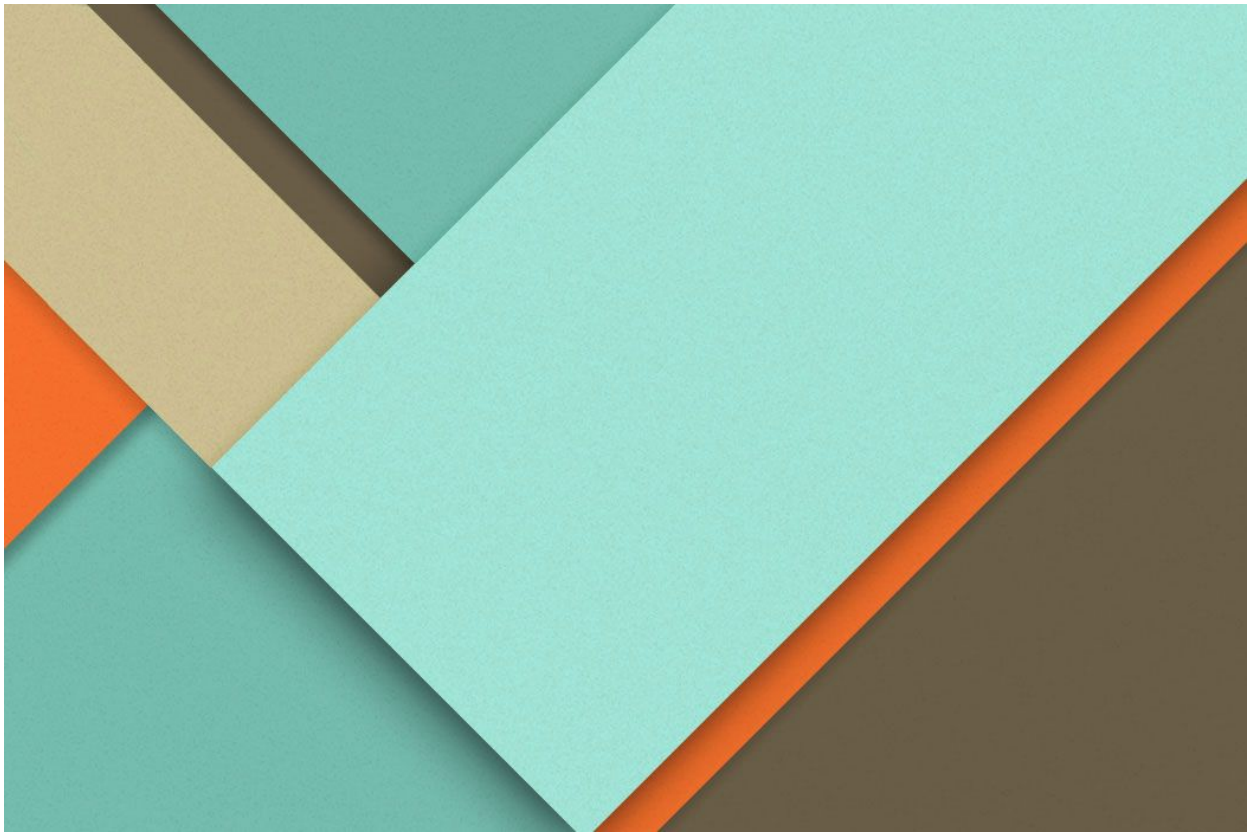


Student Travel App

Design Document

03.04.2018

Group 9

Ankush Phulia (2014CS50279)

Deepak Saini (2013CS50281)

Shashank Yadav (2013CS50799)

Vikas Chouhan (2014CS10264)

Table of Contents

Table of Contents 1

1 Introduction 3
- 1.1 Purpose 3
- 1.2 Scope 3
- 1.3 Overview 3
- 1.4 Definitions, Acronyms and Abbreviations 4
- 1.5 References 5

2 System Overview 5
- 2.1 Architectural Design 5
  - 2.1.1 Module Diagram 5
  - 2.1.2 Object Diagram 6
- 2.2 Module Definitions 7
  - 2.2.1 Journey Planner 7
  - 2.2.2 Overlap Calculator 7
  - 2.2.3 Trip Planner 7
  - 2.2.4 Schedules 7
  - 2.2.5 Search 7
  - 2.2.6 Notifications 8
  - 2.2.7 Authentication 8
  - 2.2.8 User 8
- 2.3 Technology/Tools Used 8
  - 2.3.1 Client-side / Front-End 8
  - 2.3.2 Server-side / Back-End 8
  - 2.3.3 Database 8
  - 2.3.4 External APIs 8

3 Detailed Design 9
- 3.1 Module APIs 9
  - 3.1.1 Authentication 9
  - 3.1.2 User 9
    - 3.1.2.1 Registration related 9
    - 3.1.2.2 User Profile related 9
  - 3.1.3 Journey 10
    - 3.1.3.1 Journey Creation 10
    - 3.1.3.2 Journey Completion 10
  - 3.1.4 Trip 10

3.1.4.1 Trip Creation 10
3.1.4.2 Trip Completion 11
3.1.5 Search 11
3.1.6 Schedules 11
3.1.7 Notifications 11
3.2 Database Design 12
3.2.1 Data characteristics 12
3.2.2 Database Functionality - CRUD 12
3.2.3 Sub-databases 12
3.2.3.1 Users Details 12
3.2.3.2 Travel Object Details 13
3.2.3.3 Locations 13
3.3 Screen Layouts 13
3.1.1 User Login 3.1.2 User Registration 13
3.1.3 Dashboard 3.1.5 User Profile 14
3.1.6 Search Trip 3.1.7 Create New Journey/Trip 14
3.1.8 Trip Details 3.1.9 Modify Route/Checkpoints 15
3.1.10 Select Transport 3.1.11 Travel Completed/Rate Travel 15
3.4 Use Cases 16
3.4.1 Logistics-related 16
3.4.1.1 User Login 16
3.4.1.2 User Registration 17
3.4.1.3 View Profile 18
3.4.1.4 Update Profile 19
3.4.1.5 Notifications 20
3.4.2 Undertaking a Journey/Trip 21
3.4.2.1 Create Journey 21
3.4.2.2 Create Trip 22
3.4.2.3 Search for Trip 24
3.4.2.4 Travel Completed 25
3.4.2.5 Rating Travel 26

4 Deployment Design 27
4.1 Environment 27
4.2 Version Control 27
4.3 Testing and Debugging 27
4.3.1 Functionality Testing 27
4.3.2 Usability Testing 27
4.3.3 Interface Testing 27
4.3.4 Compatibility Testing 27
4.3.5 Performance Testing 28
4.3.6 Security Testing 28

1 Introduction

Majority of the students in IIT live quite far away from campus. More often than not, when they plan a trip to their home, they do it alone. Further when students plan trips, it is often difficult to do due to all the work involved - research on destination, cost and reaching consensus. In this document, we propose a design for a seamless solution to this.

1.1 Purpose

The purpose of this document is to describe the system architecture and design of "Student Travel App", a seamless framework that allows users to plan a journey or trip and coordinate, based on the time of journey and destination. It will always reflect the current state of the system architecture, and every deliverable will be based on the description as given in the document. It is intended to be updated and revised as necessary, to reflect the iterative nature of design and the continual development of the system.

1.2 Scope

"Student Travel App" will have three major components -

1. Journey Planning - an end-to-end framework that allows planning journeys from a user's initial location(e.g. hostel), to the destination(e.g. hometown), and back
2. Trip Organisation - A planner for organising a collective trip to a destination - consensus on plan, travel to and stay at the destination
3. Central Server - web-based, manages the database and requests

The main aim of the application is to coordinate the journeys of people based on the time of journey and destination. This document will describe each of the components, the interfaces between them, as well as the interfaces to external components and the user.

1.3 Overview

The Introduction comprises the first section of the document. The remainder is organised into three sections -

- System Overview - an overview of the system functionality, interfaces and interactions
- Detailed Design - the design of each of the components in detail - the module design, system APIs, database design, user interface as well as use cases
- Deployment Design - deployment architecture and implementation specifications

1.4 Definitions, Acronyms and Abbreviations

Term	Definition
User/Normal User	Someone who interacts with the application
Admin/Administrator	System administrator who is given specific permissions for management
Destination	The end point of any travel undertaken
Journey	A one-way travel from hostel/hometown to destination
Trip	A collective travel and stay to a particular destination
Trip Leader	The administrator of a trip group, it is the creating user by default
Travel	A journey or a trip
API	Application Programming Interface
MVC	Model-View-Controller design pattern
ERD	Entity Relation Diagram
ORM	Object Relational Mapping
JSON	JavaScript Object Notation

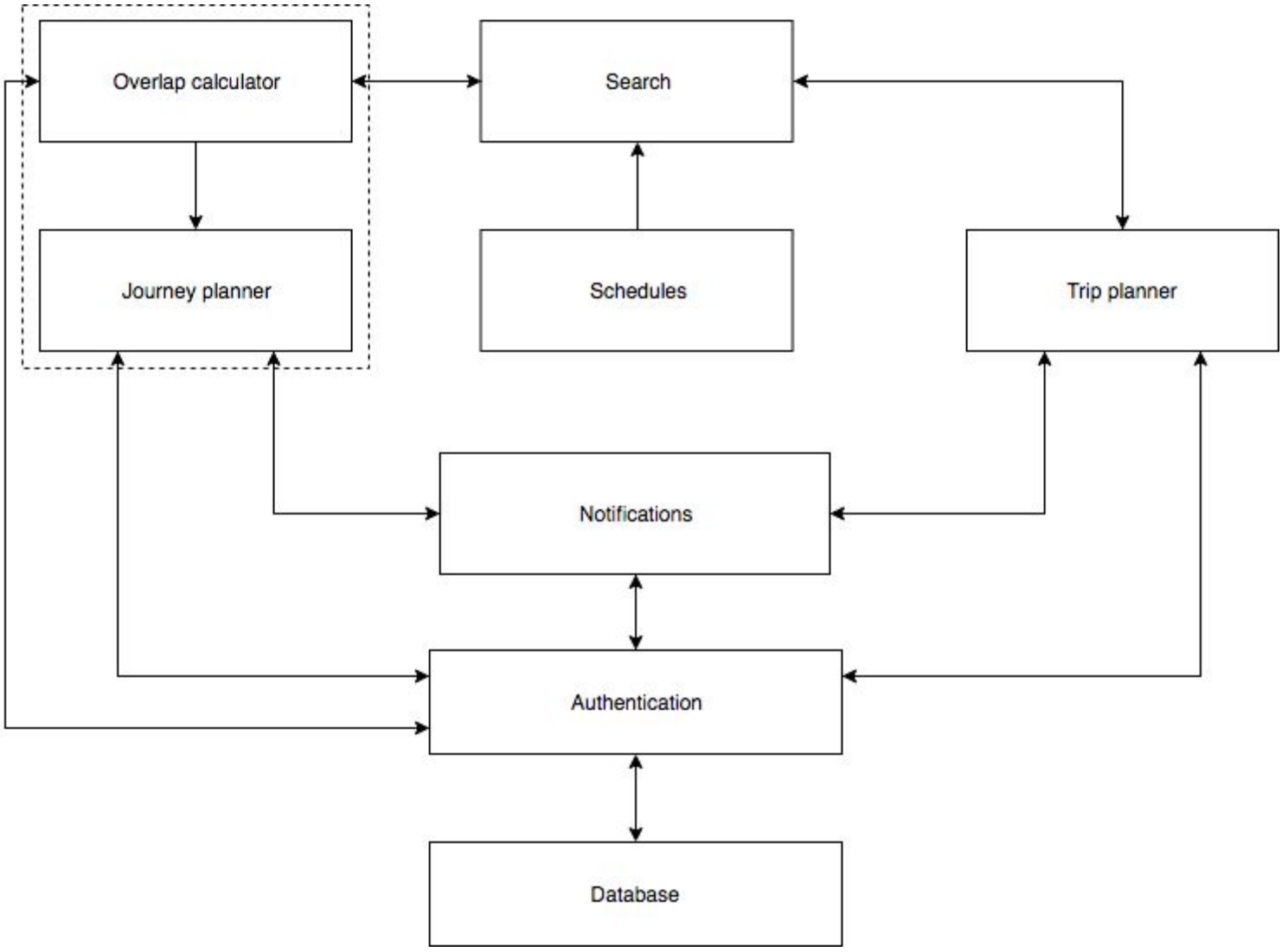
1.5 References

1. Design Format
2. Unified University Inventory System

2 System Overview

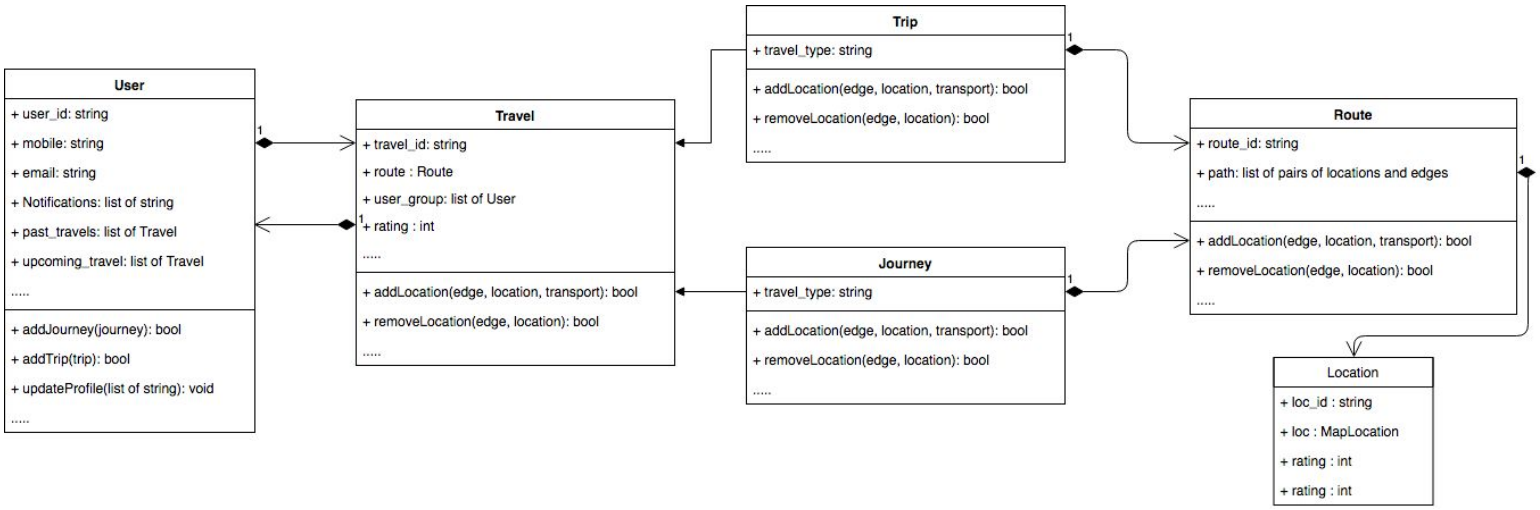
2.1 Architectural Design

2.1.1 Module Diagram



Various Interacting Subsystems in Student Travel App

2.1.2 Object Diagram



Classes comprising the modules

2.2 Module Definitions

2.2.1 Journey Planner

A module in which a user can create a tentative journey that he/she wants to take. Probable dates, times, modes of transportation can be added for different parts.

2.2.2 Overlap Calculator

Used to find the matching journeys on the basis of tentative time, route and modes of commutation and suggests suitable matches to the user. Journeys are extracted from the database and matches are stored back.

2.2.3 Trip Planner

A module in which a user can create and plan a trip. Probable dates, times, modes of transportation can be added for different parts of the trip. Voting can be done on each of the points. A trip leader is elected and finalises the details.

2.2.4 Schedules

Used to extract schedules and routes for various means of transport (buses, trains, flights etc). Information is used by journey planner, overlap calculator and trip planner.

- Get train Schedules. To do this, make API calls to eRail API.
- Get bus schedules. To get bus schedule data, scrape data from internet.
- Flight schedules. To get flight schedules, scrape data from internet.

2.2.5 Search

Module to search in the database, given keys. Compatible with different types of searches - search for matching journeys/trips based on certain criteria, search for transport, etc

- Search for a journey based on supplied checkpoint, expenses information.
- Search for a trip based on tripPoint and budget.
- Search for a tripPoint based on the time of trip and expenses. The rating of the location is also displayed to the user.

2.2.6 Notifications

Used to notify users of the changes like whether a match is found, other user has accepted/declined to travel together, proposal about the trip has been accepted or not etc. Notifications can be of three types:

- Logistics related notifications
- Trip related notifications
- Journey related notifications

2.2.7 Authentication

Validates the login and the registry of users into the database. Secures access to the database by the users by means of tokens, etc.

2.2.8 User

Central module that interacts with all other modules - essentially the ‘VIEW’ in an MVC, i.e. Model-View-Controller style architecture

2.3 Technology/Tools Used

2.3.1 Client-side / Front-End

The display of the application and the user interface / interactions

- Mobile App's front-end is Android-YAML
- Web App's front-end will be in HTML5, CSS3, JavaScript

2.3.2 Server-side / Back-End

- Mobile App's backend is Java
- Web App's backend will Django, Python

2.3.3 Database

- NoSQL database - MongoDB

2.3.4 External APIs

- Google Maps API
- Transport Schedule APIs - trains, flights, etc

3 Detailed Design

For each API, we have provided the state diagram along with a description of the major functions provided by the API.

3.1 Module APIs

3.1.1 Authentication

- **request_login**(***userId, pwd***): Make a login request to the server with username userId and password pwd.
- **authenticate_login**(***userId, pwd***): Verify the userid and password combination provided by the user.
- **login_response**(***login request object***): Depending on the response of the authenticating of user credentials, provide the response to the user.

3.1.2 User

3.1.2.1 Registration related

- **request_registration**(***form***): Make a registration request to the server by filling the form having info as userId, password, Full Name, Mobile number, Email Id.
- **authenticate_registration_request**(***form***): Validate the information provided by the user. If the form is validated, make a new user in the database. Call registraion_response to show the response to the user.
- **registration_response**(***login request object***): Depending on the response of the authenticating of the user, provide the response to the user.

3.1.2.2 User Profile related

- **get_profile**(***userId***) : Give the user profile of the user associated with userId. This includes the Full Name, Mobile number, email Id, Facebook Id.
- **update_profile**(***update_info***) : Modify the profile of the user by overwriting the information or feeding the new information if the field is empty.
- **request_contact_info**(***userId***): request the contact information of some other user.
- **allow_access**(***userId***): Provide your contact details to the user specified in the request object corresponding to the request info.

3.1.3 Journey

3.1.3.1 Journey Creation

- **specify_endpoints**(***startPoint***, ***endPoint***): Create a journey from startPoint to endPoint.
- **specify_tentative_date**(***dateTime object***): Specify the tentative date/time when the journey is going to be undertaken.
- **create_checkpoints**(***checkPoint***, ***cost***, ***meansOfTenasport***): Create a checkpoint in the journey with the preferred means of transport and the preferred estimated cost.
- **create_journey**(***journeyObject***): The journey object is created in the database and posted on the App. It will be visible to other users of the App.
- **fuse_journies**(***journeyObject1***, ***journeyObject2***): Journey objects can be fused together if two people agree to travel together. The users,destinations and checkpoints of the new journey will be the union of the two journey objects.
- **add_compinion**(***journeyObject***, ***userId***): A new user can be added to the journey, if all the existing users agree to it.

- **calculate_overlap**(***journeyObject1***, ***journeyObject2***): Given two journey objects, calculate the overlap between the two.

3.1.3.2 Journey Completion

- **close_journey**(***journeyObject***): A user can mark a journey completed.
- **rate_journey**(***journeyObject***, ***rating***): A user can rate a journey. The rating will be reflected in the rating of the users included in the journey.

3.1.4 Trip

3.1.4.1 Trip Creation

- **create_trip_intent**(***tripPoint***, ***budget***, ***tentativeDate***): A user can create an intent to undertake a trip by specifying the trip location, the tentative date and the estimated budget of the trip.
- **create_trip**(***tripObject***): The trip object is created in the database and posted on the App. It will be visible to other users of the App.
- **appoint_leader**(***tripObject***, ***userId***): If all users agree, then the user specified by userId will be made the leader of the trip. The trip leader will be point of contact for all things related to the trip.
- **add_traveller**(***tripObject***, ***userId***): A new user can be added to the trip by the leader.

3.1.4.2 Trip Completion

- **close_trip**(***tripObject***): A user can mark a trip completed.
- **rate_location**(***tripObject***, ***rating***, ***tripPoint***): A user can rate a trip destination specified by tripPoint, included in the tripObject. The rating of the tripPoint associated with the trip will be modified to incorporate the rating.
- **rate_leader**(***tripObject***, ***rating***): A user can rate a the leader of the trip.

3.1.5 Search

- **search_journey**(***journeyObject***): The overlap of journeyObject is calculated with the existing journeys and a the search results are shown to the user.
- **search_location**(***budget***, ***tentativeDate***): The locations based on the budget and tentative date are shown to the user. Their rating area also included in the search results.
- **search_trip**(***tripPoint***, ***budget***, ***tentativeDate***): A user can search for trips by specifying the trip location, the tentative date and the estimated budget of the trip.

- **search_trip**(***tripObject***) : searching can also be done for a trip based on a trip object. The function will call the above search function.

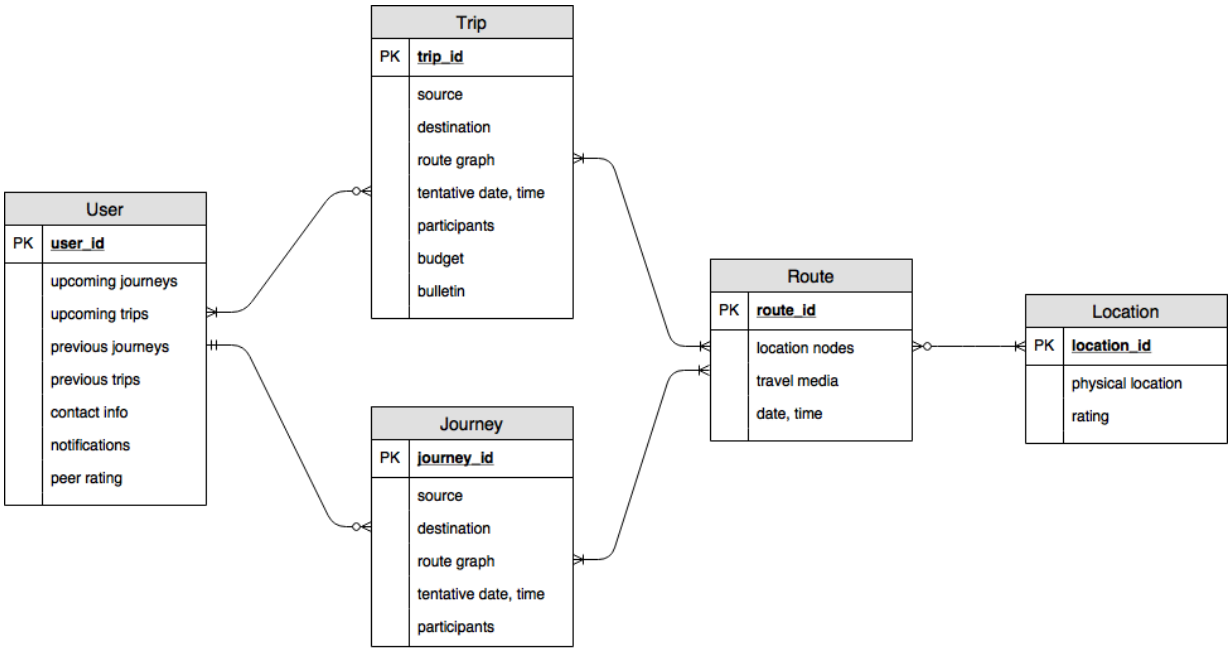
3.1.6 Schedules

- **get_trains**(***budget***, ***startPoint***, ***endPoint***): get the trains in the budget between startPoint and endPoint.
- **get_filghts**(***budget***, ***startPoint***, ***endPoint***): get the flights between startPoint and endPoint having cost in ballpark of budget.

3.1.7 Notifications

- **set_notification**(***journey***): Show a notification to the user if some overlapping journey is posted.
- **set_notification**(***tripPoint***): Show a notification to the user if some trip to the tripPoint is posted.
- **display_notification**(***userId***, ***notification***): Send the notification to the user specified by userId.
- ***check_satisfication(notification)***: Check if notification is satisfied. This function is to be called periodically.
- ***turn_off(notification)***: Turn the regular checking for the notification.

3.2 Database Design



Entity Relation Diagram

3.2.1 Data characteristics

As mentioned earlier, the data will be stored in an online, NoSQL, document store, as JSONs. Compound objects are also stored as JSONs.

3.2.2 Database Functionality - CRUD

- **Create**(***collection***, ***document***) : Insert document in the collection.
- **Retrieve**(***collection***, ***fields***) : Retrieve information from the collection with the fields specified.
- **Update**(***collection***, ***filter***, ***field***, ***new_data***) : Update data corresponding to the given field in the documents specified by the filer in the collection with the provided new_data.
- **Delete**(***collection***, ***filter***, ***field***) : Delete fields from documents specified by filter in the collection

3.2.3 Sub-databases

3.2.3.1 Users Details

Stores the profile and contact details, as well as all the upcoming and previous travels

3.2.3.2 Travel Object Details

Each travel item stores the source, destination, tentative date/time as well as the group of users who are involved in the travel. Also a route object is stored - a graph with nodes representing real-world location and edges representing cost/mode of travel.

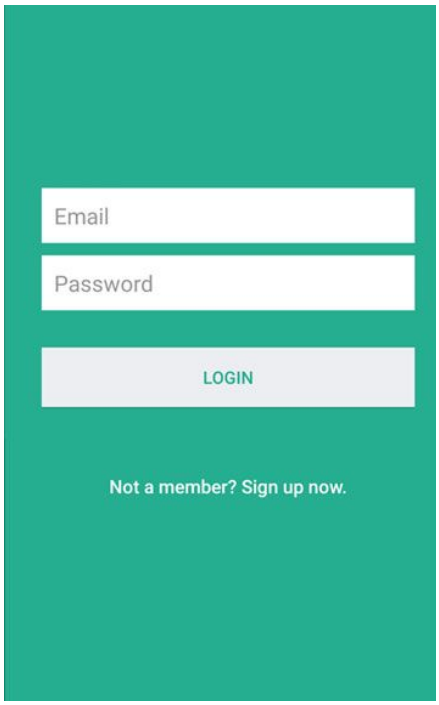
- Trips - Keeps record of the budget as well as the bulletin - all the announcements by the admin, in addition to the details kept by a travel object
- Journey - Keeps record of all the details kept by a travel object

3.2.3.3 Locations

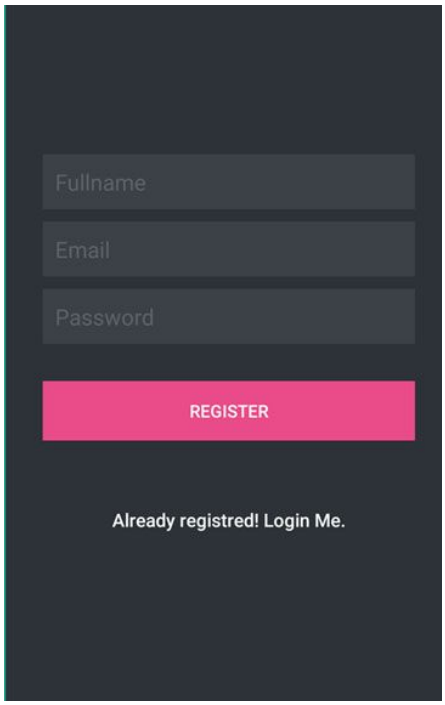
Maps a real location to a location id. Mainly used to record the rating of a travel destination

3.3 Screen Layouts

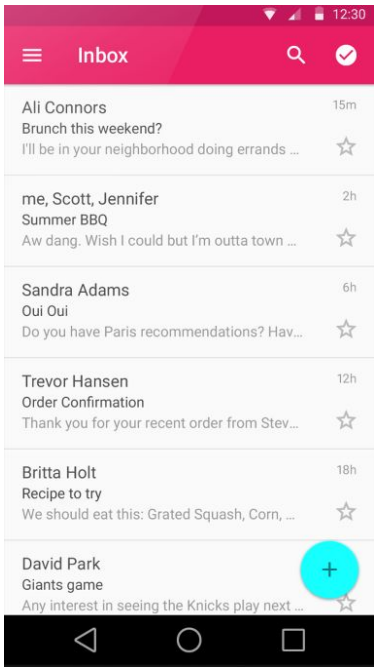
3.1.1 User Login



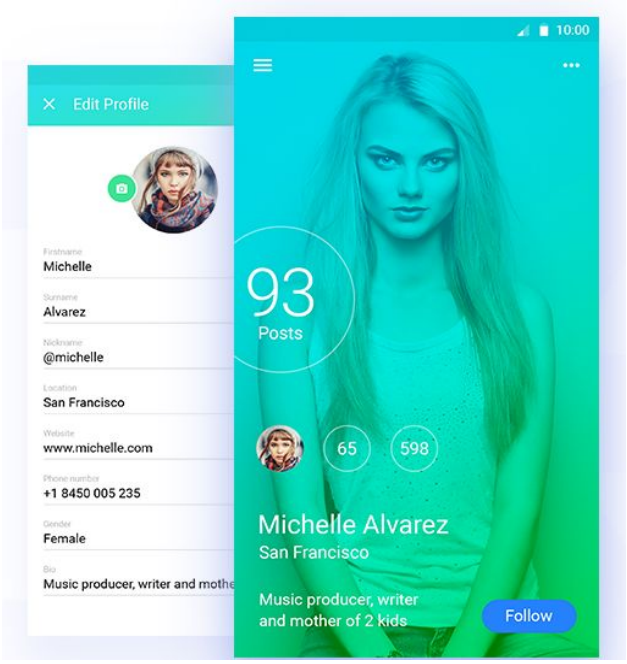
3.1.2 User Registration



3.1.3 Dashboard

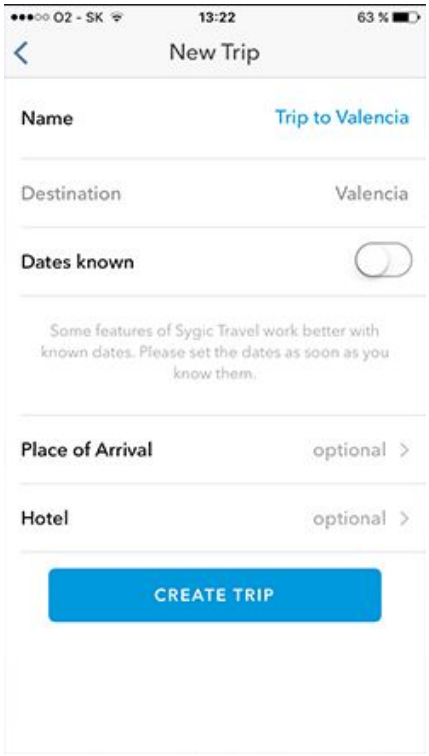


3.1.5 User Profile

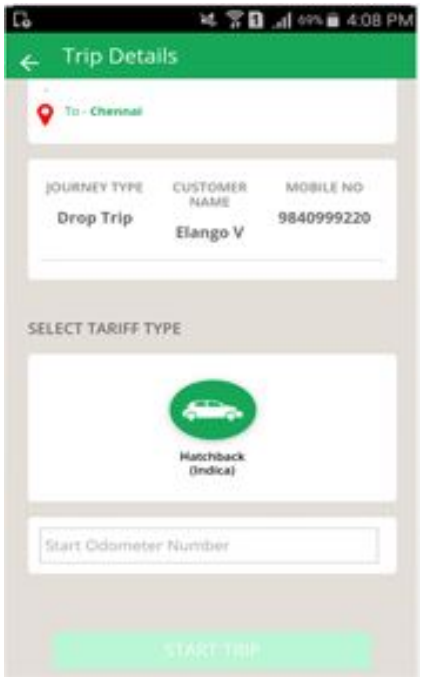


3.1.6 Search Trip

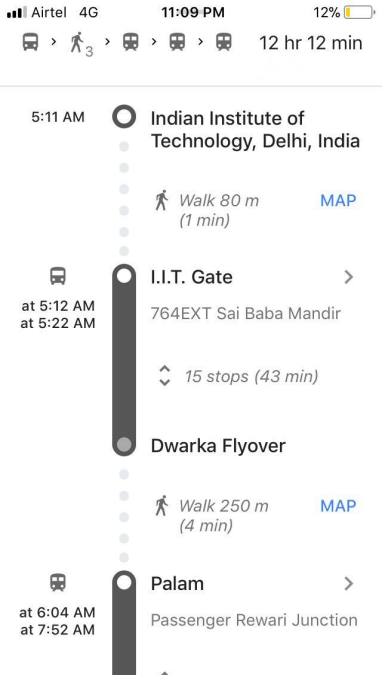
3.1.7 Create New Journey/Trip



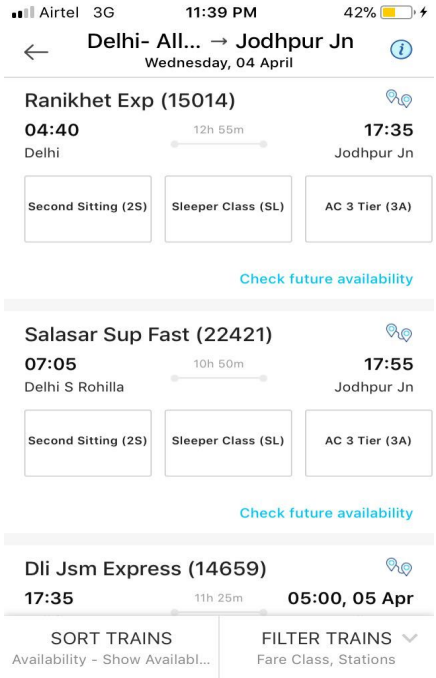
3.1.8 Trip Details



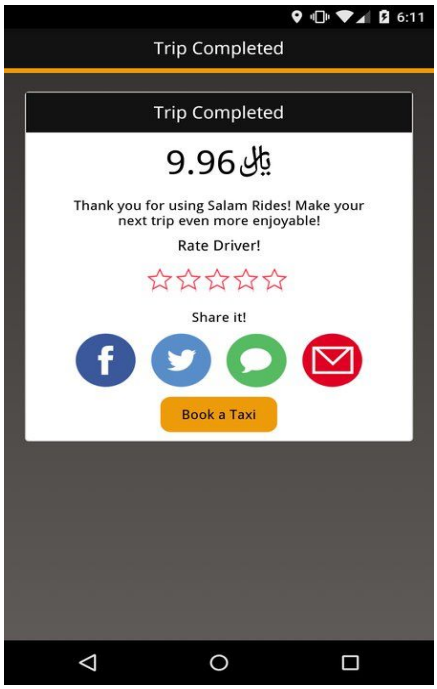
3.1.9 Modify Route/Checkpoints



3.1.10 Select Transport



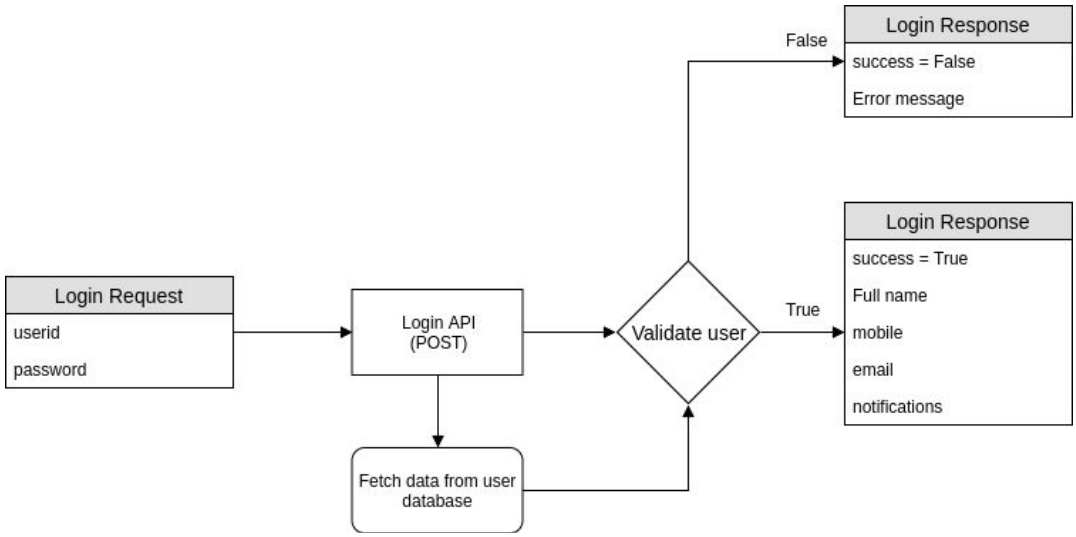
3.1.11 Travel Completed/Rate Travel



3.4 Use Cases

3.4.1 Logistics-related

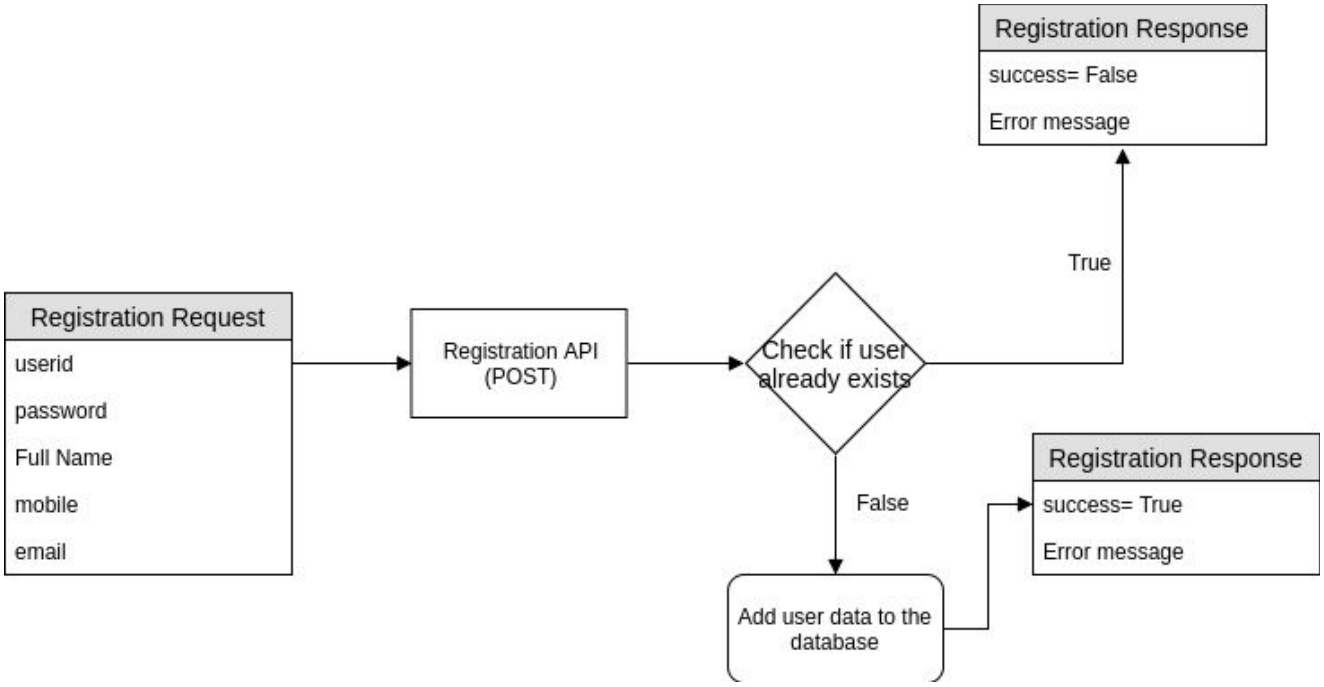
3.4.1.1 User Login



User login API

```
login(userId, pwd)
{
      request = authentication.request_login(user_id, pwd);
      isValid = authentication.authenticate_login(user_id, pwd);
      if (isValid == True)
      {
          data = user.get_profile(userId);
          view.display_data(data);
      }
      else
      {
          view.show_error_screen(authenticationError);
      }
}
```

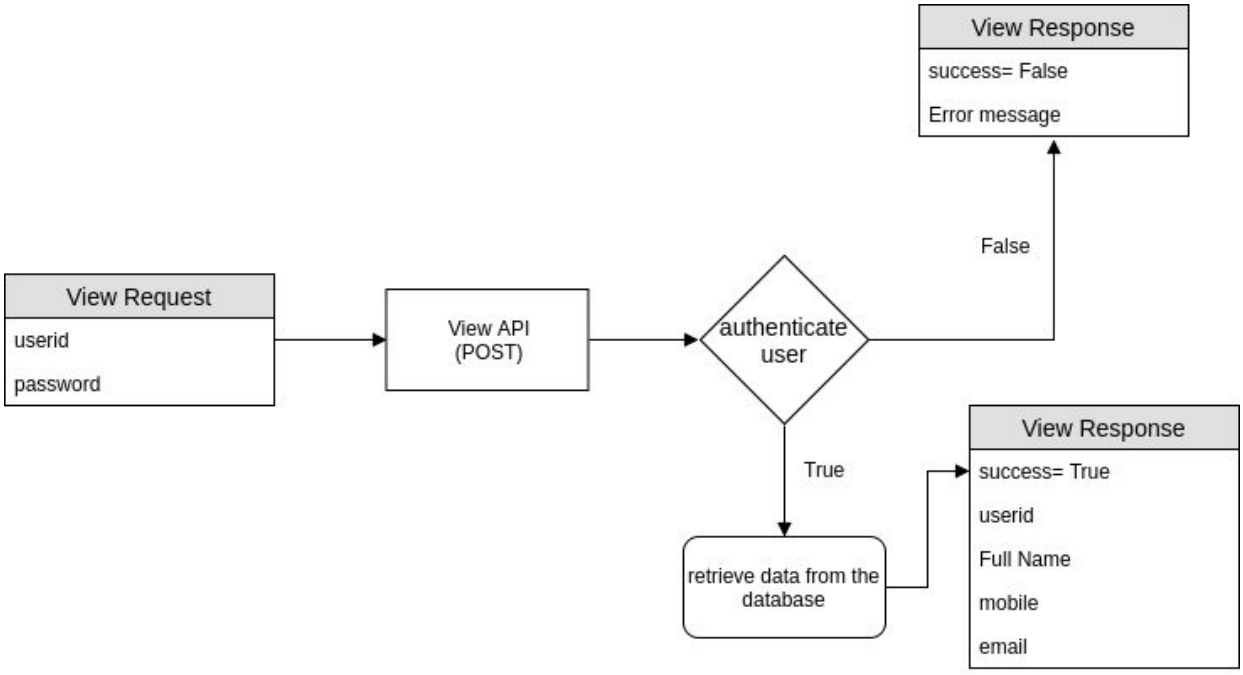
3.4.1.2 User Registration



User Registration API

```
registration(form)
{
      request = user.request_registration(form);
      isValid = user.authenticate_registration(form);
      if (isValid == True)
      {
          data = user.registration_response(request);
          view.display_data(data);
      }
      else
      {
       // There can be a lot of errors like username already taken,etc.
          view.show_error_screen(Error);
      }
}
```

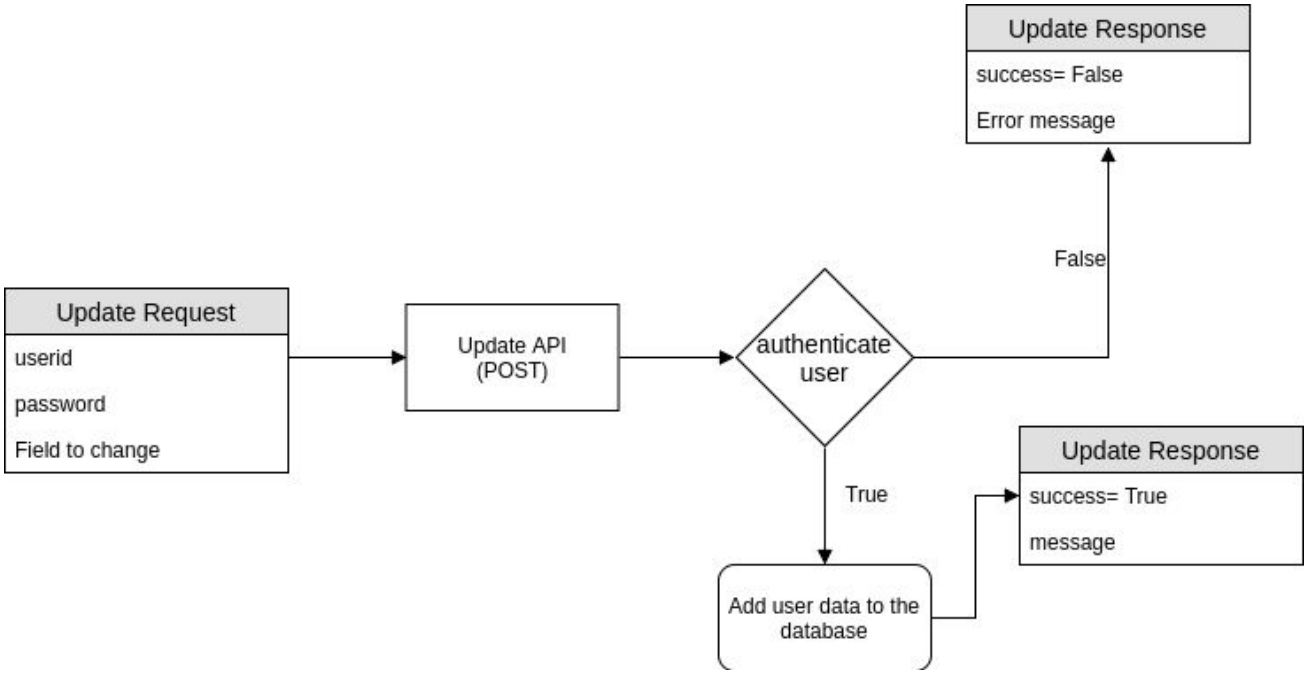
3.4.1.3 View Profile



View profile API

```
view_profile(userId)
{
    data = user.get_profile(userId);
    view.display_data(data);
}
```

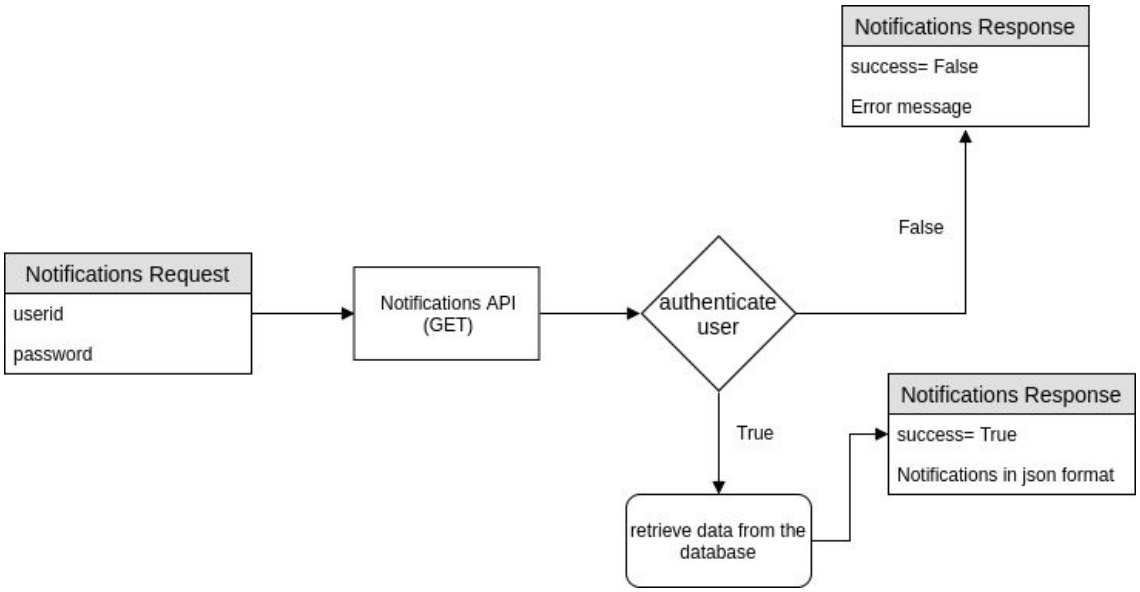
3.4.1.4 Update Profile



Update profile API

```
update(userId, pwd, data)
{
    user.request_login(userId, pwd);
    profile = user.get_profile(userId);
    for field in data
    {
       if(field in profile empty)
       {
             profile.make_key(field);
             profile[field] = data[field];
       }
       else
             profile[field] = data[field];
    }
}
```

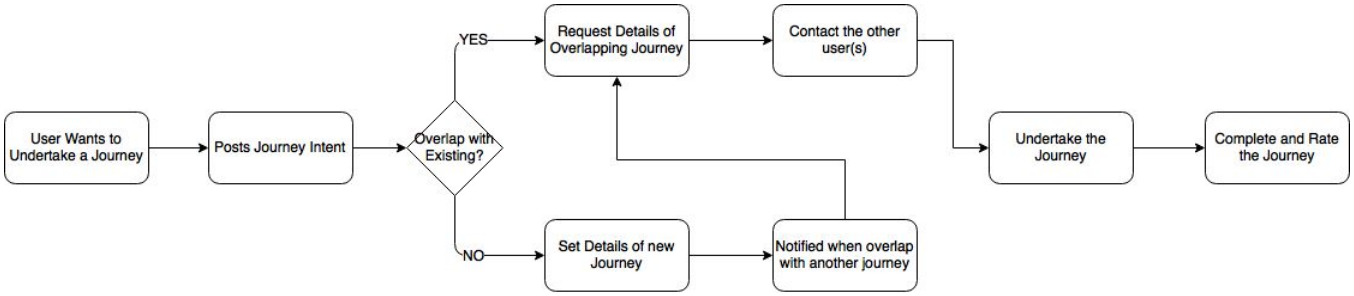
3.4.1.5 Notifications



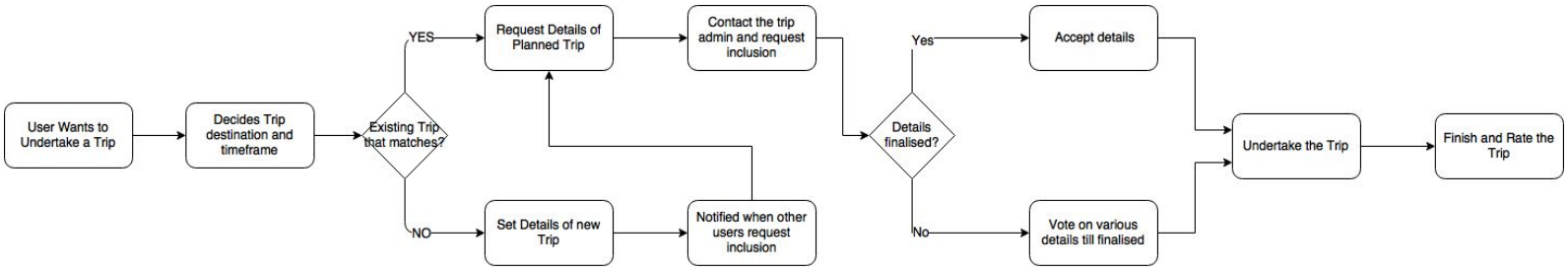
Notifications API

```
notification(userId, notif)
{
    notifications.set_notification(notif);
    while(True)
    {
       isSatisfied = notifications.check_satification(notif);
       if(isSatisfied==True)
             notifications.display_notification(notif);
       if(notifications.turn_off(notif)
       {
            break;
       }
    }
}
```

3.4.2 Undertaking a Journey/Trip

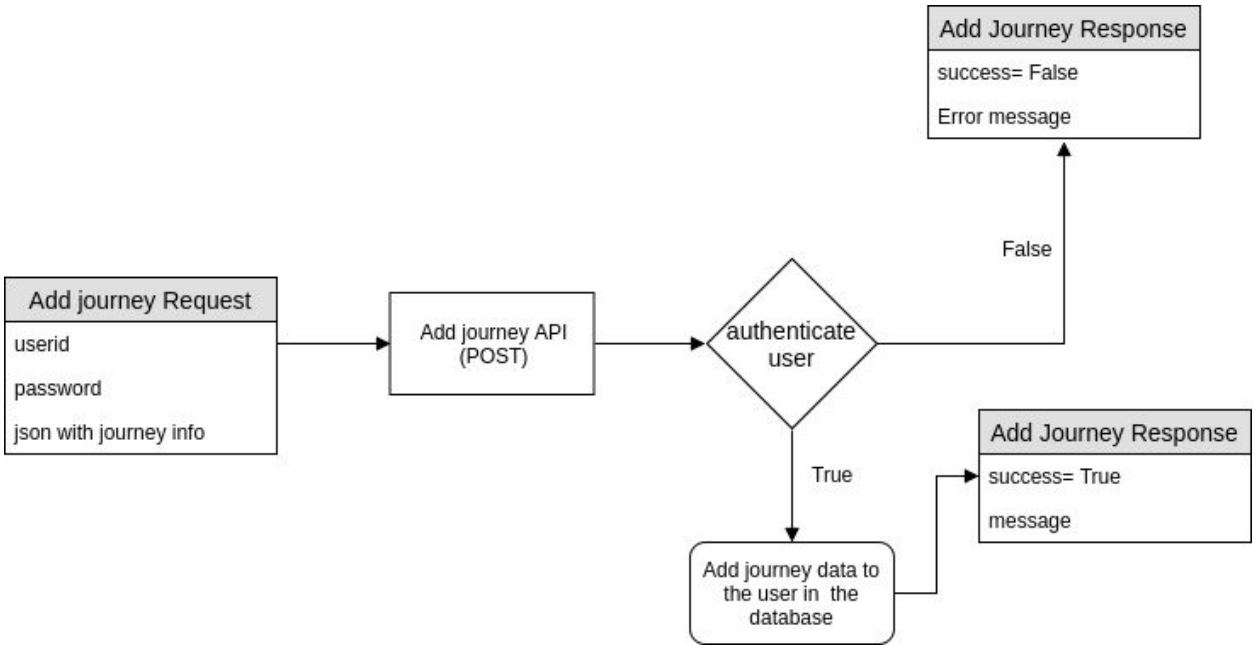


Overall flow for Journey



Overall flow for Trip

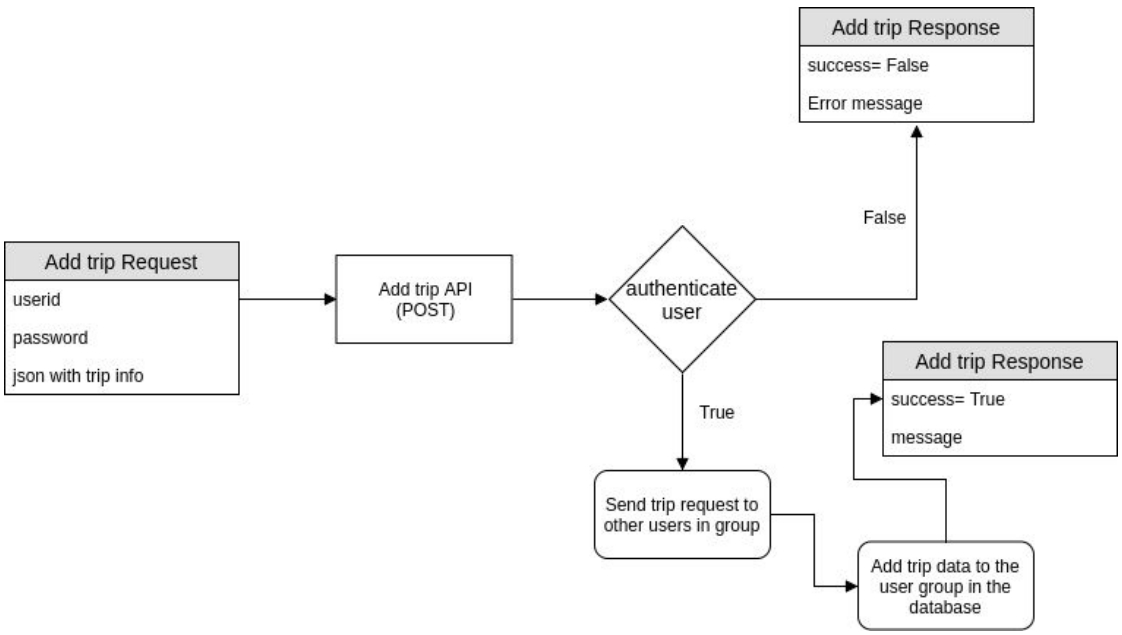
3.4.2.1 Create Journey



Add Journey API

```
create_journey()
{
    journeyObject = journey.new();
    journeyObject.specify_endpoints(startPoint, endPoint);
    journeyObject.specify_tentative_date(dateTime object);
    journeyObject.createCheckpoints(checkPoint, cost, meansOfTenasport);
    matching_journeys = search.search_journey(journeyObject);
    if(there is some matching journey) {
       view.display_data(matching_journeys);
    }
    else
    {
       view.display_message("Sorry! No overlapping journey exists. Do you
want to set notification?");
       input = view.take_input();
       if(input=="Yes")
             Notifications.set_notification(journeyObject);
    }
}
```

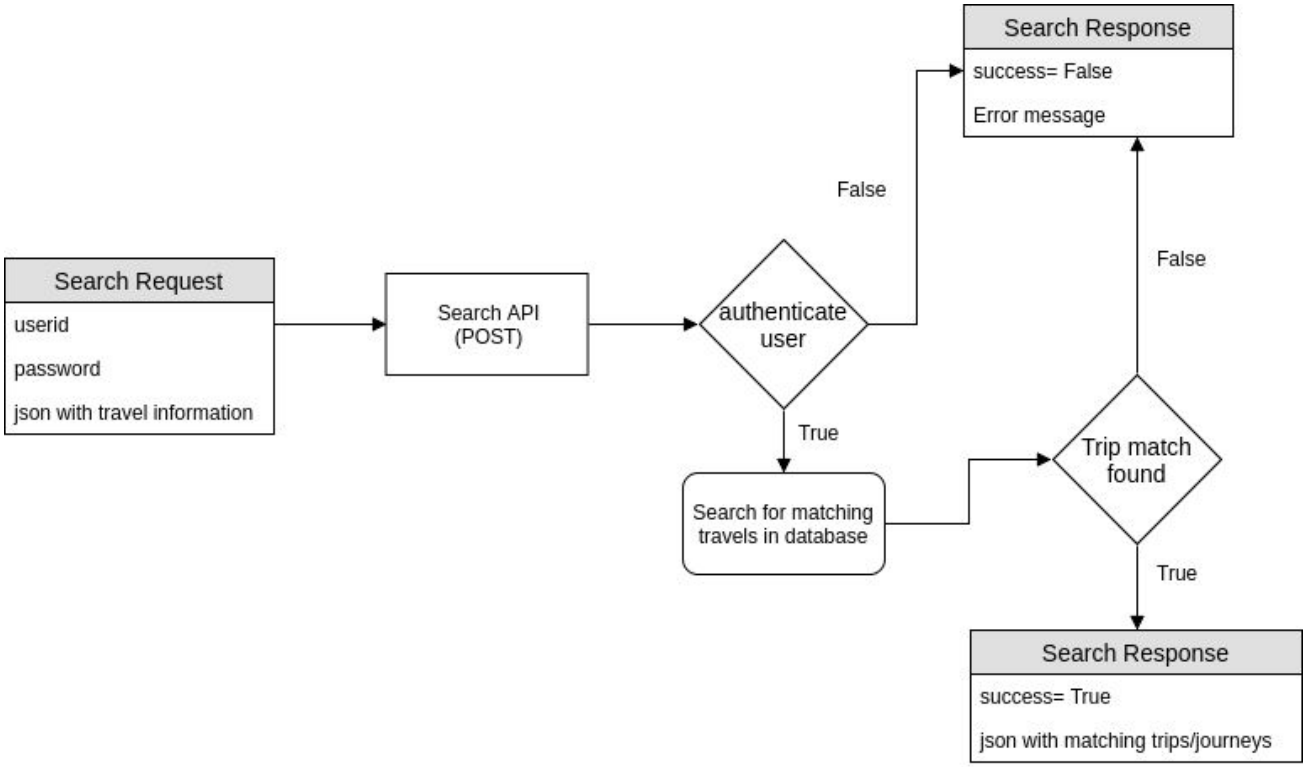
3.4.2.2 Create Trip



Add trip API

```
create_trip(userId)
{
    trip = create_trip_intent(tripPoint, budget, tentativeDate);
    trips = search.search_trip(trip);
    if(there is some matching existing trip and user wants to join)
    {
       view.display_data(trips);
       user.request_contact_info();
       //Contact the leader of the trip and ask to be added to the trip.
       if(user==trip.leader)
       {
             trip.add_traveller(userId);
       }
    }
    else
    {
       view.display_message("Do you want to create a new Trip?");
       input = view.take_input();
       if(input=="Yes")
       {
             tripObject = trip.new(leader=userId)
             trip.create_trip(tripObject);
       }
       if(input=="No")
       {
             view.display_message("Do you want to set notification?");
             input = view.take_input();
             if(input=="Yes")
             {
                   notifications.set_notification(trip);
             }
       }
    }
}
```

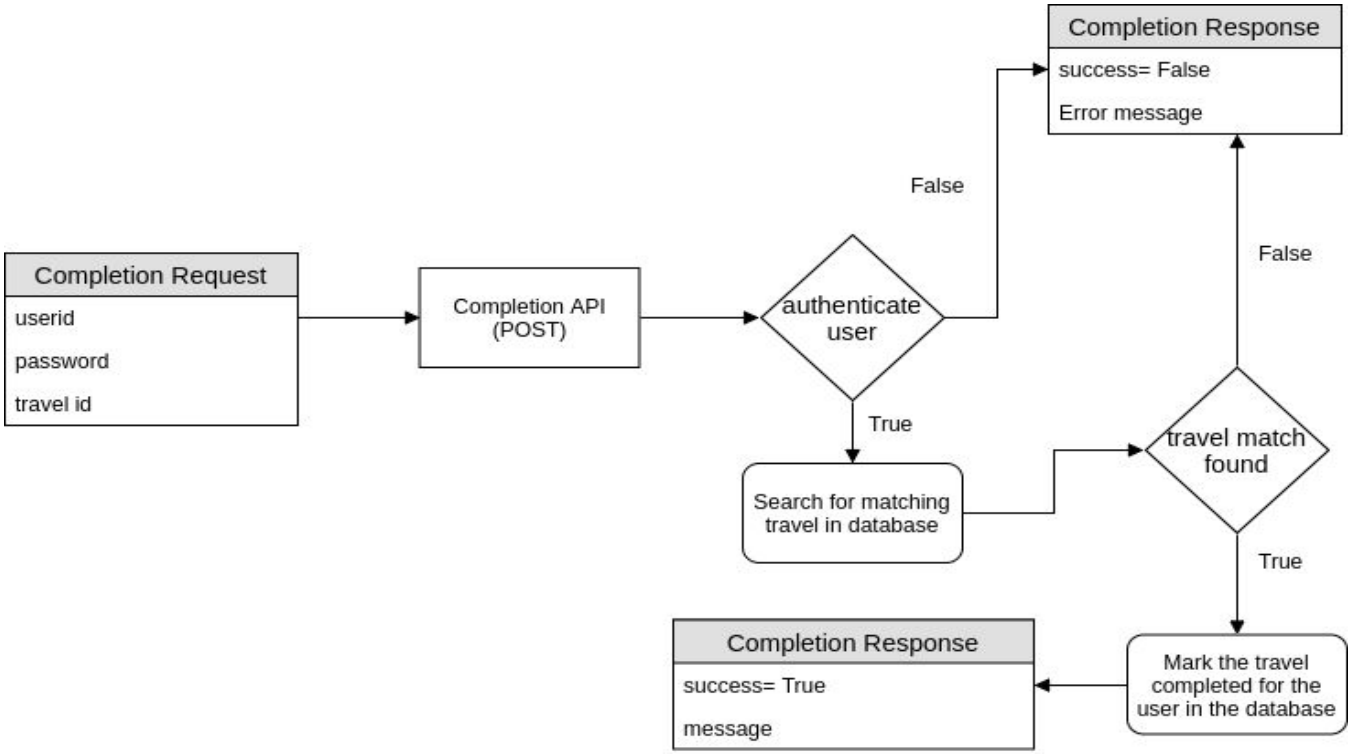
3.4.2.3 Search for Trip



Search API

```
search_trip()
{
    view.display_message("Please provide the trip details")
    trip_details = view.take_inputs();
    data = search.search_trip(trip_details);
    if(some matching trip exists)
    {
       view.display_data(data);
    }
}
```

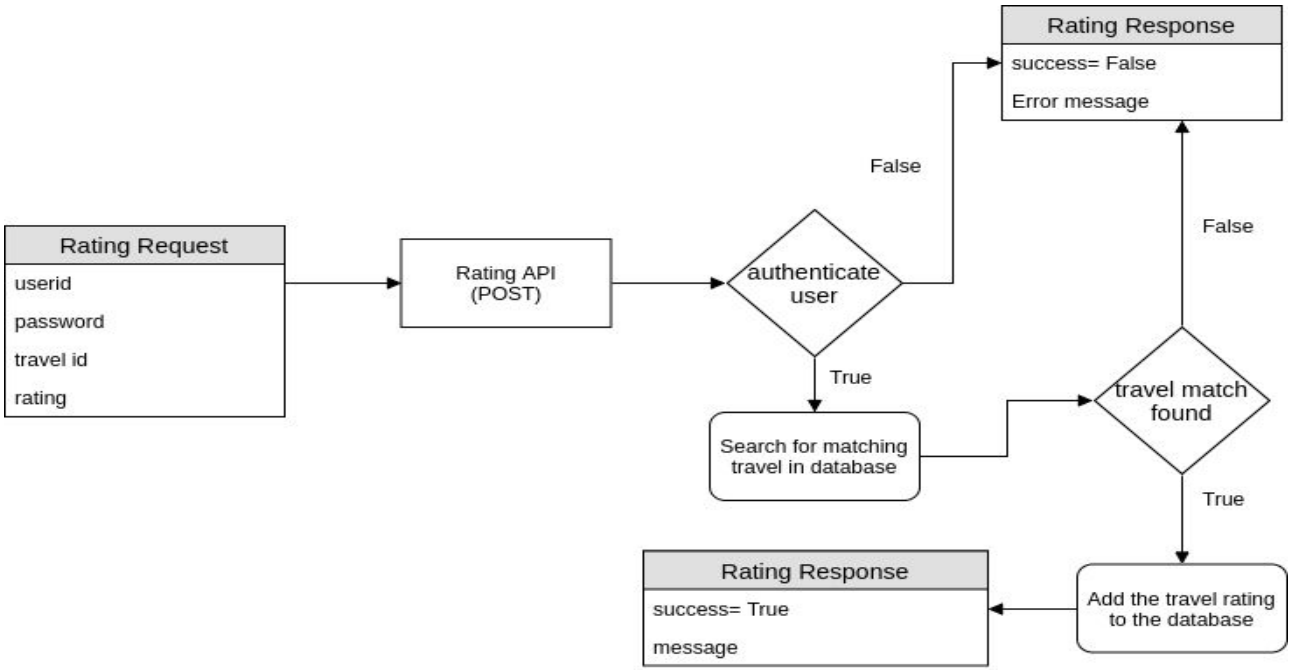
3.4.2.4 Travel Completed



Travel Completion API

```
close_trip(trip)
{
    tripObject = search.search_trip(trip);
    for(traveller in trip.travellers)
    {
       view.display_message("Do you want to close the trip?")
       input = view.take_input()
    }
    if(all inputs=="Yes")
    {
       trip.close_trip(tripObject);
    }
}
```

3.4.2.5 Rating Travel



Rating API

```
rate_trip(tripObject)
{
    if(location in tripObject.locations)
    {
       rate_location(tripObject, rating, location);
    }
    rate_leader(tripObject, rating);
}
```

4 Deployment Design

4.1 Environment

- The application development environment will be Android Studio 3.1
- Android SDK for Oreo will be used as target
- The application will support Android 4.4 (Kitkat) and above
- The project will use Gradle as the build system

4.2 Version Control

- Version Control System - Git
- Platform - GitHub or BitBucket
- Separate repositories for server and client-side, to avoid exposing client and server code with each other

4.3 Testing and Debugging

4.3.1 Functionality Testing

- **Actions**: Check that all the buttons and gestures work like intended, in various scenarios.
- **Inputs**: Check all the input/form data is successfully submitted and processed at the webserver.
- **Database**: Check if all queries and updates are consistent with the integrity constraints.

4.3.2 Usability Testing

- **Navigation**: Ensure that navigating through the app is intuitive and familiar.
- **Content**: Verify that the content is what you intended to be. Check spellings, fonts, colors etc.

4.3.3 Interface Testing

Ensure that any disconnection between the web-server and the app is handled, reported and logged properly.

4.3.4 Compatibility Testing

- **Device compatibility**: Test the application to be compatible with all the devices with targeted version of Android and the hardware features required by the application.

- **Server compatibility**: Test the server application to be compatible with Windows as well as Ubuntu servers with appropriate changes.

4.3.5 Performance Testing

- **App Stress Testing**: Verify that app does not break break or behave rampantly on invalid inputs and lifecycle actions (like switching, screen rotation, app destroyed due to low memory).
- **Load Testing**: Verify that the server reacts normally with high number of users putting load on the database and specific pages by posting or fetching huge data.

4.3.6 Security Testing

- **Authentication**: Ensure that application can only be accessed after logging in.
- **Logging**: All transactions and error messages must be logged so that security breaches can be studied later.