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

List of the main distribution technologies you will use

- *Tech 1…: What is it used for?*
- *Tech 2…: What is it used for?*

Highlight why you used the chosen set of technologies and what was it about each technology that made you want to use it.

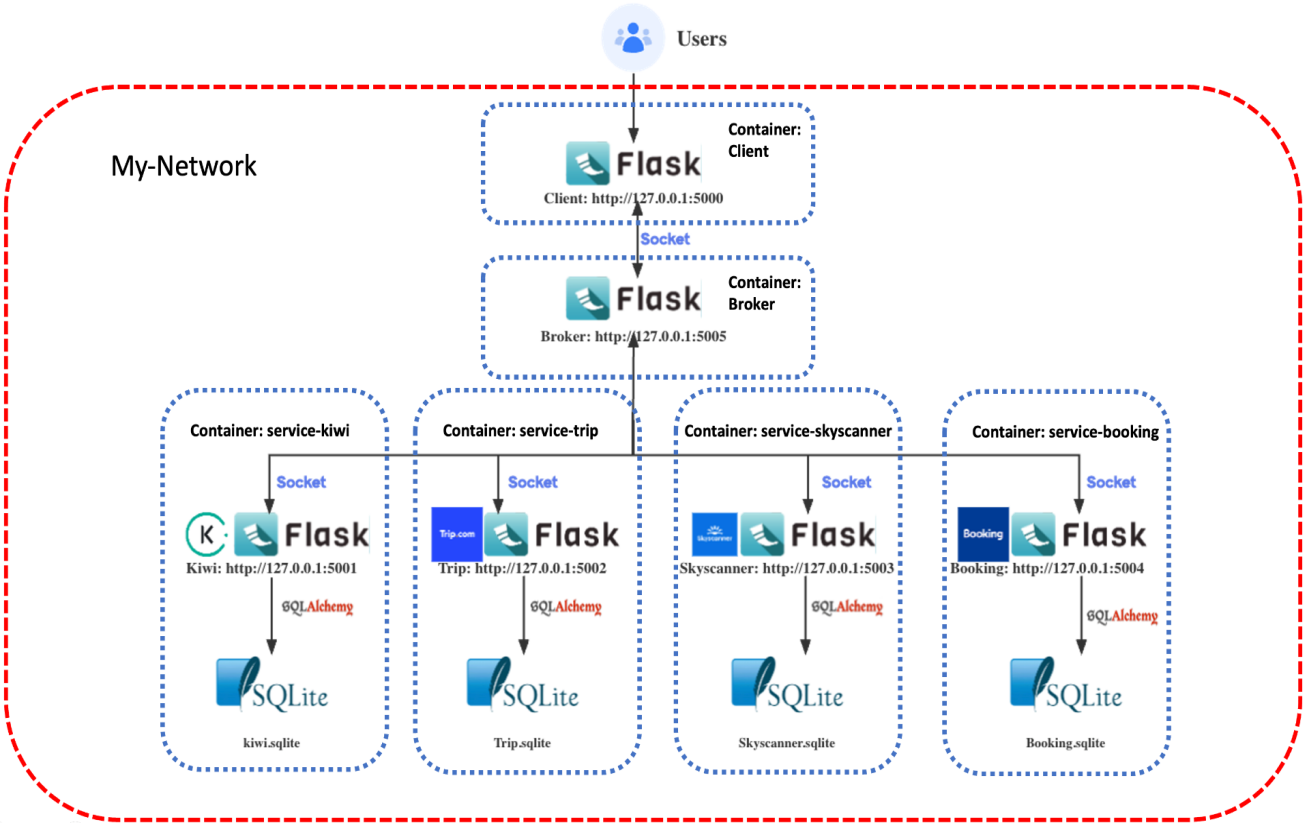
- **Python**: Python is the main programming language used in this project, this is to work with the Flask framework.
- **Flask**: Flask is a python web framework, this framework can easily build the web service with backend and frontend.
- **Websockets**: Websockets used for data transfer and interaction 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**: provides a set of tools and utilities for working with databases in a flexible and efficient manner.
- **Javascript**: Javascript 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 process running simultaneously. SQLAlchemy is used to integrate the sqlite management 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

Provide a sub section for each team member that describes their contribution to the project. Descriptions should be short and to the point.

- **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?

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

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

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 makes them running in different ports, and using the Websoket to make them can communicate with each other so that achieve the synchronization and concurrency.

Sqlite Management: In our distributed system, we access the sqlite databases frequently and the conditions is not simple. Therefore, it's complex for developers to access the database using the SQL sentences. To overcome this, we introduce the flask-SQLAlchemy to simplify the the query sentences 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 systems 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 client issues a 'get_input' event containing invalid or incomplete data, the server currently does not handle this situation.

Also, we can further extend our distributed system with more microservices. As a travel agency, we can connects to different airlines to find the cheapest flights, we also can connects to different accomodation providers to give user the most comfortable residence, also many other services like restaurant recommendations, tickets prices comparison can also be added in our distributed system.

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

Flask(Docker): Flask is a Python web framework, while Docker is a platform for containerization. Here are some benefits and limitations of using these two technologies together.

Benefits:

Simplicity: Flask is well known for its simplicity and quick setup, which makes it a great choice for developing web applications. While docker simplifies deployment by packaging the application with its dependencies in a container, which allows it to run on any system that has Docker installed.

Scalability: Docker allows for easy scaling of applications. It can quickly start multiple instances of the Flask application in separate containers and distribute traffic between them.

Isolation: Docker containers isolate Flask application and its dependencies from the rest of the system. This allows multiple different applications with conflicting dependencies running on the same server without any issues.

Consistency: Docker ensures that the application runs in the same environment regardless of where it's deployed. This eliminates the common issue of the application working on one machine but not on another due to differences in the environment.

Development and Deployment Efficiency: Docker can mirror production environments locally, improving the development process. It's also easier to update versions and push application changes using Docker containers.

Limitation:

Complexity: While Docker can simplify deployment, it also adds a layer of complexity to your project. Which means that developers need to understand how Docker works, how to write Dockerfiles, and how to manage containers.

Resource Usage: Each Docker container runs a separate instance of the application and its dependencies. If many containers are running on a single machine, it will results in more resources usage than running the applications directly on the host system.

Security: Docker containers do provide some level of isolation, but they are not as isolated as virtual machines. A vulnerability in the Docker daemon could potentially allow an attacker to gain access to the host system.

Learning Curve: For developers unfamiliar with Docker, there can be a steep learning curve. Concepts like images, containers, volumes, and Dockerfiles can take time to understand.

Websocket: WebSockets technology enables real-time, two-way communication between a client and a server. Here are some benefits and limitations of using WebSocket technology.

Benefits:

Real-Time Communication: WebSockets provide a persistent connection for real-time data transfer. Which is an advantage over HTTP, where each request requires a new connection to be established.

Two-Way Communication: Unlike HTTP, which is unidirectional (from client to server), WebSockets are bidirectional. This means data can be sent back-and-forth between a client and server at the same time.

Efficiency: Once a WebSocket connection is established, data can be transferred with very low latency since there's no need to reestablish one connection for each transfer. Also, WebSockets use less bandwidth as they remove the overhead of HTTP headers for each message.

Live Content: WebSockets are great for applications that require live content, such as online chat applications, gaming, and live news updates.

Limitation:

Browser Compatibility: Not all browsers (especially older ones) support WebSockets. Therefore, developers will have to implement fallback mechanisms.

Firewall and Proxy Issues: Some firewalls and proxy servers are not configured to allow WebSocket connections, which can prevent some users from being able to use WebSocket-based applications.

Lack of HTTP Features: WebSockets don't have some built-in features of HTTP, such as status codes and headers, which can be useful for communicating metadata about the request or response.

Increased Server Load: As WebSockets keep the connection open, a high number of open connections can lead to high memory usage on the server.