and database will be accessible through an exposed port in your systems localhost. Assuming that you followed these instructions you can access the containers like this:

`NOTE:` Always Keep in mind that the database container must be started before the auction-rest-api

Database Container

The database is exposed by the docker container in the port 6000 of your host machine to access it use psql, pgadmin4 or any other postgreSQL client

```
# Example: Accessing the database with the superuser

# Database Name: dbauction
# Database Exposed port: 6000

# Example user account information
# Username: admin
# Password: admin

psql -h localhost -p 6000 -U admin -d dbauction
```

REST API

The REST API exposed by the docker container in the port 8080 of your host machine, in order to interact with it we can use a web browser or make curl/postman calls to the following URL or other URLs derived from this one and described in the REST API endpoint specification.

`http://localhost:8080/`

Useful docker commands

Here is a list of docker commands that might be useful

```
# To see all the available commands do
docker --help

# To start a container do
docker start {container}

# To stop a container
docker stop {container}
```

```
# To run a container from an image
docker run {image}

# To list all running containers do
docker ps

# To list all running and stopped containers do
docker ps -a

# To see all the images installed in your system do
docker images

# To remove a image do
docker rmi {image_name}

# To forcefully remove all images do
docker image prune -f

# To remove a docker container do
docker rm {container_name}

# To forcefully remove all containers do
docker container prune
```

Dragon server

```
TODO: Add here information about the interaction with our server
i will do this when we complete the project
```

Project features

REST API specification

```
TODO: Add here information relative to our API
TODO: Format this better
```

- User login Query

```
curl -X PUT http://localhost:8080/dbauction/user -H "Content-Type: application/json" -d '{"u
```

- Expected Response

```
{
  "token": "Here goes a big string that is in fact a token"
}
```

- User Registry Query

```
curl -X POST http://localhost:8080/dbauction/user -H "Content-Type: application/json" -d '{'
```

- Expected Response

```json
{
  "token": "Here goes a big string that is in fact a token"
}
```

Collaborators

- Miguel Rabuge
- Duarte Dias