Created by:

Vidur Puliani, Gong Yifei, Li Jingmeng, Jiang Yanni, Donal Ngo

Content

Executive Summary

Business Background & Problem Description

In 2018, Singapore received about 18 million International tourists with an average stay of about 3.7 days per visit, making Singapore the 5th most visited city in the world and 2nd in the Asia-Pacific.

25-34 year olds were the largest (23%) age group of visitors and survey data has shown that most visitors within this age group are likely to have travelled using their own itinerary. With more than 200 attractions in Singapore, significant effort is required to find attractions that fit a traveller's interests, preferences and context. Besides finding the right attractions, it is extremely inconvenient to find the best route between attractions manually by searching every attraction on Google Maps.

Based on the above considerations, our project aims to simplify and ease the process of itinerary planning by filtering attractions to fit the user preferences and automatically search for the best travel route between attractions.

Project Objective & Scope

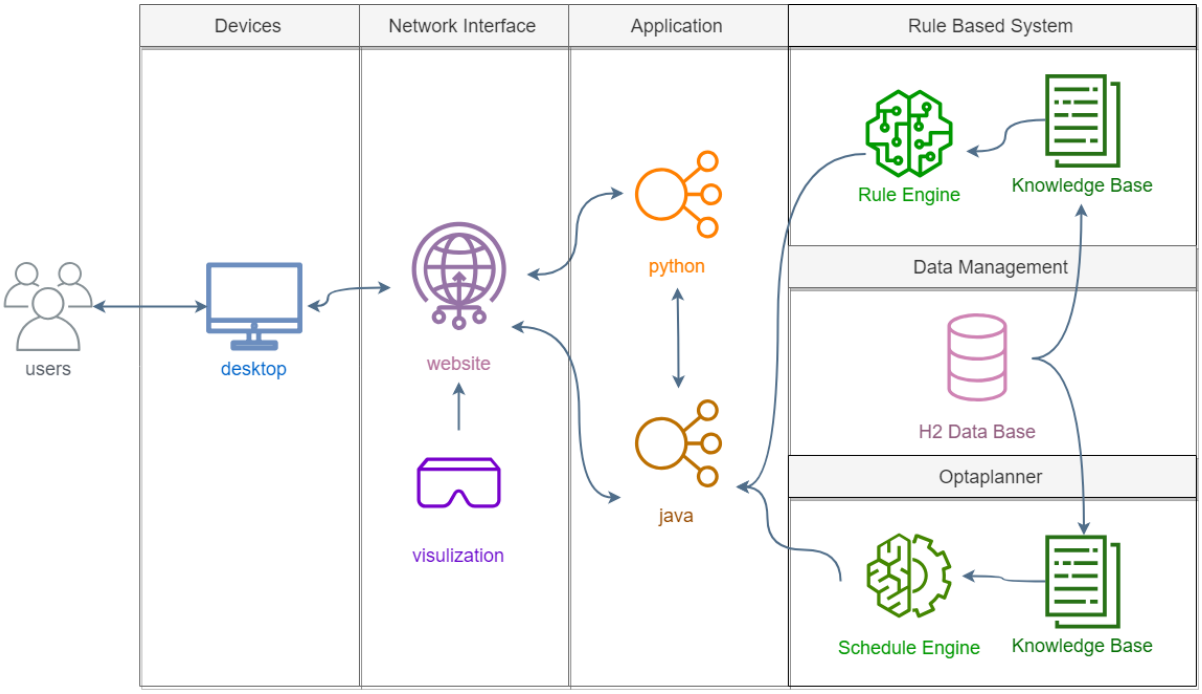
The system consists of two parts: firstly, attraction recommendation. The system recommends attractions to travellers based on their personal preferences through a ranking mechanism and enhances these recommendations with market basket analysis on the travel history of TripAdvisor users with similar interests through Drools Rule Engine.

Secondly, itinerary optimization. The system allows travellers to select from the recommended attractions and the selected attractions are then used to create an optimised travel itinerary by performing an optimization problem through optaplanner.

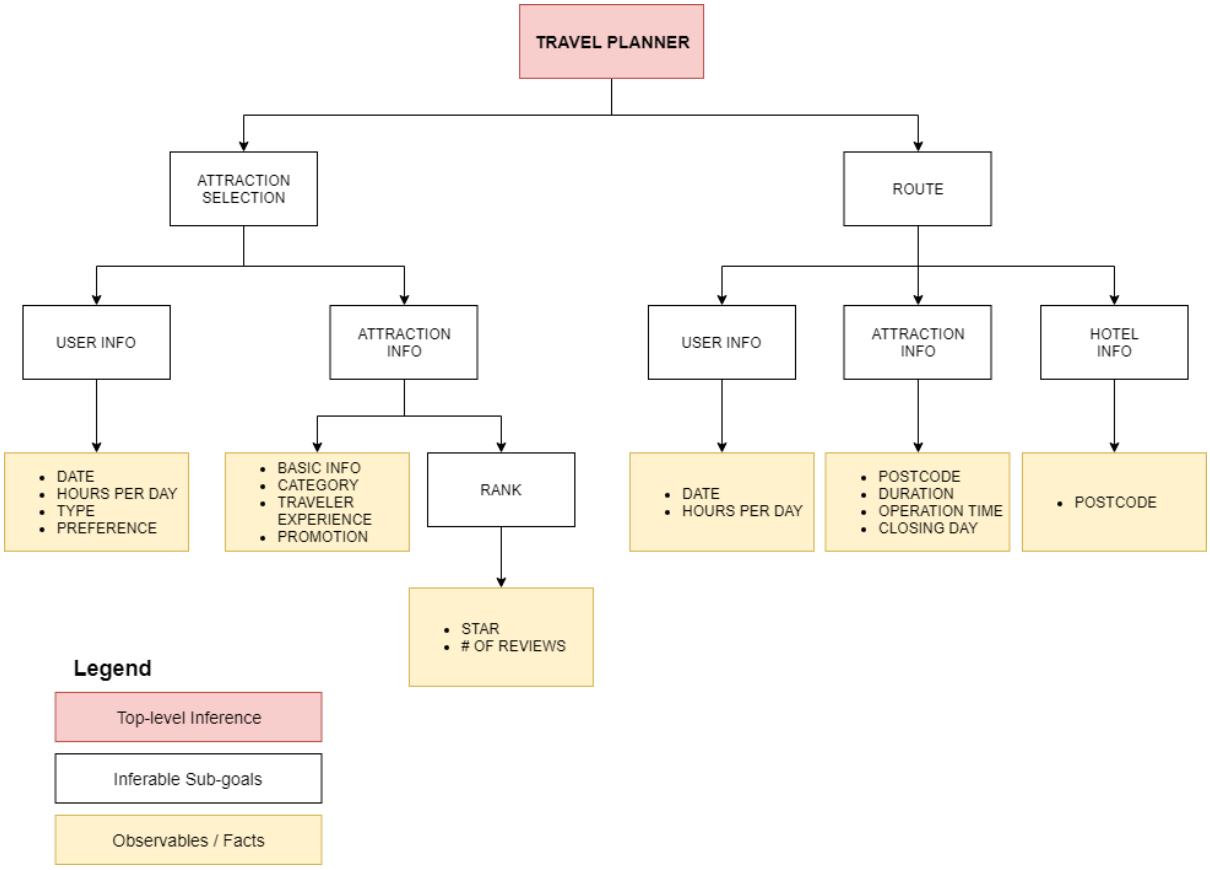
For the final result, travellers will get a full picture of how their trips in Singapore will be like: the best combinations of attractions for every day, directions between attractions, and even the promotions.

System Architecture

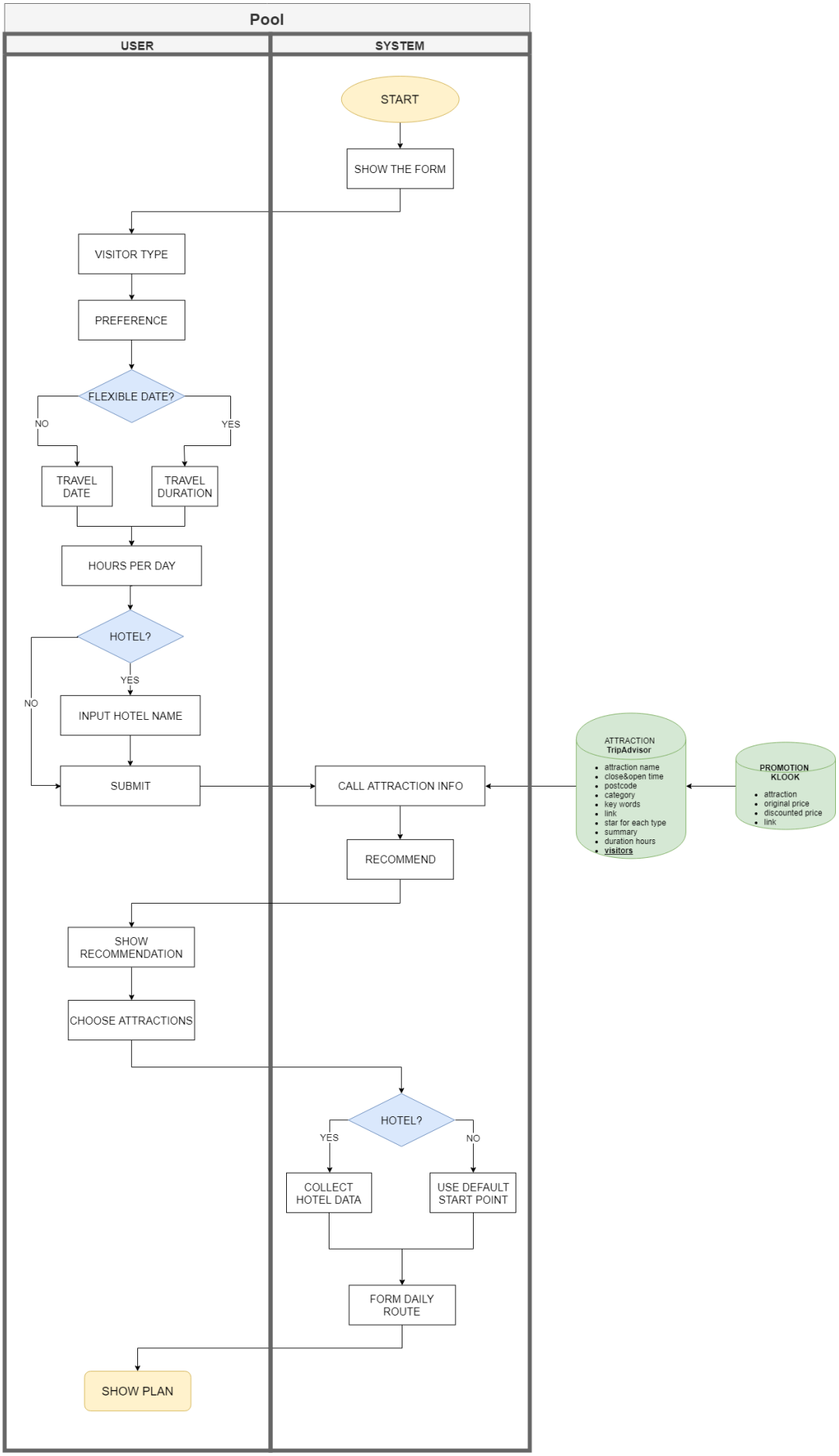
Architecture Diagram



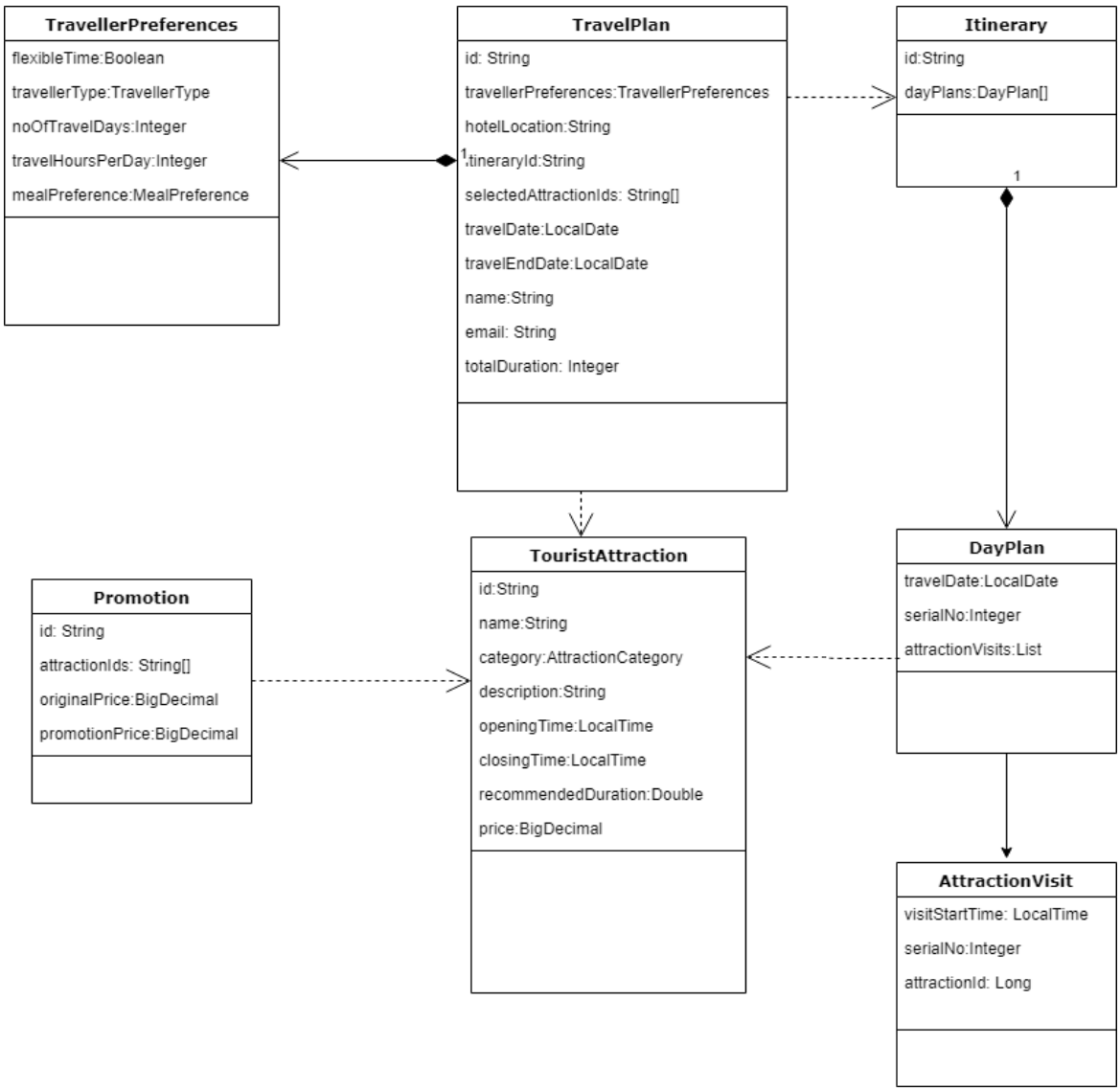
Inference Diagram



Activity Flowchart


Pool
USER
SYSTEM
START
SHOW THE FORM
VISITOR TYPE
PREFERENCE
FLEXIBLE DATE?
NO
YES
TRAVEL DATE
TRAVEL DURATION
HOURS PER DAY
HOTEL?
YES
NO
INPUT HOTEL NAME
SUBMIT
CALL ATTRACTION INFO
ATTRACTION
TripAdvisor
attraction name
close&open time
postcode
category
key words
link
star for each type
summary
duration hours
visitors
PROMOTION
KLOOK
attraction
original price
discounted price
link
RECOMMEND
SHOW RECOMMENDATION
CHOOSE ATTRACTIONS
HOTEL?
YES
NO
COLLECT HOTEL DATA
USE DEFAULT START POINT
FORM DAILY ROUTE
SHOW PLAN

Class Diagram



System Features

The SG Travel Planner has a number of features that can assist a user in planning their trip to Singapore an efficiently using their days. These functionalities have been briefly described below:

- **Recommendations:**
  The system has the capability to capture user travel preferences and interests to give personalised recommendations. Additionally, it augments these with suggestions based on travel history of users with similar travel interests.
- **Itinerary Optimisation:**
  The system allows user to select attractions from the provided recommendations that match the user's interests. The selected attractions are then used to plan an itinerary for the entire trip that optimally uses the travel days. Empty slots in the itinerary are intelligently filled with attractions based on the inferred user preferences.
- **Attraction Directions:**
  The system is integrated with Google Maps API to provide travel information from one location to another. It provides a general overview of the travel route and direction on various transport options based on the planned itinerary from the system.
- **Promotions:**
  Along with the directions and the day plans of the optimised itinerary, the system provides links for promotions from the popular Singapore promotions website Klook, allowing the user to find the best deals for attractions.

Knowledge Modelling

Knowledge Acquisition

Web Scraping with Selenium Web Crawler Bot

A web data extraction system is required to efficiently harvest data webdata by requesting unstructured HTML data as plain text which is then parsed to extract information of interest. The system will also need to identify “paths” and actions; web inputs eg. checkbox, buttons, to automatically move/crawl and extract useful information from different page to build on the knowledge base.

A Selenium-based web crawler bot is built to crawl and extract information from TripAdvisor and Klook. Based on the objectives of the project, a web scraper is developed to extract knowledge base which have been described in more detail below:

TripAdvisor KB

Fields	Sub-Fields
Attraction Name	-
Attraction Categories	-
Postal Code	-
Opening Hours	-
Closing Hours	-
Recommended Duration	-
Attraction Description	-
Attraction Keywords	-
User Reviews	Username
	Rating
	Visit Date
	Traveller Type

Klook KB

Fields
Image URL
Promotion Name
Number of Bookings
Number of Reviews
Original Price
Discounted Price
Overall Rating
SEO
URL Link

Figure : TripAdvisor and Klook website extracted data

1) TripAdvisor - Attraction Information
   All Singapore attractions listed on TripAdvisor are scraped to present useful information to front-end users, as well as the knowledge base to fulfil trip planning and optimization requirements.

2) TripAdvisor - User Review
   A large database of all users reviews are scraped from attraction reviews for association analysis and rank scoring to provide the best recommendations to end-users.

3) Klook - Promotions
   This is a large data source of promotions for single attractions as well as discounted bundles in Singapore which was scraped and stored in our application database to be used for recommending promotions for the selected attractions.

Association Mining

From the user reviews of attractions, we extract the visited attractions list of all the users as below:

user1: Mustafa Centre, Takashimaya Singapore
user2: Chinatown, City Square Mall, Gardens by the Bay, Singapore Botanic Gardens
user3: Clarke Quay, Cloud Forest, Flower Dome, Gardens by the Bay, Singapore Zoo
……

We used Apriori to extract association knowledge from the list above. Apriori is an algorithm for frequent item set mining and association rule learning over transactional databases. It proceeds by identifying the frequent individual items in the database and extending them to larger and larger item sets as long as those item sets appear sufficiently often in the database. The frequent item sets determined by Apriori can be used to determine association rules which highlight general trends in the database: this has applications in domains such as market basket analysis.

In this case, the above list is treated as a list of market baskets. Each user represents a market baskets, and the attractions that the user has visited represent the items in the corresponding market basket. To implement the Apriori Algorithm, we used the Apriori package in R.

As there are over 20,000 users and only around 200 attractions, the list is highly sparse. To effectively extract the association rules that are most representative, we set the thresholds for support and confidence to 0.01 and 0.25. Examples of the association rules are as below:

{Statue.of.Raffles} => {Sri.Mariamman.Temple}
{Tanjong.Beach} => {Floating.Bridge.at.Siloso.Beach}
{Robertson.Quay} => {Cavenagh.Bridge}
……

These rules indicate that travellers who have visited the attractions on the left hand side are also likely to visit the attractions on the right hand side.

Ranking

Tripadvisor has the ranking of attractions based on all the user reviews. But we want to generate the ranking of attractions from different types of users. As we have the rating and the type of each user for each attraction, and we can calculate the amount of each

type of users for each attraction, then we have to look for a reasonable ranking criteria. Tripadvisor popularity ranking is based on the quality, recency and the quantity of reviews. It mentioned that a business just needs to have enough reviews to provide statistical significance and allow for a confident comparison to others. But in our case, more reviews mean that this attraction is more popular in travellers than others. So our ranking algorithm gives some weight to the quantity factor and ignores the recency factor as attractions do not change a lot over time.

Google Maps

We can integrate Optaplanner with Google Maps with Google Maps API when doing route planning. The latitude and longitude of attraction can be acquired from Google Maps API with the attraction name. Then we can use the latitude and longitude info to get the transit time with public traffic between every pair of attractions. Then we can fetch distance matrix and optimize the trips with Optaplanner. After planning, Google Maps API can also help to render the route planning result in Google Maps browser. Besides, Google Maps API can get the photo and direction url of each attraction.

Promotion Matching

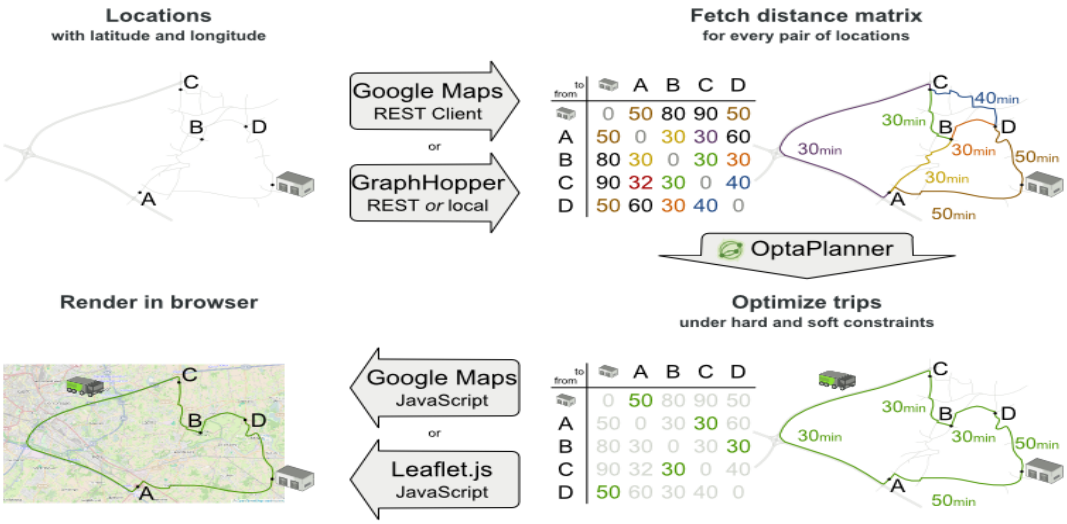
As mentioned in the previous sections, promotions and attractions were scraped from different sources. Thus, there was a need to create an automated process of identifying the attractions mentioned in a given promotion. Initial experiments with fuzzy string matching algorithms such as Levenshtein distance showed that several attractions were not accurately captured. Therefore, a moving substring window was used to iterate through all substrings(size equal to the attraction name) of promotion name and use fuzzy string matching to find attractions that cross a score threshold.

Distance Matrix

For the route planning, we have to integrate Optaplanner with real maps. Google Maps can help to calculate the distances between every pair of attractions, then we can generate the distance matrix as below accordingly. After that, OptaPlanner can optimize the trips according to the distance matrix.

Integration with real maps

Google Maps or GraphHopper (OpenStreetMap) calculate distances, OptaPlanner optimizes the trips.



Knowledge Representation

The modelled knowledge has been represented in a combination of H2 Embedded Database and KIE Drools rules. The database and drools rules are accessed using a Spring Boot application running on an embedded tomcat server. The knowledge base has been exposed as several REST APIs for easy access in the user interface. The H2 database contains the transformed data about attractions, promotions, travel distances and reviews. The KIE Drools rules were used to represent the recommendation rules derived from association mining of the reviews(described in detail in the earlier Association Mining Recommender section)

The two components are described in brief detail below

H2 Embedded Database

The H2 Embedded Database was chosen for its tight integration with Spring Boot application framework and ease of use for developing prototype applications. The Database design consists of several key entities that have been described in the class diagram under the System Architecture section of the report. This database is accessed using the JPA persistence API provided by Hibernate Object-Relational-Mapping tool. The data was imported into the database by using command line runners as import utilities that loaded the crawled JSON data files from TripAdvisor, Klook, Google Maps, and inserted it into the H2 database. The import utilities can be configured in the application.properties file using the boolean property 'trippingo.importData'.

KIE Drools

The rules derived from performing association mining on the user reviews were represented as KIE Drools rules inside the Spring Boot application. This allowed for easy integration with the rest of the application by exposing the associated recommendations service as a REST API.

All the association rules are divided into two types. One is the left hand side only has one attraction(short rules), the other is the left hand side has more than one attractions(long rules). These rules are stored in the drl files under the resources folder of the Spring Boot application.

An example of Drools rules is as below:

```
rule "AssociationRules_short__short_1"
  when
  $touristattraction:TouristAttraction(name=="The Helix Bridge")
  $recommendationBox: AssociationRecommendationDTO(associatedAttractionNames!
=null)
  then
  $recommendationBox.getAssociatedAttractionNames().add("Supertree Grove");
end
```

The "when" part of the rule is to judge if the inserted object is an instance of attraction and match the attraction with the left hand side of association rules. The "then" part of the rule is to return the right hand side of association rules, that is to return the associated attraction to be recommended.

Machine Reasoning

Intelligent Travel Recommendations

Preferences based Recommender

Preferences based Recommender Design

Through the data crawled from TripAdvisor, get the ranking and categories of each attraction and comments, scores and visitor type of all the reviewers of each attraction. According to these data, rank all the top 100 attractions for different visitor type. Store the ranking in different visitor type and categories of each attraction into our database.

APIs are created to make preferences based recommendations easier to implement. In the recommendation APIs, system need user's visitor type and preferred categories as filter to search results.

- Search the ranking of all the top 100 attractions of the specified visitor type.
- Select the attractions which belong to the specified categories.
- Based on the recommendation number i, return the first i-th attractions information

Preferences based Recommender Implementation

Our preference-based recommender use the ranking in different visitor types and categories of each attraction to make preferences based recommendation.

When user's visitor type and categories preference are given, the recommender by calling "http://localhost: 8001://recommendations?keywords= {keywords} &travellerType = {traveller Type }&count={recommendation_number}" to give recommended attractions.

Here, { keywords } is the user's preferred categories; { travellerType } is the user's visitor type; { recommendation_number } is the number of recommendations, normally it is set to 3.

Association Mining Recommender

Association Mining Recommender Design

Our association mining recommender is a kind of rule based system, which belongs to static reasoning. When attractions are recommended based on a user's preference, the association mining recommender will then insert the recommended attractions into the Drools Rule Engine. As the association rules are divided into two parts, the execution of the association rules is also divided into two parts.

Firstly, the execution of short rules. The recommender will insert the recommended attractions one by one into the Drools Rule Engine. The engine will then search only for the short rules that match the inserted attractions on the left hand side and execute the matched rules one by one.

Secondly, the execution of long rules. The recommender will generate all of the possible combinations of the recommended attractions and then insert the combinations into the Drools Rule Engine. The engine will then search only for the long rules that match the inserted combinations on the left hand side and execute the matched rules one by one.

When the preferences based recommendation have completed, we use the association mining recommender by calling "http://localhost:8001:///travelPlans/{id}/associatedAttractions?attractions={attr_names}" to give associated attractions.

Here, { id } is the travel plan's id, which will be generated after preferences based recommendation; { attr_names }is one or a list of attraction names from preferences based recommendation.

Itinerary Optimization

Optaplanner Design

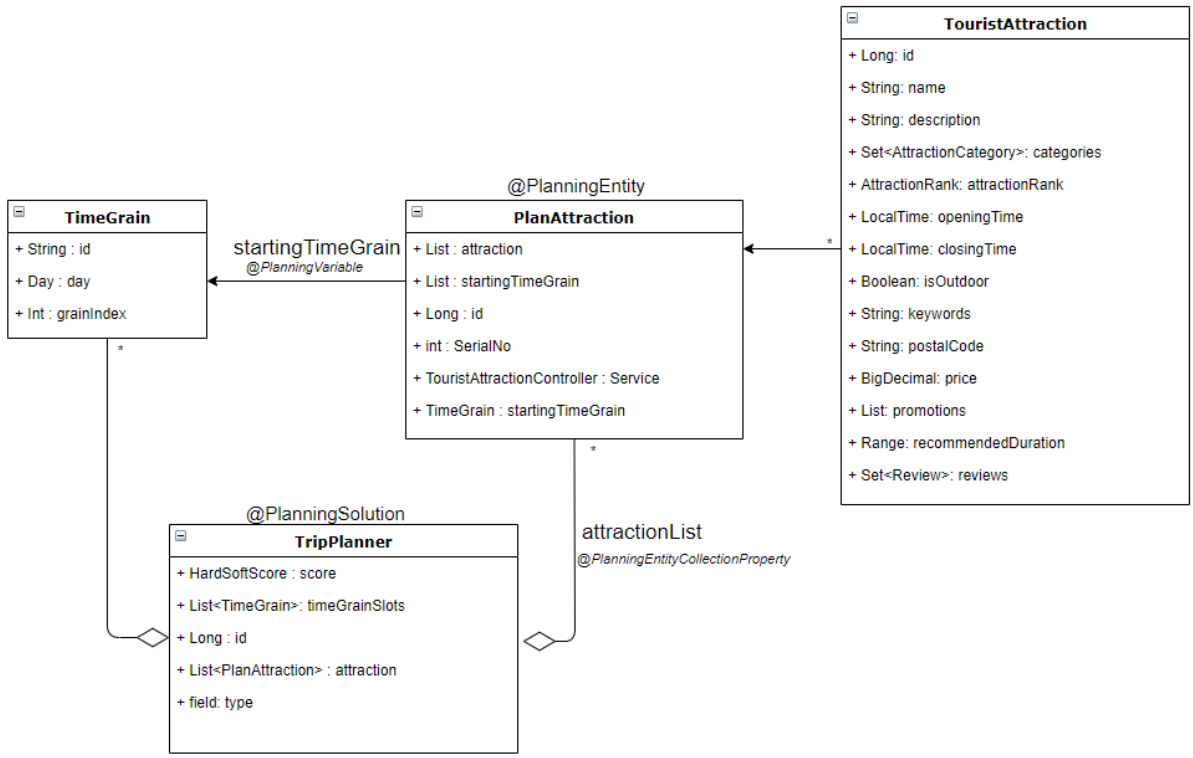


Figure : Optaplanner Class Diagram

Planning Solution

The Planning Solution class for the itinerary optimization is TripPlanner. The Planning Solution has two lists:

- Selected Attractions(List<PlanAttraction>):
  This is a list of Planning Entities that need to be optimised by the opta-planner. In this application, it represents the attractions selected by the user for their personalised itinerary. The Planning Entity variable is explained in more detail in the subsequent section.

- Time Slots(List<TimeGrain>):
  This is a list of Problem Facts that the Optaplanner algorithm needs to assign to a given Planning Entity. In this application, it refers to the time slots available for travel planning.

Planning Entity

The Planning Entity for optimizing itinerary is PlanAttraction which refers to the key information about the attractions that the user has selected from the provided recommendations. This class contains two key variables:

- Attraction Information (attraction): This variable contains all the information related to a given attraction in the application database such as its id, name, description, operating hours, recommended duration, etc.
- Time of visit (startingTimeGrain): The visit time is stored as the time slot of the day at which the traveller must reach the attraction according to the optimized travel plan.

The planning entity is also designed to maximize the amount of time spent per day on attractions within the given range of travel duration. When the sum of duration over the total number of days is not suffice, new attractions based on traveller type will be automatically filled for optimization. An attraction based on ranking will be removed for each iteration to find the most optimized solution to the user.

Planning Variable

The Planning Variable for this optimization problem is the time slots for the traveller in a given day. The time slots have been captured in a class called TimeGrain where each grain refers to a fifteen minute slot in a travel day. The search space window is limited by the starting and the ending slot determined by the user preference captured for the number of travel hours per day. Each time grain contains a grainIndex which refers to the aforementioned time slot, and a day variable which contains information about the day this particular time grain belongs to. The number of days are also derived from the captured user preference. The concept of time grains was used here to reduce the computational complexity and also convert time from a continuous variable to a discrete variable for the purpose of optimization.

Optimization Constraints

The optimization constraints have been implemented as Drools rules to utilise the built-in capabilities of Drools and Optaplanner to calculate score incrementally. The scores used in this optimization problem are a mix of Hard and Soft constraints which have been described in more detail below:

- Opening Hours (Hard Constraint) & Closing Hours (Hard Constraint)
    - Planned Timings must be within attraction operating timings
    - Penalize when Planned Hours is outside operating hours
- Duplicate Time Slots (Hard Constraint)
    - Penalize when more than 1 attraction is assigned to a single starting time grain
- Recommended Duration (Soft Constraint)
    - Balancing Travel Duration and Visit Hours
    - Give reward to maximize visit duration of attraction close to maximum recommended duration per attraction
    - Penalize to minimize travelling time to discourage to reduce the time travelling

Performance Comparison

The results were generated by running the optimization planning for 10 seconds over a fixed set of pre-loaded data for a total of 10 times. The following table shows the mean score generated when executed with different search algorithms.

Search	**Mean Score**
Hill Climbing	-5063232
Tabu Search	**-900002**
Simulated Annealing	-1002312
Late Acceptance	-1105157

The score is calculated by adding soft score and hard score scaled with a product of 1000, therefore hard score and soft scores can be determined based on their values. This is a good metric of comparison as it breaks any ties in hard score by differentiating two algorithms based on soft score. The performance comparison results show Tabu Search as the best algorithm for this optimisation problem. The optimisation algorithm uses a tabu entity ratio of 0.1 instead of a fixed entity tabu size. This allows the tabu algorithm to fine-tune its search based on the number of entities.

Limitations and Improvements

1) Attraction information and Promotions are limited to the day when the data is scraped from the website. Knowledge based should be periodically refreshed to ensure data relevancy to current trends.
2) UI can be enhanced for Across device Compatibility so users can use optimiser on the go.
3) Provide options to add itinerary onto personal calendar events for information retrieval offline
4) Optaplanner with parallel processing will provide a more optimal solution in planning within a shorter timeframe.
5) Results can be generated and provided on an email link in order to buy more time for optimization.
6) Live chat platform or Chatbot can be embedded into the website to strengthen the interaction with users.
7) This application is not only for travel planning in SIngapore, but also can be easily extended for other countries and regions or even cross country planning.

Creators and Contributors

Name	Student ID	Work Items
Vidur Puliani	A0198492L	Klook Web Scraping, Promotion-Attraction Matching, Spring Boot Application and Database design, Opta-planner integration, Recommendation User Interface
Gong Yifei	A0198495E	Data preprocessing, Prototype design, Diagrams, Implementation of rules, Itinerary interface, Video
Li Jingmeng	A0198484J	Data preprocessing, Association Mining, Association Knowledge Representation, Association Recommendation User Interface
Jiang Yanni	A0201097M	Data Preprocessing, Ranking Processing, DIstance Matrix Collection, Google Maps Interaction and Interface
Ngo Jinze Donal	A0198487A	TripAdvisor Web Scraping, Data Preprocessing, Opta-planner Integration, Promotion-Attraction Matching, Web Input interface