Iter - A Web Based Travel Planner

Anirudh Makhana
Kanokpon Nagavajara
Phurit Warapattanapong
Pitchakorn Pichetnarumit

Bachelor of Engineering in Software Engineering
Department of Computer Engineering
Faculty of Engineering
King Mongkut's Institute of Technology Ladkrabang
Academic Year 2022

Thesis - Academic Year 2022

Bachelor of Engineering in Software Engineering

Department of Computer Engineering, School of Engineering

King Mongkut's Institute of Technology Ladkrabang

Title : Iter - Web Based Travel Planner

Authors

1	Anirudh Makhana	Student ID: 62011088
2	Kanokpon Nagavajara	Student ID: 62011131
2	Phurit Warapattanapong	Student ID: 62011208
3	Pitchakorn Pichetnaruemit	Student ID: 62011213

Approved for submission

Natthapong J.

.....

(Dr. Natthapong Jungteerapanich)

Advisor

Date: May 27, 2023

Acknowledgement

We would like to express our heartfelt gratitude to all those who have contributed to the successful completion of this thesis. First and foremost, we are deeply indebted to our project advisor, Dr. Natthapong Jungteerapanich, for his invaluable guidance, support, and encouragement throughout the entire research process.

We would also like to extend our sincere appreciation to the following faculty members at the Software Engineering Department of the King Mongkut's Institute of Technology Ladkrabang: Dr. Isara, Dr. Pipat, Dr. Veera, Dr. Ukrit, Dr. Chaiwat, and Dr. Apinun, for their valuable insights, feedback, and suggestions, which have greatly enriched our thesis work.

In addition, we would like to thank the Tourism Authority of Thailand for providing us with the necessary data and information for our research. We acknowledge their valuable contribution and express our sincere appreciation for their support.

Furthermore, we extend our gratitude to our colleagues for their support and encouragement, and for the stimulating discussions and debates that have helped shape our ideas and perspectives. Finally, we express our heartfelt thanks to our families and friends for their unwavering support, understanding, and patience throughout this journey.

Abstract

The proposed project aims to develop a web-based application that simplifies the process of planning a trip to Thailand called Iter. The application will assist travelers in creating personalized and customizable itineraries, as well as motivate them to travel more in Thailand by allowing them to share their own travel guides and experiences. Leveraging data from the Tourism Authority of Thailand, Iter utilizes advanced technologies such as recommender systems, Vehicle Routing Problem (VRP) optimization, and Ant Colony Optimization (ACO) algorithms. The platform will also provide a way for a group of travelers to plan a trip together more harmoniously and conveniently. Additionally, the application will encourage travelers to explore more unseen locations in Thailand. The overall goal of the project is to make traveling to Thailand more convenient, enjoyable, and rewarding for everyone.

Contents

List of Figures

Listings

List of Tables

Chapter 1

Introduction

1.1 Motivation

Planning a trip can be a tedious and complicated process, for the fact that, interesting destination at an unfamiliar place can be difficult to discover without some familiarity of that area. Additionally, communication with friends and co-travelers is one of the obstacles while planning a trip with the current communication platform. Lastly, traveling can be expensive and it might not be affordable for some people. We aim to develop a web-based application that elevates trip planning experience along with financial solutions that are beneficial for all parties.
Our application will create a personalized travel itinerary based on the user's specifications using machine learning. After users could fill out their travel information and personal preferences, our application would suggest travel destinations, accommodations, restaurants, and transportation based. Those plans could further be customized and edited by the users. Users also could add co-travelers, co-travelers would be able to view and edit the itinerary also. Co-travelers will also be able to vote for the destinations. And to assist users financially, our application will provide questlike activities that users could participate in to receive coupons or discounts in their future travels. However, these rewards will be expired in six months, thus incentivizing users to use the reward and travel more often. After the user completed their trips, users could also rate and review the places they visited.

Overall, our application objective is to make traveling more convenient and simple for everyone by providing a personalized and customizable plan and activ-

ities to financially assist users.

1.2 Objectives

The objectives of this project are as follows:

1. Assist travelers in creating a personalized and customizable itinerary.
2. Motivate people to travel more in Thailand by allowing them to share their own travel guides and experiences
3. Provide a platform for a group of travelers to plan a trip together harmoniously and more conveniently
4. Encourage travelers to explore more unseen locations

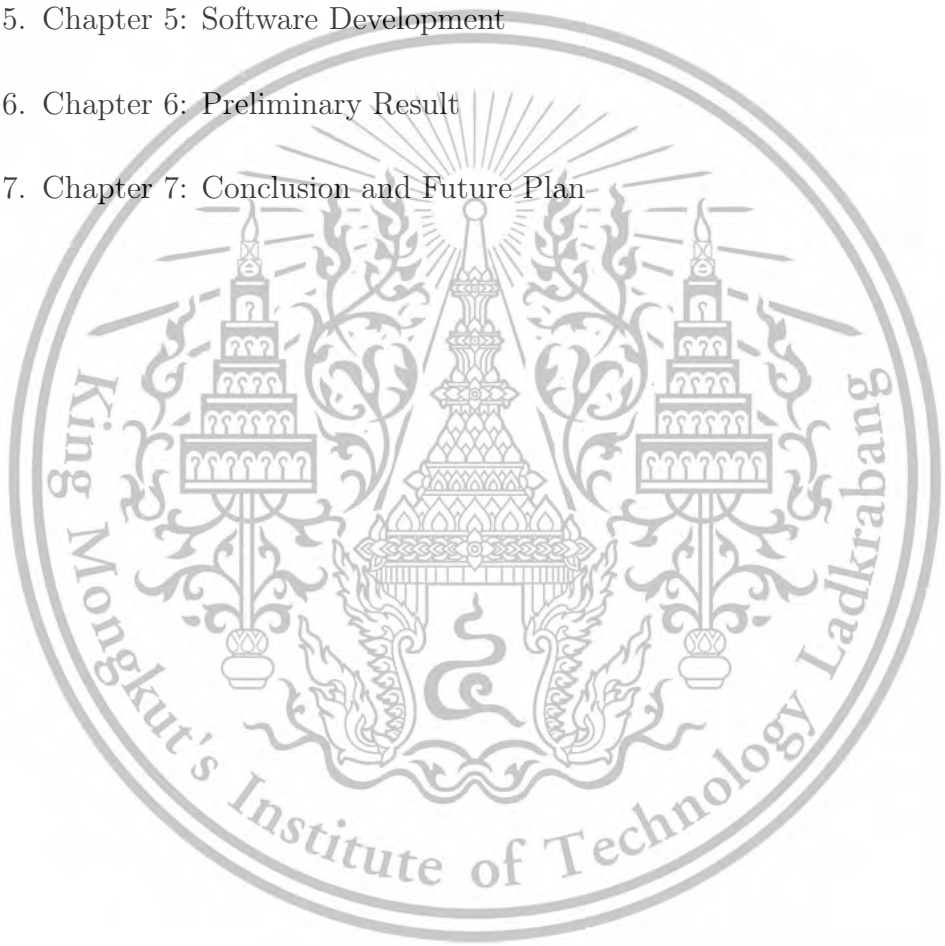
1.3 Scope of Work

The scope of work is listed below:

1. The scope includes developing a responsive web-application that provides assistance when traveling in Thailand.
2. The system does not include booking of accommodations, flights, rental cars, attractions, or restaurants.
3. Places data is obtained from the Tourist Authority of Thailand server via HTTP endpoint.
4. The system should include suggestions and details for accommodations, attractions, and restaurants.
5. The system should generate an acceptable optimized travel route by solving VRP using metaheuristic methods.
6. The system does not provide integrated GPS or other mapping details.
7. The recommender system generates correct number of places with decent distribution among the attraction types and activities.

1.4 Thesis Structure

1. Chapter 1: Introduction
2. Chapter 2: Related Works
3. Chapter 3: Background Knowledge
4. Chapter 4: Requirement Analysis
5. Chapter 5: Software Development
6. Chapter 6: Preliminary Result
7. Chapter 7: Conclusion and Future Plan



Chapter 2

Related Works

2.1 Similar Applications

2.1.1 Amazing Thailand

"Amazing Thailand" app by Tourism Authority of Thailand (TAT) as seen in [11] offers travel-related information, such as guides to destinations, events and festivals, shopping, Thai food, and other information. The app serves as your travel assistance, where users can find places to travel, routes, accommodation, restaurants, and shopping sites. The application allows users to locate the data, travel advice, and destination specifics. Amazing Thailand also provides several full day trips recommendations, and allows users to create their trip by adding destinations as well.

2.1.2 Nostra Map

Nostra Map demonstrated in [16] allows searching for places location with categories, find a route with car, motorbike, walking directions and recalculate the route from current location. It acts like a GPS to give directions to destinations. The application also allows users to add favorite points and favorite routes. The category provided includes buildings, restaurants, shopping malls, tourist attractions, ATMs, gas stations, etc.

2.1.3 Inspirock

Inspirock as seen in [15] is a travel itinerary website, which builds daily itineraries for users using AI and local knowledge. Itineraries for more than 80,000 locations throughout the world. Inspirock offers features such as generating a customizable day-to-day travel itinerary plan creation by inputting destination, dates, and a few other preferences, accommodation suggestions where users can view accommodation options and book, activity suggestions that is nearby the destination, tour suggestions including boat tours, wine tours, city walks, rental car and flight deals, booking of activity, tour, hotel, flight, car rental, travel checklist, and available purchasable travel insurance.

2.1.4 Wanderlog

Wanderlog in [3] is a travel app that allows users to create itineraries. Wanderlog offers features such as setting spending limits, managing hotel and flight bookings, exploring destinations on a map, and interacting with friends to plan all types of journeys, including road trips and group travel. Wanderlog allows users to share travel guides with other travelers and view various accommodations, activities, and flights. The application also allows users to plan your route (like Roadtrippers with timings and distances between locations.

2.1.5 TripAdvisor: Plan Book Trips

Tripadvisor as demonstrated in [22] connects people, passions, and places as a provider of travel advice. Trip Advisor allows users to plan and book trips and use it during the road. It also allows users to view reviews from other travelers to find out where to stay, what to do, and where to eat. The application allows users to explore places nearby, book accommodations at a discount, and reserve restaurant tables.

2.2 Feature Comparison of Similar Applications

Table 2.1: Feature Comparison

Feature/ Platform	**Amazing Thailand**	**Nostra Map**	**Inspirock**	**Wanderlog**	**Our Project**
Platform	Application	Application	Web-application	Application, web-application	Web-Application
Type of location	Accommodation, Restaurant, Attraction, Shopping	Accommodation, Restaurant, Attraction, Shopping	Accommodation, Restaurant, Attraction, Shopping	Accommodation, Restaurant, Attraction, Shopping	Accommodation, Restaurant, Attraction
Provide Itinerary	Yes	No	Yes	Yes	Yes
Provide Routes	Yes	Yes	Yes	Yes	Yes
Personalized Itinerary	No	No	Yes	No	Yes
Supports Booking	No	No	Yes	No	No
Supports GPS	No	Yes	No	No	No
Supports Collaboration	No	No	Yes	Yes	Yes
Weather Forecast	No	No	Yes	No	No

Chapter 3

Background Knowledge

3.1 Vehicle Routing Problem

The vehicle routing problem (VRP) is a combinatorial optimization problem, which is a subfield of mathematical optimization to find an optimal object from a finite set of objects, and an integer programming problem that involves finding an optimal set of routes for a fleet of vehicles to serve a set of customers. Figure 3.1 shows an example of a vehicle routing problem, different colors edges represent different routes of different vehicles.

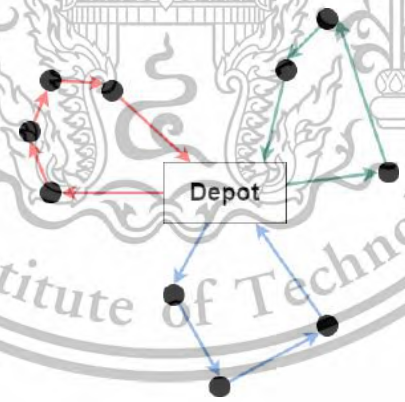


Figure 3.1: Vehicle routing problem

Finding the optimal solution to a vehicle routing problem is NP-hard[21], meaning that it is impossible to get to a final answer, so we normally search for an optimal answer instead. The vehicle routing problems can be solved by using some graph strategies to generate an initial solution and then using local search to find the optimal solution.

3.1.1 Vehicle Routing Problem with Time Windows

There are plenty of vehicle routing problem variants which are vehicle routing problem with an additional constraint. The vehicle routing problem with time windows (VRPTW) is a variant of the VRP in which each customer is only available during specific time windows.

3.1.2 Solution Methods

Since finding the optimal solution for a vehicle routing problem is NP-hard, there are two main methods for solving the problem, exact solution methods and metaheuristics.

3.1.2.1 Exact Solution Methods

The exact solution methods guarantee the optimal solution of the VRP but the algorithm required exponential time to discover the optimal solution. The VRP can be formulated using the following approaches:

- **Two index vehicle flow formulations** are network flow formulations that assign integer variables, counting the number of times that the edge is traversed by a vehicle, to each arc. The potential of this method is limited to simple problems such as the basic VRP and the VRP with backhaul constraints.
- **Commodity flow formulations** assign integer variables to the arcs representing the flow of commodities along the paths of the vehicles.
- **Set partitioning problem** has binary variables assigned to each different feasible route. Then find the set of feasible routes with minimum cost and satisfy the constraints.

3.1.2.2 Metaheuristics

A metaheuristic generates an initial solution and then provides some guidelines for searching for the optimized solution, e.g., Genetic algorithms, Tabu search, Simulated annealing and Adaptive Large Neighborhood Search (ALNS). The meta-

heuristic methods do not guarantee the optimal solution, but they require less time compared to the exact solution methods.

3.2 Ant Colony Optimization

Ant colony optimization (ACO) is an optimization algorithm that simulated ants' foraging behavior by studying how they find the shortest path between their nest and food source, without any central coordination mechanism. Generally, ants have an initial random activity pattern in the search for a food source. When the food source is located, the activity patterns become more organized, which means other ants will follow the same path, the shortest path. The autocatalytic behavior is when forager ants lay pheromones along followed trails. Paths with a larger pheromone concentration are more attractive to other ants. The more ants follow a specific trail, the more pheromone is deposited. Thus, the path is more attractive and attracts more ants to follow that path. The indirect communication of ants is called stigmergy, which referred to the behavior of ants that modify their environment to influence the behavior of other ants by laying pheromones.[4] ACO simulates the autocatalytic behavior and stigmergy by using the probabilistic technique for solving computational problems and finding the optimal solution for graph problems. Algorithm 1 shows the pseudocode of a simple ant colony optimization algorithm.

3.2.1 Path Construction

For each iteration, each ant incrementally constructs a path, which is a solution for the graph problem. At each step in constructing a path, the ant will consider traveling to the next node based on the transition probability of the remaining feasible nodes. The following equation shows the transition probability for selecting the next node $j \in N_i^k$ of ant k at node i, where N_i^k is the set of feasible nodes connected to node i with respect to ant k.[5]

Algorithm 1 Pseudocode for ant colony optimization

```
τ_ij ← small random value
t ← 0
Place n_k ants on the origin node
while stopping condition is True do
    for each ant k do
        Construct path x^k(t) to the destination node ▷ Influenced by amount of pheromone
    end for
    for each link (i, j) of the graph do
        Pheromone evaporation                          ▷ Update pheromone
    end for
    for each ant k do
        Update τ_ij                                    ▷ Each ant update pheromone of its path
    end for
    t ← t + 1
end while
```

$$p_{ij}^{k}(t) = \begin{cases} \frac{\tau_{ij}^{\alpha}(t)}{\sum_{j \in \mathcal{N}_i^k} \tau_{ij}^{\alpha}(t)}, & \text{if } j \in N_i^k \\ 0 & \text{if } j \notin N_i^k \end{cases}$$

3.2.2 Pheromone Evaporation

Pheromone evaporation is a process to enhance the algorithm's capacity for adaptability. The pheromone trails from the previous environment will not match the new environment when a dynamic change occurs, especially if the two environments are not identical.[13] Additionally, the pheromone evaporation process is also used to prevent premature convergence by controlling the pheromone level of the path as shown in the following equation, where $p \in [0, 1]$.

$$\tau_{ij}(t) = (1 - \rho)\tau_{ij}(t) \tag{3.1}$$

The rate a which pheromones evaporate is specified by ρ. The value of ρ controls the influence of search history by causing ants to forget their previous decisions. For large values of ρ, the pheromone evaporates rapidly, which means ants will always forget their previous decisions. Thus, ants will lay pheromone and update it every time. However, small values of ρ result in slower evaporation

rates. Therefore, large values of ρ implies more exploration in a search process.[6]

3.2.3 Pheromone Update

The key process of the ACO is pheromone update since it is the process of influencing ants to select the optimal path for the graph. The pheromone update aims to the pheromone values associated with good solutions, and decrease the pheromone values that are associated with bad solutions.[2] The following equation represents how the pheromone value of the edge (i, j) is updated in iteration $t + 1$ with respect to ant k.

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \sum_{k=1}^{n} {}_k\Delta\tau_{ij}^{k}(t) \tag{3.2}$$

where:

$$\Delta\tau_{ij}^{k}(t) = \frac{1}{L^k(t)} \tag{3.3}$$

$L^k(t)$ = length of the path constructed by ant k at time step t

n_k = number of ants

3.3 Recommendation System

Recommendation System as a name suggested is a system which predicts the user rating on an items. There are many ways to implement the recommendation system, however it can be categorized into two general groups.

- **Content-based filtering** which recommend item based on the item profile for instance, if user watched lots of Sci-fi movies then the system will recommend movies based on the genre of Sci-fi.

- **Collaborative filtering** items are recommended based on metrics of similarity between users and/or items. In collaborative filtering items that are recommended to the user are also preferred by similar users. This type of recommendation system can be insufficient on their own and problem like

cold-start might occur, yet there are some new algorithms have proven helpful for this type of recommendation system.

There are various use cases for recommendation system. An examples are product recommendation, product recommendation can be found commonly in the internet as almost every online store are using it to show the items that are most likely be bought by the user. [10]

3.3.1 Item profiles

Item profile is one of the crucial part for content-based recommendation system. Item profile can be describe by a record or records showing crucial information about that item for instance, consider a feature for attraction the profile of it would be types of the attraction, and activities of that place. [10]

3.3.2 Similarity Measure

A similarity measure is a way of measuring how similar two samples of data are to one another. This implies that an item is similar if the distance between two data points is small, and vice versa. To measure the distance between two items, a distance function is needed. There are many distance functions to choose from. Well-known example of this is Euclidean distance, Manhattan distance, and Minkowski distance. [1]

3.4 Data Pipeline

A data pipeline is a technique for transferring raw data from different data sources to a data store, such as a data lake or data warehouse for further usage. In most cases, data is processed before it enters a data repository. The data pipeline consists of three core steps. [24]

- **Data Ingestion** is a process of collecting data from various sources.
- **Data Transformation** is where a series of jobs are executed in order to process the data into a required format for the destination data storage.

- **Data Storage** is where a transformed data is stored for further usage.

A notable use cases for the data pipeline are:

- **Exploratory Data Analysis(EDA)**: Data scientists do exploratory data analysis to study data sets, and identify their important characteristics. By figuring out how to change data sources to get the results they need, data scientists can more easily discover patterns, spot anomalies, test theories, or confirm presumptions.

- **Machine Learning**: The obvious case of the data pipeline is machine learning since it requires data to train the model. Higher data quality will result in better model performance. Data pipeline helps machine learning engineers by reducing time spent on data cleaning.

3.4.1 Extract Transform Load and Extract Load Transform

Extract Transform Load, ETL in short, is a type of data pipeline where the process precisely follows the sequence of data ingestion, data transformation, and loading data into the data storages. Extract Load Transform, or ELT, on the other hand, loads the data into the data repository before transforming the data when needed. This approach has become more popular due to its speed, however, this requires the user of the data to transform before use. [24]

3.5 Continuous Integration / Continuous Delivery

A continuous integration and continuous deployment (CI/CD) pipeline is a series of steps that must be performed in order to deliver a new version of software. CI/CD pipelines are a practice focused on improving software delivery throughout the software development life cycle via automation.[18]

By automating CI/CD throughout development, testing, production, and monitoring phases of the software development lifecycle, organizations are able to develop higher quality code, faster. Although it's possible to manually execute each of the steps of a CI/CD pipeline, the true value of CI/CD pipelines is realized through

automation.

A pipeline is a process that drives software development through a path of building, testing, and deploying code, also known as CI/CD. By automating the process, the objective is to minimize human error and maintain a consistent process for how software is released. Tools that are included in the pipeline could include compiling code, unit tests, code analysis, security, and binaries creation. For containerized environments, this pipeline would also include packaging the code into a container image to be deployed across a hybrid cloud.

3.6 Django Framework

Django is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source. [8] Our choice of framework was Django because any machine learning model that we build will make things easier as we can deploy the model using Django REST Framework.

3.6.1 Object Relational Mapping (ORM)

Object-Relational Mapping (ORM) is a technique that lets you query and manipulate data from a database using an object-oriented paradigm. When talking about ORM, most people are referring to a library that implements the Object-Relational Mapping technique, hence the phrase "an ORM".

An ORM library is a completely ordinary library written in your language of choice that encapsulates the code needed to manipulate the data, so you don't use SQL anymore; you interact directly with an object in the same language you're using.

3.6.2 REST

REST, or REpresentational State Transfer, is an architectural style for providing standards between computer systems on the web, making it easier for systems to communicate with each other. REST-compliant systems, often called RESTful systems, are characterized by how they are stateless and separate the concerns of client and server.

In the REST architectural style, the implementation of the client and the implementation of the server can be done independently without each knowing about the other. This means that the code on the client side can be changed at any time without affecting the operation of the server, and the code on the server side can be changed without affecting the operation of the client.

As long as each side knows what format of messages to send to the other, they can be kept modular and separate. Separating the user interface concerns from the data storage concerns, we improve the flexibility of the interface across platforms and improve scalability by simplifying the server components. Additionally, the separation allows each component the ability to evolve independently.

3.6.3 CRUD

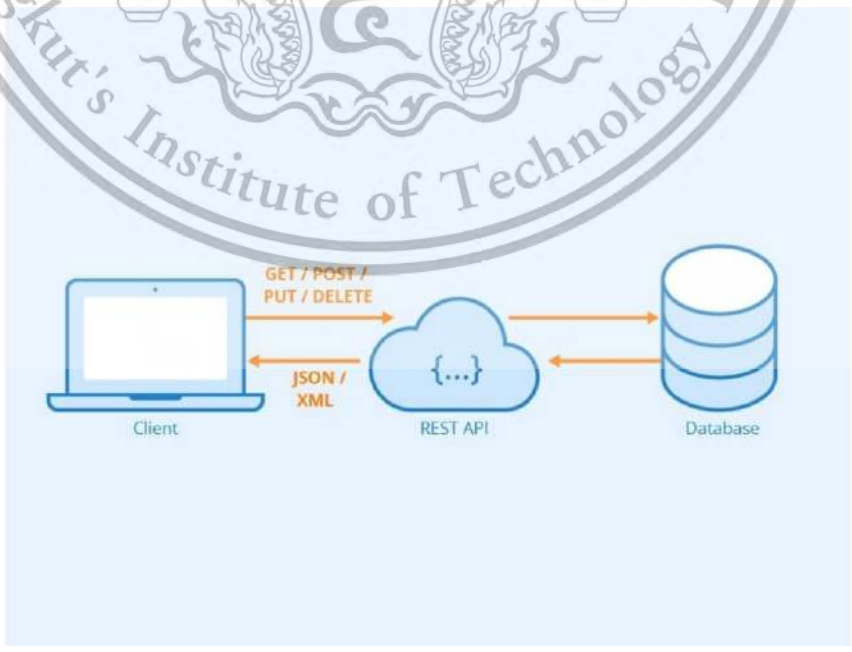


Figure 3.2: REST API Structure

CRUD stands for Create, Retrieve, Update, and Delete, which are the basic operations that can be performed on data in a RESTful API, as shown in Figure 3.2.

In Django Rest Framework, these operations are mapped to HTTP methods as follows:

- Create: POST method
- Retrieve: GET method
- Update: PUT or PATCH method
- Delete: DELETE method

By defining views in Django Rest Framework that correspond to these HTTP methods, you can create a RESTful API that allows clients to perform CRUD operations on your data.

3.7 Flask Framework

Flask is a Python micro web framework, where micro means Flask keeps the core simple and extensible.[8] It is a simple framework for creating a web service that allows developers to decide which tools they prefer such as databases. However, there are various Flask extension packages available, e.g., administrative interface, password hashing, cache support, etc.[7]

3.8 Reverse Proxy

Nginx is a web server and reverses proxy server that is commonly used in modern web applications to improve performance, scalability, and reliability. A reverse proxy is a server that sits between the client and the web server that handles requests and responses, allowing for additional features such as load balancing, caching, and security. Nginx reverse proxy is used to direct incoming client requests to the appropriate backend web server, thereby improving the performance and reliability of the application

3.9 Web Scraping

Web scraping is the process of extracting data and content from a website. However, it is not just the process of extracting displayed pixels, it is the technique of extracting the implicit HTML code of a website.[25] Scraping public data on the internet is absolutely legal. Nonetheless, some kinds of data are not allowed to be scraped and are protected by international regulations, i.e., personal data, intellectual property, and confidential data.[23]

3.9.1 Beautiful Soup Library

Python BeautifulSoup is a popular library used for web scraping, which is the process of extracting information from websites.[12] BeautifulSoup allows developers to parse and navigate HTML and XML documents easily, making it a valuable tool for data extraction and analysis. With BeautifulSoup, developers can scrape data from web pages, including text, images, links, and other elements. The library also provides a range of methods for manipulating and cleaning data, allowing developers to preprocess the scraped data before analysis. Python BeautifulSoup is highly versatile and can be used for a range of applications, from data mining to web crawling. Overall, it is a valuable tool for developers looking to extract and analyze data from websites.

Chapter 4

Requirements & Software Design

4.1 Requirements Analysis

During our online interviews with various individuals, we had the opportunity to delve into their unique perspectives on travel-related challenges. One person we interviewed was Boss, who expressed frustration over the extensive time consumption associated with researching attractions and destinations online. Mali, on the other hand, shared her primary concern for women's safety during travel, highlighting the need for reliable information and resources in this regard. Pierce emphasized his worry about exceeding his budget while traveling, indicating the importance of efficient planning and cost management.

Furthermore, we spoke with Yok, who often plans trips for groups and expressed the difficulties faced in conducting extensive research on Google to accommodate everyone's preferences and needs. Collaboration among friends during trip planning was a challenge raised by Khun, highlighting the need for streamlined communication and decision-making processes. Lastly, Aina expressed her reliance on travel blogs for inspiration and itinerary planning, but also mentioned the overwhelming saturation of recommendations when it comes to choosing destinations.

These interviews shed light on the diverse range of issues faced by individuals when planning and embarking on trips. Understanding these concerns will be invaluable in developing a comprehensive solution that addresses the specific pain

points and provides a more efficient and enjoyable travel experience for everyone involved.

4.1.1 Requirements Elicitation

- 4 out of 5 interviewees mentioned they spent over 2.5 months planning their previous international trips
- 8 out of 8 users were interested in the solutions we provide. 3 of them are willing to pay for a service that generates a travel plan for them.
- Almost all of the interviewees have previously generated their travel plan on social media platforms like LINE, Discord or planned it out in Google Sheets which allows them to collaborate with their co-travelers
- Majority of the interviewees inspire their travel plan from social media influencers, YouTube videos and casual browsing on the internet.
- All the interviewees think there should be a dedicated application that allows them and helps them generate travel plans.

To understand what travelers encounter and feel, we started drawing out a journey map of an average person's trip. In our journey graph [14] shown in figure 4.1, we notice that users feel overwhelmed and confused by research and an inefficient itinerary that had no context.

4.1.2 Functional Requirements

4.1.2.1 Summary of Functional Requirements

- System provides user authentication and registration.
- The system should generate travel plans according to the user's preference.
- The system should allow users to add multiple destinations.
- The system allows users to edit, add, or delete places in the plan.

Figure 4.1: Customer Journey

- The system allows users to write a review for the destinations.
- The system should allow users to create destinations (unseen places).
- The system verifies the user's destination when they publish to the public.
- The system should allow hosts to add co-travelers and collaborate with them.

4.1.2.2 Detailed Functional Requirements

1. **User Authentication and Registration:** The system should provide a secure user authentication and registration mechanism. Users should be able to create an account with a unique username and password, and subsequently log in to access personalized features and information.

2. **Travel Plan Generation:** The system should generate travel plans based on the user's preferences. This could include factors such as preferred destinations, travel dates, budget, interests, and any specific requirements provided by the user. The generated plan should consider available options for accommodations, transportation, attractions, and activities, aiming to create a comprehensive and optimized itinerary.

3. **Multiple Destination Support:** The system should allow users to add multiple destinations to their travel plans. Users should be able to specify

the order of destinations, travel durations, and any specific preferences or requirements for each destination.

4. **Editing, Adding, and Deleting Places:** Users should be able to modify their travel plans by editing, adding, or deleting places within the plan. This includes the ability to adjust the order of destinations, change the duration of stay, add or remove attractions or activities, and update any other relevant details.

5. **Destination Reviews:** The system should allow users to write reviews for the destinations they have visited. Users can provide ratings, comments, and recommendations based on their personal experiences, which can be helpful for other users in making decisions about their own travel plans.

6. **Adding Unseen Places:** The system should enable users to create and add new destinations that may not be commonly known or popular among travelers. This feature allows users to contribute unique and less-explored places to the travel planning community.

7. **Verification of User-Added Destinations:** When a user adds a destination to the public database, the system should verify the accuracy and legitimacy of the information provided. This verification process ensures that the destination information is reliable and trustworthy for other users who may come across it during their travel planning.

8. **Co-Traveler Collaboration:** The system should allow hosts to add co-travelers to their travel plans and collaborate with them. This includes features such as sharing and syncing itineraries, exchanging messages and notes, and jointly making decisions about the travel plan. The system should provide a seamless and efficient communication and coordination platform for co-travelers.

4.1.3 Non-Functional Requirements

4.1.3.1 Summary of Non-Functional Requirements

- The user interface should be minimalistic, user-friendly, and aesthetically pleasing.
- The system should be a responsive web application.
- The system should allow unauthorized users to generate itineraries.
- The system should not allow unauthorized users to save itineraries and invite co-travelers.
- The system should receive HTTP requests.
- The system should provide API documentation via Postman.
- The system should use a setup server provided in KMITL.

4.1.3.2 Detailed Non-Functional Requirements

1. **Minimalistic, User-friendly, and Aesthetically Pleasing User Interface:** The user interface of the web application should be designed with a minimalistic approach, focusing on simplicity and ease of use. It should provide a user-friendly experience, allowing users to navigate the app intuitively and efficiently. Additionally, the interface should be aesthetically pleasing, incorporating visually appealing design elements to enhance user engagement.

2. **Responsive Web Application:** The system should be developed as a responsive web application, ensuring that it adapts and functions well across various devices and screen sizes. Whether accessed from a desktop computer, laptop, tablet, or smartphone, the application should dynamically adjust its layout and content to provide an optimal viewing and interaction experience for users.

3. **Allow Unauthorized Users to Generate Itineraries:** The system should permit unauthorized users (users who have not created an account or logged

in) to generate travel itineraries. This feature allows anyone visiting the web application to access the core functionality of planning and creating travel plans without the need for authentication.

4. **Restrict Unauthorized Users from Saving Itineraries and Inviting Co-travelers:** Unauthorized users should not have the ability to save their generated itineraries or invite co-travelers to collaborate. These functionalities should be reserved for authenticated users who have created an account and logged in. By restricting unauthorized users from these actions, the system ensures that only authenticated users can save their travel plans and engage in collaborative features.

5. **Receiving HTTP Requests:** The system should be designed to receive and handle HTTP requests. This allows for communication and data exchange between the web application and external clients, such as web browsers or mobile apps. The system should effectively handle various types of HTTP requests, such as GET, POST, PUT, and DELETE, to perform operations and provide responses accordingly.

6. **API Documentation via Postman:** The system should provide API documentation using Postman. Postman is a popular API development and testing tool that allows developers to document, test, and interact with APIs. By providing API documentation via Postman, other developers or users can easily understand the available endpoints, parameters, and expected responses, enabling them to integrate or interact with the system's API effectively.

7. **Using a Setup Server Provided in KMITL:** The system should utilize a setup server provided by KMITL (King Mongkut's Institute of Technology Ladkrabang). This implies that the web application should be deployed and hosted on a server infrastructure provided by KMITL, ensuring a reliable and stable environment for the system's operation. The server should meet the necessary requirements and configurations to support the web application's functionality and performance.

4.2 Use Case Diagram

Figure 4.2: Backend Django Class Diagram

4.3 Class Diagram

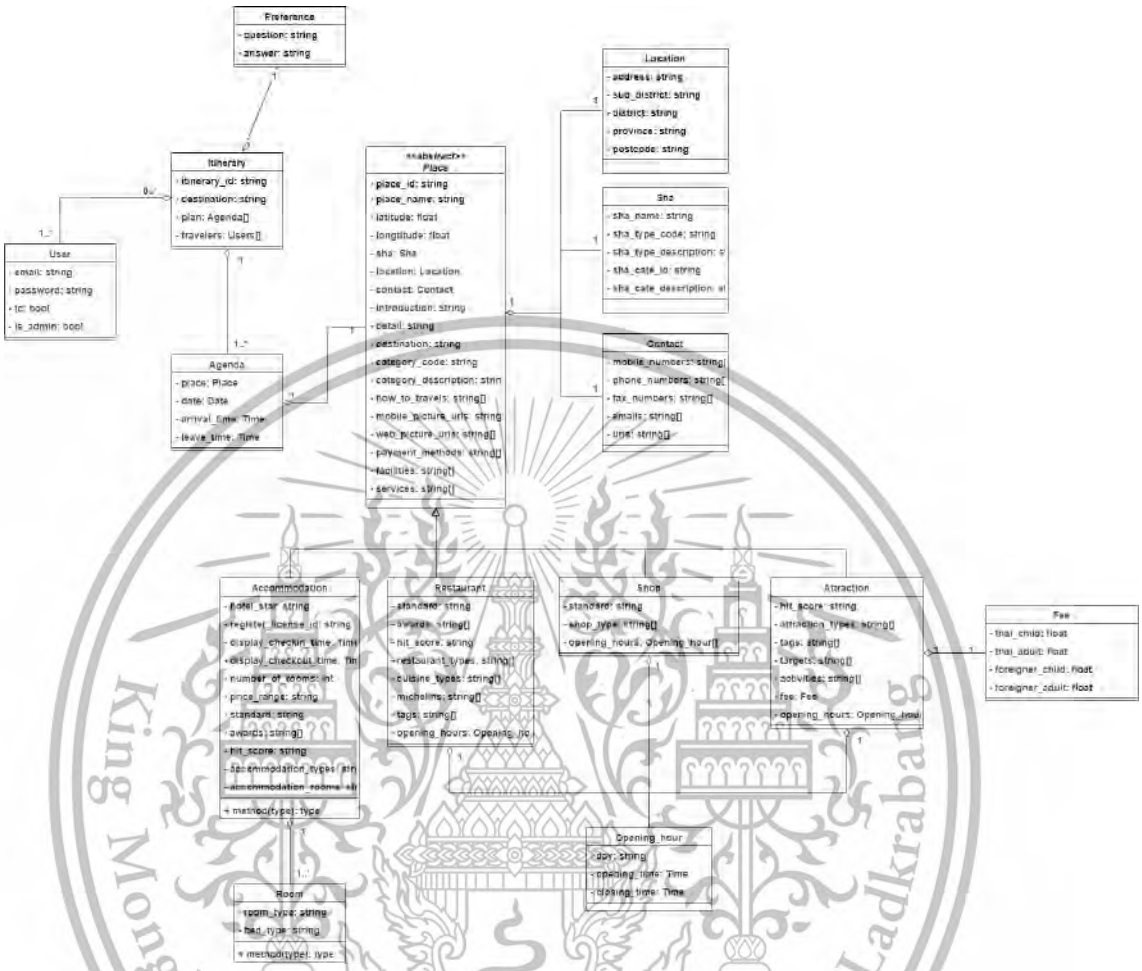


Figure 4.3: Backend Django Class Diagram

4.4 System Architecture

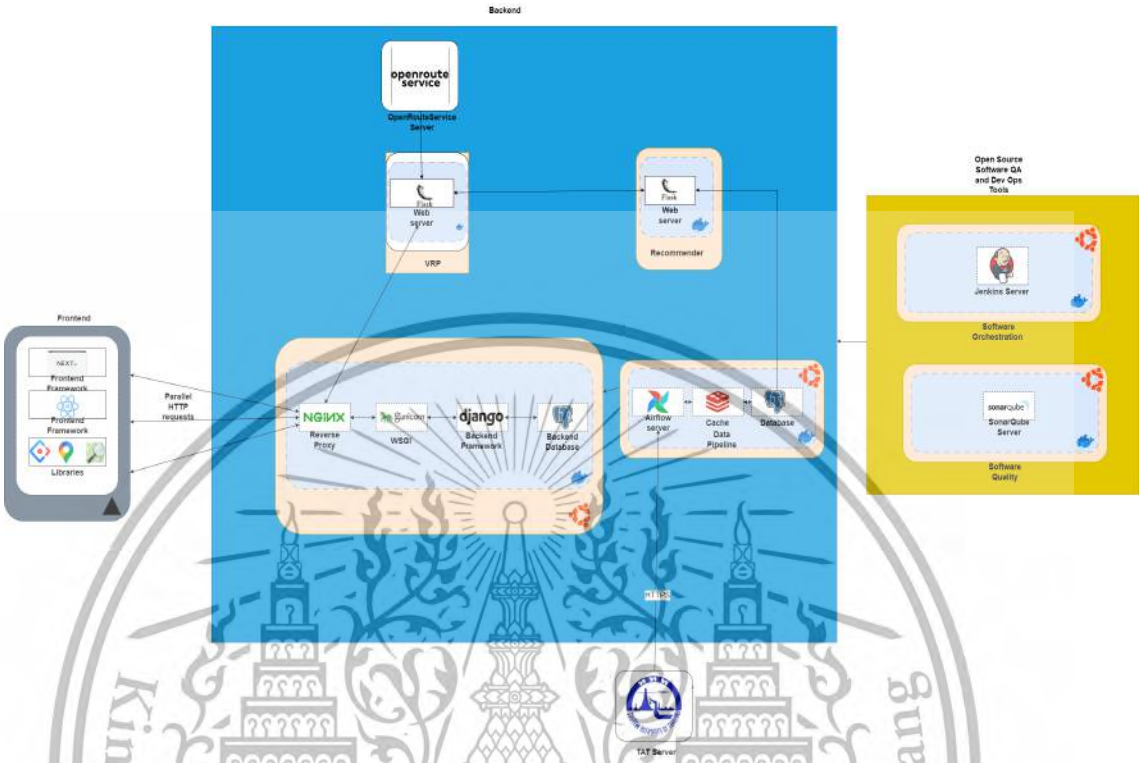


Figure 4.4: System Architecture

4.4.1 Client-Side Components

4.4.1.1 Application Structure and Component Design

The client-side application is built using React. The UI is primarily customised, based on using the Ant Design library.

4.4.1.2 State Management and Data Handling

State management and data handling are achieved through the use of Axios. Axios, which is a promise-based HTTP client, enables efficient communication with the server by making asynchronous API calls to fetch and send data.

4.4.1.3 Geographic and Calendar Functionality

Leaflet, an open-source for interactive maps, and OpenStreetMap, an open-source mapping project, are utilized to create geographical data and map functionality.

Users can view and interact with their travel plans in a geographically. Moreover, FullCalendar is used to visually represent travel plans in a calendar layout to better demonstrate the travel plan.

4.4.1.4 Client-Side Routing

React Router is used to handle client-side routing, enabling navigation through different pages in the application, and supports the implementation of protected routes which are only accessible based on user authentication and authorization status.

4.4.1.5 Security and User Authorization

The application incorporates robust security measures and user authorization checks on the client side. Input validation is done using the Ant Design (Antd) Forms library and custom validation logic. Additionally, unauthorized user access controls are enforced, restricting the visibility of trip details based on user roles and permissions.

4.4.2 Server Side Components

4.4.2.1 Django Backend

The Django backend is the core of the system which takes care of storing user data, storing places as well as managing itinerary for each individual user. It has been connected to Nginx, and Gunicorn WSGI so it can serve to its users.

4.4.2.2 Itinerary Scheduling System

The itinerary scheduling system is the system that is implemented to generate a travel plan, as a list of agendas, by inputting the list of places and details of the trip, e.g., start time and end time of each day, trip start date, number of days in a trip, etc. The system is implemented by applying the Ant Colony Optimization (ACO) algorithm to solve the Vehicle Routing Problem (VRP) that represents the scheduling problem of trip planning.

4.4.2.3 Recommender System

The recommender system is one of the crucial parts of the system which is being utilized for generating a list of places that the user might be interested in for further use by the VRP. The recommender system is connected to the PostgreSQL database so that it has access to the fresh data provided by the Airflow data pipeline on a weekly basis. The recommender system leverages machine learning techniques to provide personalized place recommendations to users.

4.4.3 Server Side Components

4.4.3.1 Django Backend

The Django backend is the core of the system which takes care of storing user data, storing places as well as managing itinerary for each individual user. It has been connected to nginx, and gunicorn wsgi so it can serve to its users.

4.4.3.2 Itinerary Scheduling System

The itinerary scheduling system is the system that is implemented to generate a travel plan, as a list of agendas, by inputting the list of places and details of the trip, e.g., start time and end time of each day, trip start date, number of days in a trip, etc. The system is implemented by applying the ACO algorithm to solve the VRP that represents the scheduling problem of trip planning.

4.4.3.3 Recommender System

The recommender system is one of the crucial part of the system which is being utilized for generating a list of places that the user might interested for further use by the VRP. The recommender system is connected to the PosgreSQL database so that it has access to the fresh data provided by the airflow data pipeline on the weekly basis.

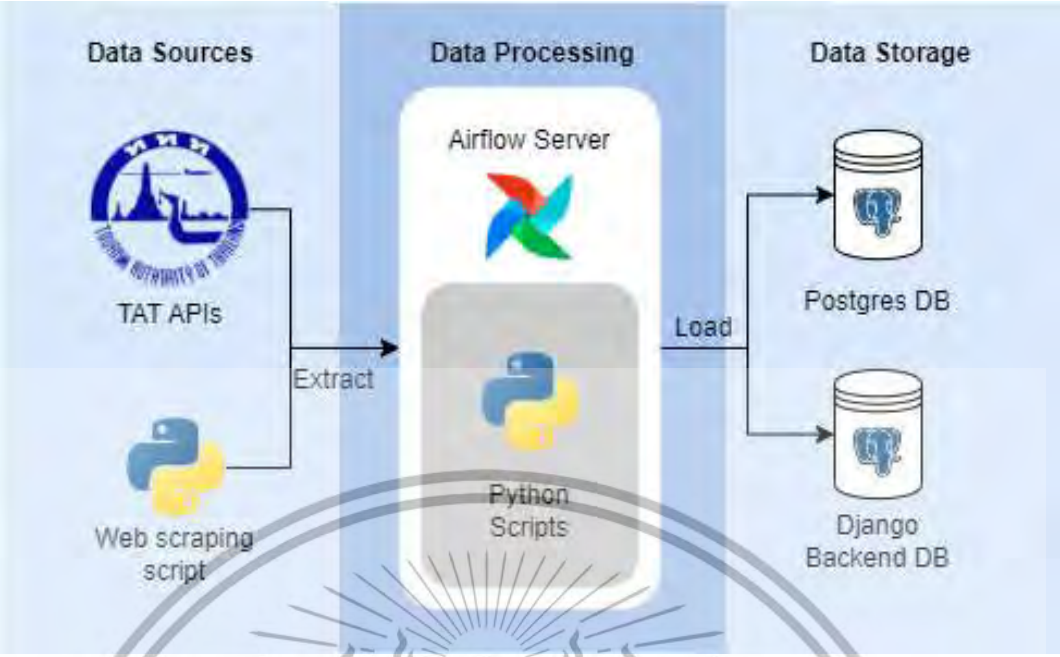


Figure 4.5: Data Pipeline

4.5 Data Pipeline

The data pipeline in this project consists of 3 mains components: Data source, Data processing, and Data storage. The pipeline is utilizing ETL process which extract the data, transform the data into desire format and then load into the data storage. This data pipeline automatically runs on weekly basis and aims to collect the place data, process, and load into our database to be use by both front-end part of the project and also use by the recommender system to suggest personalized travel location.

4.5.1 Data Source

Data source is a crucial parts for our project since it provide us with critical data needed by the recommender system and the application interface for the users. There are various sources of data that we can gather for our use for instance, web scraping, Tourism Authority of Thailand API, or Google API. The chosen data sources for this project are TAT API for place data and Web scraping for place popularity. The data obtained from TAT api contains data about places. Crucial information being utilized by the recommender system are attraction types,

activities, and destination.

- *Attraction types* are types of attractions for instances, museums, islands, beaches, etc.
- *Activities* are the activities offered by that attractions.
- *Destination* is the location of that attraction for example, Bangkok.

4.5.2 Data Processing

After we obtained the required data the next part is to clean and transform the data into the correct format. Data processing method can ensure data is cleaned to a desirable level and transform to a appropriate format before storing it in the database and will reduce the work needed in order to clean the data in the future usage.

4.5.3 Data Storage

The last part of the pipeline is to load the cleaned and transformed data into the database. The processed data is stored in two places relational database via PostgreSQL, and database of the backend server. Both database contain similar data, however it is separate so that the workload between frontend and recommender system is splitted. These databases allow the recommender system to access the data it needs to recommend places for the users and the frontend to display the place information to the users. Figure 4.6 shows the ER diagram for the PostgreSQL database

Figure 4.6: Data Pipeline ER diagram

Chapter 5

Software Development

5.1 Data Engineering Process

5.1.1 Data Gathering

Since the scope of the project is limited to traveling in Thailand, the dataset consists of places in Thailand that are required. Thus, we construct the dataset from the Tourism Authority of Thailand's web service APIs. The data-gathering process is done by implementing Python script using Requests and JSON libraries. The Requests library is used to interact with the Tourism Authority of Thailand's API services and convert the API response into a JSON object for the response to be processed more comfortably. The JSON library is utilized to convert the JSON object created from the API response into a JSON string, which is then saved into a JSON file. We created the dataset by calling the `getPlaceSearch` API and incrementing the `pagenumber` query parameter iteratively to gather a list of the places' basic data, which includes the `place_id` key attribute.

```
{
    "place_id": "P02000002",
    "place_name": "Amari Watergate Bangkok",
    "latitude": 13.751371, "longitude": 100.540248,
    "sha": {
        "sha_name": "Amari Watergate Bangkok",
        "sha_type_code": "2", ...
    },
    "place_information": {
        "introduction": "The magnificent Amari Watergate Hotel is...",
        "detail": "The magnificent Amari Watergate Hotel is...",
        "accommodation_types": [
            {
            "code": "18",
```

```
            "description": "Registered Hotel"
            }
        ], ...
    },
    "location": {
        "address": "847 Petchburi Road", ...
    },
    "contact": {
        "phones": [ "02-653-9000" ], ...
    },
    "facilities": [
        {
            "code": "PK",
            "description": "Parking"
        }, ...
    ], ...
}
```

Listing 5.1: Example of detailed data from getAccommodationDetail API

The `place_id` is then used to query the detailed data of each place via the `get<placeCategory>Detail` API. Since the detailed data does not contain `category_code` and `category_description` of the place, those fields were added to the detailed data separately. The detailed data was collected and saved to the JSON file as a JSON array, so it could be inserted into the raw data database effortlessly.

Algorithm 2 Pseudocode for creating the dataset from the TAT APIs

```
Create a JSON file for writing the data
write '[' into the file
response ← True
pageNo ← 1
firstObject ← True
while response do
    response ← getPlaceSearch(pagenumber = pageNo)
    pageNo ← pageNo + 1
    for each place textbfin response do
        detail ←get⟨place['category_code']⟩Detail(place_id
      = place['place_id'])
        detail['category_code'] ← place['category_code']
        detail['category_description'] ← place['category_description']
        if firstObject = True then
            writedetailintothedatafile
            firstObject ← False
        else
            write ',' and detail into the data file
        end if
    end for
end while
write ']' into the file
```

5.1.2 Data Cleaning

After we got the raw data from the previous section, the data must be cleaned before it would be used in further processes. Data cleaning is the process of removing corrupted, incomplete, or duplicate data within a dataset [9]. Some of the data might have different character cases, e.g., Bangkok, bangkok, and BANGKOK, or different values representing empty data. Thus, we need to standardize the entire dataset to be under one standard. We also include the restructuring of the data in this step to facilitate further processes the raw data have different attributes among places categories. In the data cleaning process, Pandas is the main Python library that we used.

5.1.2.1 Remove Duplicate or Irrelevant Data

The dataset contains uncategorized places, the majority of which are only place names and localities and lack sufficient information to be useful. Some of the places do not have latitude and longitude information which is critical. Duplication of `place_id`, which is the primary key of the data, must not be allowed. Thus, we decided to remove those uncategorized places, places with duplicate `place_id`, and places lacking coordinate information from the dataset.

5.1.2.2 Data Restructuring

Initially, the raw data have a nested structure, and many significant attributes are stored in the `place_information` attribute. The attributes stored in the `place_information` field are different among the different place categories. Therefore, we decided to extract those attributes from the `place_information` and separate the data into four sub-datasets for four place categories. All the JSON objects in each sub-dataset have the same attributes, to be easier for further cleaning processes and for constructing the relation database table from the data. Furthermore, there are some common attributes that have nested attributes, e.g., `location`, `contact`, etc. Thus, we replace those attributes with the attributes inside of them. The results of this step are four separate JSON files for four categories of places that contain the same attributes for each file.

```
{
    "place_id": "P02000002",
    "place_name": "Amari Watergate Bangkok",
    "latitude": 13.751371,
    "longitude": 100.540248,
    "sha_name": "Amari Watergate Bangkok",
    "sha_type_code": "2", ...
    "introduction": "The magnificent Amari Watergate Hotel is...",
    "detail": "The magnificent Amari Watergate Hotel is...",
    "accommodation_types": [ "Registered Hotel" ],
    "register_license_id": "TTS-202-003-590001-0",
    "hotel_star": "5",
    "display_checkin_time": "14:00",
    "display_checkout_time": "12:00",
    "number_of_rooms": 569,
    "price_range": "2810",
    "address": "847 Petchburi Road",
    "sub_district": "Thanon Phhaya Thai",
    "district": "Ratchathewi",
    "province": "Bangkok",
    "postcode": "10400",
    "phones": [ "02-653-9000" ],
    ...
}
```

Listing 5.2: Example of data in JSON file storing accommodation data

5.1.2.3 Fix Structural Errors

Some attributes have different values representing an empty value, e.g., NaN, an empty string, None, an empty array, etc. Since we would insert the data into a relational database, the value representing the empty cell should be a null value. So, we filter out the possible values that can represent the null value, then replace them with None, to avoid the problem while inserting the data into the relational database.

5.1.2.4 Address Cleaning

The `destination` column in the place table and `province`, `district`, `sub-district` in the location table contains a lot of common items with different letter cases for instance, Bangkok, BANGKOK, or even in Thai. Moreover, some columns' data type needs to be adjusted to their correct types to reduce the space needed for storing and correctness of the display values, for example, the postcode is first stored in float format and later adjusted to int. These values need to be cleaned in order for the recommender system to perform as expected. Algorithm 3 shows pseu-

docode for cleaning `destination` and Algorithm 4 shows pseudocode for cleaning the location table.

Algorithm 3 Pseudocode for destination column cleaning

```
Read a place CSV file
Read province CSV file
dataFrame ← dataFrame['destination'].apply(lambdax : x.upper())
for index in range of province do
    thai_name ← province.iloc[index]['ThaiName']
    eng_name ← province.iloc[index]['Name']
    dataFrame['destination'] ← dataFrame['destination'].replace(
  thai_name, eng_name.upper())end for
    extra_location ← [itemsthatisnotinprovincetable]
    eng_ver ← [englishversionofextra_location]
    index ← 0
    for place in extra_location do
        dataFrame['destination'] ← dataFrame['destination'].replace(
  place, eng_ver[i].upper())
        index ← index + 1
    end for
    Write dataFrame to CSV
```

Algorithm 4 Pseudocode for location table cleaning

```
Read location table CSV
dataFrame ← dataFrame['postcode'].fillna(0)
dataFrame ← dataFrame['postcode'].astype('int32')
for province in set of dataFrame['province'] do
    dataFrame['province'] ← dataFrame['province'].replace(
  province, province.upper())end for
    district in set of dataFrame['district']
    dataFrame['district'] ← dataFrame['district'].replace(
  district, district.upper())
        subdistrict in set of dataFrame['sub_district']
        dataFrame['sub_district'] ← dataFrame['sub_district'].replace(
  subdistrict, subdistrict.upper())
            Write dataFrame to CSV
```

5.1.3 Airflow Data Pipeline

5.1.3.1 Extract

Extract part of the pipeline consists of 3 tasks: get_raw_data, get_detail and get_popularity. The functionality of the three functions can be describe as follow.

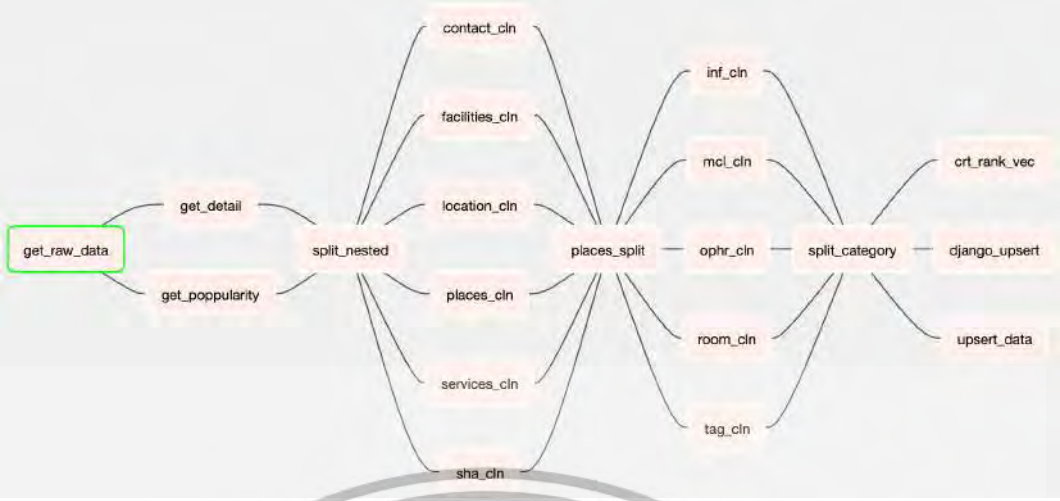


Figure 5.1: Airflow Data Pipeline

- *get_raw_data*: extract data from the TAT's GetPlaceSearch API, the API returns basic information of each places which the place's id will be used to obtained more detailed information in the following tasks.
- *get_detail*: extract in-depth details of each places utilizing the place id from get_raw_data task.
- *get_popularity*: scrape popularity of the places using place id, place name, and destination from search count of each places in Google.

5.1.3.2 Transform

There are multiple tasks being used to process the extracted data into a required format. Generally, process that are used to transform the data consists of flatten, clean, and transform. Where flatten functions remove the nested object inside the data. The clean functions cleanse the missing data via multiple approaches appropriate for each cases and also standardize the string for some attributes. Lastly, the transform functions restructured the data into the specific format so that it can be correctly upsert into the database.

5.1.3.3 Load

The pipeline loads data into two places: Django backend's database and PostgreSQL database. Firstly, the upsert task delete the existing table in the database

to prevent any conflict when updating. After removing tables are done, it upsert the Pandas Dataframe into corresponding table. PostgreSQL Database consists of 24 tables containing all information about the places. In addition, the database is normalized to 5NF to ensure that the tables are organized.

5.1.3.4 Upsert Data to Django

When dealing with data, it is often a good practice to store different types of data in separate databases. In the case of the Airflow instance that is pulling data from the Tourism Authority of Thailand and making POST requests to a Django backend on a weekly basis, it would be beneficial to store the incoming data in a separate database from the Django backend's database.

By storing the incoming data in a separate database, it ensures that the data is isolated and can be managed independently of the Django backend's database. This provides several advantages, such as improved scalability, better security, and easier maintenance.

One of the biggest advantages of storing data in separate databases is improved scalability. By separating the incoming data from the Django backend's database, it reduces the load on the backend database, which can lead to improved performance and scalability. This is especially important when dealing with large amounts of data, as it allows for better management and optimization of resources.

Another advantage is better security. By storing the incoming data in a separate database, it ensures that any security breaches or vulnerabilities in the incoming data are isolated from the backend database. This reduces the risk of data loss or corruption and improves overall security.

Finally, storing data in separate databases makes maintenance and updates easier. It allows for easier backups and restores, as well as simpler maintenance and updates for each database. This reduces downtime and improves overall system reliability.

5.1.3.5 Code Optimization

According to Velotio Technologies [20], asynchronous programming in Python can offer significant performance benefits.

To optimize code being published to the django server, we have made use of asynchronous programming with batching to make sure we can upload large stream of data without slowing down the server and affecting its performance.

Here is an example code snippet that runs on the airflow server:

```
async def create_place(token, payload_request):
    '''
    API request to create a place in the django server at port 1337
    '''
    headers = {
        'Authorization': f'Bearer {token}',
        'Content-Type': 'application/json'
    }

    payload = payload_request
    session = requests.Session()
    session.trust_env = False

    response = session.post('http://dev.se.kmitl.ac.th:1337/api/places/',
        ↪ headers=headers, json=payload)

    if response.status_code == 201:
        print('Place created successfully!')
    elif response.status_code == 400 and "Place with this ID already
        ↪ exists." in response.json():
        print('Place already exists, skipping creation')
    else:
        print('Error creating Place:', response.content)

async def create_places(token, places_json, batch_size):
    '''
    Batch API request to create multiple places in the django server at
        ↪ port 1337
    '''
    tasks = []
    for i in range(0, len(places_json), batch_size):
        batch = places_json[i:i+batch_size]
        for j in range(len(batch)):
            task = asyncio.create_task(create_place(token, batch[j]))
            tasks.append(task)
        await asyncio.gather(*tasks)
```

Listing 5.3: Code to create places

The code defines two async functions. The first function `create_place` creates a single place on the Django server by sending an API request with the given

payload. It then checks the response code and prints the appropriate message based on the response.

The second function `create_places` is an async function that creates multiple places in batches by calling the `create_place` function. It takes three parameters: `token`, `places_json`, and `batch_size`. The `places_json` parameter is a list of JSON objects, where each object represents a place. The `batch_size` parameter is an integer that specifies the number of places to create in each batch.

The `create_places` function first divides the `places_json` list into batches of `batch_size` and then creates a task for each place in each batch using the `asyncio.create_task` function. The `create_place` function is called for each task with the given `token` and the current place in the batch. The tasks are then gathered using the `asyncio.gather` function, which returns when all tasks are complete.

This code is well-optimized as it makes use of asynchronous programming to create multiple places in batches. This reduces the amount of time taken to create all the places. Since the creation of each place is independent of the others, it can be done asynchronously. By using async functions, the code can execute multiple tasks concurrently, which is more efficient than executing them sequentially.

5.2 Frontend Development

5.2.1 OpenStreetMap and Leaflet

OpenStreetMap and Leaflet are tools for integrating interactive maps into web applications. OpenStreetMap is an open-source project that provides geospatial data, while Leaflet is designed for building interactive maps. React-Leaflet is provided for the integration with React.js, which offers React components representing Leaflet layers, controls, and more. Using these components, it's possible to create a dynamic map with pins and connections between different locations. Marker components are used to place pins on the map, and Polyline ise used

for establishing connections between places. React's state and props mechanism paired with the responsive nature of Leaflet make real-time map updates and user interaction.

5.2.2 FullCalendar

FullCalendar is an open-source JavaScript library that provides interactive calendar functionalities for scheduling and event management in web applications. It offers a range of capabilities like drag-and-drop, event resizing, and various view modes for dynamic scheduling. Full calendar also provides React-FullCalendar, that offers React components for the FullCalendar API. The React's state and props mechanism features can be controlled in real-time.

5.2.3 Ant Design

Ant Design, or antd, is a React component library, it allows flexibility to create customizable components that can follow the specific requirements. Each component is encapsulated, keeping the React principle of a single responsibility pattern. Most components provide a number of properties and methods that can be used to customize their behavior and appearance. Moreover, antd offers built-in internationalization and theme customization capabilities.

5.2.4 Responsiveness and Accessibility

The application is designed to ensure optimal responsiveness and accessibility. Flexbox is predominantly used in CSS styling for layout arrangement, providing flexibility and efficiency in scaling components depending on the container size. Sizing units like percentages and rem are utilized to allow elements to respond to their container's size, further improving the application's responsiveness.

Furthermore, CSS Media queries are employed to adjust styling based on the user's device screen size and resolution. This allows for changes in the layout, such as stackable columns and scalable text size, to ensure experiences on different evironments like desktops, tablets, and mobile devices.

5.2.5 User Experience

Designs are based on UI/UX principles, which aims to deliver an interface that is both user-friendly and intuitive. With the use of UI components from Ant Design, ensuring a cohesive aesthetic across the platform. For instance, based on User Control and Freedom principles, users are offered the choice between system-generated itineraries and the creation of their own custom itineraries. By facilitating these diverse user needs, the platform provides an efficient and user-centric experience, enhancing overall satisfaction and engagement.

User testing and feedback collection have been established in the application's design process, by engaging users in the design process, conducting usability tests at various development stages to identify potential issues. These iterative feedback loops are ensuring the final product effectively meets user needs.

5.2.6 Services and Backend Interaction

The frontend of the application maintains a connection with the backend through Axios. This connection is established with a deployed backend endpoint, ensuring a reliable and consistent data flow. Moreover, the Axios instance is customized to fit the specific requirements of the application, enabling precise control over request configurations, headers, and response handling. Additionally, interfaces are used to enforce a strong structure between the frontend and the backend, ensuring that the data passed across this connection aligns to a defined structure.

5.3 Backend Development

5.3.1 Server

The backend development project is built using the Django framework and incorporates the Django REST Framework (DRF) to create an efficient and effective RESTful API. The project contains multiple apps, including authentication, place creation, and itinerary creation. The authentication app provides secure login and registration functionality, allowing users to create accounts and access the pro-

tected API endpoints. The place creation app allows users to create new places, with fields such as name, address, and description. The itinerary creation app allows users to create itineraries, which can be made up of multiple places and include details such as the start and end dates.

The DRF is used to create a well-structured API, with clear endpoints and standardized responses. The project incorporates features such as serializers, views, and routers to simplify the process of creating and consuming the API. The API is designed to be easily extendable, with the ability to add new endpoints or apps as required.

5.3.1.1 Authentication App

The authentication app in the Django project provides secure login and registration functionality. Users can create an account with their email address and a secure password, and then log in to access the protected API endpoints. Passwords are stored securely using a hashing algorithm to ensure that they are not accessible in plain text. Additionally, the app provides token-based authentication, which allows users to authenticate once and then access the API without having to enter their password for subsequent requests. This app is crucial for maintaining the security of the API and ensuring that only authorized users can access the data.

Here is a list of endpoints that are implemented in this app:

Service	Base URL	Endpoint
Register	http://dev.se.kmitl.ac.th:1337/api/user	/register
Login	http://dev.se.kmitl.ac.th:1337/api/user	/login
Profile View	http://dev.se.kmitl.ac.th:1337/api/user	/profile
Change Password	http://dev.se.kmitl.ac.th:1337/api/user	/changepassword

Table 5.1: Endpoints related to user and authentication

5.3.1.2 Place Creation App

The place creation app in the Django project allows users to create new places, which can be retrieved by other parts of the API. When a user creates a new place, they provide details such as the name, address, and description. The app validates the input data to ensure that it meets the required format, and then stores the place in the database. When a user requests information about a place, the app retrieves the data from the database and returns it in a standardized format using the DRF serializers. This app is essential for providing the data that other parts of the API use to create itineraries and other functionality.

Here is a list of endpoints that are implemented in this app:

Service	Base URL	Endpoint
Create and read all places	http://dev.se.kmitl.ac.th:1337/api/places	/
CRUD individual place	http://dev.se.kmitl.ac.th:1337/api/places	/:placeId
CRUD individual accomodation	http://dev.se.kmitl.ac.th:1337/api/places	/accomodations/:placeId
CRUD individual shops	http://dev.se.kmitl.ac.th:1337/api/places	/shops/:placeId
CRUD individual attractions	http://dev.se.kmitl.ac.th:1337/api/places	/attractions/:placeId
CRUD individual restaurants	http://dev.se.kmitl.ac.th:1337/api/places	/restaurants/:placeId
GET all accomodation	http://dev.se.kmitl.ac.th:1337/api/places	/accomodations/
GET all shops	http://dev.se.kmitl.ac.th:1337/api/places	/shops/
GET all attractions	http://dev.se.kmitl.ac.th:1337/api/places	/attractions/
GET all restaurants	http://dev.se.kmitl.ac.th:1337/api/places	/restaurants/

Table 5.2: Endpoints related to Places

5.3.1.3 Itinerary Creation App

The itinerary creation app in the Django project allows users to create itineraries, which are collections of places and other details. Users can specify the start and end dates for the itinerary, as well as the places that they want to visit. The app validates the input data to ensure that it meets the required format, and then stores the itinerary in the database. When a user requests information about an itinerary, the app retrieves the data from the database and returns it in a standardized format using the DRF serializers. This app is crucial for providing users with the ability to plan and track their trips, and for providing the data that other parts of the API use to create other functionality.

Service	Base URL	Endpoint
Create and read all itinerary	http://dev.se.kmitl.ac.th:1337/api/itinerary	/
Create and read all agendas	http://dev.se.kmitl.ac.th:1337/api/itinerary	/agendas
CRUD individual itinerary	http://dev.se.kmitl.ac.th:1337/api/itinerary	/:itineraryId
CRUD individual agenda	http://dev.se.kmitl.ac.th:1337/api/itinerary	/agenda/:agendaId

Table 5.3: Endpoints related to Itinerary, agenda and user preference

5.3.2 Itinerary Recommendation Module

The itinerary recommendation module is to recommend the whole trip plan for the user based on their preferences, e.g., place features, activities, cuisine types, number of days in a trip, etc. The module calculates the number of attractions and restaurants required for the specified number of days in a trip using the recommended place generation module. Then, request the recommended attraction and restaurants based on the user's preferences. After it retrieved the recommended places, then request the recommended accommodation based on the recommended attractions and restaurants. Lastly, it generates the itinerary by sending the request with the list of recommended place to the itinerary scheduling module.

5.3.3 Recommended Place Generation

The recommendation system generates a list of places according to the user preferences for their trips. The recommender include 3 main recommendation functions, attraction recommendation, restaurant recommendation, and accommodation recommendation. Figure 5.2 and 5.3 shows the flowchart for attraction recommendation and restaurant recommendation respectively.

Table 5.4 is the list of endpoints for recommender system.

Service	Base URL	Endpoint
Accommodation recommendation	http://dev.se.kmitl.ac.th:3400/api/recommendaccommodation	/
Attraction recommendation	http://dev.se.kmitl.ac.th:3400/api/recommendattraction	/
Restaurant recommendation	http://dev.se.kmitl.ac.th:3400/api/recommendrestaurant	/

Table 5.4: Endpoints related to recommendations

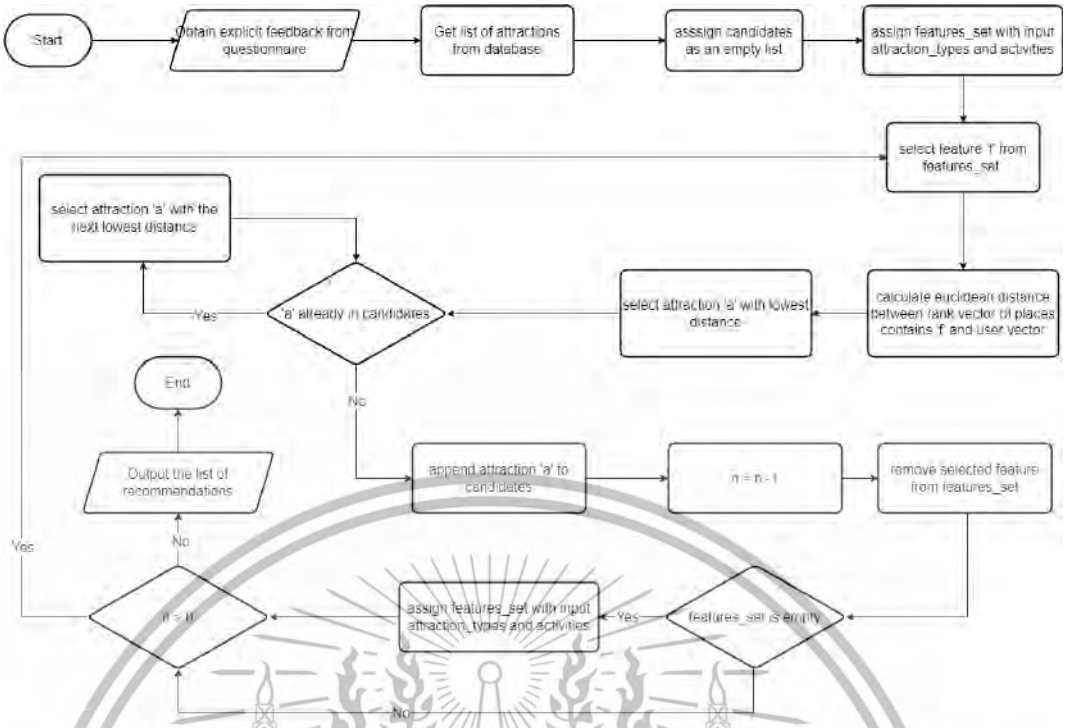


Figure 5.2: Attraction Recommendation Flowchart

5.3.3.1 Rank Vector

Rank vector is essential for the attraction recommendation as it allows the recommender to select most popular places for each types of attraction and activities. The to calculate the rank for each place, there are two functions handling this task calc_rank and norm_rank. Those two function calculates rank using popularity data and normalized the rank into value between 0-1 respectively. The listing 5.4, 5.5 and 5.6 code snippet and pseudo code for calculating rank, normalizing rank, and creating rank vector is as follows.

```
def calc_rank(feature, attractions):
    df = attractions[attractions.features.apply(lambda x: contain(x,
        ↪ feature))].sort_values('popularity', ascending=False).copy
        ↪ ()
    df.loc[:, 'rank'] = range(len(df))
    return df
```

Listing 5.4: Code to calculate rank

```
def norm_rank(feature, attractions):
    df = attractions[attractions.features.apply(lambda x: contain(x,
        ↪ feature))].copy()
    normalized_rank = 1 - (calc_rank(feature, attractions)['rank']/len
        ↪ (df))
    df.loc[:, feature] = normalized_rank
```

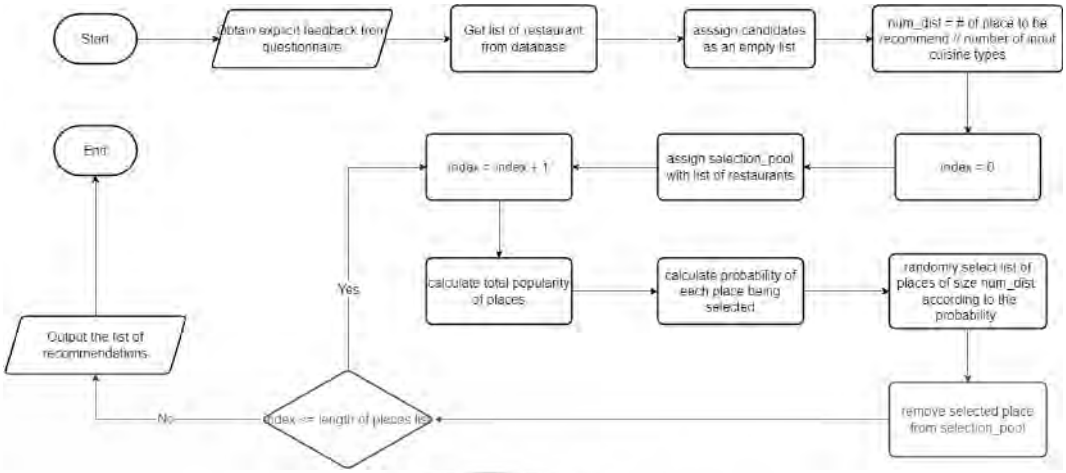


Figure 5.3: Restaurant Recommendation Flowchart

```
    return df[['place_id', feature]].sort_values(feature, ascending=
        ↪ False)
```

Listing 5.5: Code to calculate normalized rank

```
def create_rank_vector():
    attractions = dataframe containing place_id, attraction_types,
        ↪ activities, popularity
    attractions['features'] = concat attraction_types and activities
        ↪ columns
    features = get unique list of features from attractions
    for type in features:
        rank_vector.append(norm_rank(type, attractions))
    return rank_vector
```

Listing 5.6: Pseudo code to create rank vector

5.3.3.2 Similarity Measure

After we obtained the rank vector, calculating distance between user vector and the rank vector is done to measure the similarity between those two vectors. The listing 5.7 and 5.8 code snippet for calculating euclidean distance and similarity score respectively.

```
    def dist(self, vec_a, vec_b):
        return sqrt(sum((e1-e2)**2 for e1, e2 in zip(vec_a, vec_b)))
```

Listing 5.7: Code to calculate euclidean distance

```
    def calc_dist(self, user_vec, place_vec):
        distances = place_vec.loc[:, place_vec.columns != 'place_id'].
            ↪ apply(lambda row: self._dist(row, user_vec.values.flatten()
```

```
        ↪ ), axis=1)
    df = place_vec.copy()
    df['score'] = distances
    return df
```

Listing 5.8: Code to measure item similarity

5.3.3.3 Roulette Wheel Selection

Roulette wheel selection randomly selects a place with a weight provided by popularity, this means place with higher popularity will gain a higher chance of being picked by the selector. The listing 5.9 shows the implementation of roulette wheel selector.

```
def roulette_selector(self, places, size):
    pop_fitness = places['popularity'].sum()
    places['probability'] = places.popularity / pop_fitness
    return np.random.choice(places['place_id'], size=size, p=places['
        ↪ probability'], replace=False)
```

Listing 5.9: Code for Roulette wheel selection

5.3.3.4 Attraction Recommendation

Recommending the attractions used the rank vector to calculates the similarity between places and user preference. This approach selects one of the features from the set of features which then used to select the most popular place of that feature from the rank vector. When the features set is empty, it will populates the features set by the input features. This repeats until the amount of recommended places meets the desire amount. The algorithm 5 shows the recommendation algorithm for attraction.

5.3.3.5 Restaurant Recommendation

Unlike attraction recommendation, objective of the restaurant recommendation is to generate a equally distributed list of restaurant chosen by cuisine types without having to concerned about a restaurant that fulfill multiple types of cuisine. With the reason stated earlier, recommending restaurant only utilized the roulette wheel selector to select a list of restaurant for each cuisine types preferred by the user.

Algorithm 5 Attraction recommendation

Initialize *candidates* as an empty set
Initialize $features\_set$ as set of input attraction types and activities
Initialize *attractions* as a list of places containing attraction in *destination*
Initialize top_n as a number of places that will be return
while $top\_n > 0$ **do**
  if $features\_set$ is empty **then**
    $features\_set \leftarrow$ set of input from attraction types and activities
  end if
  $feature \leftarrow$ random item from the $features\_set$
  $user\_vector \leftarrow$ vector of 0,1 from $features\_set$
  $result \leftarrow calc\_dist(user\_vector, attrations)$
  $candidates \leftarrow result$
  $top\_n \leftarrow top\_n - 1$
  $features\_set \leftarrow$ remove $feature$ from $features\_set$
end while
$returncandidates$

5.3.3.6 Accommodation Recommendation

The accommodation recommendation is to suggest accommodation options based on a given list of places, which are other recommended attractions and restaurants. It connects to a database, retrieves relevant data about accommodations, including their opening hours and popularity, and stores it in a Pandas data frame. Then calculates the center point of the other places and iterates through the accommodation list to find the nearest place based on spherical distance. Therefore, we efficiently recommend accommodations by considering their proximity to the center of other places, making it easier for users to find nearby accommodation options.

5.3.4 Itinerary Scheduling Module

The itinerary scheduling API is one of the most significant parts of the application. The overall process is to generate a set of recommended destinations based on users' preferences. It is implemented using Python 3.10 to implement the ant colony optimization to generate an optimal schedule based on the recommended places from the place recommender system. Then the set of recommended destinations is used for generating the route and plan for each trip day.

5.3.4.1 Modified Vehicle Routing Problem Algorithms

The idea of using a vehicle routing algorithm to generate an itinerary came from the itinerary planning approach using K-means clustering combined with the traveling salesman problem [17]. In K-means clustering with the traveling salesman approach, the researchers used the K-means algorithm to group destinations in multiple clusters for each traveling plan. The number of clusters is the number of days in a trip. Then a genetic algorithm is used to solve traveling salesman problems in each cluster. As shown in Figure 5.4, mentioned that hotel blocks in Figure 5.4 represent the same hotel, combining K-mean clustering with the traveling salesman problem has similar characteristics with the vehicle routing problem in Figure 5.5. Thus, a vehicle routing problem with the time windows algorithm is utilized to generate a travel itinerary from the list of recommended destinations created by the recommender system. In vehicle routing problems, multiple routes are generated for multiple vehicles in a fleet, which seems equivalent to generating multiple routes for multiple days in a trip. Thus, we can generate a sequence of places to visit each day with optimal traveling time. Furthermore, we can apply time constraints in vehicle routing problems with time windows to consider the opening time of each place and the time for each meal.

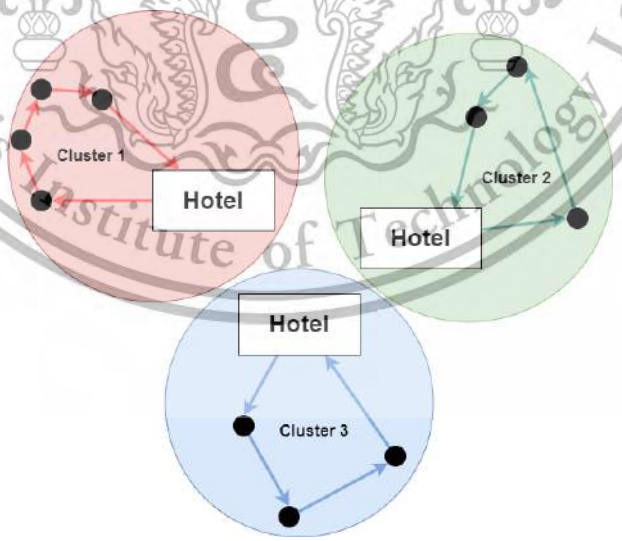


Figure 5.4: K-means clustering with traveling salesman problem approach

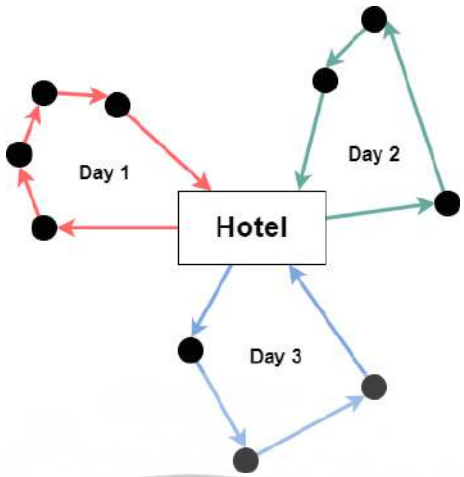


Figure 5.5: Applied vehicle routing problem approach

5.3.4.2 ACO For Modified VRP

In this project, the ACO algorithm is employed to solve the modified VRP and generate itineraries from a list of places. Inspired by the behavior of ant colonies, the algorithm mimics the process of ants searching for optimal routes. Initially, the algorithm initializes the number of days for the trip and the best path variable. Then, a group of ants is created, each representing a potential itinerary. The ants move through the places, guided by pheromone trails and a heuristic based on the closeness of the nodes. At each step, the ants select the next node probabilistically, considering the pheromone levels and closeness factor. The algorithm updates the pheromone levels on the edges based on the paths chosen by the ants. It continues iterating for a defined number of iterations, allowing the ants to explore differently. The Algorithm 6 delineates how ACO has been implemented for solving the VRP.

The Algorithm 6's *select_next_node* function, as mentioned in Algorithm 7, describes the process of selecting the next node for each ant in the modified VRP. This function determines the optimal node for the ant to move to based on a combination of pheromone levels and a heuristic value. The function starts by checking if the *node_to_visit* list is empty, indicating that the ant does not have any possible places to visit. If so, the next node is set as the hotel. Otherwise, the function calculates the distance matrix and the corresponding closeness values based on the current time. The transition probabilities are then calculated by multiplying the pheromone levels with the closeness values raised to the power of a

Algorithm 6 ACO for VRP

```
Initialize num_days as the number of days in a trip
Initialize best_path as None
for iter ← 0, 1, ..., max_iteration do
    ants ← List of num_ants ants
    for each ant in ants do
        while not ant.node_to_visit_is_empty() and num_days > 0 do
            next_node, num_days ← select_next_node(ant, num_days)
            ant.move_to_next_node(next_node)
            update_edge_pheromone(ant.current_node, next_node)
        end while
        if ant.travel_path[−1] ≠ hotel then
            ant.move_to_next_node(hotel)
            update_edge_pheromone(ant.current_node, hotel)
        end if
    end for
    scores ← Calculate the score of the travel path of all ants
    best_index ← Index of the ant with the best score from ants
    if best_path is None or scores[best_index] < best_score then
        best_path ← ants[best_index].travel_path
        best_path_score ← ants[best_index].score
    end if
    update_global_pheromone(best_path, best_score)
end for
Create an itinerary from best_path
```

bias parameter, β. These probabilities are normalized to ensure a valid probability distribution. The next node is selected randomly from the remaining nodes using the transition probabilities. If the selected node does not satisfy certain conditions, the function attempts to find an alternative node. If no suitable node is found, the ant moves to the hotel, and the number of days is decremented. Finally, the function returns the selected next node and the updated number of days for the ant.

Algorithm 7 Next node selection function

```
β is the bias toward a heuristic(closeness)
function select_next_node(ant, num_days)
    if node_to_visitisempty then
        next_node ← hotel
    else
        distance_matrix ← calculate_distance(ant.current_time)
        closeness ← 1/distance_matrix
        transition_prob ← pheromone_matrix[current_node][node_to_visit]
     ∗closeness[current_node][node_to_visit]^β
        transition_prob ← transition_prob/sum(transition_prob)
        next_node ← Random node in node_to_visit based on
transition_prob
        if not ant.check_condition(next_node) then
            Try other nodes
            if No such node satisfies then
                next_node ← hotel
                num_days ← num_days − 1
            end if
        end if
    end if
    return next_node, num_days
end function
```

5.3.4.3 Feasible Node Conditions

The *check_condition* function, for which the pseudocode is provided in Algorithm 8, plays a crucial role in determining whether the next node is feasible to visit for each ant in the algorithm. This function assesses several conditions to ensure the feasibility of the next node.

First, it calculates the travel time required to reach the next node from the current node and retrieves the visit duration at the next node. By adding the

travel time and visit duration to the current time, the function determines the estimated arrival time at the next node.

Next, it checks if the arrival time plus the visit duration exceeds the user's preferred trip end time or the closing time of the next node. If either of these conditions is true, it means that the ant won't be able to complete the visit within the given time constraints, and thus, the function returns $False$.

Additionally, the function considers whether the next node is a restaurant and checks if the arrival time falls outside the designated meal time or if the next node is not a restaurant and the arrival time falls within the designated meal time. If any of these conditions are met, it implies that the ant's visit to the next node would conflict with the meal schedule, and the function returns $False$.

If none of the above conditions are met, the function concludes that the next node is feasible to visit and returns $True$. This allows the ant to proceed with the itinerary and move to the next node in its path.

By performing these checks, the *check_condition* function ensures that the ants only visit nodes that are within the allowed time limits, taking into account the user's preferences and any specific constraints related to meal times or closing times.

5.3.4.4 Update Pheromone

The pheromone update function described by the equation 5.1 is a common formulation used in ant colony optimization algorithms. This function updates the pheromone level on an edge from node i to node j at iteration $t + 1$.

The term $\tau_{ij}(t)$ represents the current pheromone level on the edge at iteration t. The first part of the equation, $(1 - \rho)\tau_{ij}(t)$, represents the evaporation of the pheromone on the edge. Here, ρ is the evaporation rate, and $(1 - \tau)$ scales down the existing pheromone level based on this rate. This simulates the natural decay

Algorithm 8 Check condition function of each ant

An ant keeps track of simulated current time and current day of the week
function CHECK_CONDITION(next_node)
  $travel\_time \leftarrow$ travel time for traveling from the current node to the next node
  $visit\_duration \leftarrow$ visit duration at the next node
  $arrive\_time \leftarrow current\_time + travel\_time$
  $open\_time \leftarrow$ opening time of the next node on the current day of the week
  $close\_time \leftarrow$ closing time of the next node on the current day of the week
  if $arrive\_time + visit\_duration >$ user preferred trip's end time **or**
   $arrive\_time + visit\_duration > close\_time$ **or**
   ($next\_node$ is a restaurant **and** $arrive\_time$ is not within meal time) **or**
   ($next\_node$ is not a restaurant **and** $arrive\_time$ is within meal time) **then**
    return False
  end if
  return True
end function

of pheromones over time. The second part of the equation, $\rho\tau(0)$, represents the pheromone deposit on the edge. $\tau(0)$ represents the initial pheromone value, which is typically a small positive constant. The term ρ scales this initial pheromone value to determine the amount of pheromone to be deposited on the edge.

By combining these two parts, the function effectively balances the decay of existing pheromones with the deposit of new pheromones. This update rule allows the pheromone levels to evolve dynamically throughout the algorithm's iterations. Over time, edges that are part of favorable paths are likely to accumulate higher pheromone levels, making them more attractive to future ant agents and guiding the exploration of the solution space.

The global pheromone update function 5.2 is a key step in the ACO algorithms. It updates the pheromone level on an edge from node i to node j at iteration $iter + 1$, taking into account the best-constructed path found so far, represented by $best_score$. The term $(1 - \rho)\tau_{ij}(iter)$ represents the evaporation of the existing pheromone on the edge, mimicking the decay of the pheromone over time. The term $\frac{\rho}{best\_score}$ corresponds to the amount of pheromone to be deposited, where a smaller $best_score$ indicates a more favorable and promising solution. By updating the pheromone levels based on the quality of the best path found, this

function guides the search process towards more optimal solutions, encouraging the exploration of promising edges that lead to improved paths.

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t) + \rho\tau(0) \tag{5.1}$$

$$\tau_{ij}(iter+1) = (1-\rho)\tau_{ij}(iter) + \frac{\rho}{best\_score} \tag{5.2}$$

5.3.4.5 Generated Path Evaluation

In evaluating the travel paths generated by the ACO algorithm, we calculate a score for each itinerary. The goal of the ACO algorithm is to minimize this score. The score is determined by subtracting the place visit score from the total travel time, as shown in the equation 5.3.

To calculate the place visit score, we consider the travel path visited by an ant, excluding the hotel node. The path is analyzed to determine the number of unique places visited and the frequency of different types of places, such as restaurants or attractions.

For each place in the path, we calculate a contribution to the score based on its category. If the place is a restaurant, the code considers the types of cuisine offered. The score is increased based on the rarity of the cuisine types visited relative to the total number of unique places. Similarly, for attractions, the score is influenced by the diversity of attraction types visited. The place visit score is multiplied by the number of unique places visited to emphasize the importance of exploring a wide variety of locations. By minimizing the score, the ACO algorithm aims to generate travel paths that prioritize efficient travel time while maximizing the diversity and quality of visited places.

$$F(travel\_path_a) = f_{place\_visit}(travel\_path_a) + f_{travel\_time}(travel\_path_a) \tag{5.3}$$

where:

$travel\_path$ = sequence of nodes that ant a visits

Algorithm 9 Place visiting score calculation function for a travel path

function f_{place_visit}(travel_path)
  $path \leftarrow travel\_path$ excluding a hotel
  $n\_places \leftarrow$ number of visited places in the path excluding a hotel
  $n\_attraction\_type, n\_cuisine\_type \leftarrow \{\}$
  $num\_shops, score \leftarrow 0$
  for each *place* **in** *path* **do**
    if *place* is a restaurant **then**
      for each $cuisine\_type$ **in** $place.cuisine\_types$ **do**
        if $cuisine\_type$ **not in** $n\_cuisine\_type.keys$ **then**
          $n\_cuisine\_type[cuisine\_type] \leftarrow 0$
        end if
        $score \leftarrow score + \frac{n\_places - n\_cuisine\_type[cuisine\_type]}{len(place.cuisine\_types)}$
        $n\_cuisine\_type[cuisine\_type] \leftarrow$
          $n\_cuisine\_type[cuisine\_type] + 1$
      end for
    else if *place* is an attraction **then**
      for each $attraction\_type$ **in** $place.attraction\_types$ **do**
        if $attraction\_type$ **not in** $n\_attraction\_type.keys$ **then**
          $n\_attraction\_type[attraction\_type] \leftarrow 0$
        end if
        $score \leftarrow score + \frac{n\_places - n\_attraction\_type[attraction\_type]}{len(place.attraction\_types)}$
        $n\_attraction\_type[attraction\_type] \leftarrow$
          $n\_attraction\_type[attraction\_type] + 1$
      end for
    end if
  end for
  return $-score$
end function

5.4 Deployment

For this project, we decided to use a public KMITL server with the following domain http://dev.se.kmitl.ac.th over a cloud service for the following reasons:

- Costs
- Limited Control
- Opportunity to learn and setup on-premise server manually

The public domain is hosted on a Ubuntu 22.0 LTS machine. Running a Linux server allows you to use and integrate other open-source software seamlessly. Moreover, Linux servers have better optimization and resource consumption, therefore making it easier for the machine to be running 24/7.

Three common building blocks when deploying a Python web application to production are:

- A web server (like nginx)
- A WSGI Application (Like Gunicorn)
- Your application (like django, in our scenario)

Figure 5.6 shows the communication between web server, WSGI application and the actual application.

Nginx is designed to be fast, efficient, and secure, so it's a better choice to handle incoming web requests when your website is on the public Internet and thus subject to large amounts of traffic (if you're lucky and your site takes off) and also to abuse from hackers. That's not to say that it automatically makes your site fast and secure, of course, but at least it means that you're starting with the right system. It's also much better at serving static files (like your CSS, JavaScript, images, and so on) than Django is, because that's what it was built for.

uWSGI is designed to receive incoming web requests and rapidly and efficiently delegate processing them to multiple worker processes, then collate the responses

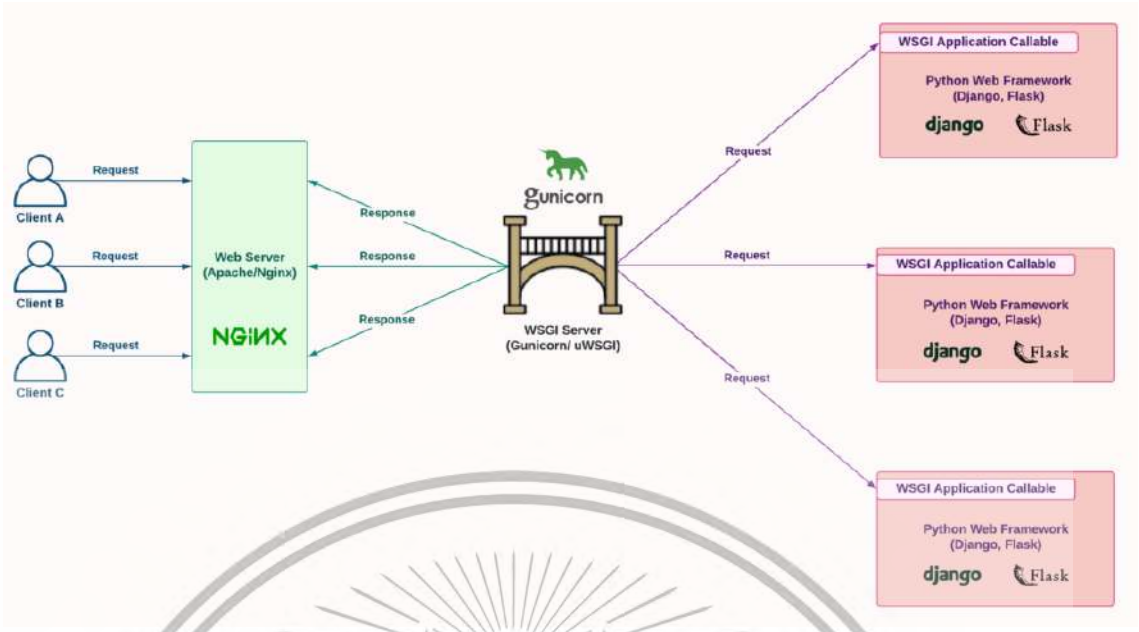


Figure 5.6: WSGI Architecture

and send them back to nginx.

uWSGI or Gunicorn takes care of everything which happens in-between the web server and your web application. This way, when coding up your Django application you don't need to find your own solutions for:

- communicating with multiple web servers
- reacting to lots of web requests at once and distributing the load
- keeping multiple processes of the web application running

If you directly point your web server to your application, it reduces the flexibility of your application. Since your web server now directly points to your web application, you are unable to swap out web stack components. Now, let's have a look at an example to make you clear about the applicability of WSGI. For instance, today you have decided to deploy your application using Gunicorn but after some years you decide to switch from Gunicorn to mod_wsgi. Now, in this case, you can easily switch to mod_wsgi without making any changes in the application or framework that implements WSGI. Hence, WSGI provides flexibility to your application.

Another reason for using WSGI is due to its scalability. Once your application is live, up and running there can be thousands of requests in your application. Hence, WSGI is capable of serving thousands of requests at a time. As we know, the WSGI server is responsible for handling the requests from the web server and takes decisions for carrying out the communication of those requests to an application framework's process. Here, we can divide the responsibilities among the servers for scaling web traffic.

To automate all the processes above, and continuously integrate and deliver our project, we decided to make use of Jenkins. Jenkins automates the process in which the developers are required to commit changes to the source code in a shared repository several times a day or more frequently. Every commit made in the repository is then built. This allows the teams to detect the problems early. Apart from this, depending on the Continuous Integration tool, there are several other functions like deploying the build application on the test server, providing the concerned teams with the build and test results, etc.

5.5 Docker and Docker Compose

5.5.1 Docker

We have made use of a Dockerfile that is used to build a Docker image for a Python web application. The Dockerfile has two parts - a builder stage and a final stage.

In the builder stage, the image is based on the Python 3.10-alpine image. The image sets the working directory to /usr/src/app and installs the necessary dependencies for psycopg2. The psycopg2 library is used to connect to a PostgreSQL database. The image then runs a linter (flake8) on the code and installs the application's Python dependencies using pip. The dependencies are saved to a directory called wheels.

In the final stage, the image is based on the Python 3.10-alpine image again. The image creates a user called "app" and sets the home directory to /home/ap-

p/web. The image then installs the libpq library, which is required for psycopg2. The image copies the Python dependencies from the wheels directory created in the builder stage and installs them using pip. The image copies the entire codebase into the container and runs the "collectstatic" command to collect static files. It then sets the owner of the files to the "root" user. Finally, the image sets the entry point to the "entrypoint.prod.sh" script.

The "entrypoint.prod.sh" script is responsible for starting the web server and any other necessary services. The script is also responsible for migrating the database schema and starting a Celery worker for background tasks. The script takes environment variables defined in the ".env" file, which are used to configure the application. For example, the script sets the database name, host, port, and user to the values defined in the ".env" file.

```
############
BUILDER
############

pull official base image
FROM python:3.10-alpine as builder

Set the user to root
USER root

set work directory
WORKDIR /usr/src/app

set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

install psycopg2 dependencies
RUN apk update
&& apk add postgresql-dev gcc python3-dev musl-dev

lint
RUN pip install --upgrade pip
RUN pip install flake8==3.9.2
COPY . .
RUN flake8 --ignore=E501,F401 ./iterthesisproject

install dependencies
COPY ./requirements.txt .
RUN pip wheel --no-cache-dir --no-deps --wheel-dir /usr/src/app/wheels -r
    ↪ requirements.txt

#########

FINAL
#########
```

```
pull official base image
FROM python:3.10-alpine

create directory for the app user
RUN mkdir -p /home/app

# create the app user
RUN addgroup -S app && adduser -S app -G app

create the jenkins user
RUN addgroup -S jenkins && adduser -S jenkins -G jenkins
create the appropriate directories
ENV HOME=/home/app
ENV APP_HOME=/home/app/web
RUN mkdir $APP_HOME
WORKDIR $APP_HOME

install dependencies
RUN apk update && apk add libpq
COPY --from=builder /usr/src/app/wheels /wheels
COPY --from=builder /usr/src/app/requirements.txt .
RUN pip install --no-cache /wheels/*

copy entrypoint.prod.sh
RUN ["chmod", "+x", "./entrypoint.prod.sh"]
COPY ./entrypoint.prod.sh .
RUN sed -i 's/\r$//g' $APP_HOME/entrypoint.prod.sh
RUN chown root:root $APP_HOME/entrypoint.prod.sh
RUN chmod +x $APP_HOME/entrypoint.prod.sh

copy project
COPY . $APP_HOME
RUN python3 manage.py collectstatic --noinput

chown all the files to the app user
RUN chown -R root:root $APP_HOME

RUN chown -R jenkins:jenkins $APP_HOME
# change to the app user
USER app
change to the jenkins user
USER jenkins
run entrypoint.prod.sh
ENTRYPOINT ["/home/app/web/entrypoint.prod.sh"]
CMD ["/home/app/web/entrypoint.prod.sh"]
```

5.5.2 Docker Compose

The following Docker Compose file defines a multi-container application with five services: web, db, pgadmin, nginx, and portainer.

```
version: "3.8"

services:
web:
...
db:
...
```

```
pgadmin:
...
nginx:
...
portainer:
...
```

The web service builds an image based on the Dockerfile.prod file in the ./iterthesisproject directory. The resulting image is tagged with am/web:latest. The command specifies that the web server will run Gunicorn and bind to port 8000. The volumes section maps the static_volume and media_volume volumes to directories within the container. The env_file section specifies that environment variables will be loaded from the .env file in the current directory. Finally, the depends_on section ensures that the db service is started before the web service.

```
web:
build:
context: ./iterthesisproject
dockerfile: Dockerfile.prod
image: anirudhmakhanakmitl/web:latest
command: gunicorn iterthesisproject.wsgi:application --bind 0.0.0.0:8000
volumes:
- static_volume:/home/app/web/staticfiles:z
- media_volume:/home/app/web/mediafiles
expose:
- 8000
env_file:
- ./.env
depends_on:
- db
```

The db service specifies an image based on postgres:13.0-alpine. The volumes section maps the postgres_data volume to the container's /var/lib/postgresql/-data/ directory. The env_file section specifies that environment variables will be loaded from the .env.prod.db file in the current directory.

```
db:
image: postgres:13.0-alpine
volumes:
- postgres_data:/var/lib/postgresql/data/
env_file:
- ./.env.prod.db
```

The pgadmin service specifies an image based on dpage/pgadmin4. The environment section specifies default values for the email and password of the pgadmin user. The ports section maps port 5050 on the host machine to port 80 on the

container. The depends_on section ensures that the db service is started before the pgadmin service.

```
pgadmin:
image: dpage/pgadmin4
environment:
- PGADMIN_DEFAULT_EMAIL=pgadmin4@pgadmin.org
- PGADMIN_DEFAULT_PASSWORD=pgadmin4
ports:
- "5050:80"
depends_on:
- db
```

The nginx service builds an image based on the Dockerfile file in the ./nginx directory. The volumes section maps the static_volume and media_volume volumes to directories within the container. The ports section maps port 1337 on the host machine to port 80 on the container. The depends_on section ensures that the web service is started before the nginx service.

```
nginx:
build: ./nginx
volumes:
- static_volume:/home/app/web/staticfiles
- media_volume:/home/app/web/mediafiles
ports:
- 1337:80
depends_on:
- web
```

Finally, the portainer service runs the Portainer container management system. The image used is portainer/portainer-ce:latest. The ports section maps port 9000 on the host machine to port 9000 on the container. The volumes section maps the /var/run/docker.sock file on the host machine to the same file within the container. This allows the Portainer container to interact with the Docker daemon on the host machine and manage other containers.

```
portainer:
image: portainer/portainer-ce:latest
ports:
- 9000:9000
volumes:
- /var/run/docker.sock:/var/run/docker.sock
```

In summary, this docker-compose.yml file defines several services that work together to create a web application environment. The web service runs the Django

application using Gunicorn as the server. It depends on the db service, which runs a PostgreSQL database. The pgadmin service provides a web interface for managing the PostgreSQL database. The nginx service serves as a reverse proxy and serves static and media files. Finally, the portainer service provides a web interface for managing the other containers.

5.6 CI/CD Jenkins Pipeline

5.6.1 CI/CD Design

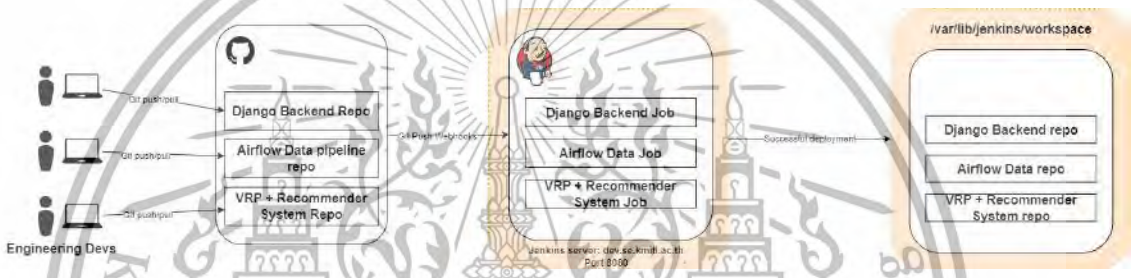


Figure 5.7: CI/CD Architecture

Continuous Integration/Continuous Deployment (CI/CD) is an essential process for software development that helps teams to deliver software faster and with higher quality. Jenkins, GitHub, and Ubuntu Server can be used together to create an efficient and effective CI/CD pipeline.

The process begins with the developer pushing code changes to the GitHub repository. Jenkins monitors the repository for any new changes and automatically starts the build process when a change is detected. The build process involves compiling the code, running tests, and generating artifacts.

Once the build process is complete, Jenkins deploys the code to a test environment on the Ubuntu Server. This environment allows the team to test the changes and ensure that they are working as expected. If any issues are found, the team can quickly make changes and push them back to the GitHub repository.

When the changes are ready for production, Jenkins deploys the code to the production environment on the Ubuntu Server. This process ensures that the software is deployed quickly and efficiently, without any downtime or disruptions.

To ensure that the process is secure, Jenkins and GitHub are configured to use

secure connections and credentials. Additionally, the Ubuntu Server is configured with appropriate security settings and permissions to protect against unauthorized access. In our project, the CI/CD pipeline was implemented using Jenkins, an open-source automation server. The pipeline consisted of several stages, including "Build," "Deploy," "Setting up Django," "Test Accounts," "Test Places," "SonarQube Analysis," and "SonarQube Quality Gate."

During the "Build" stage, the project was built using Docker containers specified in the docker compose file. The "Deploy" stage involved starting the Docker containers and making the entrypoint file executable.

The "Setting up Django" stage performed essential setup tasks specific to the Django framework, such as running database migrations, collecting static files, etc.

The "Test Accounts" and "Test Places" stages executed Django tests for the respective modules to ensure their functionality and detect any issues or bugs.

SonarQube [19], a popular static code analysis tool, was integrated into the pipeline for code quality and security analysis. In the "SonarQube Analysis" stage, the SonarQube scanner was used to analyze the project codebase, detecting potential bugs, vulnerabilities, and code smells. The results were then evaluated against predefined quality standards.

Finally, in the "SonarQube Quality Gate" stage, the pipeline waited for the Quality Gate to complete, which involved evaluating the project against defined quality criteria. If the Quality Gate status was not "OK," indicating a failure to meet the quality standards, the pipeline was aborted.

By incorporating SonarQube into our CI/CD pipeline, we ensured that our codebase was continuously analyzed for quality issues, enabling us to identify and address potential problems early in the development process. This helped us maintain code quality, improve overall software reliability, and deliver a more

robust and secure application to our users. This code represents a Jenkins pipeline script, which is used for automating the build, deployment, testing, and analysis processes in a software development project. Let's go through the different stages and their respective steps:

Build Stage

This stage is responsible for building the project. It consists of two steps:

- `sh 'whoami'`: Prints the username of the user executing the Jenkins job.
- `sh 'docker-compose -f Docker-compose.prod.yml build'`: Builds the Docker containers specified in the `docker-compose.prod.yml` file.

Deploy Stage

This stage handles the deployment of the project. It includes a single step:

- `sh 'docker-compose -f Docker-compose.prod.yml up -d'`: Starts the Docker containers specified in the `docker-compose.prod.yml` file in detached mode.

Setting up Django Stage

This stage performs setup tasks specific to the Django framework. It includes three steps:

- `sh 'docker-compose -f Docker-compose.prod.yml exec -T web python manage.py makemigrations -noinput'`: Executes the Django `makemigrations` command to generate database migration files.
- `sh 'docker-compose -f Docker-compose.prod.yml exec -T web python manage.py migrate -noinput'`: Applies the pending database migrations.
- `sh 'docker-compose -f Docker-compose.prod.yml exec -T web python manage.py collectstatic -no-input -clear'`: Collects and updates static files for production.

Test Accounts Stage

This stage runs the tests for the `account` module. It includes a single step:

- `sh 'docker-compose -f Docker-compose.prod.yml exec -T web python manage.py test account'`: Executes the Django tests for the `account` module.

Test Places Stage

This stage runs the tests for the `places` module. It includes a single step:

- `sh 'docker-compose -f Docker-compose.prod.yml exec -T web python manage.py test places'`: Executes the Django tests for the `places` module.

SonarQube Analysis Stage

This stage performs the SonarQube analysis of the project. It includes a single step:

- `script`: Specifies a block of script to be executed.
- `def scannerHome = tool 'sq1-scanner'`: Sets the `scannerHome` variable to the SonarQube scanner tool.
- `withSonarQubeEnv()`: Sets up the environment for running SonarQube analysis.
- `sh "${scannerHome}/bin/sonar-scanner"`: Executes the SonarQube scanner to perform the analysis.

Chapter 6

Results

6.1 Frontend

6.1.1 Authentication

6.1.1.1 Register

Users need to fill in their names, birth dates, email addresses, and passwords. After completely registering, user can proceed to login.

Figure 6.1: Register Page - Desktop

Figure 6.2: Register Page - Mobile

6.1.1.2 Login

In the login page, users need to provide the correct credentials, including email and password.

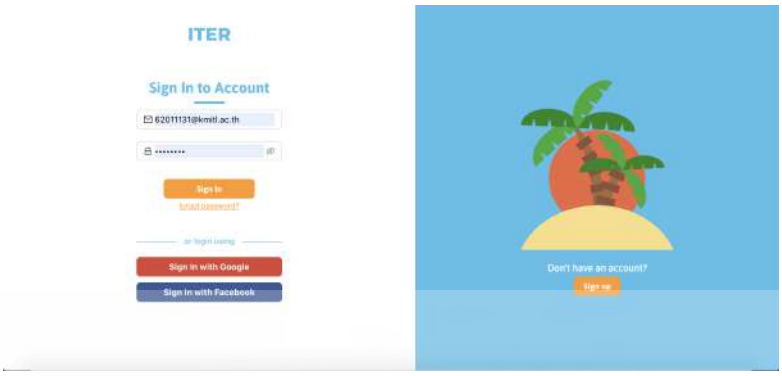


Figure 6.3: Login Page - Desktop

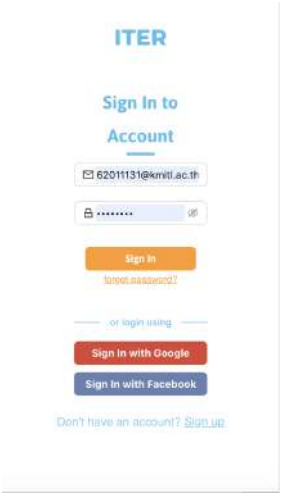


Figure 6.4: Login Page - Mobile

6.1.2 Itinerary Creation Feature

6.1.2.1 Personalized Itinerary

Users input their desired destinations, travel dates, and preferences, and the system uses this information to generate an itinerary tailored to their specific needs. The following questions wil be dynamic, based on the previous questions and filter out unrelated questions

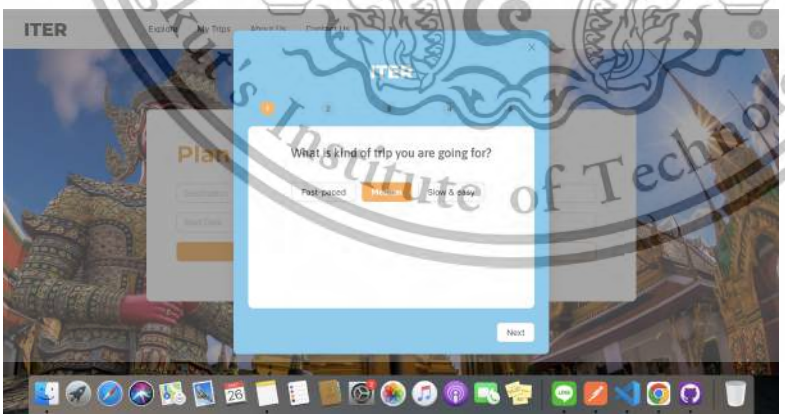


Figure 6.5: Personal Question - Desktop

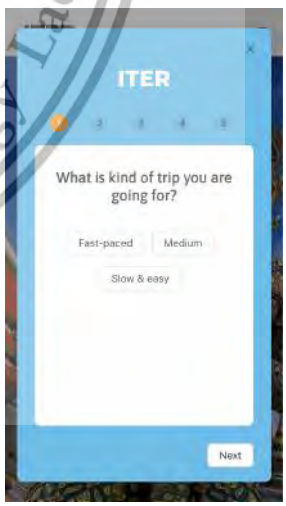


Figure 6.6: Personal Question - Mobile

6.1.2.2 Blank Itinerary

Alternatively, users can choose to create a blank itinerary, choose their own travel plans. They can manually add destinations, restaurants, and accommodations. This feature offers maximum flexibility and control to the user.

Figure 6.7: General Question - Desktop

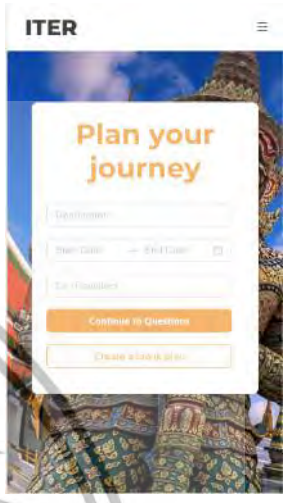


Figure 6.8: General Question - Mobile

6.1.3 Itinerary Display Feature

6.1.3.1 Timeline View

Once an itinerary is created, users can view their plan in a timeline format. This view demonstrate a sequence of agendas in chronological order, including place details, contact information, travel time, and etc. A navigational panel on the left provides an effortless way to jump to a specific date within the itinerary. Additionally, the "Go to Top" button facilitates swift navigation back to the top of the page, enhancing user experience.

6.1.3.2 Calendar View

Itinerary is presented in a calendar format where events correspond to their specific dates. This interface also includes drag-and-drop functionality for individual items and allows for item resizing, as well as warning when events are overlapped. By clicking on the side tab, users can easily navigate and zero in on a specific day's plan.

Figure 6.9: Timeline View - Desktop

Figure 6.10: Timeline View - Mobile

Figure 6.11: Timeline View(2) - Desktop

Figure 6.12: Timeline View(2) - Mobile

6.1.3.3 Map View

The map view provides geographical visual representation, incorporating pin markers for distinct places and visualizes connections between various locations, providing a spatial context for the planned trip.

6.1.3.4 My Trips

Users can view all their created itineraries, as well as duplicate and delete. Each itinerary can also be accessed from this section.

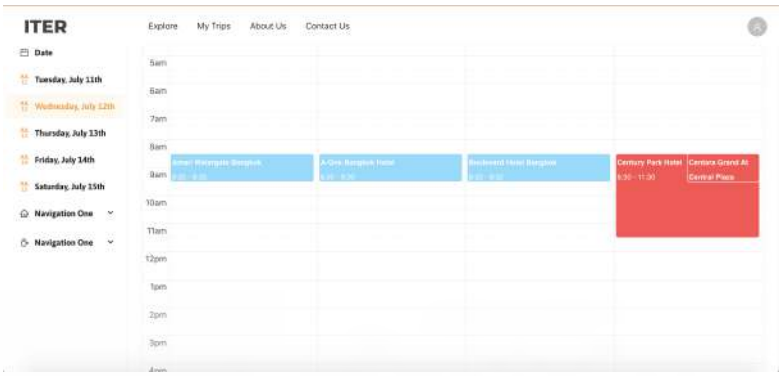


Figure 6.13: Calendar View - Desktop

Figure 6.14: Calendar View - Mobile

Figure 6.15: Map View - Desktop

Figure 6.16: Map View - Mobile

6.1.4 Itinerary Manipulation Feature

6.1.4.1 Add Place

Users can click on the calendar which directs them to the 'Explore' page. From here, they can select the desired place and add it directly to their itinerary. Alternatively, places can also be added via the 'Timeline' view by navigating to the top of the itinerary page and following a similar process.

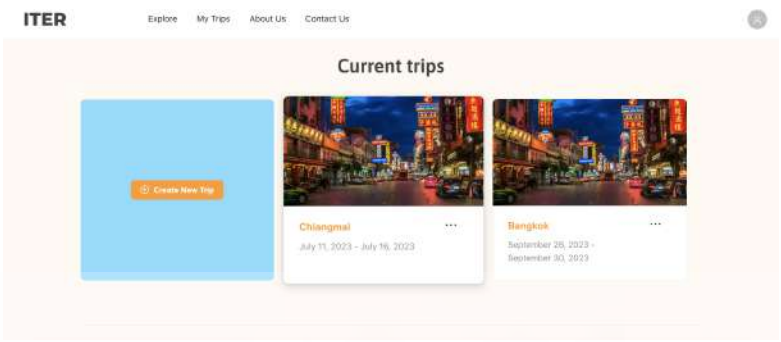


Figure 6.17: My Trips - Desktop

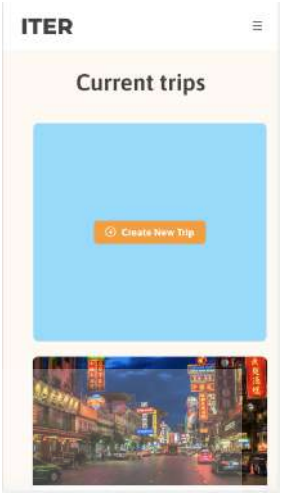


Figure 6.18: My Trips - Mobile

Figure 6.19: My Trips(2) - Desktop

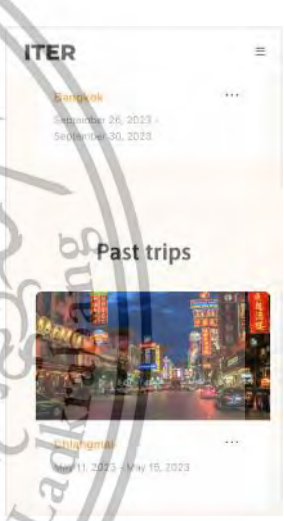


Figure 6.20: My Trips(2) - Mobile

6.1.4.2 Edit and Delete Place

Editing or deleting a place from the itinerary can be done from either the 'Calendar' or 'Timeline' views. In the 'Calendar' view, clicking on an event item opens an edit modal where dates and times can be modified. Deletion of the place can also be carried out from this modal. Similarly, in the 'Timeline' view, the ellipsis button on an itinerary item opens a menu providing the option to edit or delete the place, and following a similar process.

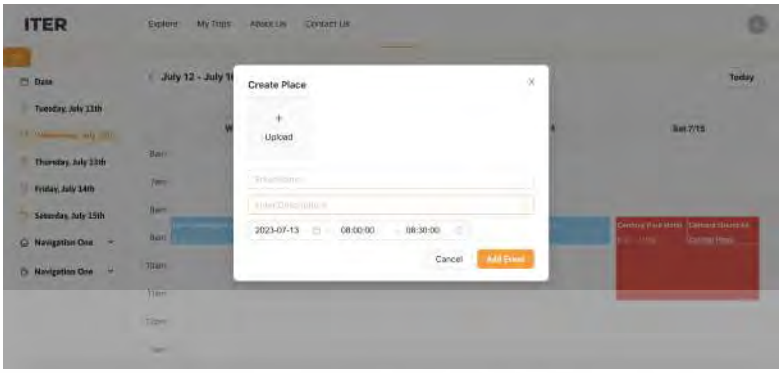


Figure 6.21: Add Place - Desktop

Figure 6.22: Add Place - Mobile

Figure 6.23: Edit Delete Place - Desktop

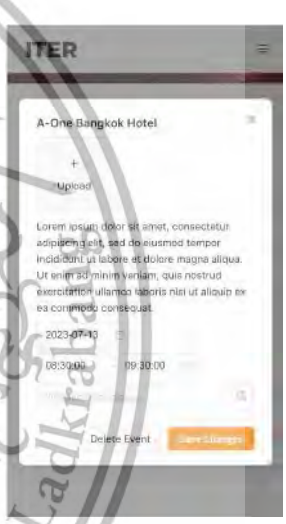


Figure 6.24: Edit Delete Place - Mobile

6.2 Backend

6.2.1 Container Management Dashboard

Portainer is an open-source container management tool that provides a web-based user interface (UI) for managing containerized applications. It offers an intuitive and user-friendly interface to manage Docker containers, Docker Swarm clusters, and Kubernetes clusters. Portainer simplifies the deployment, management, scaling, and monitoring of containers, making it easier for developers and system administrators to work with containerized environments.

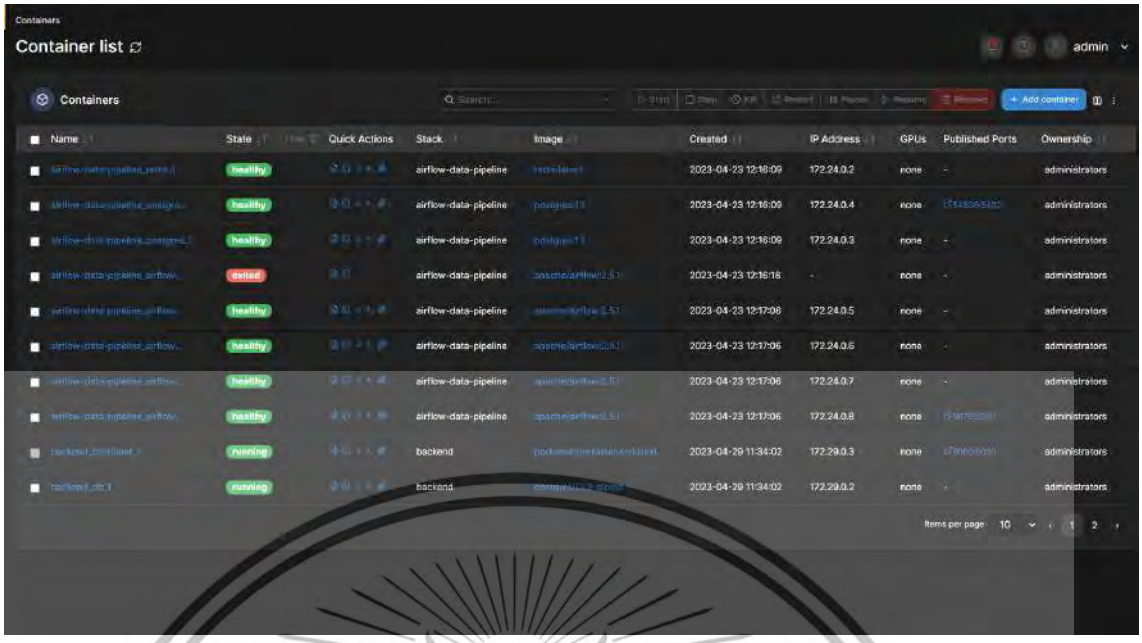


Figure 6.25: Containers

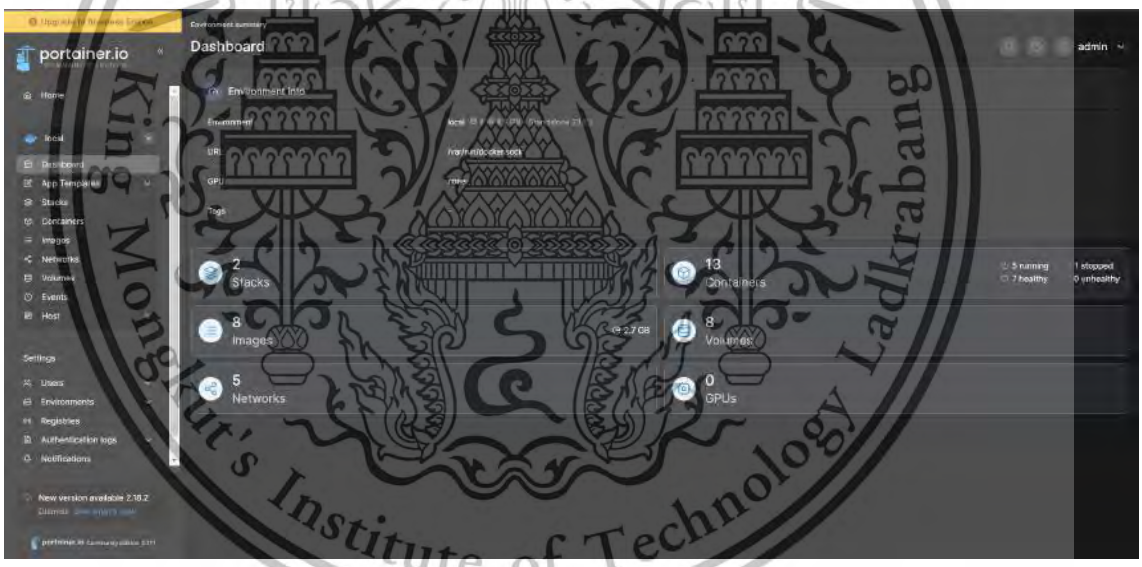


Figure 6.26: Overview

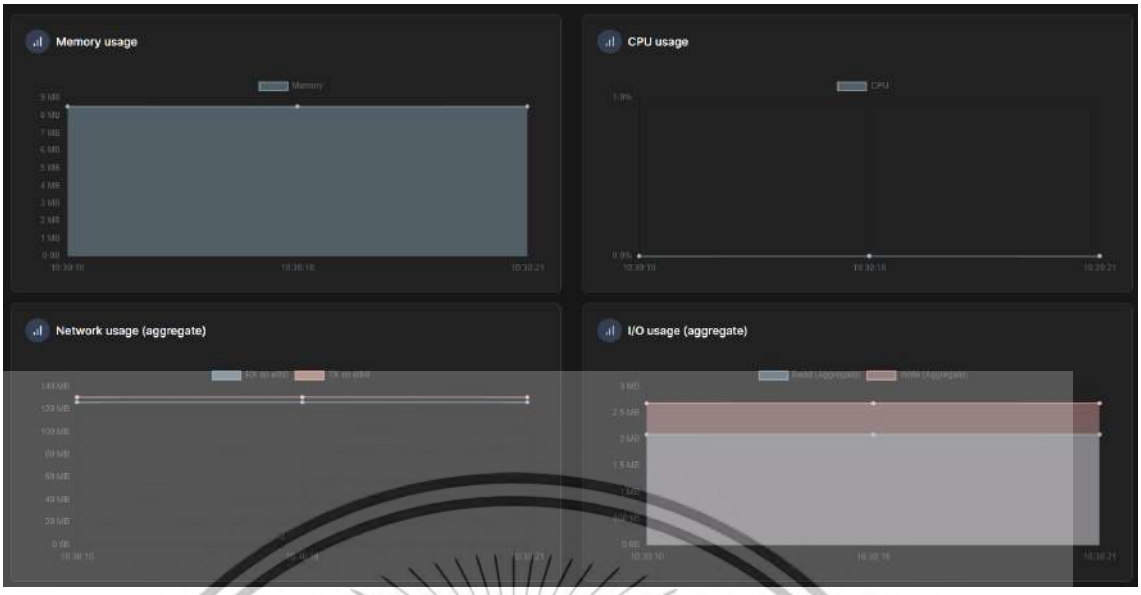


Figure 6.27: Resource Statistics

6.2.2 Jenkins Pipeline

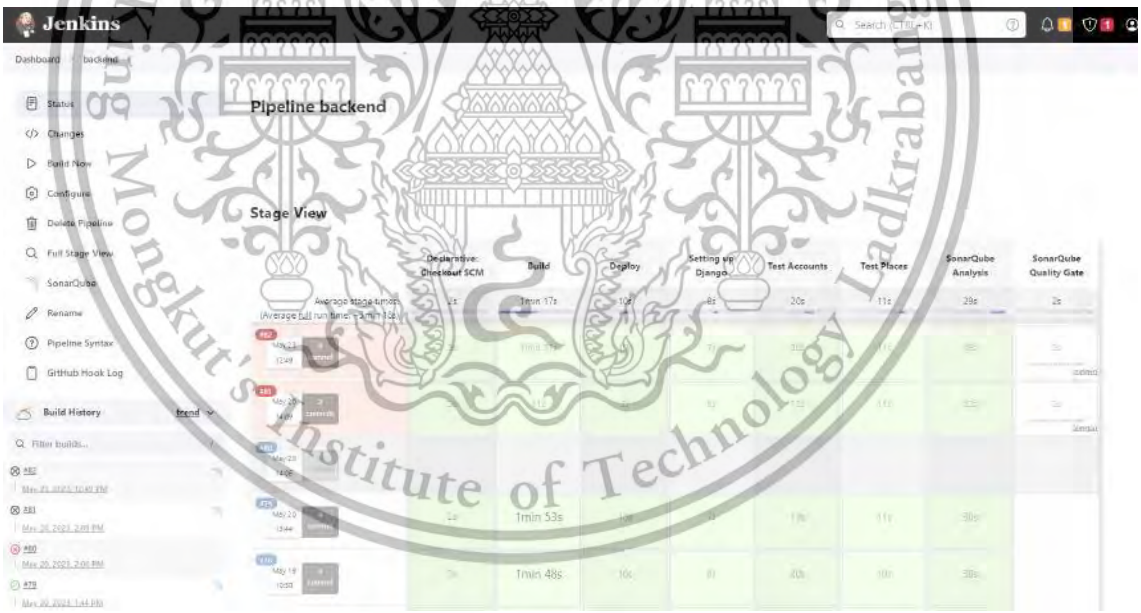


Figure 6.28: Jenkins Dashboard

Jenkins is an open-source automation server that facilitates the continuous integration and delivery (CI/CD) of software applications. It provides a wide range of features and plugins that make software development easier in several ways:

- Continuous Integration: Jenkins automates the process of integrating code changes from multiple developers into a shared repository. It monitors the

repository for changes and triggers builds automatically. This allows developers to identify integration issues early and ensures that the codebase is always in a working state.

- Build Automation: Jenkins automates the build process, including compiling source code, running tests, and packaging the application. It provides a platform-agnostic build environment and supports various build tools and languages. This automation saves time and effort, as developers no longer need to manually execute these tasks.

- Automated Testing: Jenkins allows for the execution of automated tests as part of the CI/CD pipeline. It integrates with popular testing frameworks and tools, enabling developers to run unit tests, integration tests, and other types of automated tests automatically. This ensures that code changes do not introduce regressions and maintains the quality of the software.

6.2.3 Recommender System

The recommender system returns a list of places with the attributes of place_id, place_name, attraction_types, category_code, latitude, longitude, and opening_hours. Listing 6.1 and 6.2 shows the sample output for the recommender system.


```
{
    "recommended_attractions": [
        {
          "place_id": "P03000191",
          "place_name": "Goods and Imitation Goods Museum",
          "attraction_types": [
            "Museums"
          ],
          "category_code": "ATTRACTION",
          "latitude": 13.68315046,
          "longitude": 100.5472721,
          "opening_hours": [
            {
              "day": "Sunday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
              "day": "Monday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
```

```
            "day": "Tuesday",
            "opening_time": "11:30",
            "closing_time": "22:30"
          },
          {
            "day": "Wednesday",
            "opening_time": "11:30",
            "closing_time": "22:30"
          },
          {
            "day": "Thursday",
            "opening_time": "11:30",
            "closing_time": "22:30"
          },
          {
            "day": "Friday",
            "opening_time": "11:30",
            "closing_time": "22:30"
          },
          {
            "day": "Saturday",
            "opening_time": "11:30",
            "closing_time": "22:30"
          }
        ]
      },
      .
      .
      .
    ]
}
```


Listing 6.1: Json response of attraction recommendation


```
{
    "recommended_restaurants": [
        {
          "place_id": "P08000010",
          "place_name": "Blue Elephant (Bangkok)",
          "cuisine_types": [
            "Thai",
            "Other"
          ],
          "category_code": "RESTAURANT",
          "latitude": 13.718591,
          "longitude": 100.521477,
          "opening_hours": [
            {
              "day": "Sunday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
              "day": "Monday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
              "day": "Tuesday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
```

```
            {
              "day": "Wednesday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
              "day": "Thursday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
              "day": "Friday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            },
            {
              "day": "Saturday",
              "opening_time": "11:30",
              "closing_time": "22:30"
            }
          ]
        },
        .
        .
        .
    ]
}
```

Listing 6.2: Json response of restaurant recommendation

6.2.4 Itinerary Scheduling System

The listing 6.3 represents the example of JSON response from the ACO itinerary generation module by inputting the list of recommended places. The JSON contains the itinerary information and a sequence of agendas.

```
{
    "destination": "bangkok",
    "start_date": "2023-07-11",
    "end_date": "2023-07-11",
    "start_time": "08:00:00",
    "end_time": "19:00:00",
    "plan": [
        {
            "place_id": "P02012763",
            "date": "2023-07-11",
            "arrival_time": "08:00:00",
            "leave_time": "08:00:00",
            "travel_time": {
                "P08002357": 16
            }
        },
        {
            "place_id": "P08002357",
            "date": "2023-07-11",
            "arrival_time": "08:16:00",
            "leave_time": "09:46:00",
```

```
            "travel_time": {
                "P03025433": 17
            }
        },
        {
            "place_id": "P03025433",
            "date": "2023-07-11",
            "arrival_time": "10:03:00",
            "leave_time": "12:03:00",
            "travel_time": {
                "P08013933": 19
            }
        },
        {
            "place_id": "P08013933",
            "date": "2023-07-11",
            "arrival_time": "12:22:00",
            "leave_time": "13:52:00",
            "travel_time": {
                "P03014388": 31
            }
        },
        {
            "place_id": "P03014388",
            "date": "2023-07-11",
            "arrival_time": "14:23:00",
            "leave_time": "16:23:00",
            "travel_time": {
                "P03013485": 9
            }
        },
        {
            "place_id": "P03013485",
            "date": "2023-07-11",
            "arrival_time": "16:32:00",
            "leave_time": "18:32:00",
            "travel_time": {
                "P02012763": 19
            }
        },
        {
            "place_id": "P02012763",
            "date": "2023-07-11",
            "arrival_time": "18:51:00",
            "leave_time": "18:51:00",
            "travel_time": {}
        }
    ],
    "co_travelers": [
        1
    ],
    "owner": 1
}
```


Listing 6.3: Json response of itinerary scheduling

6.2.4.1 Experimentation Results

For evaluating our ACO algorithm, we implemented two simple programs to solve the VRP and generate an itinerary as same as our ACO. The first program is using Genetic Algorithm (GA), which is a meta-heuristic algorithm. Another program is implemented using a depth-first search algorithm, which is the exact method to solve the VRP.

The experimentation was performed using five test cases as listed below:

- 10PA1D: Given set A of 10 places, generate a 1-day trip.
- 10PB2D: Given set B of 10 places, generate a 2-day trip.
- 20PC2D: Given set C of 20 places, generate a 2-day trip.
- 20PD2D: Given set D of 20 places, generate a 2-day trip.
- 30PE3D: Given set E of 30 places, generate a 3-day trip.

The table 6.1 presented compares the performance of three algorithms (ACO, DFS, and GA) for generating itineraries in different test cases. Each algorithm, except DFS, is executed ten times per test case to ensure reliable measurements. The table displays average scores and elapsed times, representing the quality and computational speed of each algorithm. However, it is worth noting that DFS was not evaluated for test cases with 20 or 30 places due to its significantly long runtime. In these cases, DFS requires an extensive amount of time, with the 20-place test case taking over an hour and the 30-place test case running indefinitely. The comparison helps in understanding the trade-offs between the algorithms and selecting the most suitable one based on specific requirements.

In our experimentation comparing the Ant Colony Optimization (ACO) algorithm with Depth-First Search (DFS) and Genetic Algorithm (GA) for solving Vehicle Routing Problems (VRP) to generate an itinerary, we observed distinct characteristics among these approaches. GA exhibited remarkable speed, quickly generating solutions, but unfortunately, the obtained results were unsatisfactory, lacking optimization. Moreover, the GA is not well-support the controlling the search behavior. Thus, it is much more difficult to deal with the conditions and

Test Case	Avg. Score(Lower is better)			Avg. Execution Time (s)		
	DFS	**GA**	**ACO**	**DFS**	**GA**	**ACO**
10PA1D	53.5	142	85.5	≈ 1	≈ 1.6	≈ 3.2
10PB2D	107.5	296	190	≈ 17.7	≈ 1.7	≈ 3.2
20PC2D	-	327.5	139	> 3600	≈ 3.2	≈ 7.2
20PD2D	-	378	133	> 3600	≈ 2.8	≈ 5.4
30PE3D	-	331	162	> 3600	≈ 3.5	≈ 13.5

Table 6.1: Comparison of ACO, DFS, and GA for VRP

time windows. On the other hand, DFS displayed a thorough exploration of the search space, leading to optimal solutions. However, its drawback was the significantly slower execution time, making it less practical for real-time scenarios. ACO, striking a balance between speed and solution quality, demonstrated acceptable performance, generating reasonably good results. Hence, ACO can be considered an efficient option for VRP itinerary generation, providing a compromise between speed and quality.

Chapter 7

Conclusion

7.1 Project Summary

To conclude, Iter is a travel itinerary application that leverages a recommender system to provide personalized recommendations to its users based on their preferences and past behavior. With the help of data obtained from the Tourism Authority of Thailand, Iter is equipped with a rich database of information to enhance the user experience.

Additionally, Iter incorporates its own Itinerary Generation module, which employs advanced techniques such as Vehicle Routing Problem (VRP) and Ant Colony Optimization. These algorithms optimize the itinerary creation process, taking into account various factors like travel constraints, user preferences, and available resources. By utilizing these intelligent algorithms, Iter ensures that the recommended itineraries are not only tailored to the user's preferences but are also optimized for efficiency and enjoyment.

In summary, Iter brings together the power of a robust recommender system, data from the Tourism Authority of Thailand, and its advanced Itinerary Generation module to deliver a comprehensive and personalized travel planning experience. Users can rely on Iter to effortlessly generate optimized travel itineraries that perfectly align with their preferences and constraints, making their journeys truly memorable and hassle-free.

7.2 Limitations and Future Works

Despite the progress made in developing Iter, there are certain limitations and areas for future improvement that we acknowledge. However, we remain committed to enhancing the application and exploring new avenues to expand its reach and impact.

One of the primary goals for the future is to deploy the Iter web application on cloud services. By leveraging the scalability and reliability offered by cloud platforms, we can ensure a seamless user experience even during peak usage periods. This deployment would also facilitate accessibility and availability of the application for users across the globe.

Furthermore, as part of our roadmap, we envision launching Iter to the public and commercializing its services. This step would enable us to cater to a wider audience and provide travelers with an invaluable tool for planning their journeys. We aim to gather user feedback and continuously improve the application based on their needs and preferences.

To enhance the scalability and manageability of the system, we plan to adopt container orchestration using Kubernetes. By transitioning from running individual containers to a Kubernetes cluster, we can achieve efficient resource utilization, automated scaling, and easier deployment management. This move will ensure a robust and scalable infrastructure to support the growing user base.

In addition to the technical aspects, we also recognize the importance of building a sustainable business model for Iter. We intend to explore different avenues for generating revenue, such as partnerships with travel service providers, targeted advertisements, and premium subscription plans. This would allow us to invest in the continual improvement and expansion of Iter's features and services.

To support these ambitious plans, we will actively seek funding opportunities, including grants and investments. Securing adequate funding will empower us to further enhance the application's functionality, refine the recommendation algorithms, and explore new collaborations to provide an unparalleled travel planning experience.

In conclusion, while Iter has shown great potential, we are aware of its lim-

itations and are determined to address them. Our future endeavors include deploying the application on cloud services, launching it to the public, adopting Kubernetes for efficient management, building a sustainable business model, and seeking funding to accelerate the project's growth. With these strategic steps, we aim to transform Iter into the go-to travel itinerary application for travelers worldwide.

Bibliography

[1] *17 types of similarity and dissimilarity measures used in data science.* URL: https://towardsdatascience.com/17-types-of-similarity-and-dissimilarity-measures-used-in-data-science-3eb914d2681.

[2] Marco Dorigo. *Update Pheromones.* URL: http://www.scholarpedia.org/article/Ant_colony_optimization.

[3] THE PARTYING TRAVELER ELI. *Wanderlog: The Travel Planning app you've been waiting for.* 2020. URL: https://thepartyingtraveler.com/2020/11/04/wanderlog-travel-planning-app-youve-been-waiting-for/.

[4] Andries P. Engelbrecht. "Ant Algorithms". In: *Computational intelligence: An introduction.* Second. John Wiley, 2007.

[5] Andries P. Engelbrecht. "SACO: Path Construction". In: *Computational intelligence: An introduction.* Second. John Wiley, 2007.

[6] Andries P. Engelbrecht. "SACO: Pheromone Evaporation". In: *Computational intelligence: An introduction.* Second. John Wiley, 2007.

[7] *Flask Extensions.* Extensions Registry | Flask (A Python Microframework). URL: https://web.archive.org/web/20180517082208/http://flask.pocoo.org/extensions/.

[8] *Foreword*. Flask Documentation (0.10). URL: https://web.archive.org/web/20171117015927/http://flask.pocoo.org/docs/0.10/foreword.

[9] *Guide to data cleaning: Definition, benefits, components, and how to clean your data*. URL: https://www.tableau.com/learn/articles/what-is-data-cleaning#:~:text=tools%20and%20software-.

[10] A. Rajaraman J. Leskovec and J. D. Ullman. "Recommendation System". In: *Mining of massive datasets*. Third. Cambridge University Press, 2022.

[11] Thitima K. *Amazing Thailand eBook*. 2020. URL: https://promotions.co.th/smartphone/inapps-discount/answer-all-your-holiday-trips-with-the-amazing-thailand-ebook-app.html.

[12] Hajba Gábor László. *Website scraping with python: Using beautifulsoup and Scrapy*. Apress, 2018.

[13] Michalis Mavrovouniotis and Shengxiang Yang. "Ant colony optimization with self-adaptive evaporation rate in dynamic environments". In: *2014 IEEE Symposium on Computational Intelligence in Dynamic and Uncertain Environments (CIDUE)*. 2014, pp. 47–54. DOI: 10.1109/CIDUE.2014.7007866.

[14] Tiffany Ng. *UX Case Study: Simplifying Travel Planning*. Medium. 2023. URL: https://ngtiffanytw.medium.com/ux-case-study-simplifying-travel-planning-c2ebccaa02e3 (visited on 05/27/2023).

[15] Nostra. *Inspirock Review: does it really make planning travel easier?* URL: https://wanderlog.com/blog/2021/09/09/inspirock-review-does-it-really-make-planning-travel-easier/.

[16] Nostra. *Nostra map*. URL: https://www.nostramap.com/mapservice/.

[17] Septia Rani, Kartika Nur Kholidah, and Sheila Nurul Huda. "A development of travel itinerary planning application using traveling salesman problem and K-means clustering approach". In: *Proceedings of the 2018 7th International Conference on Software and Computer Applications* (2018). DOI: 10.1145/3185089.3185142.

[18] Redhat. *What is a CI/CD pipeline?* Redhat. n.d. URL: https://www.redhat.com/en/topics/devops/what-cicd-pipeline.

[19] SonarSource. *SonarQube.* 2023. URL: https://www.sonarsource.com/products/sonarqube/ (visited on 05/20/2023).

[20] Velotio Technologies. "An Introduction to Asynchronous Programming in Python". In: *Velotio Perspectives* (2021). URL: https://medium.com/velotio-perspectives/an-introduction-to-asynchronous-programming-in-python-af0189a88bbb.

[21] Paolo Toth and Daniele Vigo. *The vehicle routing problem.* Vol. 9. Society for Industrial and Applied Mathematics, 2002.

[22] Tripadvisor. *Travel better with the new Tripadvisor App.* URL: https://www.tripadvisor.com/app.

[23] Ondra Urban. *Is web scraping legal?* Apr. 2023. URL: https://blog.apify.com/is-web-scraping-legal/#:~:text=Web%20scraping%20is%20completely%20legal,intellectual%20property%2C%20or%20confidential%20data..

[24] *What is a data pipeline?* URL: https://www.ibm.com/topics/data-pipeline#:~:text=pipelines%20and%20IBM-,What%20is%20a%20data%20pipeline%3F,usually%20undergoes%20some%20data%20processing..

[25] *What is scraping: About price amp; web scraping tools: Imperva.* Dec. 2019. URL: https://www.imperva.com/learn/application-security/web-scraping-attack/#:~:text=Web%20scraping%20is%20the%20process,replicate%20entire%20website%20content%20elsewhere..