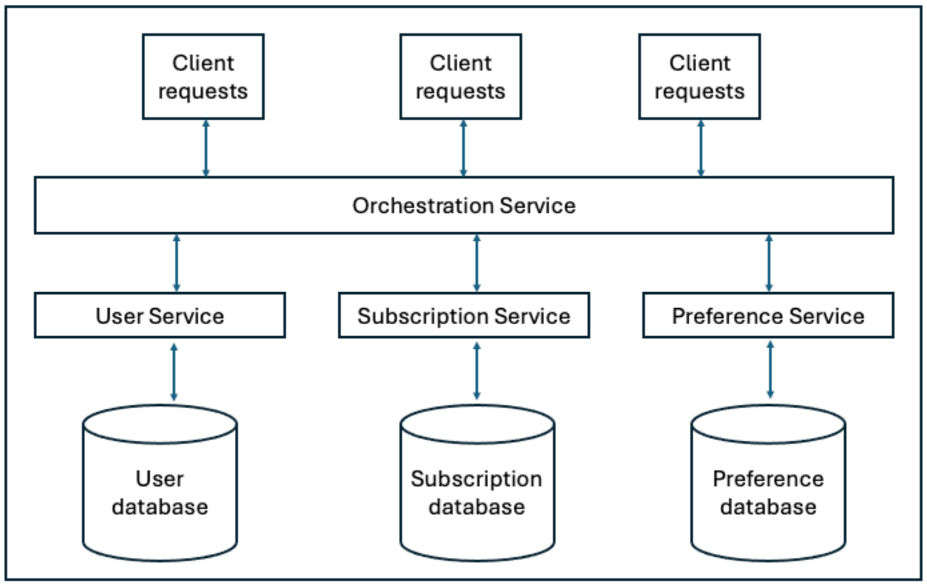
Orchestration Microservice Documentation

Introduction

This orchestration microservice is intended to demonstrate full-stack development skills, specifically: an understanding of microservices architecture and the ability to design a scalable system.

System Architecture

Overall architecture



A centralized orchestration model was used, mainly to simplify the coordination, management and error handling of the workflows.

Tech Stack

- Python 3.11.4
- Flask
- MySql

Installation

You can clone the project files by running `git clone https://github.com/CodeNameJacks/Orchestrate.git` in the command line.

To set up and run locally:

1. In the terminal command line use the command `mkdir <name of folder>` to create the following three folders in your project directory: `user, subscription, preference` and `orchestrationService.`

Perform steps 2 - 8 in each folder.

2. Install Python 3.x
3. Create a virtual Python environment in the folder by entering `python3 -m venv venv`
4. To activate the virtual environment run: `. venv/bin/activate`
   Now that virtual Python environment is running, install the following:
5. Run the `requirements.txt` file to install the dependencies. Should the dependencies fail to load you can enter them manually be entering `pip install flask python-dotenv requests` in the command line.
6. Also install the mySQL connector by running `pip install mysql-connector-python` in the command line.
7. You need to create a file called `.flaskenv` in each folder. Place this in the root directory. In that file, place the following on their own line: `FLASK_APP=<name of the api.py for each folder>`, `FLASK_DEBUG=1`, and `FLASK_RUN_PORT=<port number for the service>`.
8. In the root directory create a file called `.env`. Place your non-Flask environment variables in this file.

Running Application

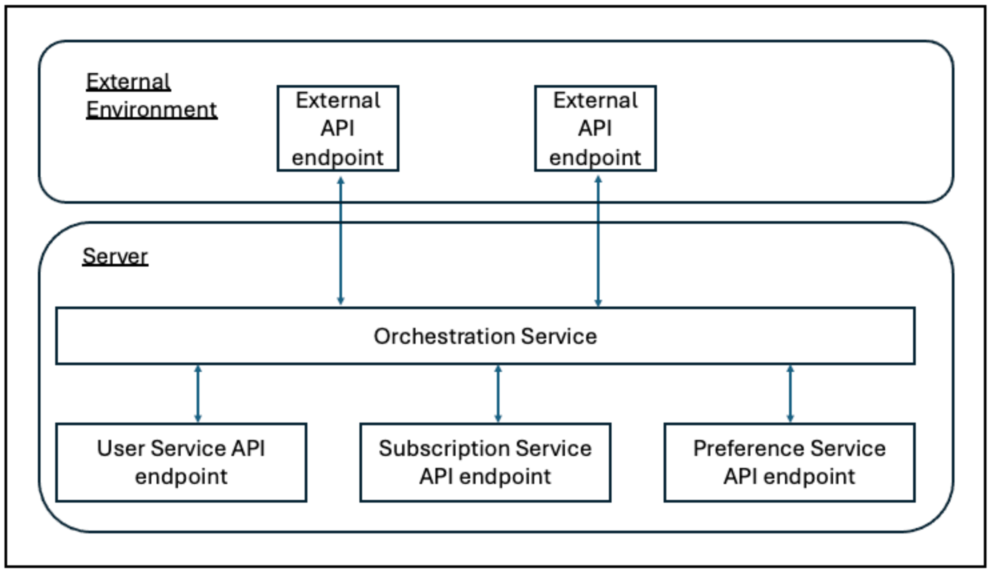
In the terminal command line `cd` to each folder one by one and run the command `flask run`. Your servers should now be running on their respective ports.

You can test each server to ensure it is running by opening a web browser and entering `localhost:<port number>/api/test` in the url. You should get the following response `<Name of service> service is working and backend is running` for the respective service.

To use the API enter the API endpoint in the web browser url. The response should show in the browser.

API

Overall API architecture


External Environment
External API endpoint
External API endpoint
Server
Orchestration Service
User Service API endpoint
Subscription Service API endpoint
Preference Service API endpoint

The API has 5 endpoints. 2 are external and accessible via the web browser, 3 are internal and accessible from the OrchestrationService. For those purposes of this demo and it being locally hosted, the use of security API-keys have not been implemented. So, while absent the API-keys, the 3 internal endpoints would be accessible externally, for demonstration purposes, we will assume that the internal endpoints are only accessible by the OrchestrationService.

External endpoints

1. http://localhost:5006/user/subscription/{id}

This endpoint retrieves the user information along with their active subscription information.

{id} is the user id being queried and can be a value between 1 and 1000.

2. http://localhost:5006/user/details/{id}
This endpoint retrieves the user information their subscription status information and preferences for all products.
{id} is the user id being queried and can be a value between 1 and 1000.

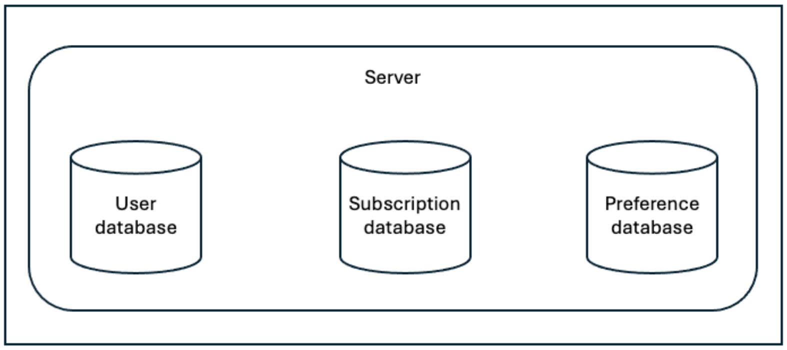
<u>Internal endpoints</u>

1. http://localhost:5000/user/{id}
This endpoint retrieves the user information based on user id.
{id} is the user id being queried and can be a value between 1 and 1000.

2. http://localhost:5002/subscription/{id}
This endpoint retrieves the user's subscription information based on user id.
{id} is the user id being queried and can be a value between 1 and 1000.

3. http://localhost:5004/preference/{id}
This endpoint retrieves the user's preferences based on user id.
{id} is the user id being queried and can be a value between 1 and 1000.

Database Design

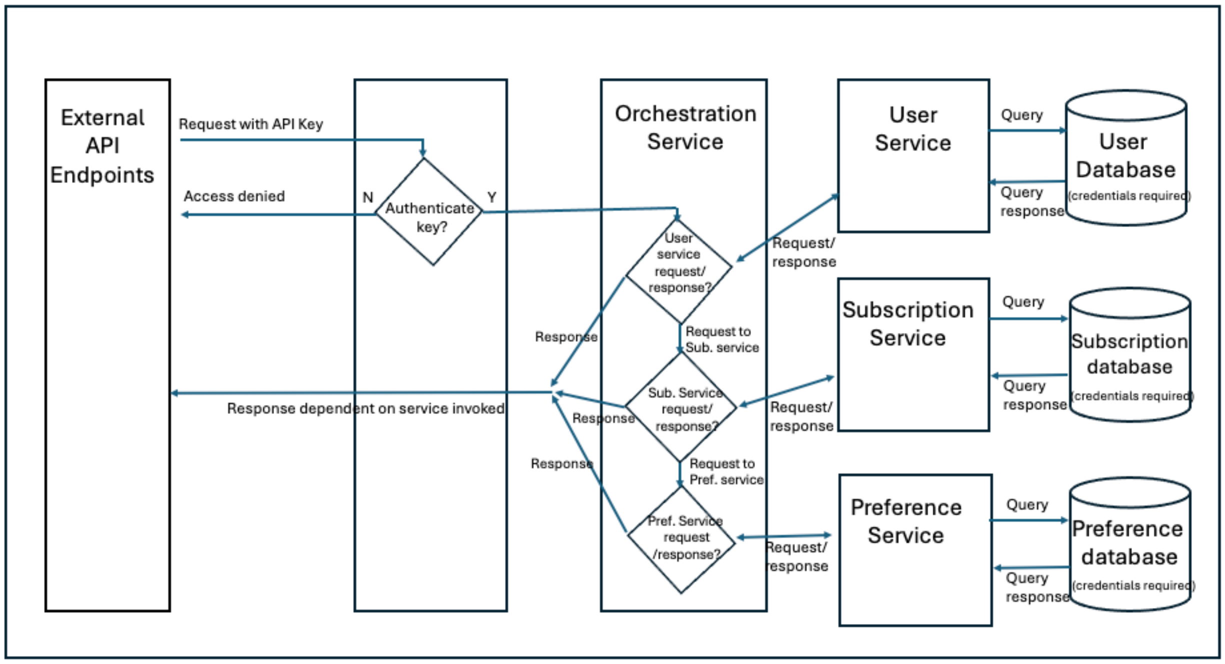


Dummy data has been manufactured via the Mockaroo API. The data is hosted on a remote database server (Aiven.io). The data is contained in databases for each respective service. The data base credentials are contained in the .env file (note this file is not in the Github repository).

.sql files containing the dummy data are found in the sql directory.

Security

While an API-Key authentication has not been implemented. The flowing chart shows the flow of data, where credentials are required and includes an API-key authentication.



Deployment

As this is a development environment, deployment to staging and/production is not covered. Please refer to the Installation and Running the Application sections for instructions on local set up.