Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

Plan My Vacation

Distributed Systems Lab Project

Prashanth Balasubramanian, Erjon Paçarizi
balasubp@student.ethz.ch, erjonp@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich

Supervisors:
Georg Bachmeier, Gino Brunner
Prof. Dr. Roger Wattenhofer

April 7, 2017

Acknowledgements

We would like to thank Georg and Gino for helping us complete this project and Prof. Dr. Roger Wattenhofer for allowing us to pursue this project. With the weekly meetings, we ensured that a certain portion of the tasks were completed and thanks to Georg and Gino we were able to follow this weekly "sprint" type methodology of software engineering, to have an update for the application every week. Putting together the whole API and connecting the android application was by no means an easy task and would not have been possible without the help of our supervisors.

Abstract

This project describes a novel application that aggregates data together from the most popular websites/applications on the Internet, that one would use separately, while planning a trip (For example: Booking.com, AirBnb, TripAdvisor etc.). This project provides a framework to solve the cumbersome problem of manually planning a trip. All current applications focus on an individual aspect of trip planning and require extensive user input. With the Plan My Vacation application, user input is minimized to purely the essentials and the framework automatically provides suggestions to the user while learning from their actions. Ideally the application is aimed at users who have a specific budget or time frame in mind. The application accounts for budget flights and hotels and provides a thorough list of possible day to day activities that fit the given time frame. Challenges include processing a large amount of data to compile a suitable and convenient travel plan, ensuring that this data is valid, gathering this data and minimizing query time. Future improvements would include the addition of learning algorithms that observe user behaviour and optimize future trips based on past behaviour.

Contents

Chapter 1

Introduction

With the abundance of travel applications these days, end consumers have the option of planning each component of a trip with hundreds of options. For example: There are about 100 different popular applications/websites that allow a user to book flights and hotels, then there are applications that allow a user to book local transport, and mapping applications that allow users to plan routes between places that they would like to visit while travelling, i.e., each component of a trip can be planned individually if desired. This process is cumbersome and there is no application available that unifies all of these individual aspects of trip planning. Not only is there no way of making a comprehensive trip, but neither is there a way for a user to possibly know all the attractions that a new city has to offer. In this project we solve this exact problem by using data from the most popular websites, aggregating this data and then cleaning it and presenting useful information to the user in a neat and concise manner such that while planning a trip, a consumer does not have to spend days researching and reading reviews and can instead plan one in a couple of minutes.

1.1 Related Work

Skiplagged

Skiplagged[2] is a relatively new player in the flights industry but after some research we found that the flight data that is offered by Skiplagged is among the most comprehensive. Skiplagged was infact sued for undercutting flight prices[3] by exposing the concept of "hidden cities" that airlines do not want passengers to exploit. Naturally, this is something that we would like our users to take advantage of because ideally the target users for this application are those between the ages of 18 and 50, a majority of whom would travel on a budget. Minimizing flight prices was of the highest priority while selecting a source of data for flights and Skiplagged was the best option. The advantages and disadvantages of this API are outlined in Section 1.1.2 and 2.1.

Booking.com

Booking.com has risen to be the most popular hotel booking website, used by millions of end consumers worldwide. The data about hotels for a particular city available on Booking.com is comprehensive and exhaustive. This was naturally a choice for a data source for hotels for this project. Each hotel presents information about the hotel, location, review, photographs, rating, availability and price.

Airbnb

Airbnb is an application that aggregates home owners to rent out vacant rooms/apartments to travellers for a fee. This application has a user base of over 10 million and offers some excellent accommodation in foreign cities with trusted home owners, for excellent prices. Thus, while aggregating accommodation data, Booking.com and Airbnb were our primary and most important sources.

The main feature of our application includes the fetching and combining of sparse data about attractions/popular places in a city from multiple sources, into an intuitive form. The following are the sources used:

FourSquare

A relatively old platform for travellers to check-in, review and rate places of interest. FourSquare exposes a powerful and exhaustive public API that is exploited here[8].

TripAdvisor

One of the biggest players in the attractions industry, a platform that aggregates user reviews about cities all over the world and rates destinations and trips according to popularity. The data that TripAdvisor has is valuable and they do not expose this through a public API anymore and it is not possible to access their data without manually crawling the website. However, for this project we were able to gain access to their API through unconventional means[9].

Google Places

The largest and the most useful database of popular locations and reviews. We combine the Google Places API along with Google queries that a consumer would normally use to find places of interest that are not listed in the aforementioned sources[10].

Thus, by combining the three largest sources of attraction information, we were able to create a unique list of recommendations for the most relevant attractions.

1.2 Motivation

All of the above listed sources of information, each have their own shortcomings. However, it is already seen that out of all the biggest and most popular travel applications, each of these applications provide a method of planning a single, individual component of a trip but not an entire comprehensive trip. For example to plan a trip, the first thing one would do is look for the best flight options. This can be done through Skiplagged, SkyScanner etc. Once flight tickets are booked, the next logical step is to search for accommodation. One would prefer accommodation close to the center of the city, while someone else would prefer a quiet place outside the city, filtered by price or even luxury accommodation. In any case, the planner would visit at least two or three hotel booking sites to compare and match prices before deciding on a place of stay. Next, the planner would want to visit some interesting places in the city. This comes down to personal taste, i.e., some travellers prefer quiet, non-mainstream attractions while others prefer to visit the most popular tourist attractions. This is the most cumbersome part of planning a trip. Users now have data from TripAdvisor, FourSquare, Google and even Facebook to compare, match and find places that interest them. Thus, it is clear that in this entire process of trip planning, a user would spend a considerable amount of time visiting various websites and consulting different sources, before finalizing a trip. With all these sources of data around, we believe that this entire process of trip planning can be made intuitive, easy, fun and quick, by bringing together the most common elements of trip planning into a single comprehensive application.

Chapter 2

Flight Data

2.1 Getting data from Skiplagged

Skiplagged is a private provider of flight information, and they expose a simple JSON API endpoint that provides access to their database[1]. This by itself is not very useful and thus a Javascript wrapper is written around this endpoint, to access their database with three different parameters in order to get different categories of flights, namely :

- Shortest Flights
- Cheapest Flights
- Trips with the least number of layovers

On a high level, the wrapper is responsible for using the IATA code for the origin and destination, parsing the duration of the flight, converting this to the right time with respect to the user's timezone and setting the option of including or excluding hidden city trips.

2.2 Hidden City Ticketing

Hidden City Ticketing[6] can be explained with the help of a short example :

- You want to book a flight from point A to point B, let's say Zürich to New Delhi.
- Skiplagged will find you a one-way ticket from point A to point C with a stopover in B. For example, there may be a flight from Zürich to Nepal, with a stopover in New Delhi, that is cheaper than a direct flight to New Delhi.

[1] www.skiplagged/api/search.php

- Book a flight from point A to point C (Zürich to Nepal) and get off at New Delhi.

2.2.1 Disadvantages

- No checked bags are allowed
- Cannot book round trips. If you miss any part of your trip, the rest of the ticket is cancelled.
- If the stopover destination changes, there is nothing that can be done.
- You may need an additional visa.

2.3 Flight selection and ordering

The Javascript wrapper returns data in a formatted JSON object to the android application, as shown in Figure 2.1.

The server requires a three letter abbreviation of the airport codes, called the IATA code, e.g., Zurich = ZRH, Prishtina = PRN, etc. From the user input we have only the address of the user location and the place of destination. Therefore, before calling the API two things need to be done: Find the nearest airport from the point of departure and destination, find the abbreviation for that airport. For implementing this we use iatageo.com which is an API to get IATA code from latitude and longitude coordinates.

The server returns three flights: the shortest flight, the cheapest flight, and a flight with the least number of layovers. In some cases, all three flights are the same, and in this case the user sees only one flight. The button to book the flight redirects the user to the skiplagged.com booking page for that particular flight. Also when clicking one of the flight cards, the details for that flight are shown in a new screen like the flight number or full addresses as can be seen in Figure 2.2.

```
{
   "flightData":[
      {
         "price":"$371.37",
         "price_pennies":37137,
         "duration":"13 h ",
         "duration_seconds":46800,
         "departure_time":"Sunday, May 21st 2017, 06:20am",
         "arrival_time":"Sunday, May 21st 2017, 01:20pm",
         "legs":[
            {
               "airline":"TAP Portugal",
               "flight_number":"TP929",
               "duration":"2 h 45 Minutes ",
               "duration_seconds":9900,
               "departing_from":"Zurich, ZRH, Zurich, Switzerland",
               "departure_time":"Sunday, May 21st 2017, 06:20am",
               "departure_time_formatted":"2017-05-21T06:20:00+02:00",
               "arriving_at":"Lisboa, LIS, Lisbon, Portugal",
               "arrival_time":"Sunday, May 21st 2017, 08:05am",
               "arrival_time_formatted":"2017-05-21T08:05:00+01:00"
            },
            {
               "airline":"TAP Portugal",
               "flight_number":"TP217",
               "duration":"7 h 35 Minutes ",
               "duration_seconds":27300,
               "departing_from":"Lisboa, LIS, Lisbon, Portugal",
               "departure_time":"Sunday, May 21st 2017, 10:45am",
               "departure_time_formatted":"2017-05-21T10:45:00+01:00",
               "arriving_at":"General Edward Lawrence Logan Intl, BOS, Boston, United States",
               "arrival_time":"Sunday, May 21st 2017, 01:20pm",
               "arrival_time_formatted":"2017-05-21T13:20:00-04:00"
            }
         ],
         "flight_key":"58f3a73",
         "flight_key_long":"4d7ded7b19fd4946b25526c94ef76a3634ed8fac852a28bece2368b73ee2800fcc
      }
   ]
}
```


Figure 2.1: Plan My Vacation API - Flight List

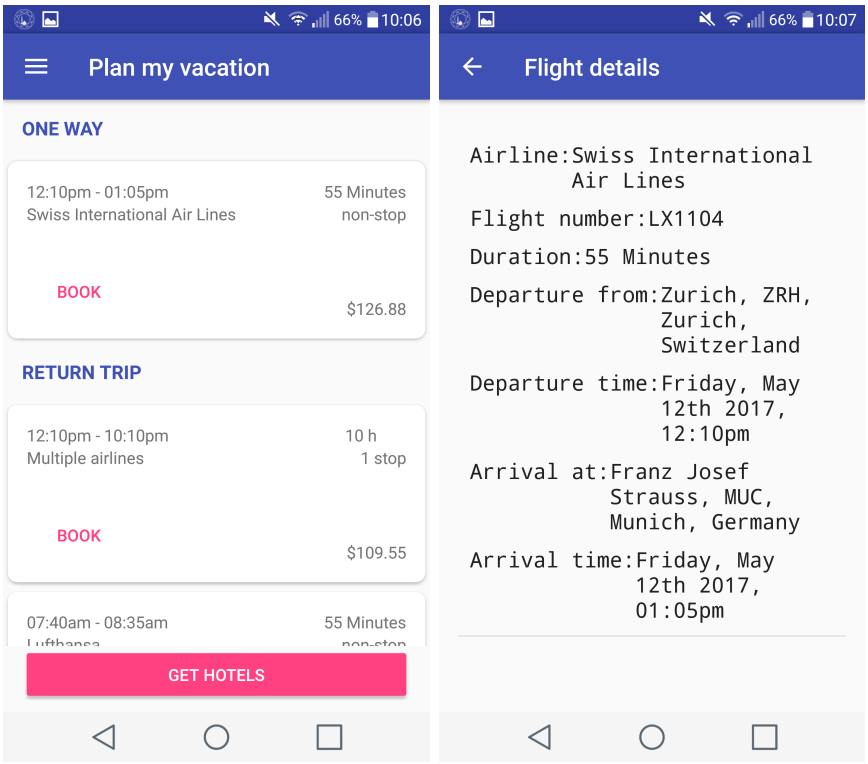

Figure 2.2: Android Application - Flight List

Chapter 3

Hotel Data

The next step in the trip planning process, is the booking of accommodation. The two most popular platforms for this are Booking.com and AirBnb. Booking.com offers hotels from the range of budget to very expensive while AirBnb offers accomodation option more towards the budget range, but at convenient locations around any city. Neither of these platforms have an official publicly accessible API.

3.1 The official un-official Airbnb API

This publicly accessible endpoint requires a unique "API" key that needs to be associated with an Airbnb account. There is officially no public API for Airbnb, however, for a normal user, the client server communication on the Airbnb platform happens through the means of a REST API. This API requires a key as well and each user profile is associated with one particular key. This is visible in the network logs through any browser. For this project, we created a new user and used the key associated with that account to access Airbnb data. The data returned from this API is a large list of key-value pairs for a given place with parameters for reviews, location, price, rating, pictures etc. The server processes this data into a neat JSON and the android application parses this data to display on the application.

3.2 Scraping Booking.com

Booking.com does have a dataset, but not a publicly accesible REST API. They have provisions for embedding iframes onto a website so that they can track bookings made through that fragment. However, for an application that requires up-to date data about hotels, as seen on the website, this is infeasible. By registering on the Booking.com developer site, one can access their worldwide dataset of all hotels. Using this would mean that each query would have to be manually matched against a massive set of possible matches, to return relevant

hotels, in addition to the likely problem of the data-set being outdated. This again is infeasible.
Thus, our approach uses a simple python script to parse the raw HTML of the Booking.com webpage and use a headless browser package to inject the required parameters into the webpage fields and then again parse the resulting HTML into a sensible form and return this to the android application in a standard JSON format. This standard JSON format and the methods of calling this endpoint can be found in Appendix 1.

3.3 Hotel selection and ordering

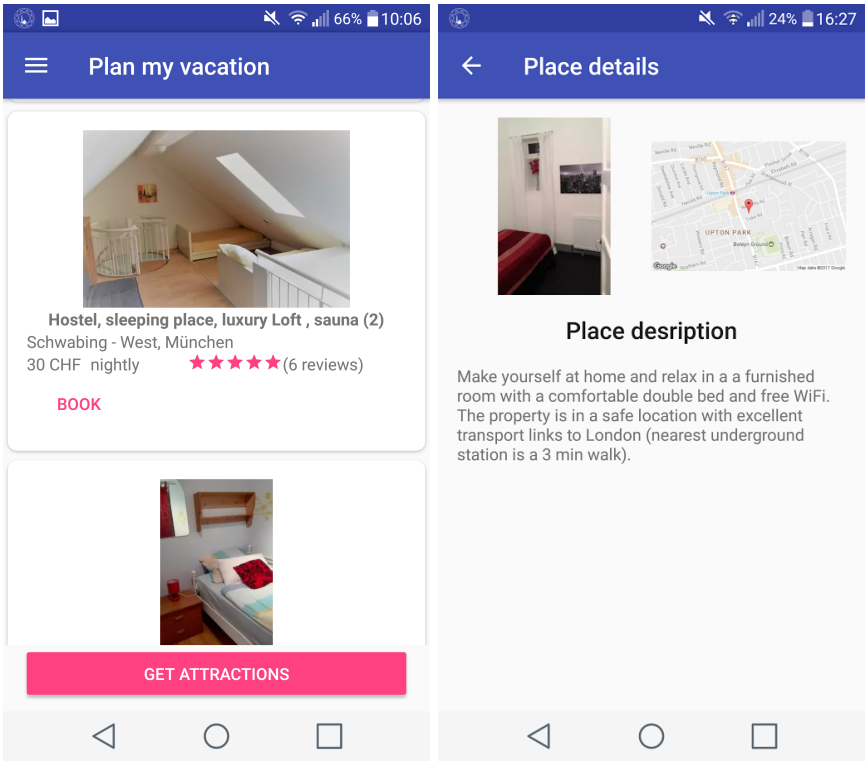

Figure 3.1: Android Application - Hotel List

For the hotel selection we are showing to the user 20 places, 10 from Airbnb and 10 from Booking.com. For each, The first 5 are the cheapest, the next 5 are the ones with the highest reviews. A snippet of the first 2 of these can be seen in Figure 3.1. The number of reviews from booking.com is much higher than the number of reviews from airbnb, moreover, when only a few users review a place, this does not reflect the same as when a high number of users review a different place. Because of these reasons we are using weighted reviews also known as

Bayesian rating[11]:

$$WR = \frac{v}{v+m} \cdot R + \frac{m}{v+m} \cdot C \tag{3.1}$$

where R is average rating for the hotel (mean), v is number of votes for the hotel, m is minimum votes required to be listed in the top 5, m = 1 is used in our case, C the mean rating across all hotels shown.

Booking a place is similar to booking a flight, the app redirects to the respective site (airbnb.com or booking.com) in their page of booking that particular place which is clicked. If a place card is clicked, the application opens the details for that place if that place is listed on Airbnb. For Booking.com unfortunately we are unable to crawl specific hotel information, as they do not expose an API. The place details for Airbnb places include a gallery of images, a location map (redirects to Google Maps) and a description of the place.

Chapter 4

Attractions

With the flights and accommodation booked, the traveller would now like to visit some places in the city. When it comes to any new city, there are always several things that would interest a traveller. This comes down to personal taste where some travellers would prefer popular tourist attractions while others prefer quite places or even nature. This data is available all over the internet and can get overwhelming really fast if the user has a specific notion in mind. Here, we gather data from the most popular sources that covers almost all types of places that any traveller would like to visit, and present it in an intuitive and interactive manner.

4.1 FourSquare

FourSquare is a social network of sorts and has data that comes directly from users. Users check-in to places that they visit and provide ratings and reviews for those places. This really comes in handy when finding data for users that do not want to visit popular tourist attractions but would like to see the lesser known places in the city or visit events that are currently happening. The FourSquare API for attractions is very well documented and publicly accessible with an API key and a client secret. More about the exact nature of the calls are described in the following section.

4.1.1 Categories

On a high level, the API allows us to use FourSquare as though we were using the mobile application. Thus, it is possible to get data about a particular city/place/spot based on what other users said about the spot or which other users checked into that particular spot. This led to data that was unclean. The limitations are described in the following section. The following calls can be made to our data to get the annotated and cleaned data from FourSquare :

- **getFourSquareCategories**?lat=47.3769&lon=8.5417

This call is made to get a list of available categories of attractions/user check-ins at a particular latitude and longitude.

- **getFourSquareTrending**?lat=47.3769&lon=8.5417&radius=2000
  A list of the most trending places in that particular area is returned. Trending is defined by FourSquare as a place with a lot of user activity.
- **getFourSquareExplore**?lat=47.3769&lon=8.5417&category=casino
  Returns a list of all establishments that match that particular category, retrieved from **getFourSquareCategories**.
- **fourSquareSearch**?near=Hannover,DE&radius=5000&qTerm=donut
  Performs a general search of all establishments in a particular area, filtering on a query term.

4.1.2 Limitations

FourSquare is designed to be more of a social networking platform rather than an attractions database like TripAdvisor, thus, a large portion of the data returned for places on FourSquare contains irrelevant information such as updates from users while checking into an airport or a railway station. However, this information is annotated in the R-Tree with timing and this helps filter out irrelevant information to a certain degree.

4.2 TripAdvisor

TripAdvisor's attractions database is of most interest to us for this application, however, TripAdvisor no longer provides access to their public API, and through trial and error, we were able to use an API key to access some of their functions. This data returned is limited and cannot be queried with additional parameters, thus, this is incorporated as is.

- **getAttractions**?lat=47.3769&lon=8.5417
  Fetches a limited number of attractions for the specified place. This number changes periodically.

4.3 Google

While planning a trip, the most common thing to do while looking up a new place is to make a Google query about the place and reading reviews and calculating distances. We automate this process in this exact manner, providing the most useful information to the user. The API calls used are listed in the following section.

4.3.1 Places API

- **getTopAttractions**?location=london&radius=5000
  With this query, we perform a raw google search for the specified place. For example, this is the equivalent of searching Google for "Points of interest in London".

- **getAttractionDetails**?placeid=ChIJ2dGMjMMEdkgRqVqkuXQkj7c
  The previoous call returns a list of attractions for the specified place, with an ID for each attraction. Using this ID more information can be acquired, for that place.

- **getTopAttractions**?pagetoken=CoQC8g..
  The first call also returns a "pageToken" parameter, that can be used to retrieve more places of interest.

4.3.2 Limitations

The google API has a rate limit of 1000 free calls for every 24 hour period. This proves to be extremely limiting for our application as each call to "getTopAttractions" on an average makes 30 calls to the google API. We were able to double this limit as can be seen in Section 5.2.

4.4 Creating an Annotated R-Tree and Trip Generation

The downfall of planning a trip is the estimation of the amount of time that one would spend on an average at each place. Especially with high dimensional and unordered data, estimating this time is a daunting task. In this project, we manually group attractions into several categories (For example, Museums, Churches, Historical sites, etc.) and annotate each category with an approximate amount of time that one would spend at that place (For example, Church = 45 minutes, etc.) and this annotation is done as soon as any of the attraction endpoints above are called.
The data is annotated and then inserted into an R-Tree [7]. This is our data structure of choice here because we are dealing with geo-spatial data with a latitude and longitude for each location, allowing us to create a large ordered tree of attractions which can be queried using bounding rectangles, to fetch results at a particular node of the tree, efficiently, without parsing the entire tree.

- **getAnnotatedTree**
  Returns the entire tree.
- **queryAnnotatedTree**?lat=51.50195785&long=-0.130199&length=10
  Queries the tree, specifying the bounds of the bounding rectangle's diagonal.
- **clearAnnotatedTree**
  Clears the entire tree.

4.4.1 Trip Ordering and Itinerary Generation

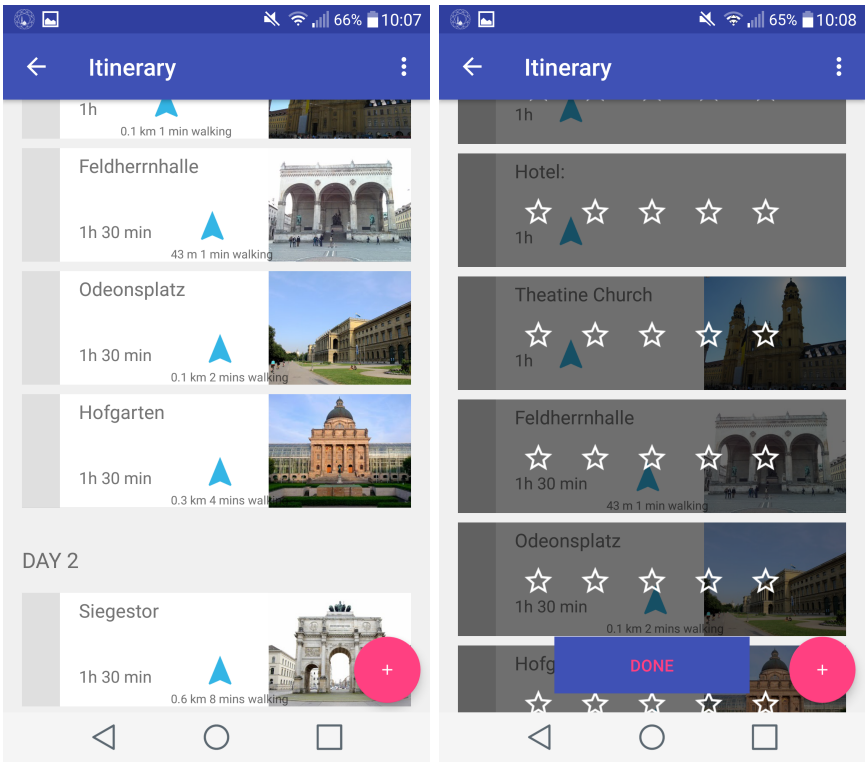

Figure 4.1: Android Application - Attraction List

When ordering the attractions for generating the itinerary different parameters are taken into account. First, the list of attractions are filtered based on user preferences, by removing the places that the user does not like or places that belong to categories which the user does not like. Trip duration and daily tour interval parameters also are important to define the number of total attraction that can be visited and the number of attractions per day. Also, the arrival time is added into the calculation for the first day of the tour. In the case that the arrival time is later than the time the tour starts for the day, then the tour

duration for that day is shorter. Similarly, this is also applied to the departure time on the last day of the trip. The hotel location is used to calculate the distance for the first attraction of each day and also for navigation. In addition, the attractions are ordered based on the reviews, from highest to lowest. Then, for each of the days starting from the first day the best attraction is assigned to that day, while the other attractions are chosen based on the distance from that attraction. The idea behind this is to minimize the distance that the user will have to travel while moving between places. A sample ordering of attractions based on these parameters can be seen in Figure 4.1.

User Preferences

This itinerary can be customized by the user by rearranging the attractions, removing them, or adding new ones. After visiting an attraction, users can remove that attraction from the list by swiping it to the right, indicating that it has been visited. If the user wants to remove an attraction from the itinerary he/she can swipe left and provide a reason for doing so:

- Does not like that particular place
- Does not like the category which that place belongs to
- The place is too expensive
- None of the above

The feedback from this screen will be used in the future to provide better itineraries. For example, when a user does not like museums, he would swipe left and choose the option “I do not like museums”. While generating future trips, there will be no museums shown for this user. The user can change this at any time by simply opening the preferences screen and removing the category of museums from the list of places that he/she doesn't like. The list of categories shown on the preference tab come from FourSquare while the categories assigned to a place that a user rejects while browsing places on the attractions screen are assigned based on their source, i.e., Google or FourSquare. The user preferences, including likes/dislikes as explained above, are summarized in a preferences view, where the user can add/remove categories/places to his likes/dislikes. Also, the user can change the duration of an itinerary which is 9am to 5pm by default. An example of this screen is shown in Figure 4.2.

Another feedback mechanism is implemented, where in the user can rate attractions by pressing the menu button, followed by ”Rate this trip”. By doing this, the user is given the option of rating individual attractions on a scale of 1 star to 5 stars. This rating can be factored in to make better recommendations to future travellers. This can be seen on the right in Figure 4.1.

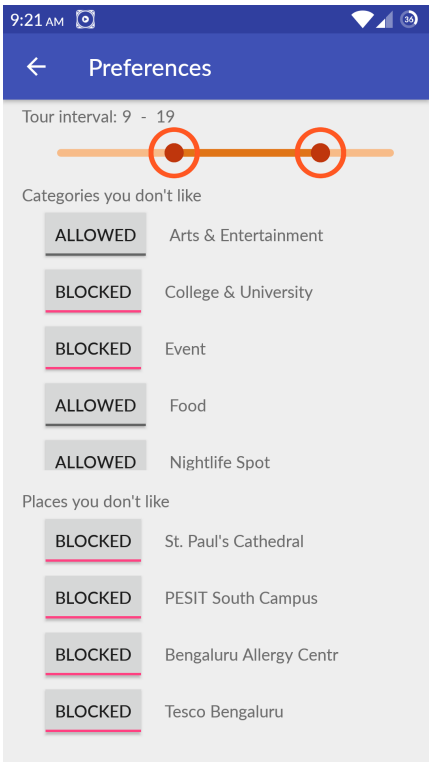

Figure 4.2: Android Application - Preferences View

One of the ways to customize a trip is by adding new places. This can be done by pressing the floating action button on the bottom right of the screen embedded with a "+" sign. The screen shown on the left in Figure 4.3 will appear. As seen in the figure, the user can search for a particular place by inputting a keyword and pressing search, or by choosing one of the categories shown. When choosing a category the user will be on the screen shown on the right in Figure 4.3, where he/she can check all the places that he/she wants to add and then press the floating action button on the bottom right corner with check-mark symbol, to add them to their current itinerary. The user has the option to save a generated trip. This option is under menu, followed by "Save trip". In the navigation drawer menu there is a view for all saved trips as shown in Figure 4.4. Another option is to reset the trip. This may be useful after a lot of manual rearrangements.

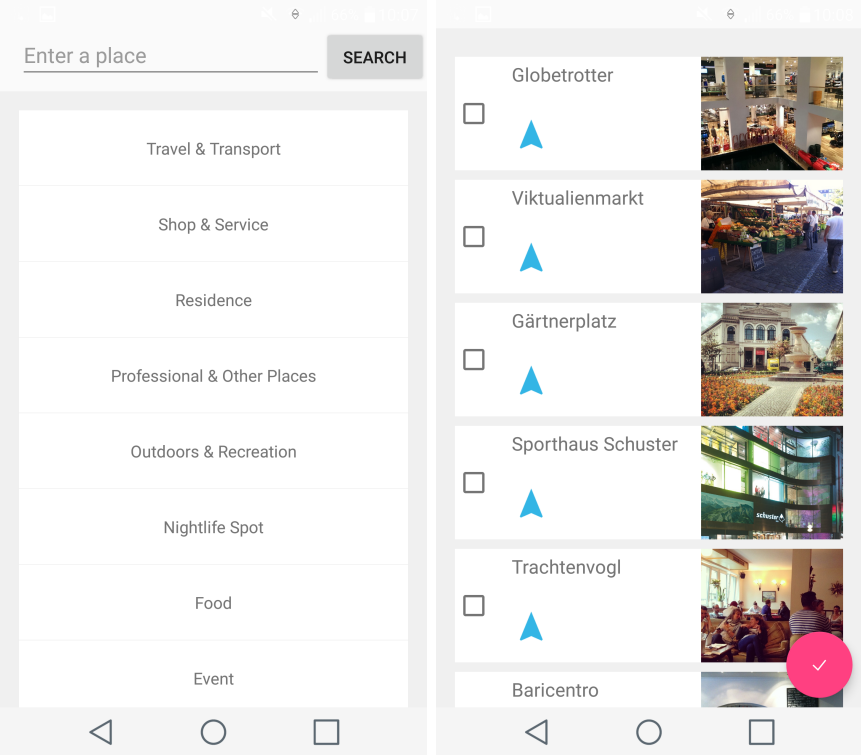

Figure 4.3: Android Application - Category List

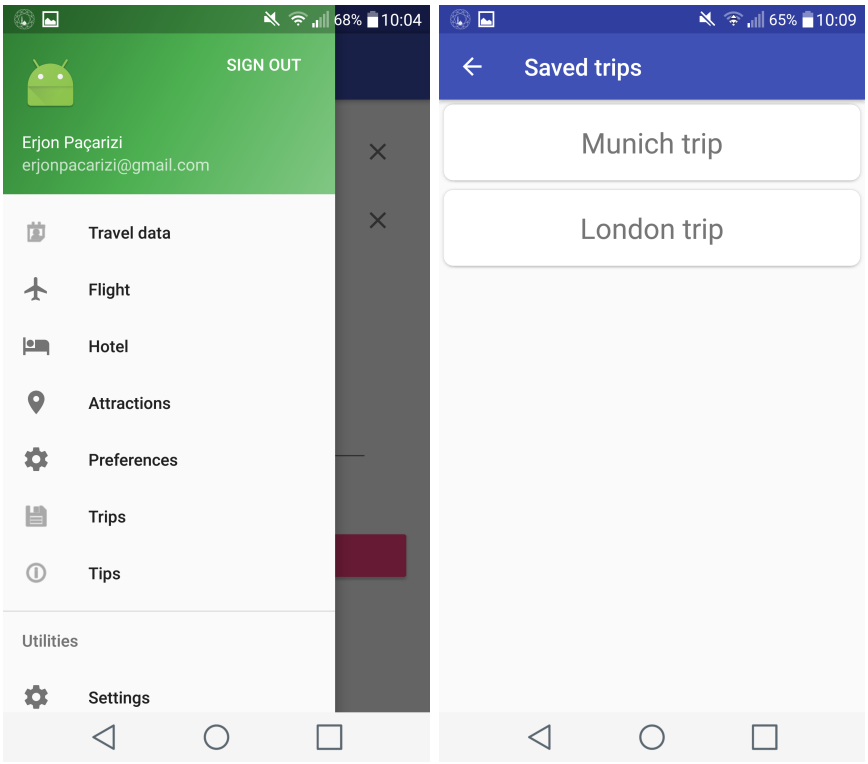

Figure 4.4: Android Application - Trip List

Chapter 5

System architecture

5.1 Architecture overview

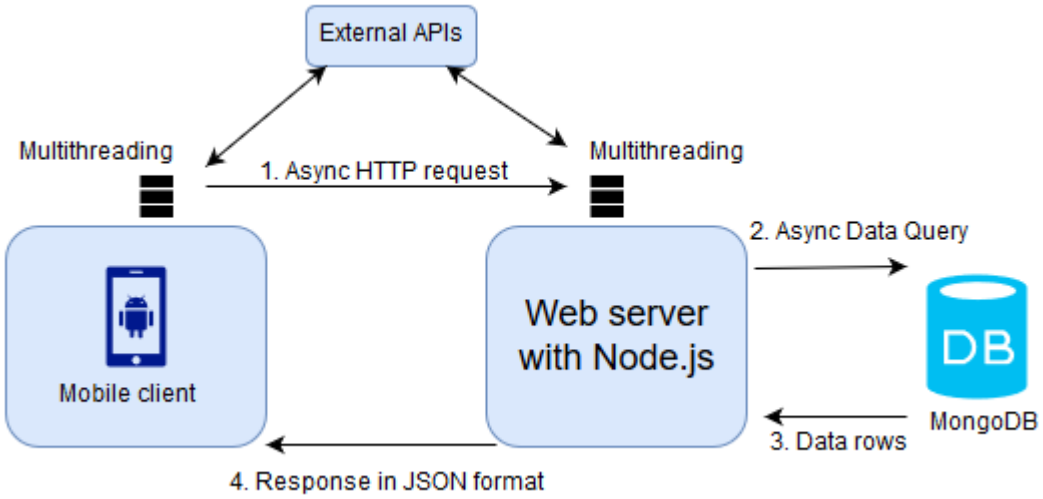

Figure 5.1: Plan My Vacation API - General Architecture

The architecture of this application is divided into two main components, i.e., client and server. The client side consists of the Android user who submits requests to external APIs, e.g., location services, Google places API for fetching travel data and loading images from attraction providers. Other requests are sent to the server, which communicates with the database for user specific requests, otherwise sends requests to external APIs, as can be seen in Figure 5.1. Communication is done using REST style techniques, while the data format used is JSON.

5.2 Server implementation

All server side programs are implemented using Node.js for high performance asynchronous execution, with port-parallelism, as described in the figure in Section 5.1, with the exception of a python script to parse Booking.com. The design of all programs is in line with standard REST API guidelines, to allow for simple and secure client-server communication.
The server follows a standard format of JSON responses with three components, namely,

- Status Code
- Status Description
- Payload data (If any)

There is also a MongoDB server running on our server that stores user information, trip information etc., and will be described in more detail in the following sections. The structure of each mongo document is as follows :

```
_id: Unique Identifier,
email: "example@example.com",
name: Full Name,
photoURL: Profile Image URL,
media: Facebook/Google,
registrationDate: Date.now(),
likes:[List of likes],
rejects:[List of Dislikes],
savedTrips:[List of Saved Trips]
```

5.3 Client implementation

The client side was developed in Java and XML using the Android SDK. The minimum SDK version required is 15 which corresponds to Android version 4.0.3 (ICE_CREAM_SANDWICH_MR1) which supports around 99% of Android devices [1]. Users have their own profiles where they can save trip related data. To support this, the first thing users will do when they open the app is to authenticate themselves using one of two external authentication providers: Google and Facebook, shown in Figure 5.2.

After users are authenticated they will enter basic travel data: Places of departure and destination, and dates of the trip start and finish. The app will ask the user for location permission so it can get the current user location. This

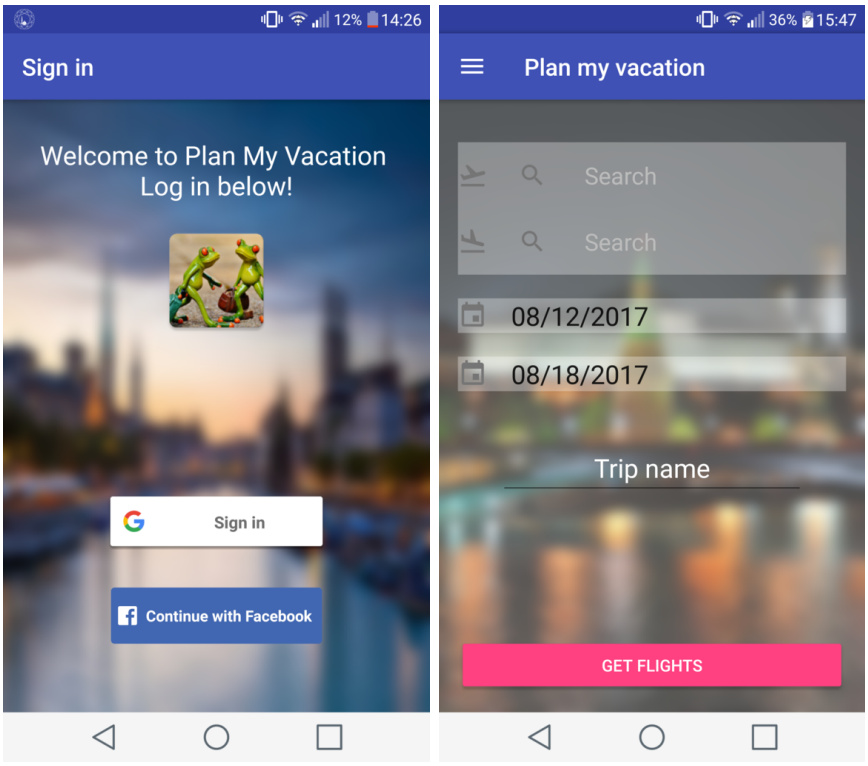

Figure 5.2: Android Application - Login and Landing Screen

is useful for filling the departure place and when generating the itinerary the user can use his location to navigate to the airport or any attraction. The trip name will automatically be filled using destination place name and adding the word “trip” e.g., “Munich trip”. Users can change this. This will be useful if the user plans to save trips for later reuse so that the user can look up a trip based on the name. Departure and destination location use Google Places API to filter the search box and auto complete.

5.4 Database - MongoDB

Basic user information, i.e., email, username, and profile photo URL are saved in the database for creating user profiles. These data are taken from the authentication providers (Google and Facebook).

Trips are saved in JSON format. This way, making a cross platform application would be easy. One trip is saved based on trip name and all the attractions that will be visited with the detailed data for each trip.

Chapter 6

Future Work

There are several approaches for adding new features to this project. Porting the application to iOS and creating a web platform would be an important one. We implemented a feature called "tips" where we added a tip for the user to download the offline map before visiting a particular location. Also, added a prototype for the weather. It would be helpful for the user to know the weather for each day during the trip and adding this weather data directly to the trip itinerary would be useful. Currently, the itinerary is shown in a "daily" approach where the attractions are sort by days. It would be a useful feature to have a "hourly" view, where the user can see the itinerary for each hour.
Incorporating the user's budget is a crucial task, and here we have made use of whatever budget data was available, to display the cheapest hotels and flights, which usually make up the bulk of the cost of a trip. However, at this point in time, data about attractions is scarce in itself and there is no reliable source of data to factor in the cost for different attractions. For instance, the Google places API can give us all the data we need about a particular place or location, however, entry costs/other costs for a particular attraction is not available. This is understandable as most attractions have fluctuating entry prices, special entry prices for age groups and so on. This cannot be factored in easily and thus, one useful task would be to scrape the web and maintain a database of attraction costs. This would even be of commercial value if executed correctly.
A simple framework of accessing data for collaborative filtering has already been set up[1]. This can be taken further to make trip recommendations.
Another useful feature would be to create a "matchmaking" algorithm for users travelling within the same time frame to the same location, to provide shared accommodation/group flight discounts. This can be done with a large user base and would be of commercial value. An important addition to this application would be to actually sign up for the booking.com affiliate program and get real-time comprehensive access to their API to make hotel bookings completely in-app with the best prices. The same cannot be said about flights unfortunately as at the moment there is no flight provider willing to open their doors to provide

[1] http://82.130.102.16:9771/generateMatrix?uid=211

a seamless API integration. In the future if this is possible, this would make the application a single complete package. User interface and User experience improvements can be made as well. At the moment, the goal was to set up a complete framework that connected several providers to package them into a single application and the UI was made as interactive as possible within the scope of this project, however there are many subtle changes that can complement the overall experience, such as animations and helpful tips.

Bibliography

[1] Android versions market share https://developer.android.com/about/dashboards/index.html

[2] SkipLagged https://www.skiplagged.com

[3] https://www.reddit.com/r/IAmA/comments/3ux82r/united_airlines_sued_me_last_year_for_creating/

[4] Booking.com Affiliate Program http://www.booking.com/affiliate-program/faq.html

[5] The Official Un-Official Airbnb API http://airbnbapi.org/

[6] Zizhuo Wang and Yinyu Ye *Hidden-City Ticketing: the Cause and Impact.* September 2013.

[7] Guttman, Antonin. *R-trees: a dynamic index structure for spatial searching.*. Vol. 14. No. 2. ACM, 1984.

[8] FourSquare https://developer.foursquare.com/

[9] TripAdvisor https://developer-tripadvisor.com/content-api/description/

[10] Places API https://developers.google.com/places/documentation/

[11] Xiao Yang, et. al. *Combining prestige and relevance ranking for personalized recommendation.* CIKM '13 Proceedings of the 22nd ACM international conference on Information Knowledge Management

Appendix A

Appendix 1

A.0.1 Overview of the Plan My Vacation server side application

All of the above described API endpoints are running on a reverse proxy NGINX linux server, serving a node application for each API endpoint on different ports. The following are the categories and sample REST API endpoints for Plan My Vacation :

Endpoint parameter reference -

Most parameters are self-explanatory, a few important ones are listed below

- lat: Latitude
- lon: Longitude
- radius: Radius of the circle within which the endpoint will return results for, with lat/lon as center.
- category: Any category of attractions that you can think of. If it exists it will be returned.
- qTerm: Query term - Can be a word or combination of words
- near: Close to - Format ¡City Name, 2 letter country code¿
- return/checkin/checkout/day/month/year - 2 digit numbers always
- from/to - 3/4 letter IATA code

Four Square

- http://82.130.102.16:8991/getFourSquareCategories?lat=47.3769&lon=8.5417

- http://82.130.102.16:8991/getFourSquareTrending?lat=47.3769lon=8.5417&radius=2000
- http://82.130.102.16:8991/getFourSquareExplore?lat=47.3769&lon=8.5417&radius=2000&category=casino
- http://82.130.102.16:8991/getFourSquareExplore?lat=47.3769&lon=8.5417&category=casino
- http://82.130.102.16:8991/getFourSquareExplore?lat=47.3769&lon=8.5417&radius=2000&category=Movie Theater
- http://82.130.102.16:8991/getFourSquareExplore?lat=47.3769&lon=8.5417
- http://82.130.102.16:8991/fourSquareSearch?near=Hannover,DE&radius=5000&qTerm=donut

Trip Advisor

- http://82.130.102.16:4121/getAttractions?lat=47.3769&lon=8.5417

Booking.com

- http://82.130.102.16:4015/getBookingcomListings?destination=Berlin&checkinday=21&checkinyearmonth=2016-12&checkoutday=23&checkoutyearmonth=2016-12&numberrooms=1&adults=2&children=0

Airbnb

- http://82.130.102.16:4015/getListings?Tanzania,TZ&checkin=07/12/2016&checkout=09/12/2016&guests=2&page=1
- http://82.130.102.16:4015/getInfo?id=15200584

Skiplagged

- http://82.130.102.16:4012/getCheapestFlight?from=ZRH&to=BLR&year=2016&month=12&day=15
- http://82.130.102.16:4012/getLeastLayovers?from=ZRH&to=BLR&year=2016&month=12&day=15&returnMonth=12&returnDay=20&returnYear=2016
- http://82.130.102.16:4012/getShortestFlight?from=ZRH&to=BLR&year=2016&month=12&day=15&returnMonth=12&returnDay=20&returnYear=2016

Database

- http://82.130.102.16:9771/registerUser?uid=2&name=PrashanthB &email=x@x.com&photoURL=abcd
- http://82.130.102.16:9771/getAllUsers
- http://82.130.102.16:9771/updateLikes?uid=2 &place=Empire State Building
- http://82.130.102.16:9771/deleteUser?uid=2
- http://82.130.102.16:9771/updateRejects?uid=1place=Broadway Musical &reason=Too Expensive&reasonCategoryNumber=3
- http://82.130.102.16:9771/updateCategoryRejects?uid=1&category=casino &reason=no money&reasonCategoryNumber=3
- http://82.130.102.16:9771/addTrip?uid=1&tripName=kosovoTrip &tripData=firstStop:beach,secondStop:hotel
- http://82.130.102.16:9771/getTrip?uid=1&tripId=2
- http://82.130.102.16:9771/addReview?uid=211&place=NYC&review=tall building
- http://82.130.102.16:9771/removeLikes?uid=211 &place=Empire State Building
- http://82.130.102.16:9771/removeRejects?uid=211&place=Broadway Musical
- http://82.130.102.16:9771/removeCategoryRejects?uid=211&category=casino
- http://82.130.102.16:9771/getUserById?uid=211
- http://82.130.102.16:9771/generateMatrix?uid=211
- http://82.130.102.16:9771/renameTrip?uid=666&tripId=2 &newTripName=testTrip
- http://82.130.102.16:9771/deleteTrip?uid=666&tripId=1

Category Tree

- http://82.130.102.16:1947/getTopAttractions?location=london&radius=5000
- http://82.130.102.16:1947/getAnnotatedTree
- http://82.130.102.16:1947/queryAnnotatedTree?lat=51.50195785&long=-0.130199 &length=10
- http://82.130.102.16:1947/clearAnnotatedTree