Globant's Data Engineering Coding Challenge

Johan Pineda

In the following document, the advancements for the Globan'ts data engineering coding challenge is presented. The primary objective was to create an API that accomplishes three key tasks:

- Receive historical data from CSV files and upload to a database
- Return number of employees hired for each job and department in 2021 divided by quarter.
- Return a list of ids, name and number of employees hired of each department that hired more employees than the mean of employees hired in 2021 for all the departments

When tackling this coding challenge, various factors needed to be considered. In this case, the different implementations were designed with an emphasis on time development, accessibility, and scalability. As a result, both the database and the API are hosted on AWS services due to their provision of free service access, which allowed the operation within the minimum free tier in terms of computational power and storage. The following information is relevant regarding the coding challenge's implementation.

The code is store in a github repo with the following url:

- https://github.com/jmpt97/GlobantTechInterview

The Database is stored in AWS Amazon RDS as an MSSQL DB with the following server:

- globant-tech.chsprj1nr44a.us-east-1.rds.amazonaws.com

And finally, the API is hosted with AWS lightsail, that allows the user to upload docker image containers. The url for the API is the following:

- https://jmpt-tech-int.0tgg51p70fnse.us-east-1.cs.amazonlightsail.com/

For this case, the challenge can be divided in to 4 principal aspects Database, API and Tests.

Database:

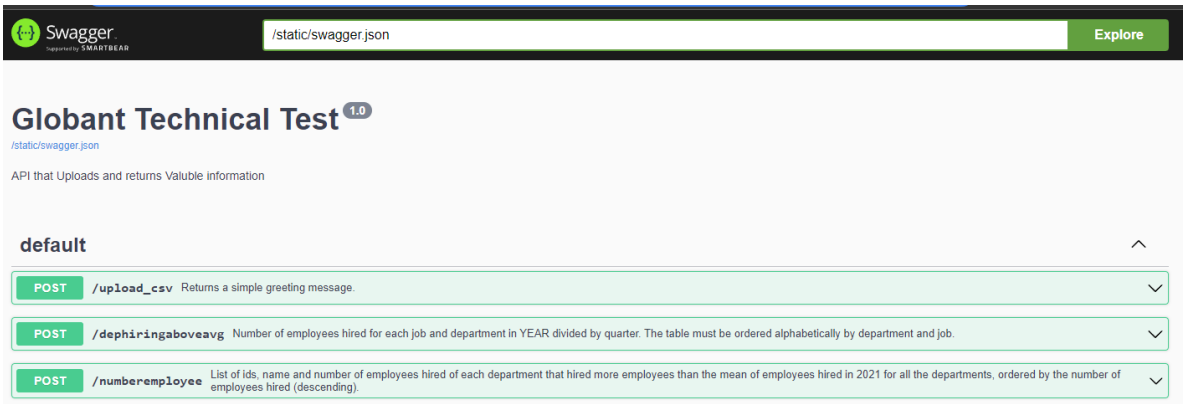
The database selected for the solution is a MSSQL database with 2 vCPUs, 1 GiB of RAM and a Network speed of 2085 Mbps.

For this part of the solution, the Database ***GlobantTech*** was created. In this database, 3 tables and 2 stored procedures were developed to fulfill the requirements of the solution and the necessary tasks.

- **Tables**:
    1. hired_employees: This table contains information about employees who have been hired. It includes details such as IDs, names, hire_datetime, deparment_id, job_id.
    2. departments: This table is related to organizational departments within a company or institution. It contains data about various departments, including a unique identifier and department name.
    3. jobs: This table is related to job positions within an organization. It includes data about different job titles and including a unique identifier.
- **Stored Procedures**:
    1. GetDepartmentsWithHiringAboveAverage: This stored procedure retrieves departments with hiring counts above the average for a specific year. It takes a single input parameter @year, which is an integer representing the year for which you want to perform the analysis. This procedure identifies departments that have hired more employees than the average hiring count for a specified year.
    2. GetQuarterlyHiringStatistics: This stored procedure retrieves quarterly hiring statistics for a given year, department, and job. It has three input parameters: @year (optional), @department (optional) and @job (optional). The last two can be used to query different type of departments and jobs that both can be separated by a semi-colon. This procedure allows a user to get a summary of quarterly hiring statistics for specific departments, job positions, and years. If no specific department or job is provided, it brings all the information available.

API:

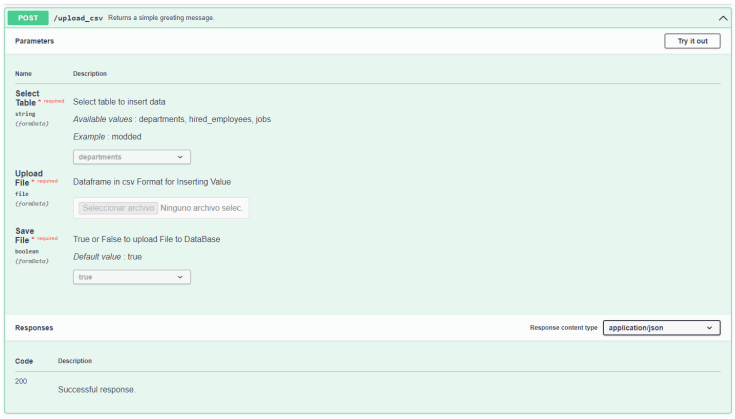
The API was built using python and Flask. As a preference in easiness and user design, the API was implemented hand by hand with the SWAGGER interface. In the following image, the API swagger interface can be observed:



The API has three principal POST methods:

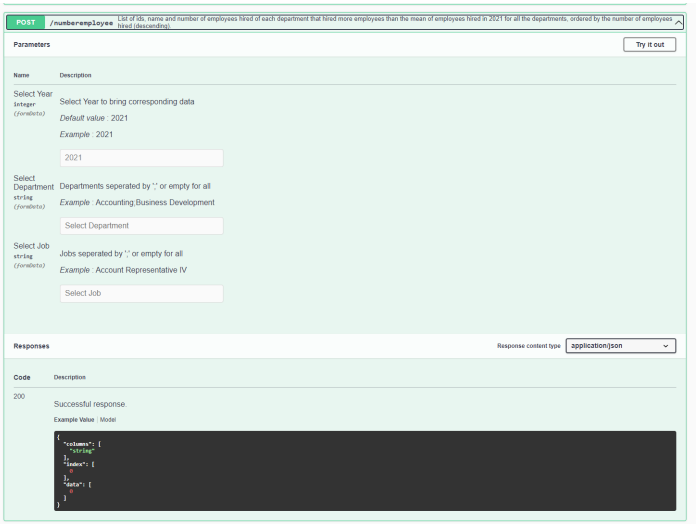
- **/upload_csv**: Is in charge of uploading a CSV file and save it to the specified database table regarding the users desired input.
- **/numberemployee**: This method retrieves quarterly hiring statistics based on user-provided criteria implementing the stored procedure *GetQuarterlyHiringStatistics*
- **/dephiringaboveavg**: This method retrieves the department hiring information above average for a given year with the execution of the stored procedure GetDepartmentsWithHiringAboveAverage.

Each method has the option to accept different values or search criteria that the user can define. For ***/upload_csv*** the user can input the table where the data is going to be inserted, the file, and select if the data is going to be saved in the DB. This can be seen in the following image:



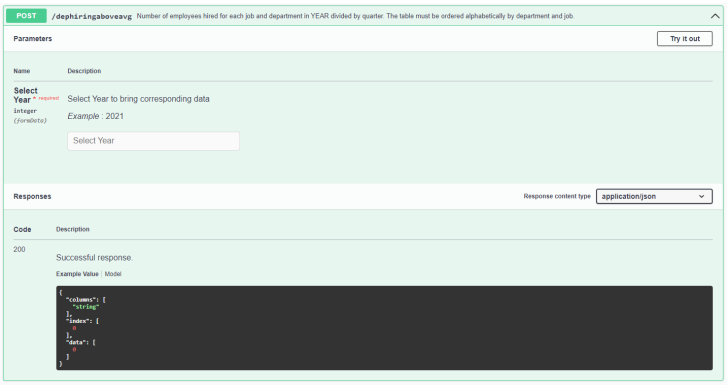
Additionally, the method ***/upload_csv*** has verifications regarding the length of the columns expected for each table. An example of the response received when uploading a file is seen in the image below:

For the method ***/numberemployee*** the user can define the year of the query, select the departments and select the jobs that want to be queried:

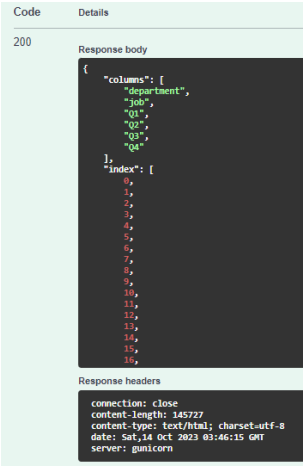


An example of the response received when uploading a file is seen in the image below:

For ***/dephiringaboveavg*** the user can select the year which is desired to be queried as observed in the image below:



An example of the response received when uploading a file is seen in the image below:



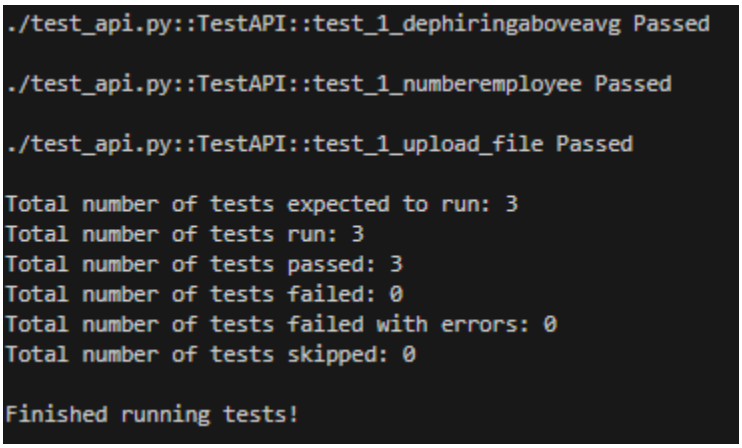
For correct coding practices, the flask application was done following PEP8 rules and sonar lint verifications implemented with the source-code editor.

Tests:

For the test, the library implemented was unittest implemented. In this case, the test was created and verified manually. For each method of the API a response status code 200 assert was created. This means that if the response of the API was different the successful, the test was going to fail. The tests are:

- **test_1_upload_file:** Test the '/upload_csv' endpoint by uploading a CSV file.
- **test_1_dephiringaboveavg:** Test the '/dephiringaboveavg' endpoint by sending a request with a specified year.
- **test_1_numberemployee:** Test the '/numberemployee' endpoint by sending a request with specific criteria.

In the following images, the execution of the unittest can be observed:



As a side note, these tests are only an example of a surface level creation.