TeamNotFound

Link to the video: COMP30220(2023-2024) Group Project

Nan Wu	Yiming Zhao	Weijiong You	Xinyu Dong
18210184	19353281	19206208	19206377

Synopsis:

This project aims to create an integrated central travel planning website. The project can provide the user with some quoted prices from different travel agencies/websites. The project requires the user to enter some basic requirements of the ideal trip, i.e. start and end date, city of travel, type of travel (all inclusive, semi-inclusive or just transport and hotel). The website then collects prices from related travel agencies/websites and returns these prices to the user.

Technology Stack

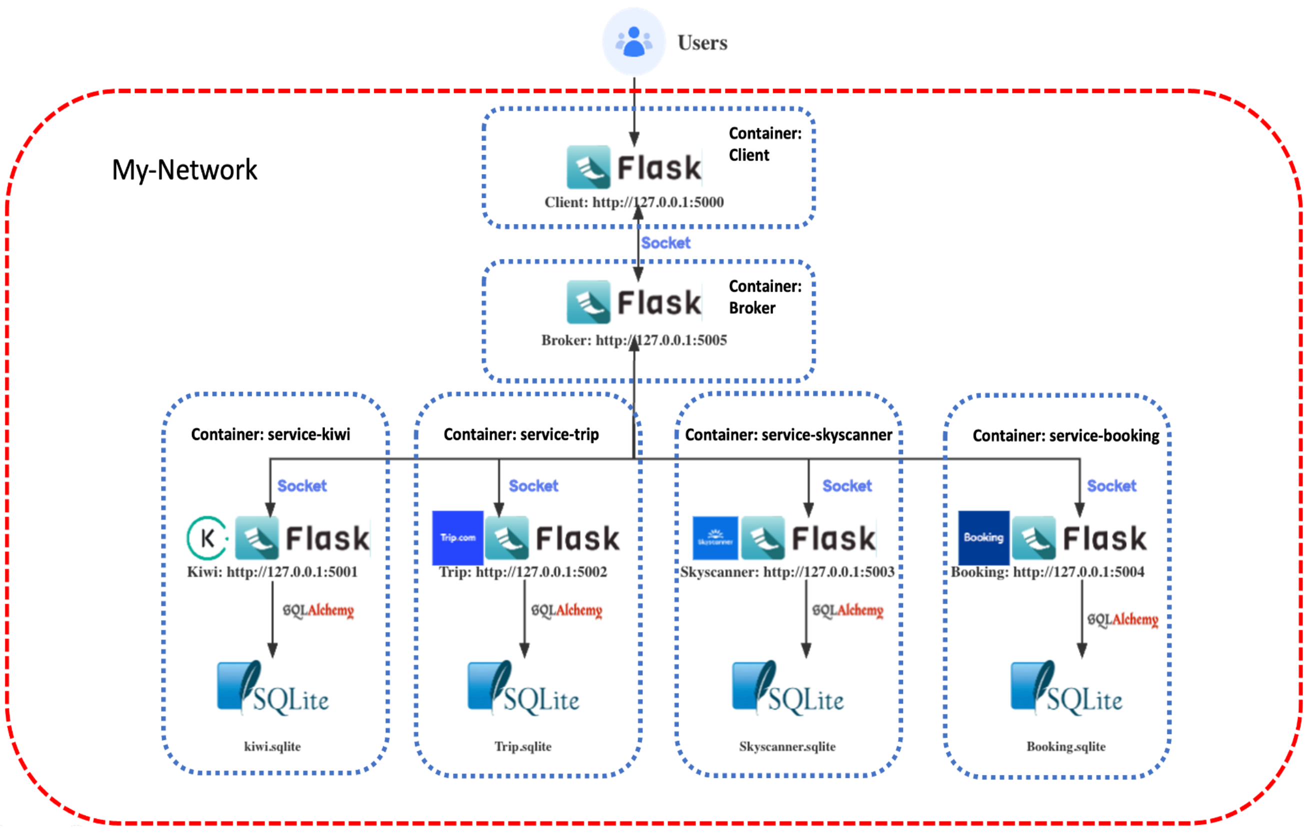
- **Python**: Python is the main programming language used in this project, this is to work with the Flask framework.
- **Flask**: Flask is a python web framework, this framework can easily build the web service with backend and frontend.
- **Websockets**: Websockets used for data transfer and interaction between services and the client. It can build the bidirectional communication channel between each two flask applications, and the data transfer in one channel won't interfere with the other transfer in another channel.
- **SQLite**: Used as the backend database, can be easily connected to the flask framework and is suitable for the project.
- **HTML+CSS**: HTML and CSS used to layout the frontend page.
- **SQLAlchemy**: SQLAlchemy is a database toolkit for Python, it provides a set of tools and utilities to access and manage the SQL database with Python.
- **Javascript**: Javascript is used to pass the parameters between front-end and back-end, also for design and layout of the front-end website.
- **Docker**: Docker used for containerisation of the project, 'docker compose up' used to run the project without using multiple commands to run each service.

System Overview

Describe the main components of your system.

The system includes the front-end page (client), a broker, four services (serviceKiwi, serviceBooking, serviceTrip, serviceSkyscanner).

Include your system architecture diagram in this section.



Explain how your system works based on the diagram.

The Client end which directly connects with users asks the user to select/enter a number of options, including start date, end date, city, type of travel and promo code if applicable. Then the client takes all the input parameters from the user to the broker. And the broker links with four different services and sends the input to them, each service gets the same input parameters but they act differently and distributed, it connects to its own backend database and applies an algorithm and calculates the corresponding price. Finally, the prices from each service are sent back to the broker and the broker will send them together to the client and presented to the user. The communication in all systems is based on the Websocket to build a bidirectional communication channel between each pair of flask applications, which allows multiple communication processes running simultaneously. SQLAlchemy is used to integrate the SQLite database so that we can access and process the data more efficiently.

Explain how your system is designed to support scalability and fault tolerance.

As the services are distributed and they're containerised in different containers, each service runs individually and isolated, which means that a service failure cannot affect other services, this feature greatly supports the stability and fault tolerance of the whole system. The broker can easily connect to more services by adding a service with different urls and containers, and each service connects to a scalable database that can easily add more services/data by using the interfaces provided by SQLAlchemy, showing great scalability.

Contributions

- **Nan Wu**: Responsible for the Kiwi service, the front-end and its connection to broker and services, the promo code system, and fixed the connection issue with containerisation.
- **Yiming Zhao**: Responsible for the Trip service, improvement of date selection, optimise the CSS layout of the front-end website and several bug fixes (error display, promo code bug).

- **Weijiong You**: Responsible for the Booking service, the basic architecture of the distributed system, database connection, and containerisation using Docker.
- **Xinyu Dong**: Responsible for the SkyScanner service, the initial user interaction section (user inputs), and several bug fixes.

Reflections

What were the key challenges you have faced in completing the project? How did you overcome them?

Synchronization and concurrency: Managing concurrent connections and ensuring data synchronization across multiple services can be challenging. This is solved by using Flask to make them run in different ports, and using Websockets to make them communicate with each other so that they achieve synchronization and concurrency.

Database management: In this distributed system, we use SQLite as the backend database for each service. Accessing and managing the database by SQL query may be easier, but integrating it with the Python project can be challenging. To overcome this, we introduce the Flask SQLAlchemy to simplify the query sets and manage the database connections for each Flask application.

Connections between containers: In the process of containerisation, we find that the data can not be transferred between different containers. To overcome this, we just put all containers into the same network and then the system works.

What would you have done differently if you could start again?

We can design more robust error handling and input validation. For example, when we design the client's website, sometimes we may get an invalid input, such as the start date is after the end date, a proper error handling and promotion can help and notify our user to avoid this situation.

Also, we can further extend our distributed system with more microservices, which requires a complete plan ahead. As a travel booking system, we can connect to different transport and accommodation providers, and many other services such as restaurant recommendations, ticket price comparison can also be added to our distributed system.

What have you learnt about the technologies you have used? Benefits? Limitations?

Flask: The Flask framework has great simplicity and scalability. The developer can quickly start the project, it provides a good setup for frontend and backend and can be very useful for building a web application. It also provides a good and handy way to connect the database.

Docker: Docker is a great tool to containerize the project. It can easily package the project and run on any system without considering the current environments. It can easily isolate different parts/dependencies of the project and add more. There is also a limitation with Docker. It adds complexity to the project, the developer needs a bit of extra work to write and use it, it also requires the developer to learn it as using Docker may not be an easy thing for the learner. The resource usage and security of Docker can also be considered.

WebSockets: WebSockets is a great tool for real-time data transfer between a client and a server. Unlike HTTP requests a connection every time and only provides a one-way transfer, it always provides a two-way connection. It also transfers data with very low latency and is a good tool for live data transfers. Some limitations are that it is not as powerful as HTTP and may be restricted by browser type or firewall. It also increases the load on the service as the connection will not be closed.