**Transaction Reporting Application**

A Flask web application that interacts with the Financial House Reporting API to fetch and display transaction reports, transaction details, and client information. The application includes an Ajax-based fluent interface for querying transactions.

**Table of Contents**

* [Project Overview](#project-overview)
* [Project Requirements](#project-requirements)
* [Installation](#installation)
  + [Prerequisites](#prerequisites)
  + [Steps](#steps)
* [Usage](#usage)
  + [Running the Application Locally](#running-the-application-locally)
  + [Available Endpoints](#available-endpoints)
* [Testing](#testing)
  + [Running Tests](#running-tests)
  + [Test Coverage](#test-coverage)
* [Deployment](#deployment)
  + [Deploying to Heroku](#deploying-to-heroku)
  + [Environment Variables](#environment-variables)
* [Project Structure](#project-structure)
* [Dependencies](#dependencies)
* [Contributing](#contributing)
* [License](#license)

**Project Overview**

This application serves as a client for the Financial House Reporting API. It allows users to:

* Generate transaction reports within a specified date range.
* Retrieve a list of transactions based on various filters.
* View detailed information about a specific transaction.
* Obtain client information associated with a transaction.
* Use an interactive web interface for querying transactions using an Ajax-based fluent approach.

**Project Requirements**

Below is a list of the project requirements along with explanations of how each has been fulfilled.

1. **Transaction Report Generation**
   * **Requirement:** Create an endpoint that generates a transaction report based on a date range and optional merchant and acquirer IDs.
   * **Implementation:** Implemented the **/transactions/report** endpoint in **app.py** that accepts **fromDate, toDate, merchant, and acquirer** as query parameters. It uses the **TransactionReportClient** to fetch the report from the API.
2. **Transaction List with Filtering and Sorting**
   * **Requirement:** Provide an endpoint to list transactions with filtering options such as date range, status, operation, and more. Include sorting functionality.
   * **Implementation:** Created the **/transactions/list** endpoint in **app.py** that accepts various query parameters for filtering. Implemented filtering and sorting logic using list comprehensions and the **sorted()** function.
3. **Transaction Details Endpoint**
   * **Requirement:** Implement an endpoint to retrieve detailed information about a specific transaction using its ID.
   * **Implementation:** Added the **/transaction/<transaction\_id>** endpoint in app.py. It uses the **GetTransactionClient** to fetch transaction details from the API.
4. **Client Information Endpoint**
   * **Requirement:** Provide an endpoint to obtain client information associated with a transaction.
   * **Implementation:** Implemented the **/client/<transaction\_id>** endpoint in app.py. It uses the **GetClientInfoClient** to retrieve client information.
5. **Ajax-Based Fluent Interface**
   * **Requirement:** Use an Ajax-based fluent approach for the transaction query endpoint to enhance user experience.
   * **Implementation:** Created a new HTML template **transactions.html** in the **templates** directory. It includes a form for inputting search parameters and uses jQuery to send Ajax requests to the **/transactions/list/ajax endpoint**. Results are displayed dynamically without page reloads.
6. **Best Practices for Error Handling and Code Organization**
   * **Requirement:** Follow best practices for error handling and organize code appropriately.
   * **Implementation:**
     + Used try-except blocks in app.py to handle exceptions and return meaningful error messages.
     + Organized API client classes in the api\_clients package for modularity and reusability.
     + Used environment variables for sensitive information (credentials), managed with python-dotenv.
7. **Unit Testing**
   * **Requirement:** Write unit tests for the application.
   * **Implementation:** Created test cases in tests/test\_app.py using pytest. Tests cover the main endpoints and check for correct responses and error handling.
8. **Version Control with GitHub**
   * **Requirement:** Use GitHub (or GitLab/Bitbucket) for version control.
   * **Implementation:** The project is managed using Git and hosted on GitHub for version control and collaboration.
9. **Deployment to Heroku**
   * **Requirement:** Deploy the application to Heroku for public access.
   * **Implementation:** Configured the application for deployment on Heroku, including a Procfile, managing environment variables, and instructions for deployment provided in the [Deployment](#deployment) section.

**Installation**

**Prerequisites**

* **Python 3.6+**
* **pip** (Python package installer)
* **Virtual Environment** (recommended)

**Steps**

1. **Clone the repository**

* git clone https://github.com/your-username/your-repo-name.git
* cd your-repo-name

1. **Create a virtual environment (optional but recommended)**

* python3 -m venv env
  + source env/bin/activate # On Windows use `env\Scripts\activate`

1. **Install dependencies**

* pip install -r requirements.txt

1. **Set up environment variables**

* Create a .env file in the project root directory:
  + touch .env

Add the following to .env:

* MERCHANT\_EMAIL=demo@financialhouse.io
* MERCHANT\_PASSWORD=cjaiU8CV

**Note:** Replace these credentials with your own if necessary. **Do not commit .env to version control.**

**Usage**

**Running the Application Locally**

1. **Ensure the virtual environment is activated**

* source env/bin/activate # On Windows use `env\Scripts\activate`

1. **Run the Flask application**

* python app.py

1. **Access the application**

Open your web browser and navigate to <http://localhost:5000/>.

**Available Endpoints**

* **Home Page**
  + GET /
  + Renders the home page (index.html).
* **Transaction Report**
  + GET /transactions/report?fromDate=YYYY-MM-DD&toDate=YYYY-MM-DD
  + **Optional parameters**: merchant, acquirer.
  + **Example**:
    - GET /transactions/report?fromDate=2023-01-01&toDate=2023-12-31
* **Transaction List**
  + GET /transactions/list with query parameters as needed.
  + **Parameters** include fromDate, toDate, status, operation, etc.
  + **Example**:
    - GET /transactions/list?fromDate=2023-01-01&toDate=2023-12-31&status=APPROVED
* **Ajax-Based Transaction Query Interface**
  + GET /transactions
  + Provides an interactive interface for querying transactions.
* **Transaction Details**
  + GET /transaction/<transaction\_id>
  + Retrieves detailed information about a specific transaction.
  + **Example**:
    - GET /transaction/1-1444392550-1
* **Client Information**
  + GET /client/<transaction\_id>
  + Retrieves client information associated with a transaction.
  + **Example**:
    - GET /client/1-1444392550-1

**Testing**

**Running Tests**

The project includes tests using pytest.

1. **Ensure the virtual environment is activated**

* source env/bin/activate (On Windows use `env\Scripts\activate`)

1. **Run tests**

* pytest tests/

**Test Coverage**

To check test coverage, you can use coverage.py:

* pip install coverage
* coverage run -m pytest tests/
* coverage report

**Deployment**

**Deploying to Heroku**

The application is ready to be deployed on Heroku.

1. **Create a Heroku app**

* heroku create your-app-name

1. **Set environment variables on Heroku**

* heroku config:set MERCHANT\_EMAIL=your-email
* heroku config:set MERCHANT\_PASSWORD=your-password

1. **Push code to Heroku**

* git push heroku master

If you're using the main branch, use:

* git push heroku main

1. **Scale dynos**

* heroku ps:scale web=1

1. **Open the app**

* heroku open

**Environment Variables**

Ensure that you set the necessary environment variables both locally (in .env) and on Heroku.

* MERCHANT\_EMAIL: Your merchant email for API authentication.
* MERCHANT\_PASSWORD: Your merchant password for API authentication.

**Project Structure**

Homework-Python/

├── app.py

├── auth.py

├── Procfile

├── requirements.txt

├── .gitignore

├── .env (Should not be committed to version control)

├── templates/

│ ├── index.html Home page template

│ └── transactions.html => Ajax-based transaction query template

├── api\_clients/

│ ├── \_\_init\_\_.py

│ ├── base\_api\_client.py

│ ├── transaction\_report\_client.py

│ ├── transaction\_list\_client.py

│ ├── get\_transaction\_client.py

│ └── get\_client\_info\_client.py

└── tests/

├── \_\_init\_\_.py

└── test\_app.py

**Dependencies**

The application relies on several Python packages:

* **Flask**: Web framework.
* **requests**: For making HTTP requests.
* **gunicorn**: WSGI HTTP server for Unix.
* **python-dotenv**: For loading environment variables.
* **pytest**: For testing.
* **flask-cors**: (If CORS support is needed).
* **coverage**: (Optional, for checking test coverage).

All dependencies are listed in requirements.txt.

**Contributing**

Contributions are welcome! Please follow these steps:

1. **Fork the repository**
2. **Create a new branch**

* git checkout -b feature/your-feature-name

1. **Commit your changes**

* git commit -am 'Add some feature'

1. **Push to the branch**

* git push origin feature/your-feature-name

1. **Create a new Pull Request**