Smart Traveler Guide (Traveler part)

(CS 872: Software Engineering)

Simranjeet Randhawa (200412297)

Table of Contents

Problem and Requirements

Problem Definition

In today’s world, mobile devices are becoming very intelligent and many travelers have started using their mobile devices to explore the places around them. These travelers either use a travel guide application or take assistance from a tourist guide to explore the new city. There are many applications available for travelers in the market to find cheap hotels, car rentals and best dining places at the new city. But only very few applications like Sidekix offers route planner to help the travelers find the best way to get anywhere around the city without the help of a tourist guide. Also, the existing application does not consider about the user’s likes and dislikes and their availability during a day. Our mobile application as the name implies smartly examines user’s likes and the time duration the user is expected to explore a new city/place. The proposed system provides an astonishing result in the form of 2 best route to explore the city. The user is asked to fill some questionnaire when the user logs in the system. The questionnaires include questions like the user liking, start time, duration and place the user wants to explore. The proposed application smartly scans for the place based on user-specified information and helps the user to discover the places adjacent to them or throughout the world. The application displays a map with detailed description about a particular place. This application significantly minimizes the time needed by the travelers to search for a place.

Motivation

The main motivation behind developing this system was to serve the purpose of a travel guide. There are many applications in the market that are working to serve the purpose of a travel guide but only a few consider users likings and their availability to search the popular places. The smart city traveler app retrieves the popular travel places by considering user likes and time preferences. Most of the applications in the market provide the route to the user while does not consider the likes and time duration for the user. Our main motivation was to allow the user to search popular places keeping in mind the time duration the user have to explore a new place. The purpose was to replace traditional traveler guide applications that only displays all the places without considering the user likes or dislikes. This application was made keeping in mind the likes of the user.

Roles

- Traveler (primary user)
- Administrator (secondary user)

Software Quality Requirements

- Correctness
    - The system logs into the correct account to display proper data and travel plan.
- Robustness
    - The system displays with a warning or error message when incorrect data is entered by the user.
- Time-efficiency
    - System returns and displays travel plan information for users in an acceptable amount of time.
- Portability
    - Runs on both Android and iOS platform.
- Usability
    - Easy to navigate and understand the application.
- Security
    - Basic credential authentication to allow only registered users.
    - JSON Web Tokens (JWT) based authentication to the application.
- Maintainability
    - Change in one function will not affect the functionality of the whole system. It follows two design pattern – Observer design pattern and Builder design pattern.

Economic Feasibility Study

The Smart Traveler Guide serves as an improvement on existing systems. The main purpose of this app takes the likes of the users and then give the places the user can visit. Few applications like Sidekix offers route planner to help the travelers find the best way to get anywhere around the city without the help of a tourist guide. Also, Sidekix does not consider the user's likes and dislikes and their availability during a day. Our mobile application smartly examines user's likes and dislikes and the time duration the user is expected to explore a new city/place. The proposed system provides 2 plans that the traveler user can explore.

The smart city traveler app works both on Android and iOS. This cross-platform feature makes the app unique. This was possible because of the native script framework. The smart city traveler app provides places according to user likes and preferred timings. Depending on the duration entered by the user the coverage are can be extended. If the user enters more duration, then the coverage will be greater as it is understood that the user will have more time to explore the city.

Another improvement as compared to the traditional traveler guide app is that our application will only display the places open during the user timings. The traveler enters the time they want to explore a particular place and based on the time the search results are retrieved i.e. only places opened at that particular time are retrieved. Some traditional traveler apps do not have this feature in them. So, this way this app overcomes the limitation of traditional traveler applications.

Some major points that this app covers that other traditional applications does not:

- Search places based on user likes.
- Respect the time interval the traveler inputs.
- Places are retrieved according to the popularity of the place.
- Also provides the user with an alternate plan so that user can switch plans if not liking the plan.
- Works on both platform android plus iOS.

Traveler will no longer have to download separate apps if they switch from android to an iOS device as the smart city traveler works on both the platform. This was possible using Native Script (a framework to develop both iOS and Android apps). Comparing the app with some of the competitor's applications:

Comparison with Competitor application:

1. *Visit a City App*

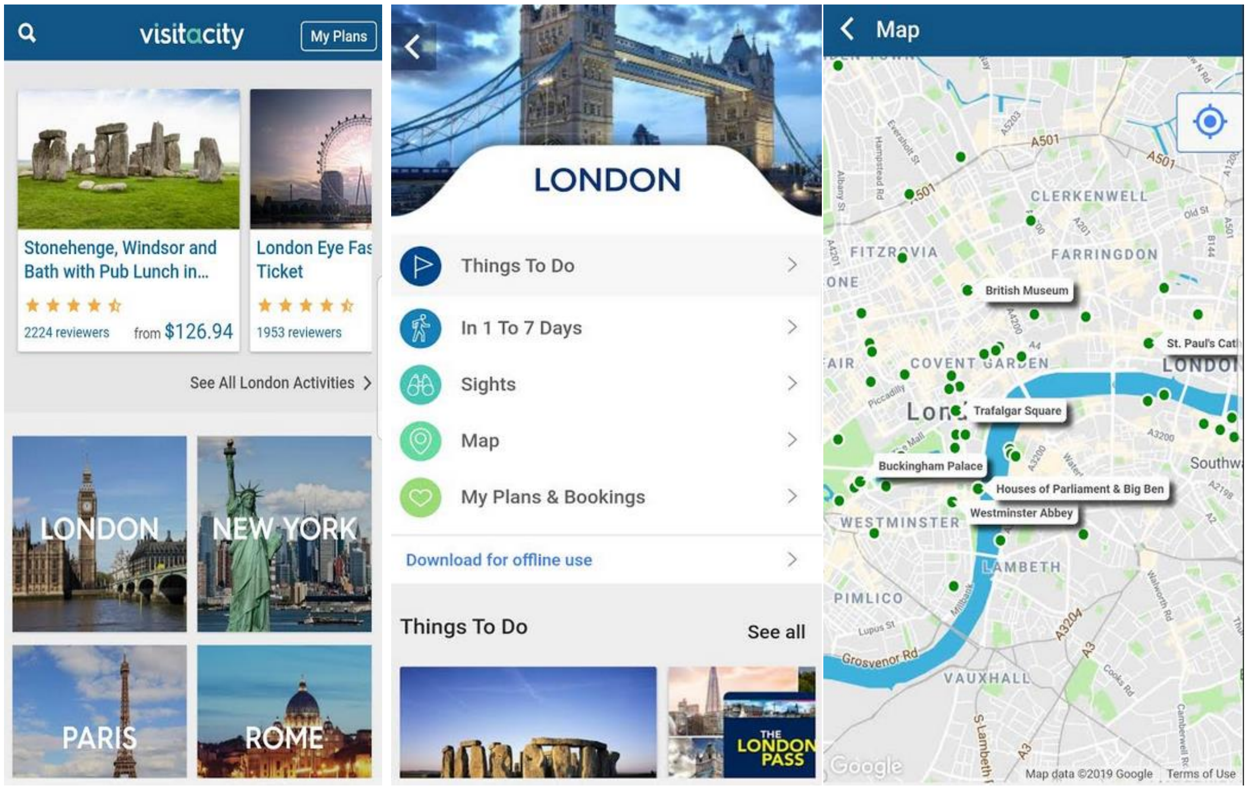


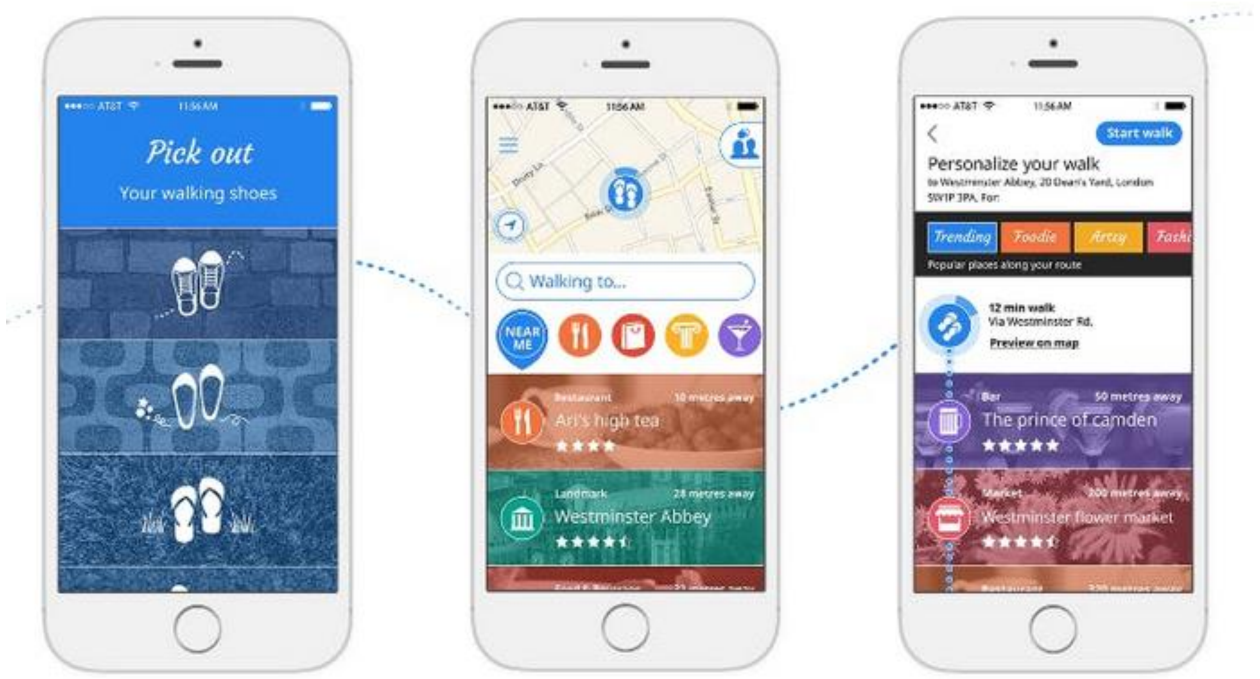
Figure 1

- Visit a city application gives the schedule by day and time. But it do the give the option to choose the time manually instead it gives the whole day schedule. On the other hand my application takes the time as input and then search the places according to the time. The closed places will not be shown using my application at that particular point of time.

- Visit a city application display all the tourist spots not considering the likes of the user. While on the other hand my application does show the places the user is usually looking for. The traveler can easily see popular places based on the likings and timing as parameters.

- Cross-platform support not provided by Visit a City App but our app supports cross-platform.

2. *Sidekix*



- Sidekix offers a route planner to help the travelers find the best way to get anywhere around the city without the help of a tourist guide. Also, the existing application does not consider the user's availability during a day. Whereas my application accepts the user likes and also values the time entered by the user.

- Unlike Sidekix, my application will cover thousands of places that Sidekix does not cover. My application takes advantage of the Four Square API to get the popular places in the vicinity.

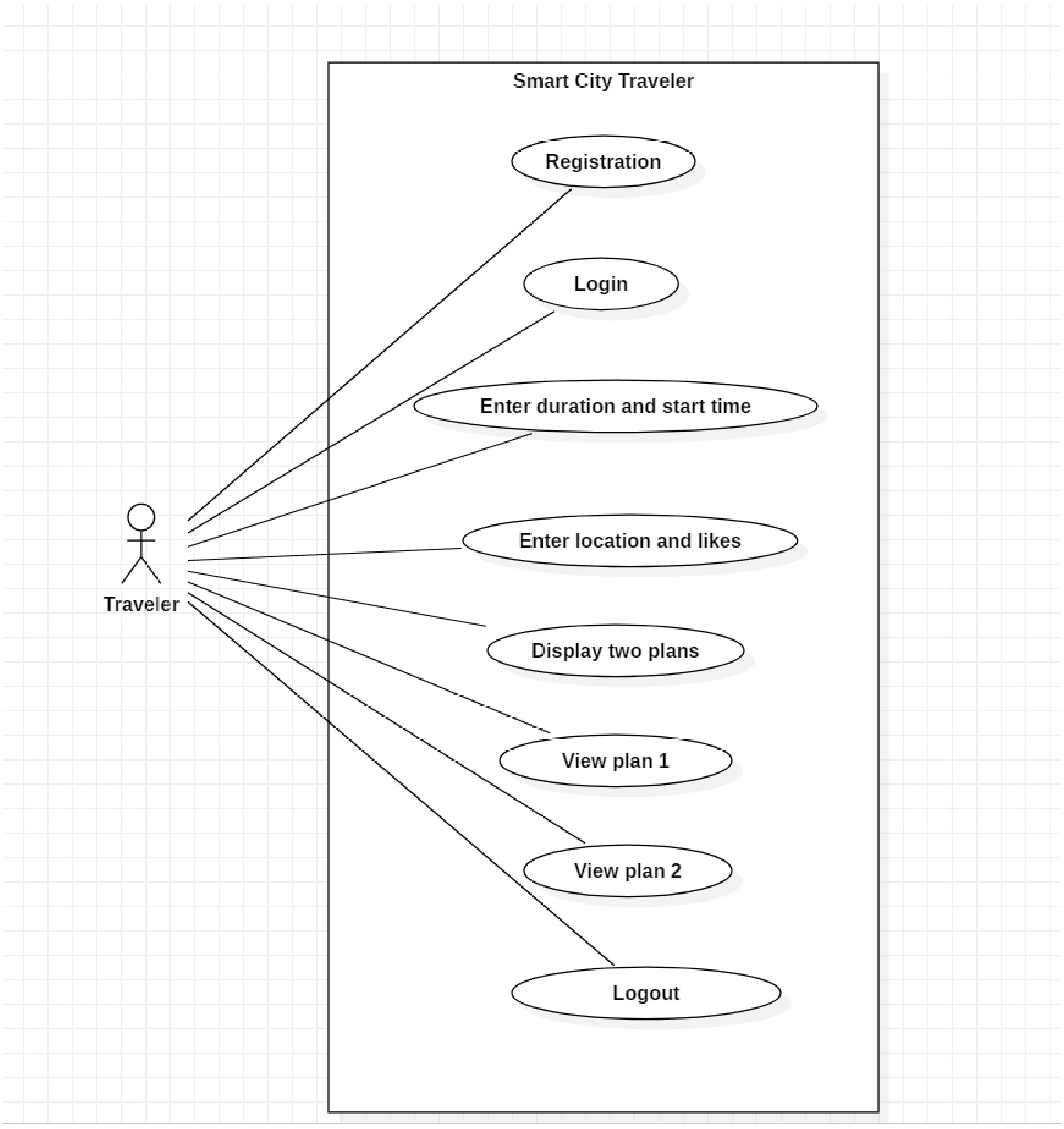
Software Requirement Specification Document

Functional requirements list (Traveler):

- A signup page for travelers to register for the mobile application.
- A login page to allow only registered users to log in and prevent any unauthorized access.
- When user login home module gets opened where a form is to be filled by the user.
- Home module with an option for the traveler to enter the timing, preferred location, their likings and the time duration they want to explore the place.
- When the user clicks submit button in the home page, the system displays a list of 2 travel plan option for the traveler to select.
- The traveler can view the plan on the map by selecting the view plan button.
- The map displays all the places the traveler can visit in the region the traveler wants to travel.
- When the user clicks the finish button, dissolve the travel plan.
- Logout option to sign off from the application.

Use Case Diagrams:

Traveler Use Case Diagram:



Use Case Summary’s:

Enter start time and duration:

- Initiating Actor: Traveler
- Pre-conditions: The user must be logged in as a traveler.
- Primary scenario:
    - Enter the start time in hh: mm format (e.g. 12:30) and duration in number of hours.
    - The system displays an error if it is the hh:mm format is not followed.
    - The system displays error message if number of hours exceeds 24.
- Exceptions: none
- Post-conditions: The traveler user will be allowed to enter location and likes.
- Benefiting actor: Traveler as a user.

Enter location and likes:

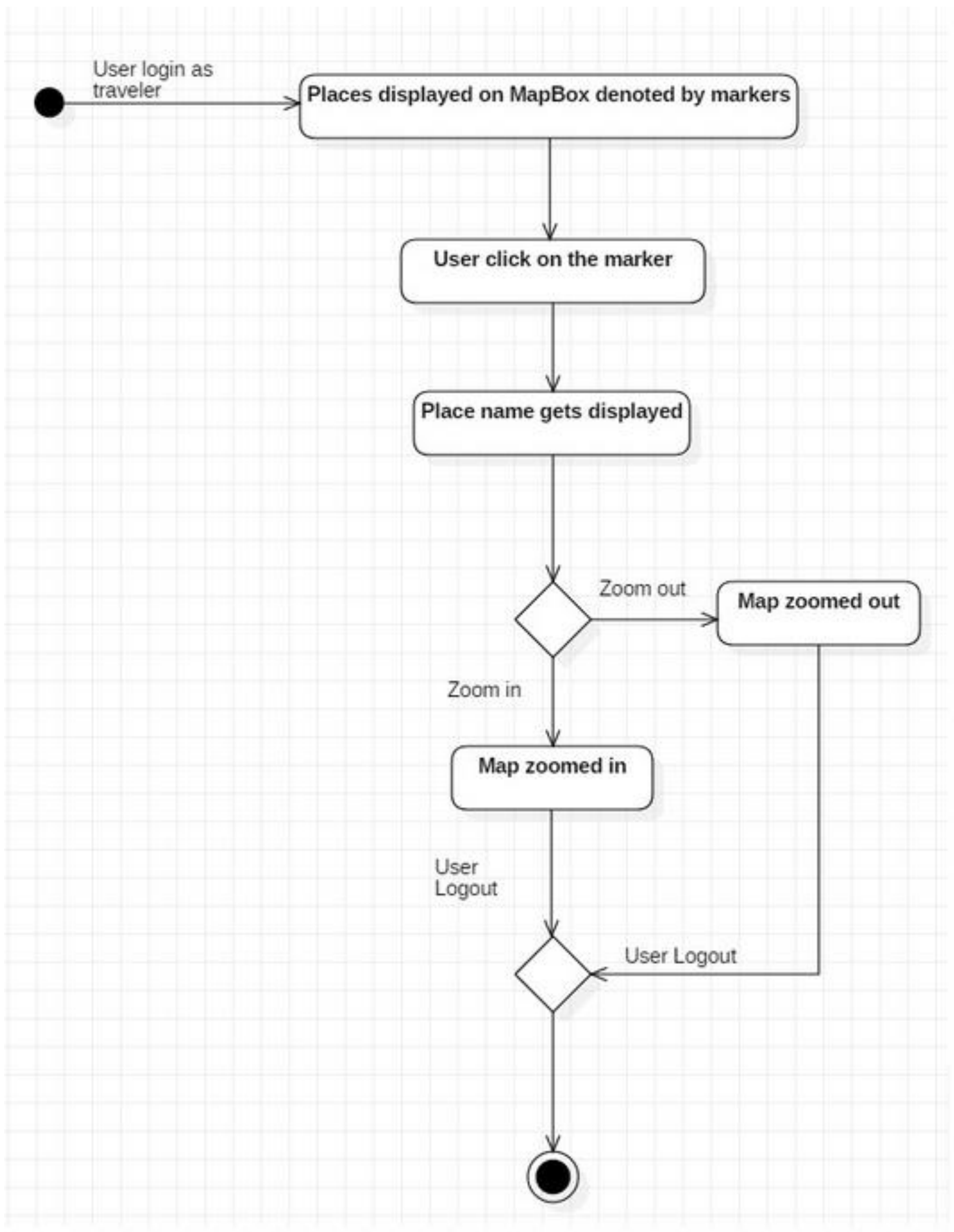
- Initiating Actor: Traveler
- Pre-conditions: Traveler must be logged in to their account.
- Primary scenario:
    - The location and likes are entered by the traveler.
    - The system displays an error if the location specified by the traveler is not valid.
    - Multiple likes can be entered by the traveler (e.g. food, drinks, etc.).
    - The system will pick popular spots if the preferences are not entered.
- Exceptions: none
- Post-conditions: The plans get displayed to the traveler.
- Benefiting actor: Traveler.

View Plan 1 or 2 Use Case:

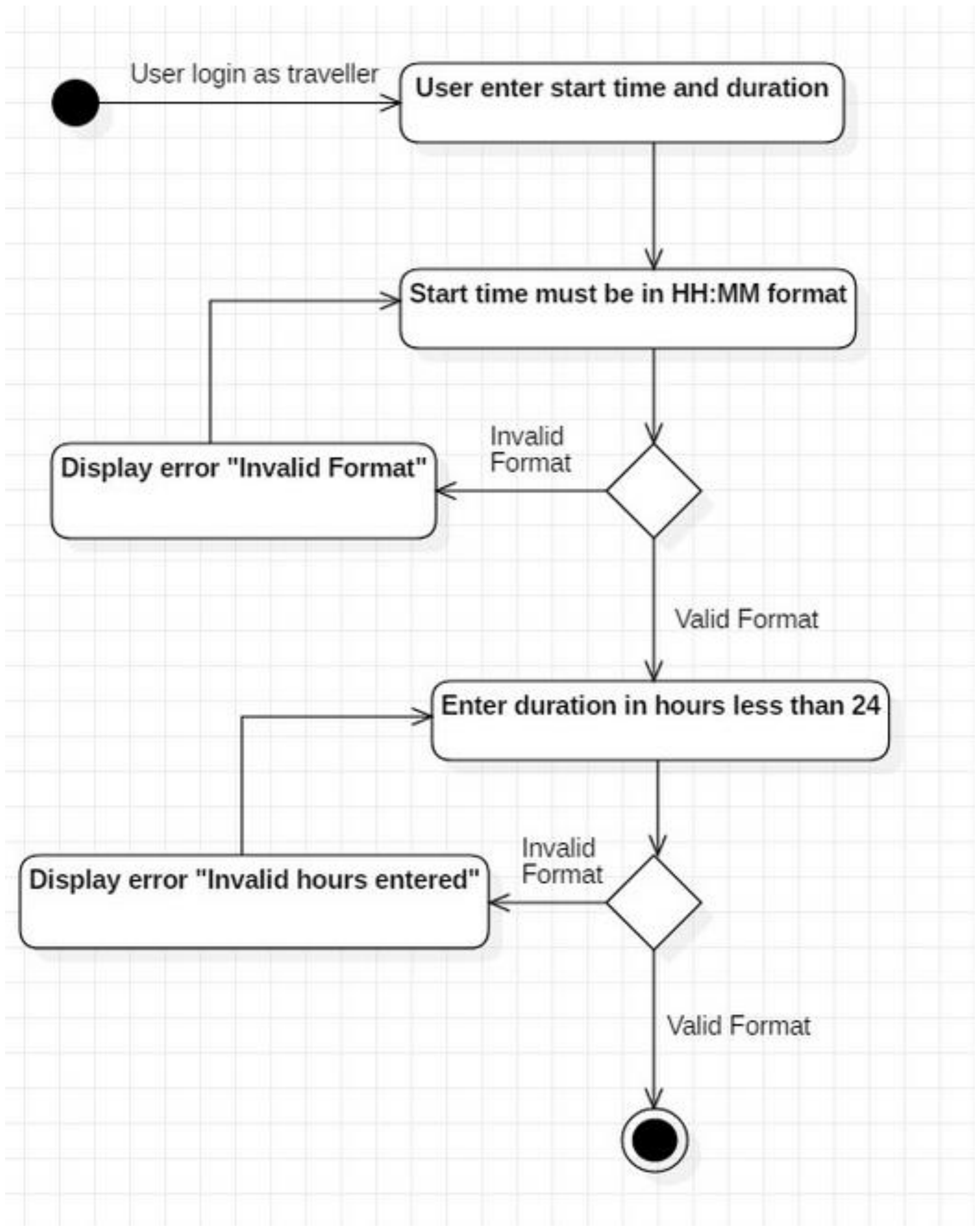
- Initiating Actor: Traveler
- Pre-conditions: Traveler must be logged in to their account.
- Primary scenario:
    - The traveler will be displayed the Map Box with markers specifying the position of the places.
    - The traveler can view place name by clicking on the marker.
    - Traveler can zoom in or zoom out the displayed map.
- Exceptions: none
- Post-conditions: The traveler can visit the locations marked on the map.
- Benefiting actor: Traveler

Activity Diagrams:

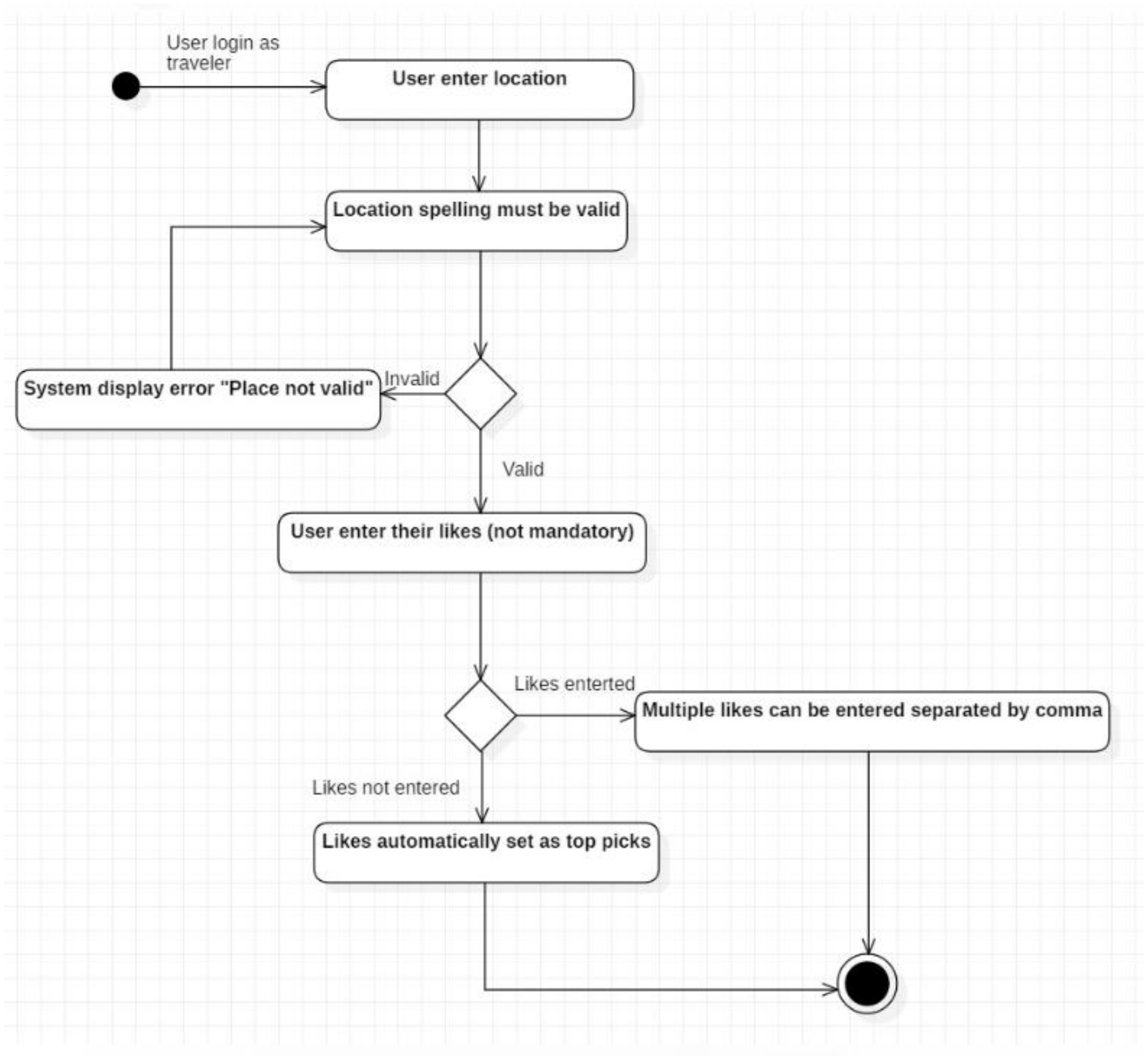
<u>View plan 1 or 2 activity diagram:</u>



Enter start time and duration Activity Diagram:

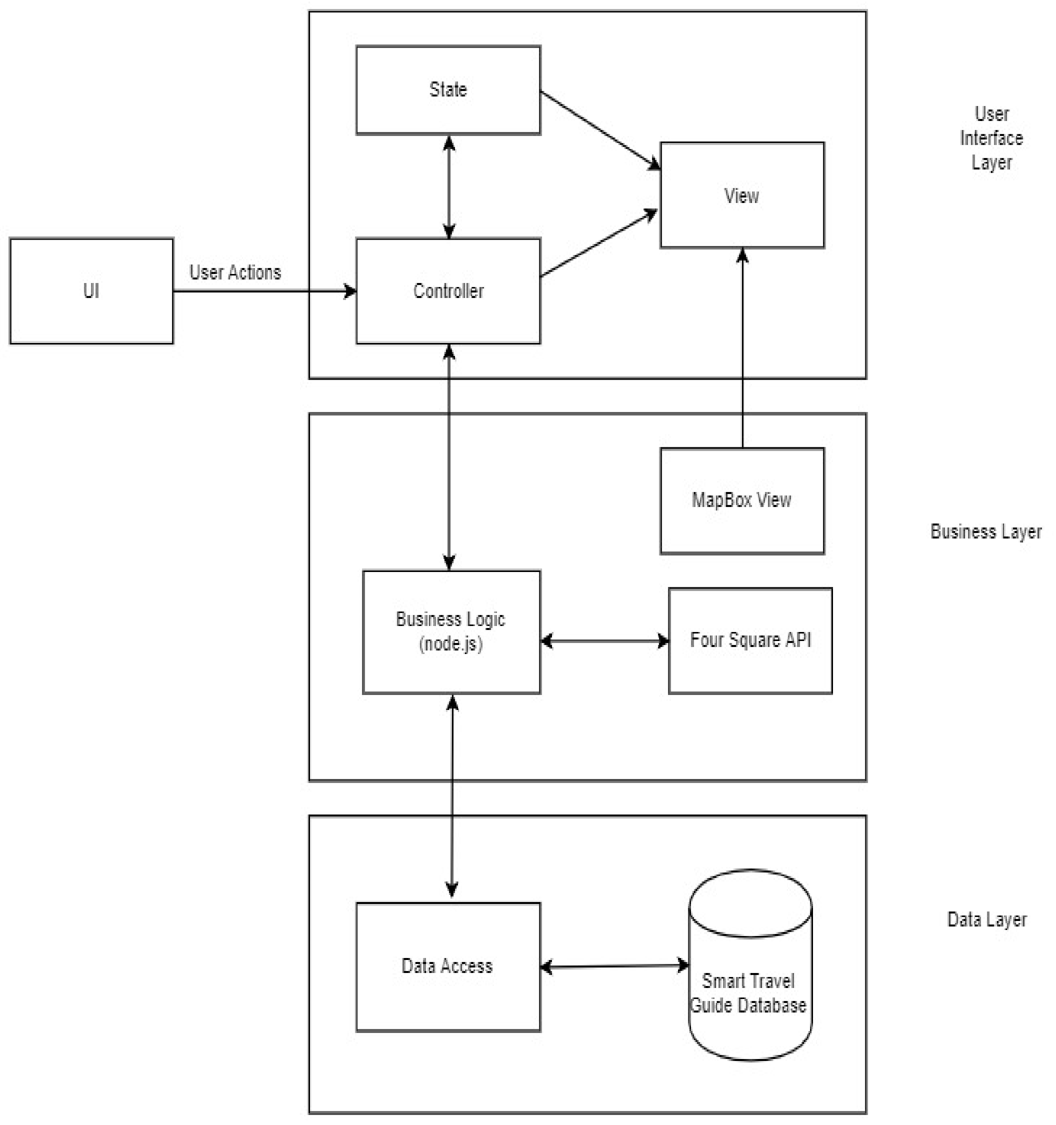

User login as traveller
User enter start time and duration
Start time must be in HH:MM format
Display error "Invalid Format"
Invalid Format
Valid Format
Enter duration in hours less than 24
Display error "Invalid hours entered"
Invalid Format
Valid Format

Enter location and likes Activity Diagram:


User login as traveler
User enter location
Location spelling must be valid
Invalid
System display error "Place not valid"
Valid
User enter their likes (not mandatory)
Likes enterted
Multiple likes can be entered separated by comma
Likes not entered
Likes automatically set as top picks

Design Specification Document

Logical Software Diagram:



Logical Software Summary:

We decided to structure the software over 3 tiers using the N layer architecture. The 3 layers consists of a user interface, one business logic and a data access layer. In addition, Model view controller helps to separate the code such that dependencies can be reduce. This style was chosen because it eased the implementation part as there were very few dependencies.

Some of the features using this architecture:

- Implementation became easy.
- Development faster.
- Testing was easy.
- Well-structured because of separation of the model, view and controller.
- Modification in some parts does not affect other parts.
- High Cohesion and Low coupling.

The Smart City Traveler app is considered a medium size application that consists of 3 layers. The app makes use of Four Square API to get the neighbourhood places.

Characteristics of N layer architecture:

- Secure
- Easy to manage
- Flexible

The separation of code makes it easier to develop as each piece of the architecture is independent. This also speeds up the development process as more than one person can work on the single piece of the website.

User Interface Layer:

We have a model, view and controller in the User Interface layer. It also has user interaction in which the traveler enters his or her preferences depending on the location the traveler wants to explore. The controller is responsible for the interaction with the business logic (node.js) to get the desired response from the server.

Business Layer:

The node.js will get request from the User interface Layer and then it calls the four-square API to get the popular neighbourhood places based on the likes of the traveler. I have used builder pattern in this layer that helps in creation of complex objects easily.

Data Layer:

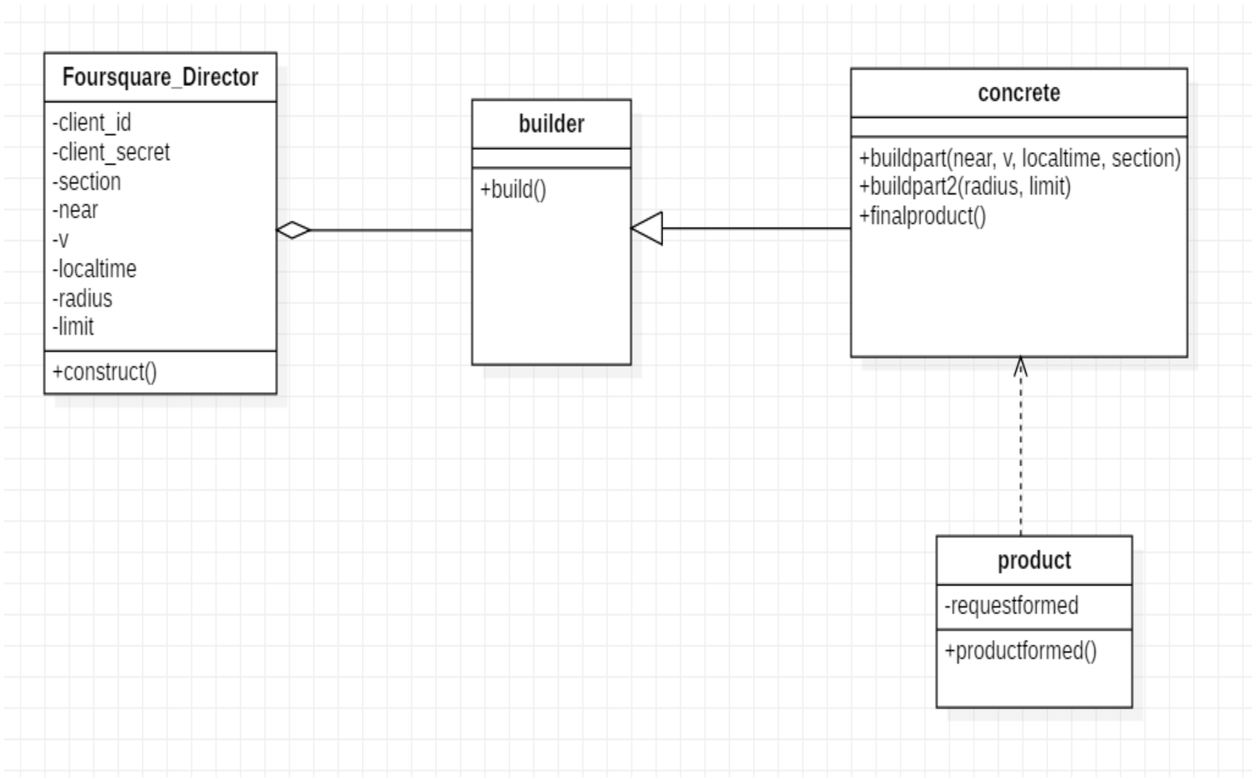
The data tier is responsible of storing the Smart City Traveler database. In this tier, all the data is stored which is used in my application. The password is stored securely in the database using bcrypt.

Motivation behind using this architectural design:

- Code became manageable.
- Model, view and controller makes development process faster.
- Modification in one part does not affect any other part of the application.
- Easier to reuse.

Design Patterns:

Builder Design Pattern Diagram:



1. *Summary*

The Design pattern I chose is the Builder design pattern. This pattern falls under the category of Creational patterns. This pattern is useful for complex object creation. It deals with how composite objects are created.

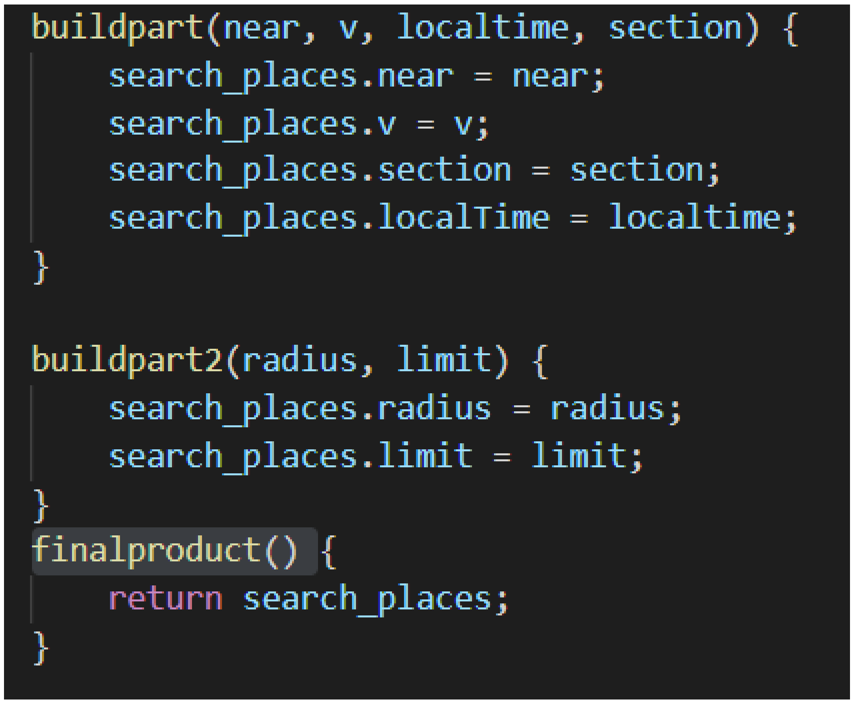
The builder pattern follows a construction process that allows different representation for the objects that are constructed. This design pattern is very useful for my project as during the Four-Square API call an object should be passed as a parameter such that the object can have different representation depending on the traveler's preferences.

2. *Motivation*

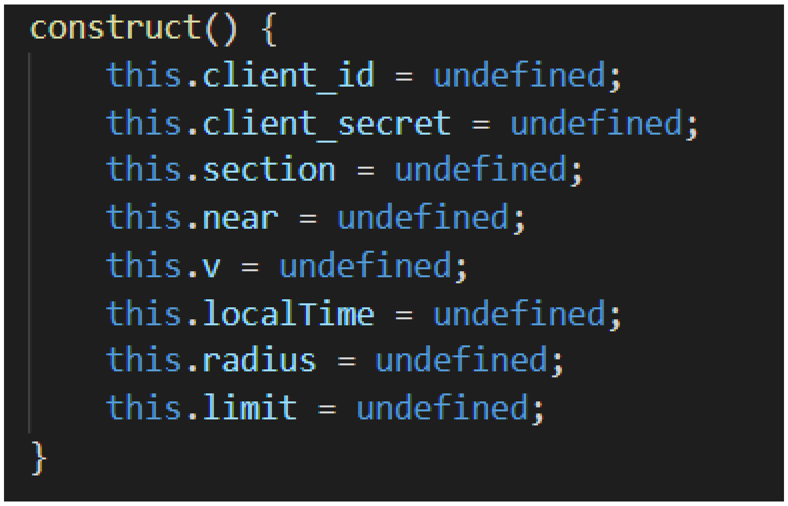
The main motivation that made me follow the builder pattern was the fact of creating complex objects easily when calling the four-square API for the location suggestions. When the traveler submits the preferences. The preferences get passed to the server. The server calls the four-square API. But for calling four square API we require certain parameters like user location, Likes, etc. As we know every user will enter a different parameter, hence this requires a complex object creation that can support all types of user input. For this purpose, I was motivated to use the builder design pattern in my application.

3. *Sample Code*

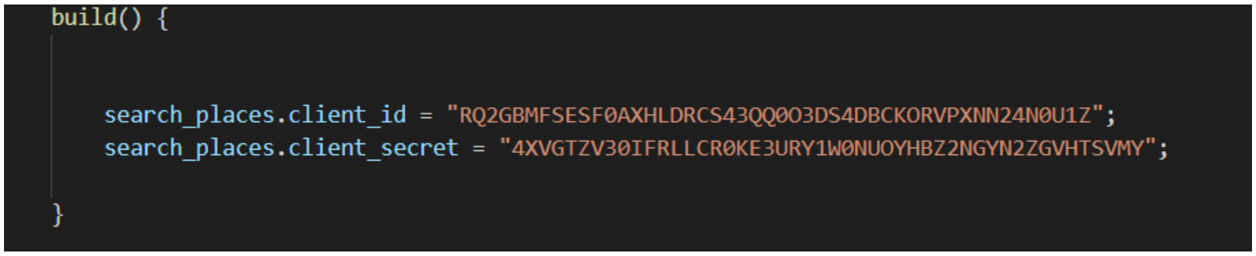
Prototype definition:



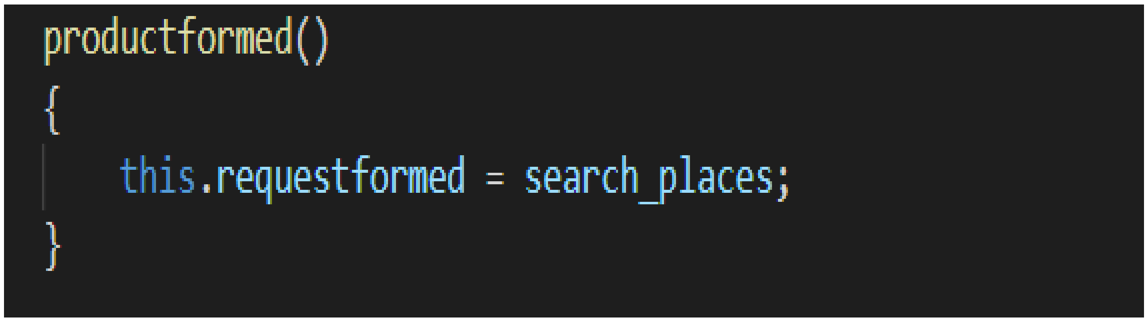
```
buildpart(near, v, localtime, section) {
    search_places.near = near;
    search_places.v = v;
    search_places.section = section;
    search_places.localTime = localtime;
}

buildpart2(radius, limit) {
    search_places.radius = radius;
    search_places.limit = limit;
}
finalproduct() {
    return search_places;
}
```



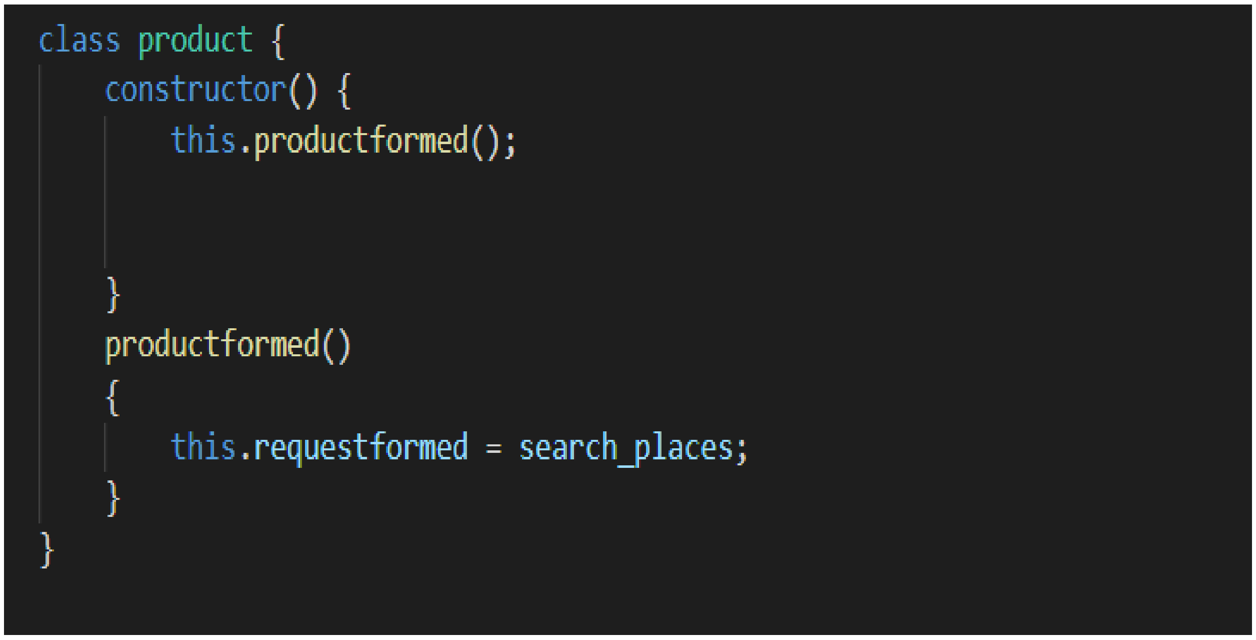
```
construct() {
    this.client_id = undefined;
    this.client_secret = undefined;
    this.section = undefined;
    this.near = undefined;
    this.v = undefined;
    this.localTime = undefined;
    this.radius = undefined;
    this.limit = undefined;
}
```



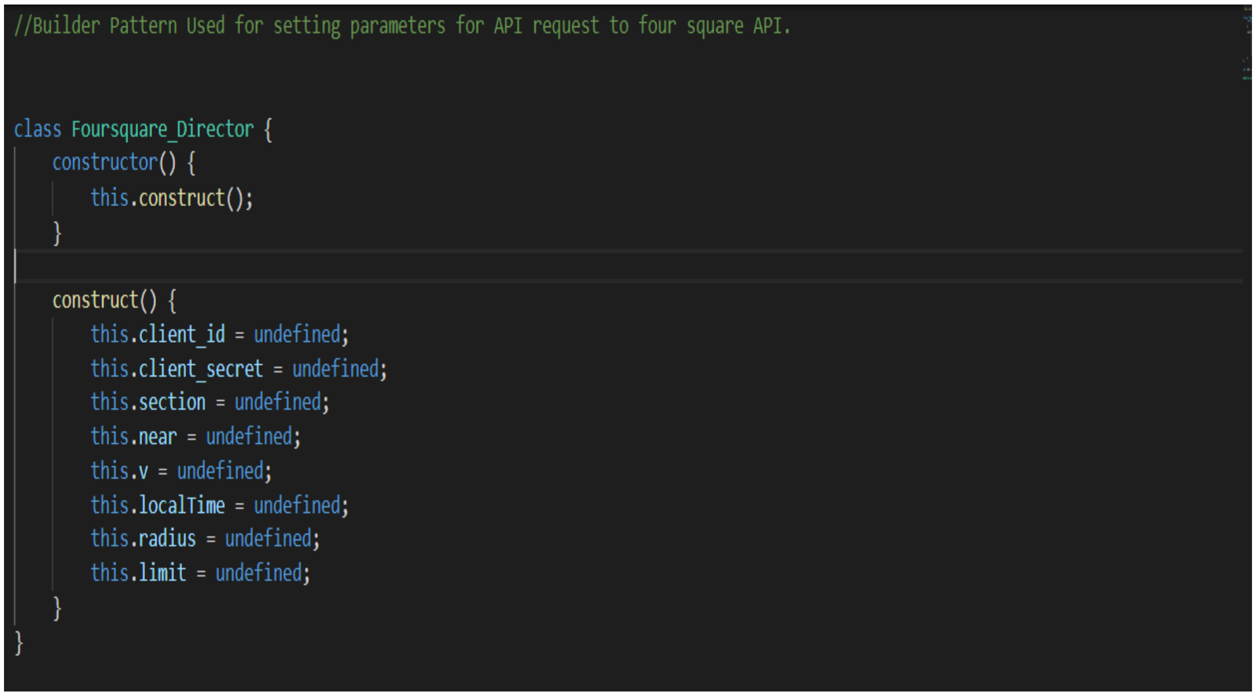
```
build() {


    search_places.client_id = "RQ2GBMFSESF0AXHLDRCS43QQ0O3DS4DBCKORVPXNN24N0U1Z";
    search_places.client_secret = "4XVGTZV30IFRLLCR0KE3URY1W0NUOYHBZ2NGYN2ZGVHTSVMY";

}
```



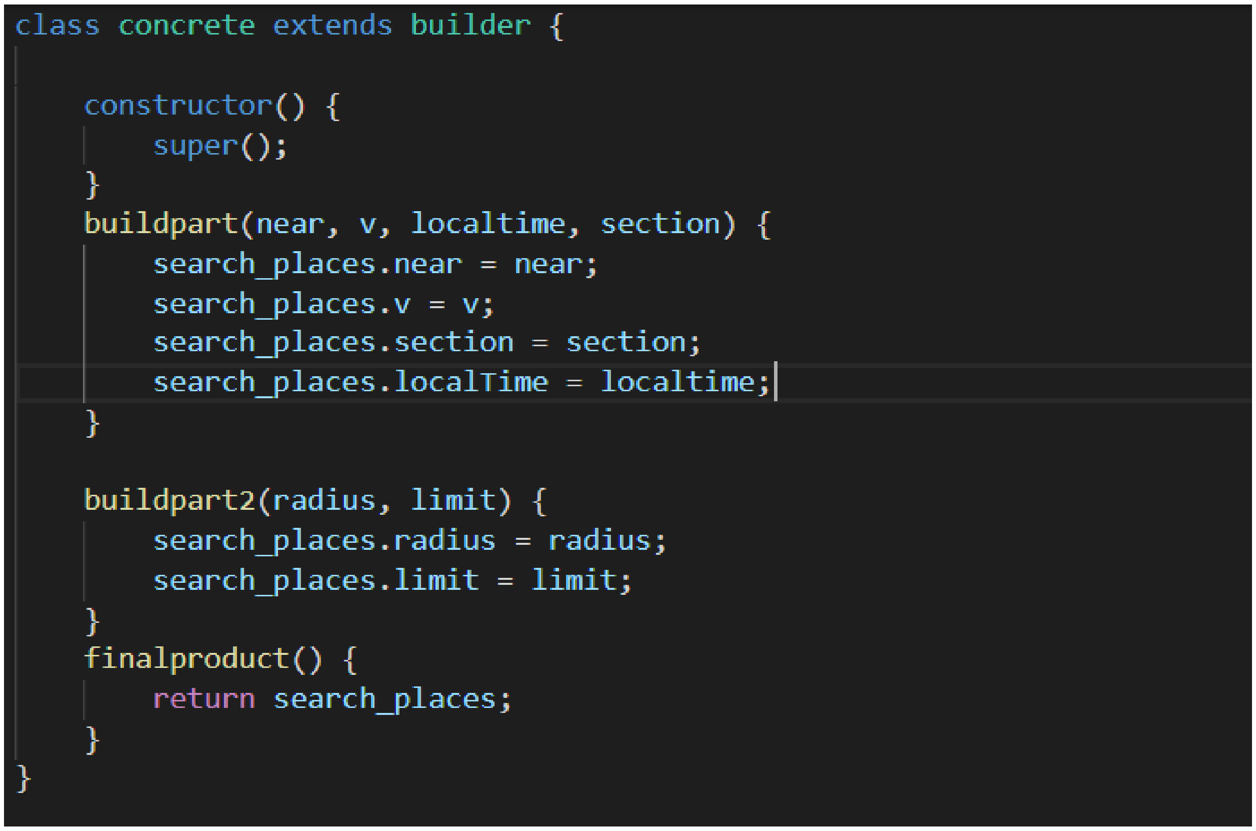
```
productformed()
{
    this.requestformed = search_places;
}
```



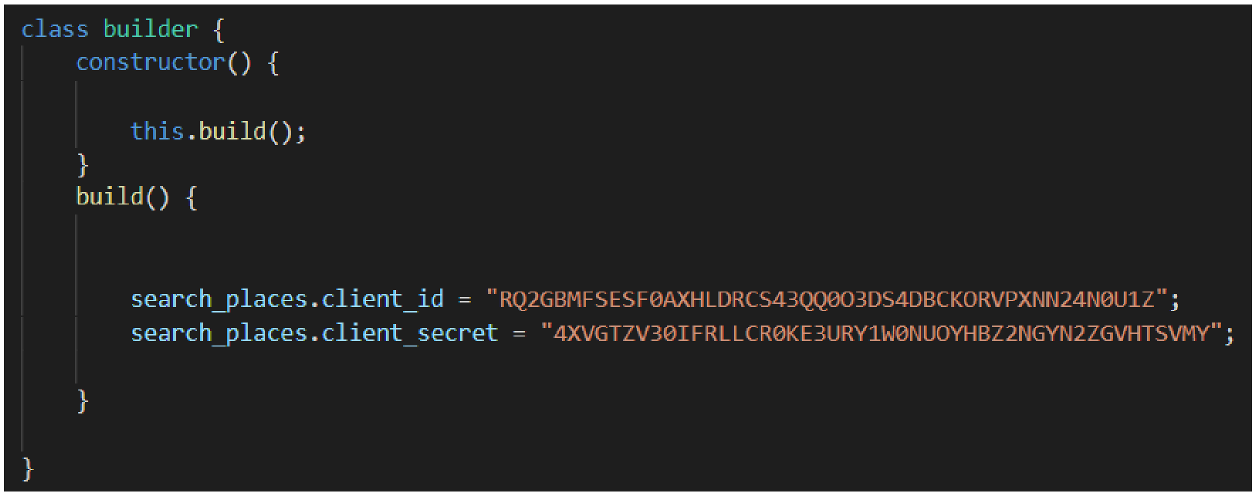
```
class product {
    constructor() {
        this.productformed();


    }
    productformed()
    {
        this.requestformed = search_places;
    }
}
```



```
//Builder Pattern Used for setting parameters for API request to four square API.


class Foursquare_Director {
    constructor() {
        this.construct();
    }

    construct() {
        this.client_id = undefined;
        this.client_secret = undefined;
        this.section = undefined;
        this.near = undefined;
        this.v = undefined;
        this.localTime = undefined;
        this.radius = undefined;
        this.limit = undefined;
    }
}
```



```
class concrete extends builder {

    constructor() {
        super();
    }
    buildpart(near, v, localtime, section) {
        search_places.near = near;
        search_places.v = v;
        search_places.section = section;
        search_places.localTime = localtime;
    }

    buildpart2(radius, limit) {
        search_places.radius = radius;
        search_places.limit = limit;
    }
    finalproduct() {
        return search_places;
    }
}
```



```
class builder {
    constructor() {

        this.build();
    }
    build() {


        search_places.client_id = "RQ2GBMFSESF0AXHLDRCS43QQ0O3DS4DBCKORVPXNN24N0U1Z";
        search_places.client_secret = "4XVGTZV30IFRLLCR0KE3URY1W0NUOYHBZ2NGYN2ZGVHTSVMY";

    }

}
```

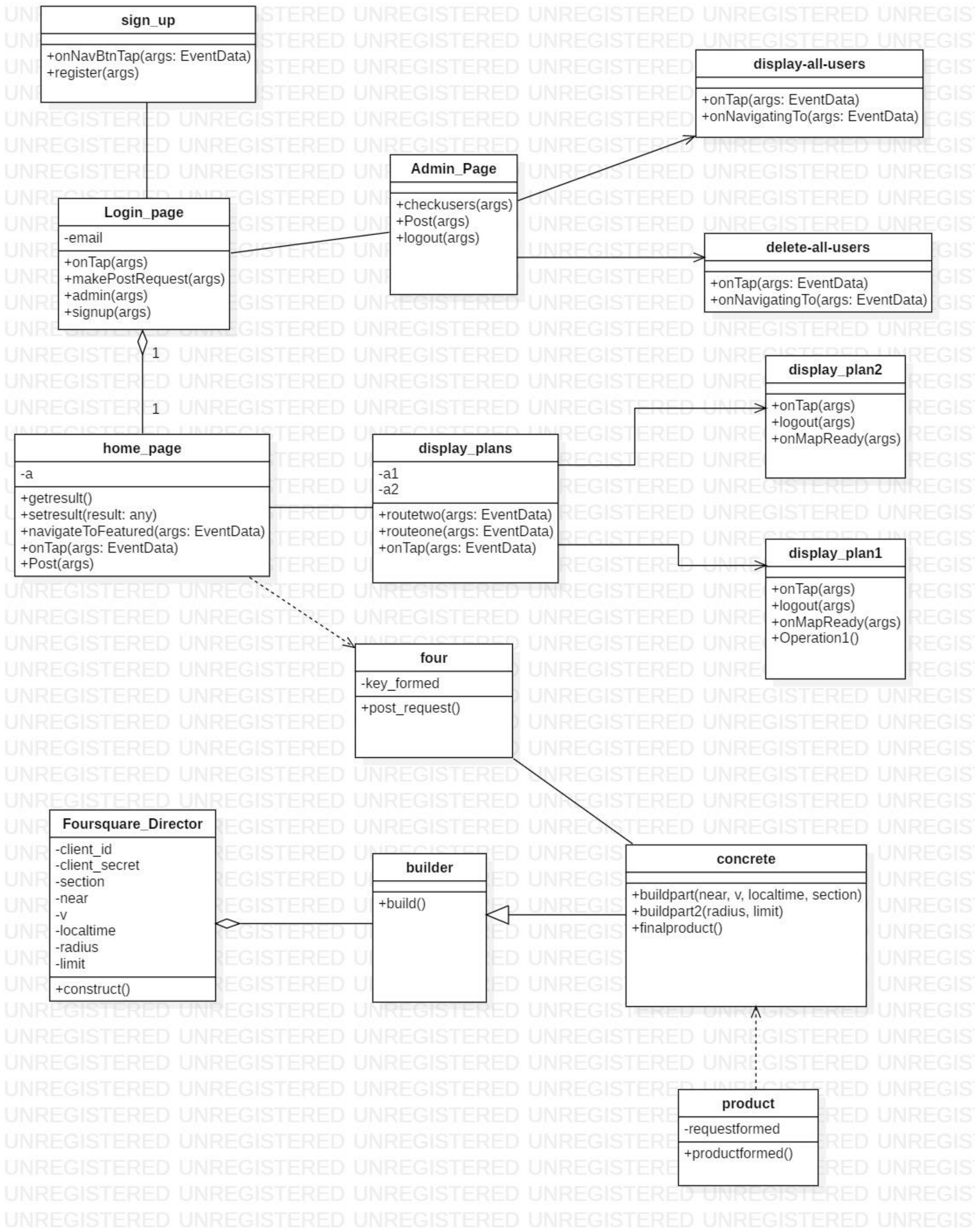
4. *Advantages*

- Better control over object construction.
- Object representation and construction are well separated.
- Improves readability.
- Object can be constructed following a series of steps.
- Easy to change internal object representations.

5. Disadvantages

- It will require a concrete builder for different products.
- Data members are not always to get initialized in some cases.

Class diagrams



Program:

List the classes and functions implemented:

- **Class**
  Login_page (in filename login-view-model.ts)
  Functions:
  onTap(args)
  makePostRequest(args)
  admin(args)
  signup(args)
  Binded with: login-page.xml

- **Class**
  display_plans
  Functions:
  onItemTap(args):void
  routetwo(args: EventData)
  routeone(args: EventData)
  onTap(args: EventData)

- **Class:**
  home_page
  Functions:
  setresult(result: any)
  getresult()
  navigateToFeatured(args: EventData)
  onTap(args: EventData)
  Post(args)

- **Class**
  routeone
  Function
  function routeone()

- **Class**
  routetwo
  Function
  function routetwo()

- **Class**
  display_plan1
  Functions
  onTap(args)
  logout(args)
  onMapReady(args)

- **Class**
  display_plan2
  Functions
  onTap(args)
  logout(args)
  onMapReady(args)

Implemented Functions and Classes (Node.js):

- **Class**
  Concrete
  Functions
  buildpart(near, v, localtime, section)
  buildpart2(radius, limit)
  finalproduct()

- **Class**
  Foursquare_Director
  Functions
  construct()

- **Class**
  Product
  Fuinction
  productformed()

- **Class**
  Builder
  Function
  build()

- **Class**
  four
  Function
  postrequest()

- **Class**
  Foursquare
  Function
  post_request()

- **Class**
  Users_data
  Functions
  getresult2()
  post_login()
  delete_user()
  getresult()

Modules:

- server.js
- app.js

Models:

user.js (Database schema)

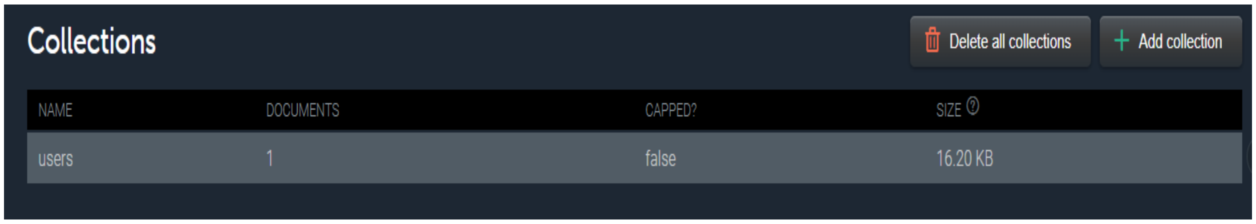
Screenshots of all the tables of the system data

I have used Mongo DB which is a NoSQL database for the storage of traveler (users). I used mLab which is a cloud database service for hosting my database.

As our project is not database oriented therefore it has only two collections i.e. admin and users
In our project we have two collections:

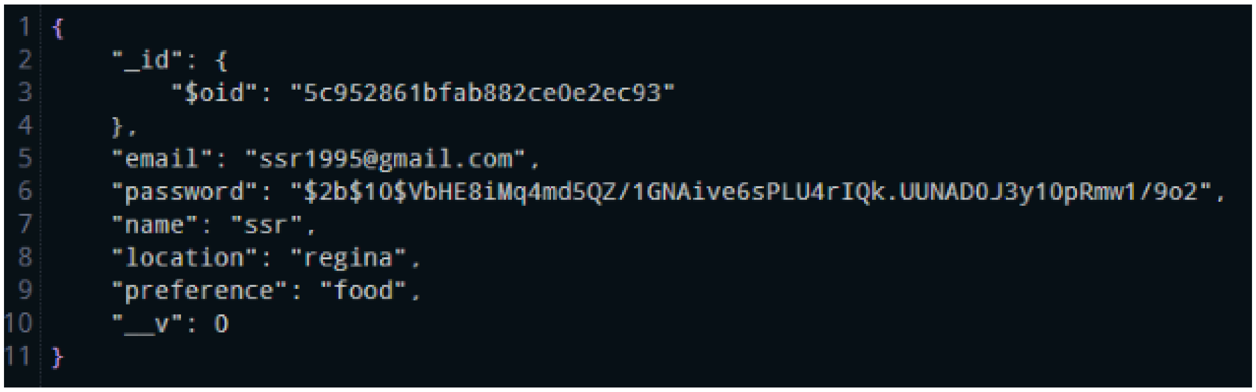
- admin
- users

Screenshot of collections



NAME	DOCUMENTS	CAPPED?	SIZE
users	1	false	16.20 KB

Document Structure of users (travelers):

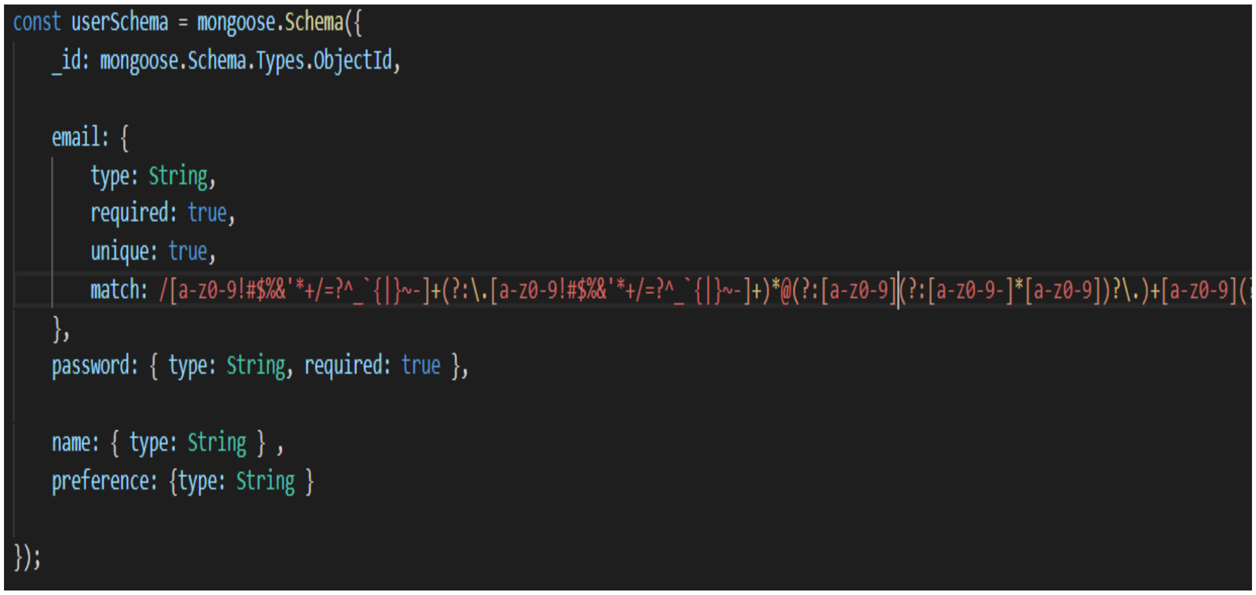


```
{
    "_id": {
        "$oid": "5c952861bfab882ce0e2ec93"
    },
    "email": "ssr1995@gmail.com",
    "password": "$2b$10$VbHE8iMq4md5QZ/1GNAive6sPLU4rIQk.UUNAD0J3y10pRmw1/9o2",
    "name": "ssr",
    "location": "regina",
    "preference": "food",
    "__v": 0
}
```

Default _id provided by MongoDB automatically acts as a primary key for all the users.

Schema:

This is the schema for the document.



```
const userSchema = mongoose.Schema({
    _id: mongoose.Schema.Types.ObjectId,

    email: {
        type: String,
        required: true,
        unique: true,
        match: /[a-z0-9!#$%&'*+/=?^_`{|}~-]+(?:\.[a-z0-9!#$%&'*+/=?^_`{|}~-]+)*@(?:[a-z0-9](?:[a-z0-9-]*[a-z0-9])?\.)+[a-z0-9](
    },
    password: { type: String, required: true },

    name: { type: String } ,
    preference: {type: String }

});
```

Technical Documentation:

Programming languages

- XML
- Typescript - https://www.typescriptlang.org/docs/home.html
- Node.js - https://nodejs.org/en/
- Plain JavaScript
- Nativescript
- CSS

Reused Algorithms and Programs:

- Express.js - https://expressjs.com/
- Fetch: https://docs.nativescript.org/ns-framework-modules/fetch
- Body-parser - www.npmjs.com/package/body-parser
- Observable: https://docs.nativescript.org/ns-framework-modules/observable
- Mongoose: https://www.npmjs.com/package/mongoose
- Axios: https://www.npmjs.com/package/axios
- Page: https://docs.nativescript.org/ui/ns-ui-widgets/page#page
- Button: https://docs.nativescript.org/angular/ui/ng-ui-widgets/button
- Bcrypt- https://auth0.com/blog/hashing-in-action-understanding-bcrypt/
- nativescript-mapbox: https://www.npmjs.com/package/nativescript-mapbox

Software Tools and Environments:

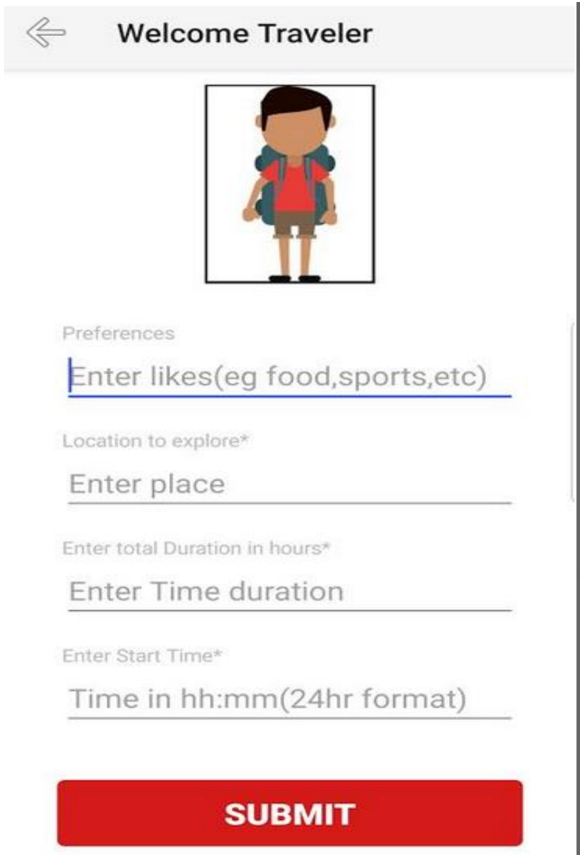
- Notepad++ - https://notepad-plus-plus.org/
- StarUML - http://staruml.io/
- Mongo DB - https://www.mongodb.com/
- Putty - https://www.chiark.greenend.org.uk/~sgtatham/putty/
- WinSCP - https://winscp.net/eng/download.php
- npm - https://www.npmjs.com/
- Amazon web service: https://aws.amazon.com/
- mlab : https://mlab.com/
- NativeScript- https://www.nativescript.org/

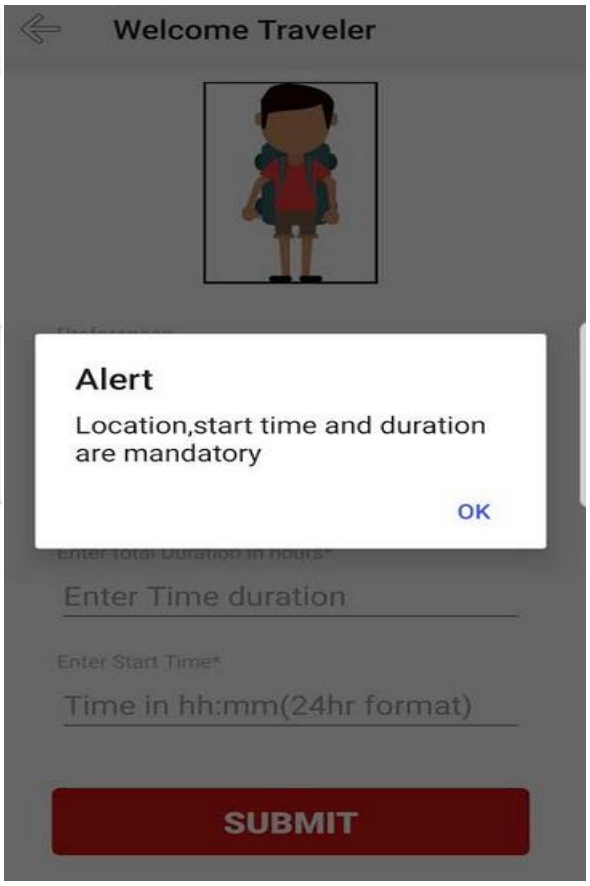
ROBUSTNESS TESTING

Test Case 1: Field mandatory validation

- User when submitting without entering anything (Location, start time and duration is mandatory to search the places). As preferences are not mandatory therefore an alert message displays Location, start time and duration mandatory. If preference is not entered then popular places randomly gets displayed.

Input **Output**





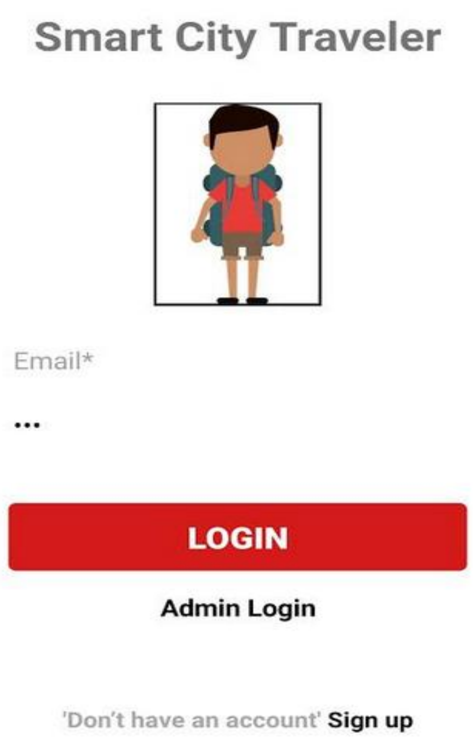
Alert

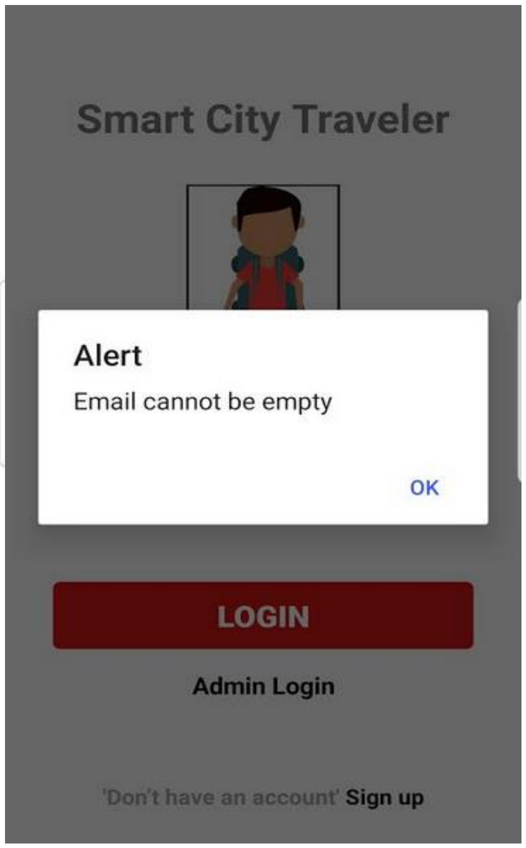
Location,start time and duration are mandatory

OK

- When user forgets to certain fields:

Input **Output**




Alert

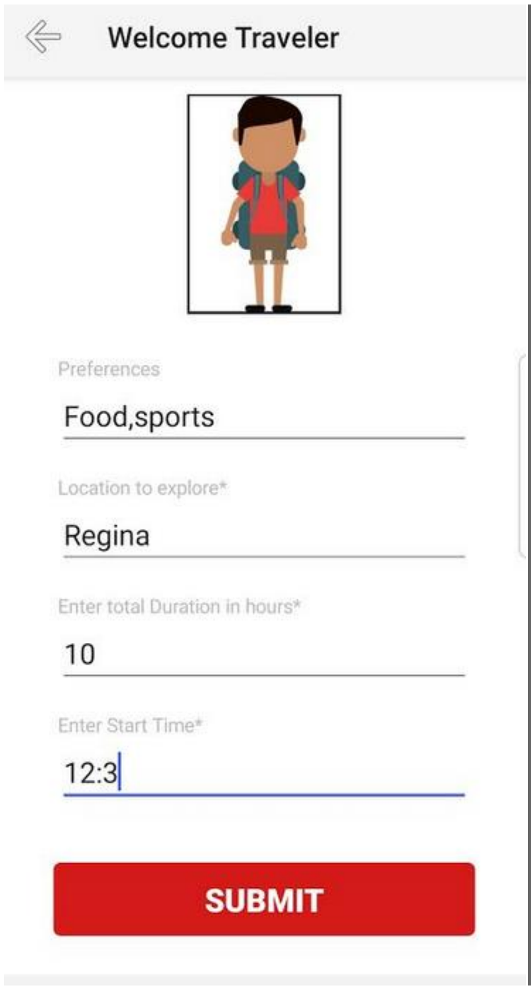
Email cannot be empty

`

Test Case 2: Start time validation (must be in hh:mm formats)

- If traveler user enters a wrong format start time:

INPUT **OUTPUT**



Welcome Traveler

Preferences

Food,sports

Location to explore*

Regina

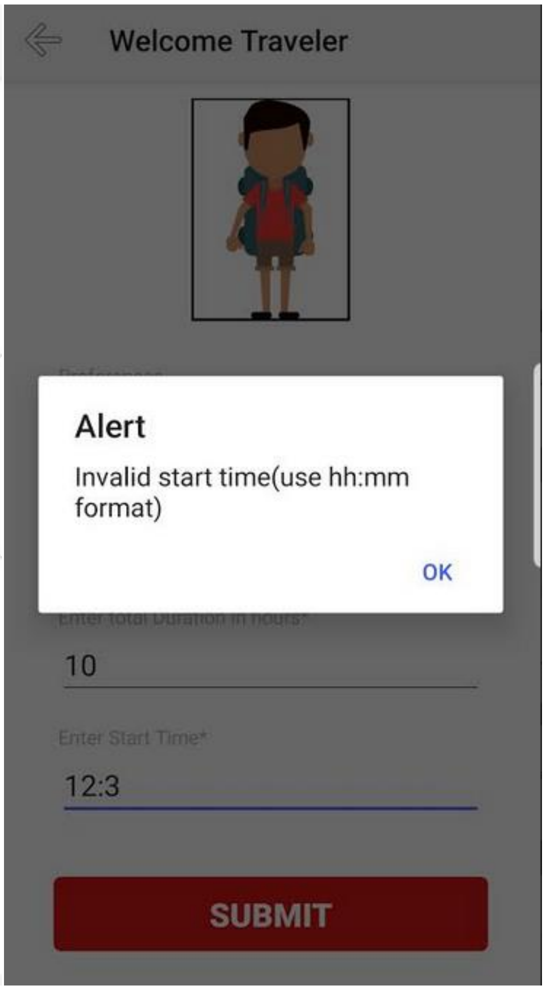
Enter total Duration in hours*

10

Enter Start Time*

12:3

SUBMIT



Welcome Traveler

Alert

Invalid start time(use hh:mm format)

OK

10

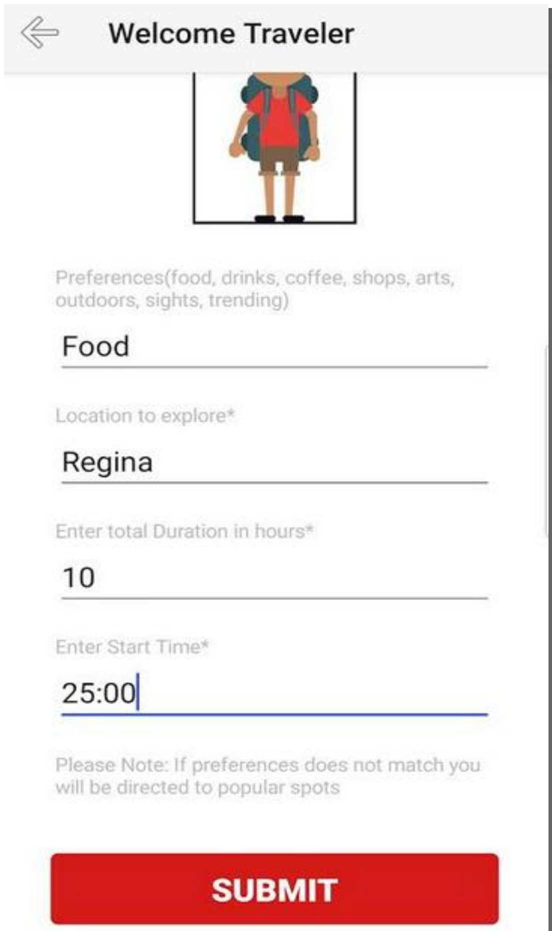
Enter Start Time*

12:3

SUBMIT

- If traveler enters a start time like 25:00 (hours can be only 24 in a day.).Hence this will again display an error message showing invalid start time.

INPUT **OUTPUT**



Welcome Traveler

Preferences(food, drinks, coffee, shops, arts, outdoors, sights, trending)

Food

Location to explore*

Regina

Enter total Duration in hours*

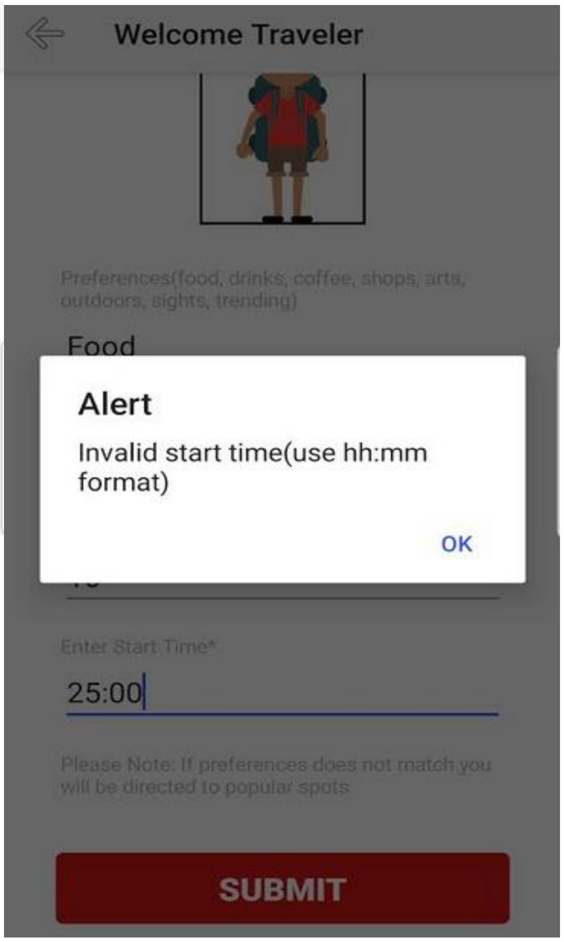
10

Enter Start Time*

25:00

Please Note: If preferences does not match you will be directed to popular spots

SUBMIT



Welcome Traveler

Alert

Invalid start time(use hh:mm format)

OK

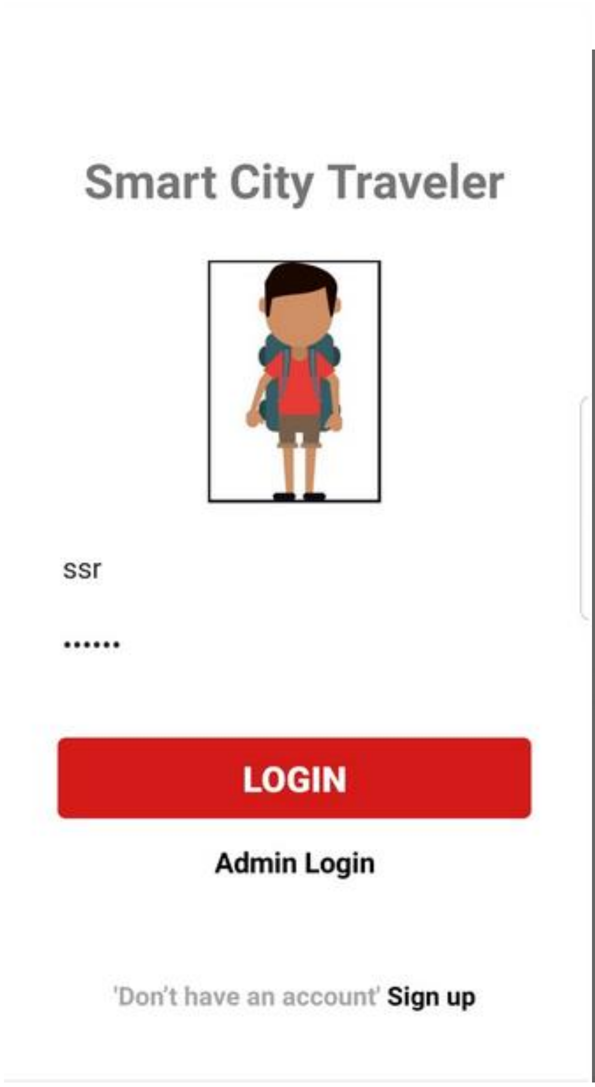
Enter Start Time*

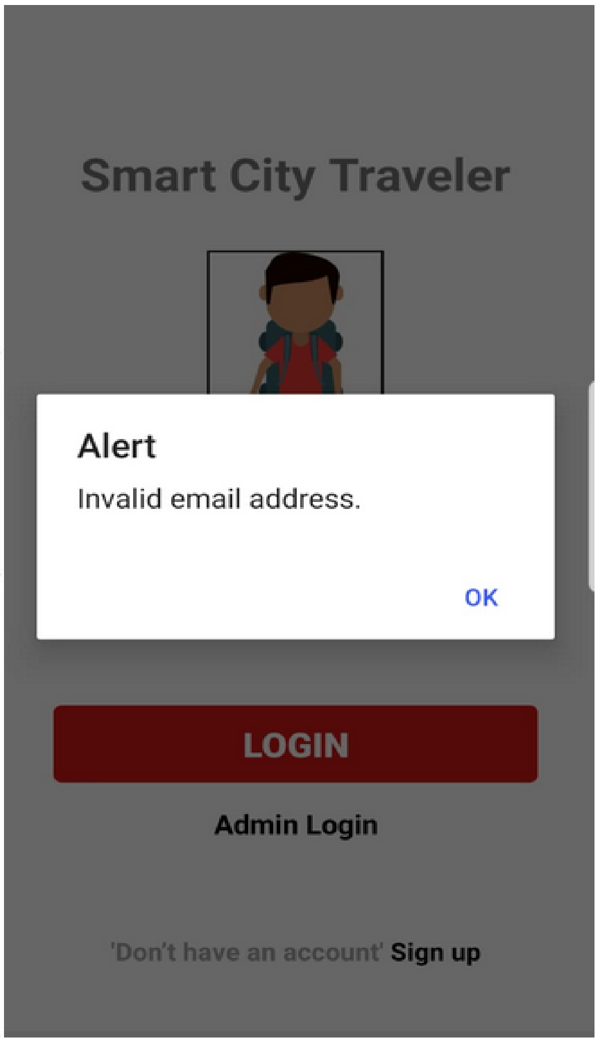
25:00

Test Case 3: Email Valid or not:

- As we know email should be in the form of something@something.com hence the input below displays an error message.

INPUT **OUTPUT**





Alert

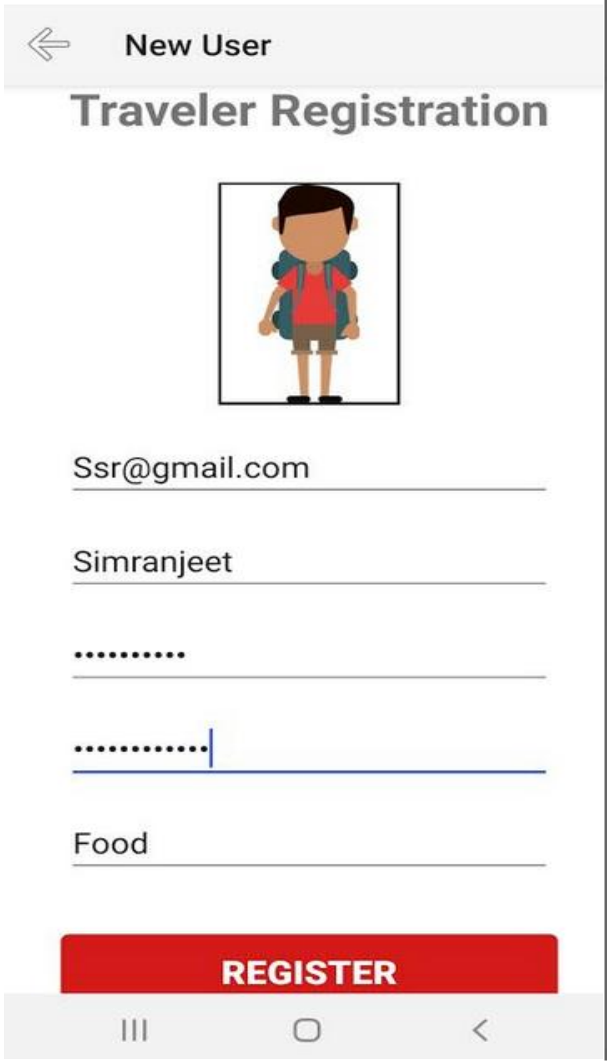
Invalid email address.

OK

Test Case 4: Traveler exists or not

- If email is already registered and user tries to register with the same email.

INPUT **OUTPUT**



New User

Traveler Registration

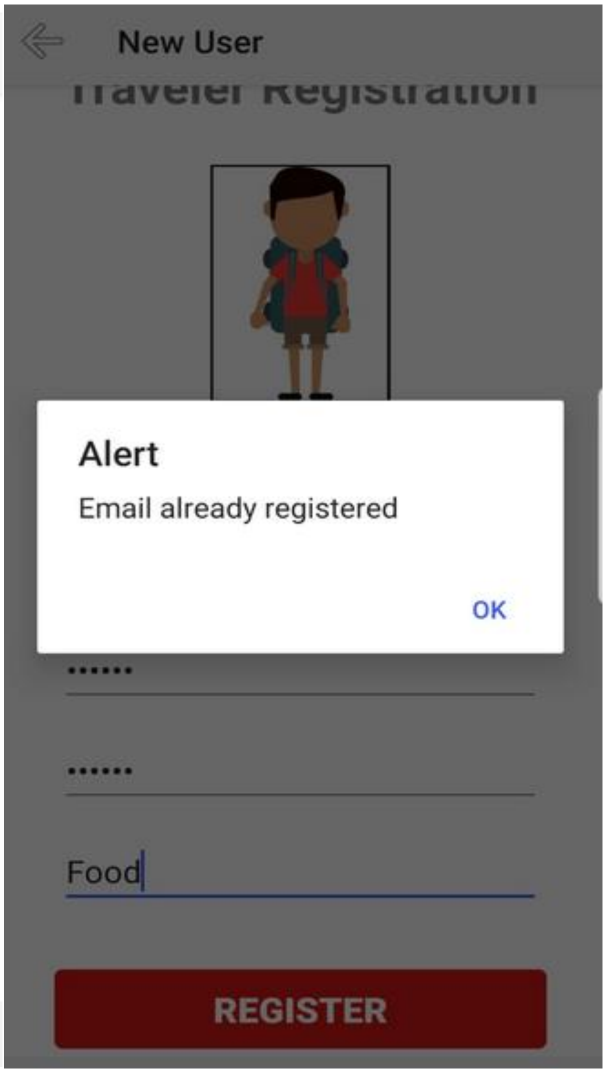
Ssr@gmail.com

Simranjeet

••••••••••

••••••••••••

Food



New User

Traveler Registration

Alert

Email already registered

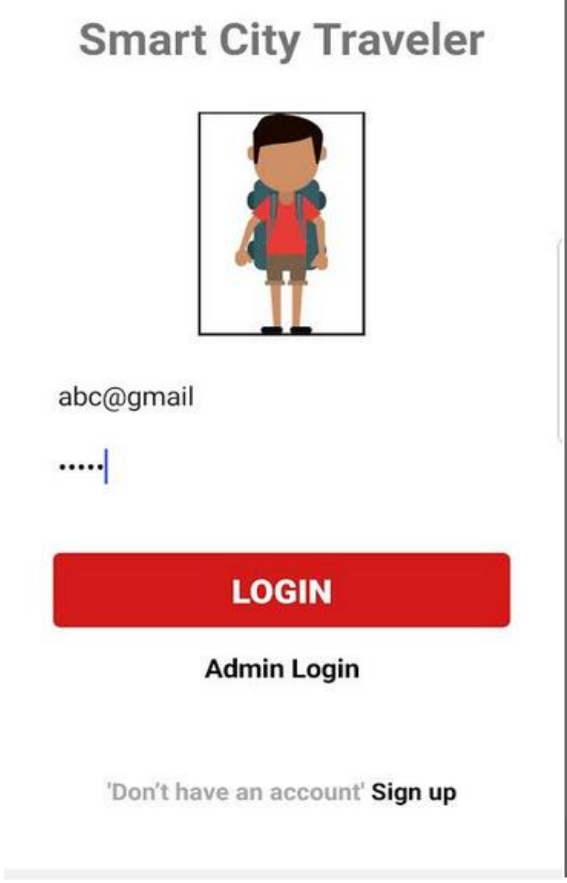
••••••

••••••

Food

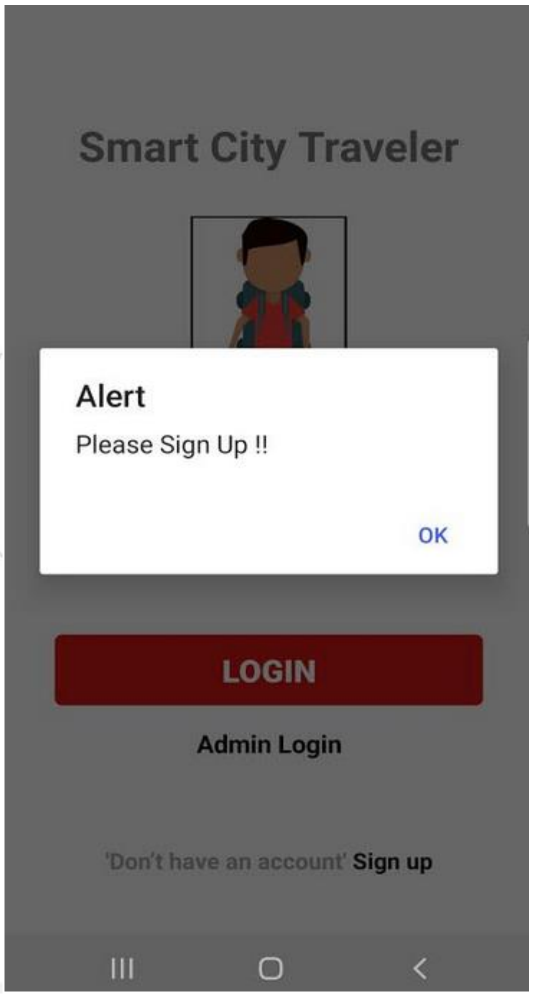
- If the traveler does not exists the message of Please Signup!! Will be displayed as shown:

INPUT **OUTPUT**


Smart City Traveler

abc@gmail

•••••


Smart City Traveler

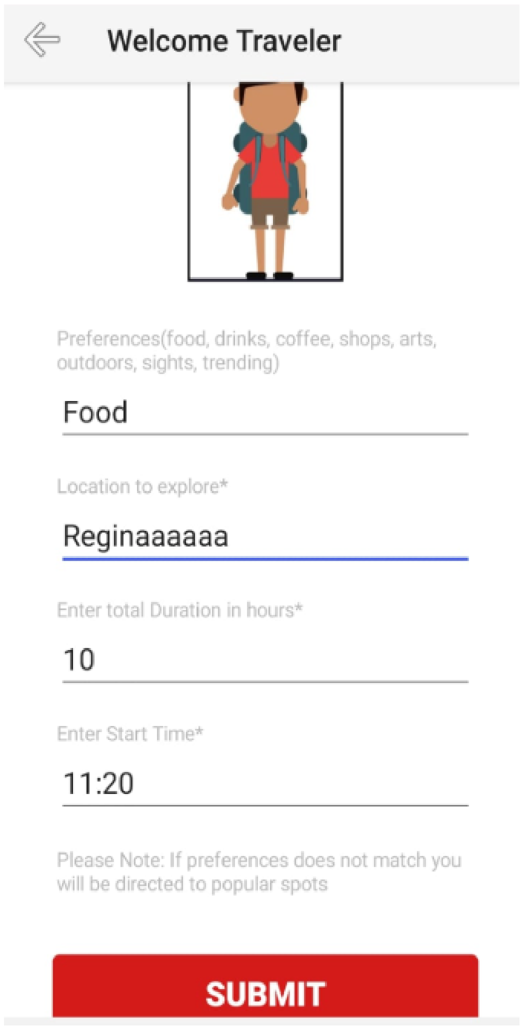
Alert

Please Sign Up !!

Test Case 5: Place name valid or not

If user enters a place name that does not exists.

Input **Output**



Welcome Traveler

Preferences(food, drinks, coffee, shops, arts, outdoors, sights, trending)

Food

Location to explore*

Reginaaaaaa

Enter total Duration in hours*

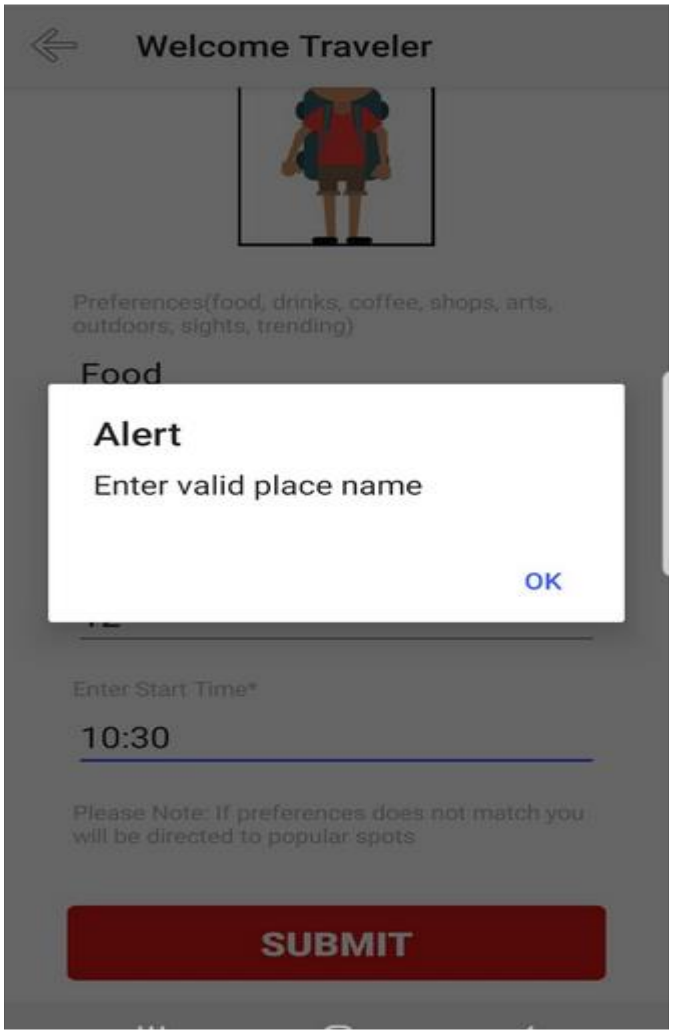
10

Enter Start Time*

11:20

Please Note: If preferences does not match you will be directed to popular spots

SUBMIT



Welcome Traveler

Preferences(food, drinks, coffee, shops, arts, outdoors, sights, trending)

Food

Alert

Enter valid place name

OK

Enter Start Time*

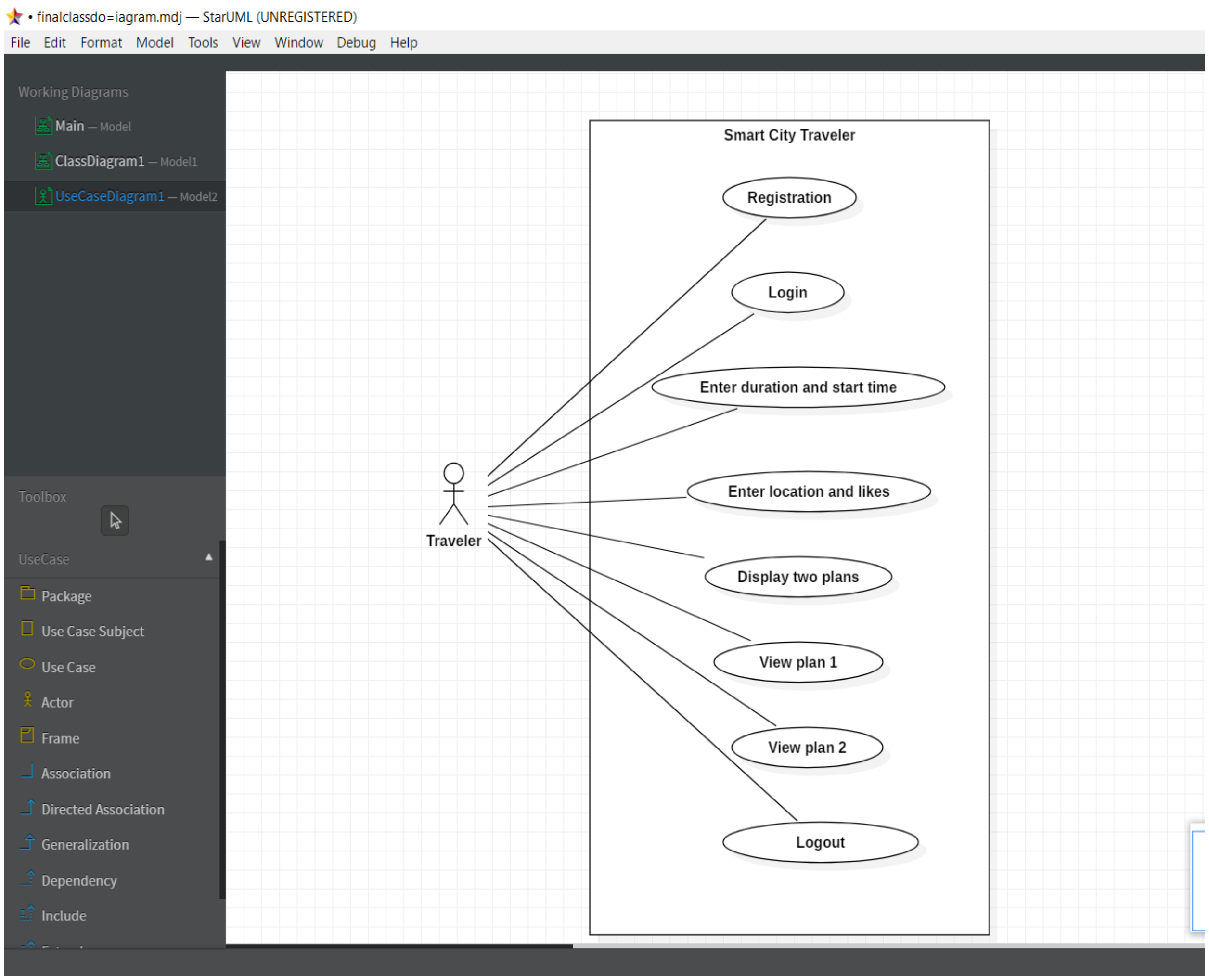
10:30

Please Note: If preferences does not match you will be directed to popular spots

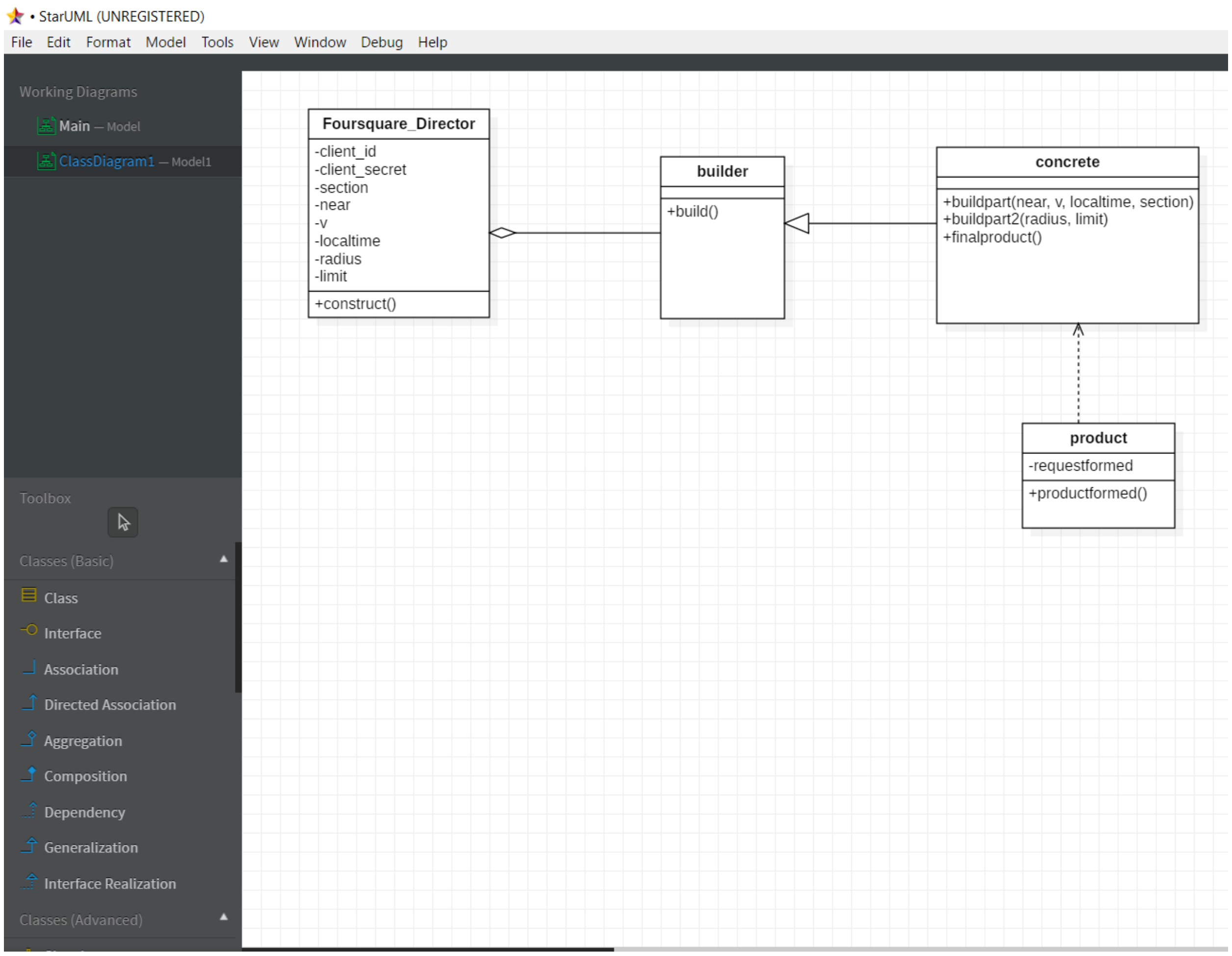
SUBMIT

UML SCREENSHOTS:

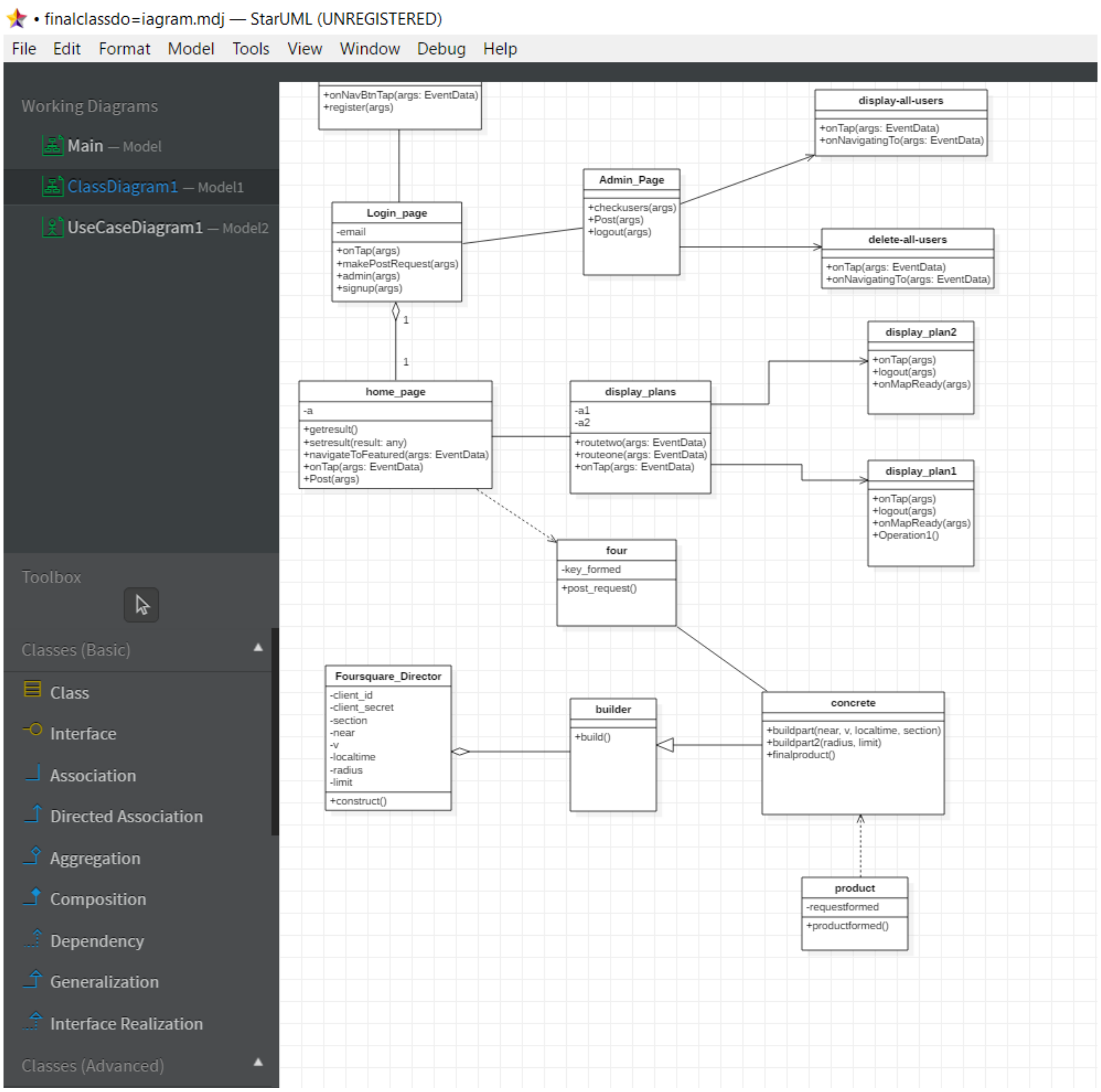
Use Case



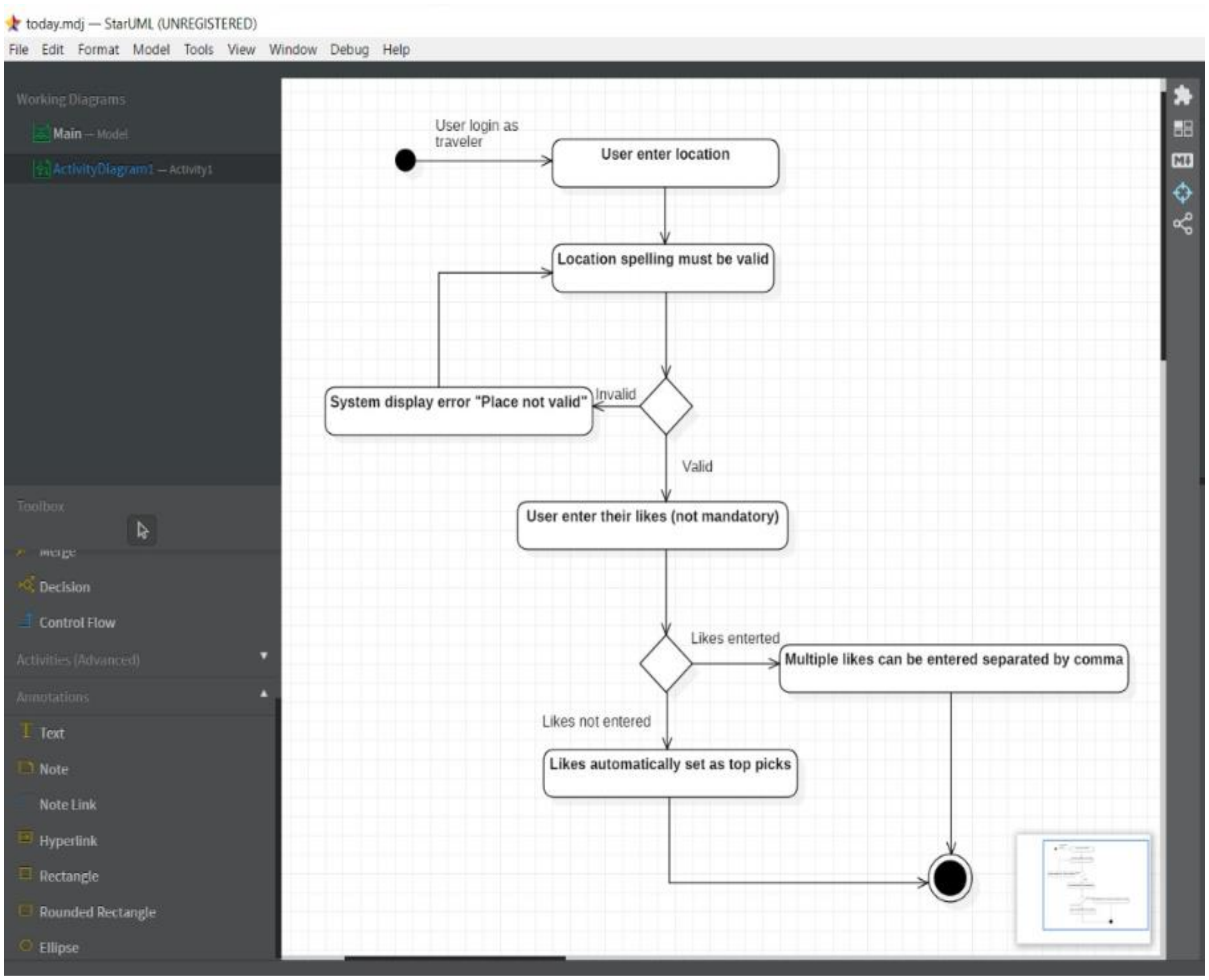
Builder Design Pattern(Class Diagram)



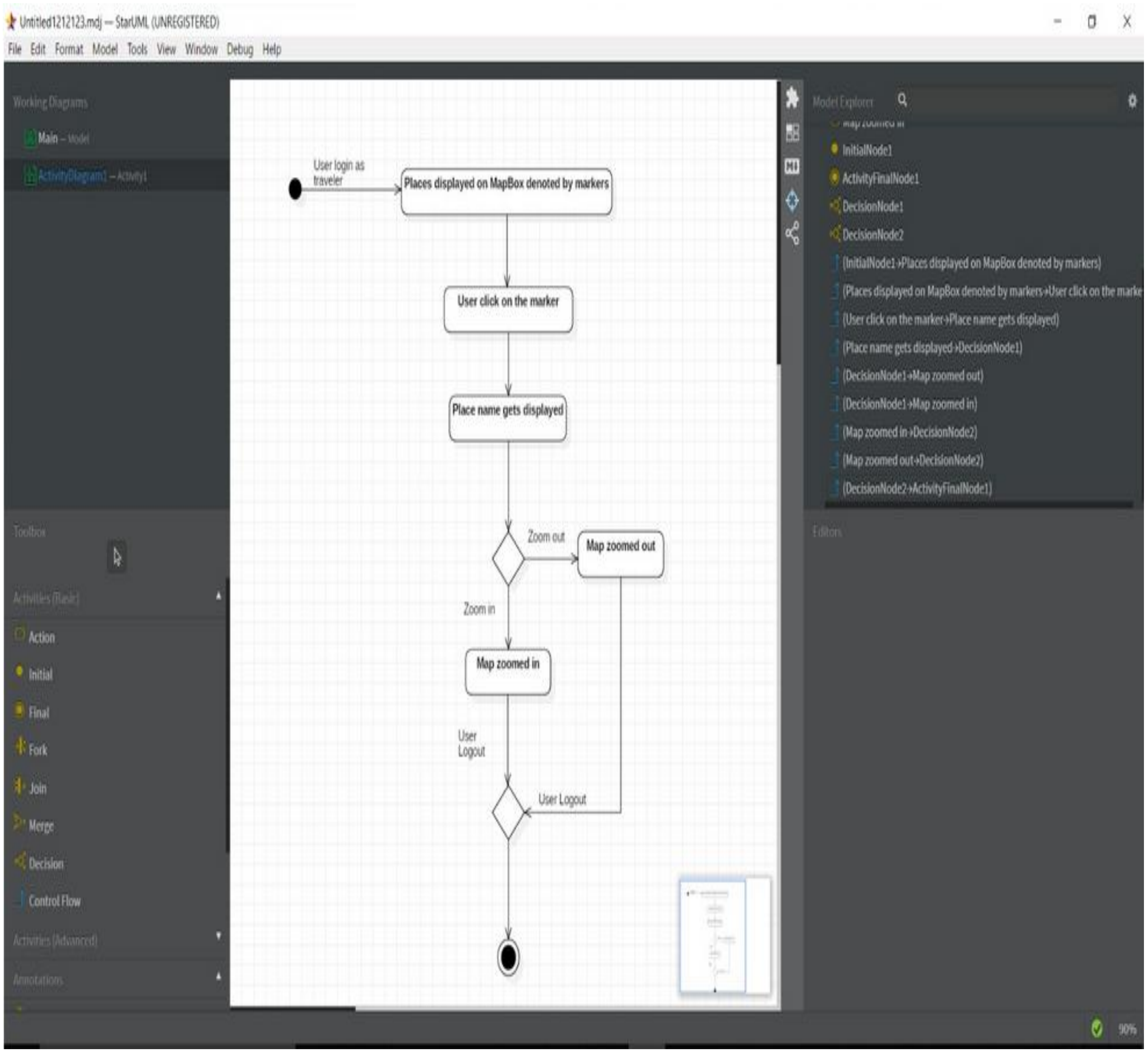
Class Diagram



Enter location and likes Activity Diagram

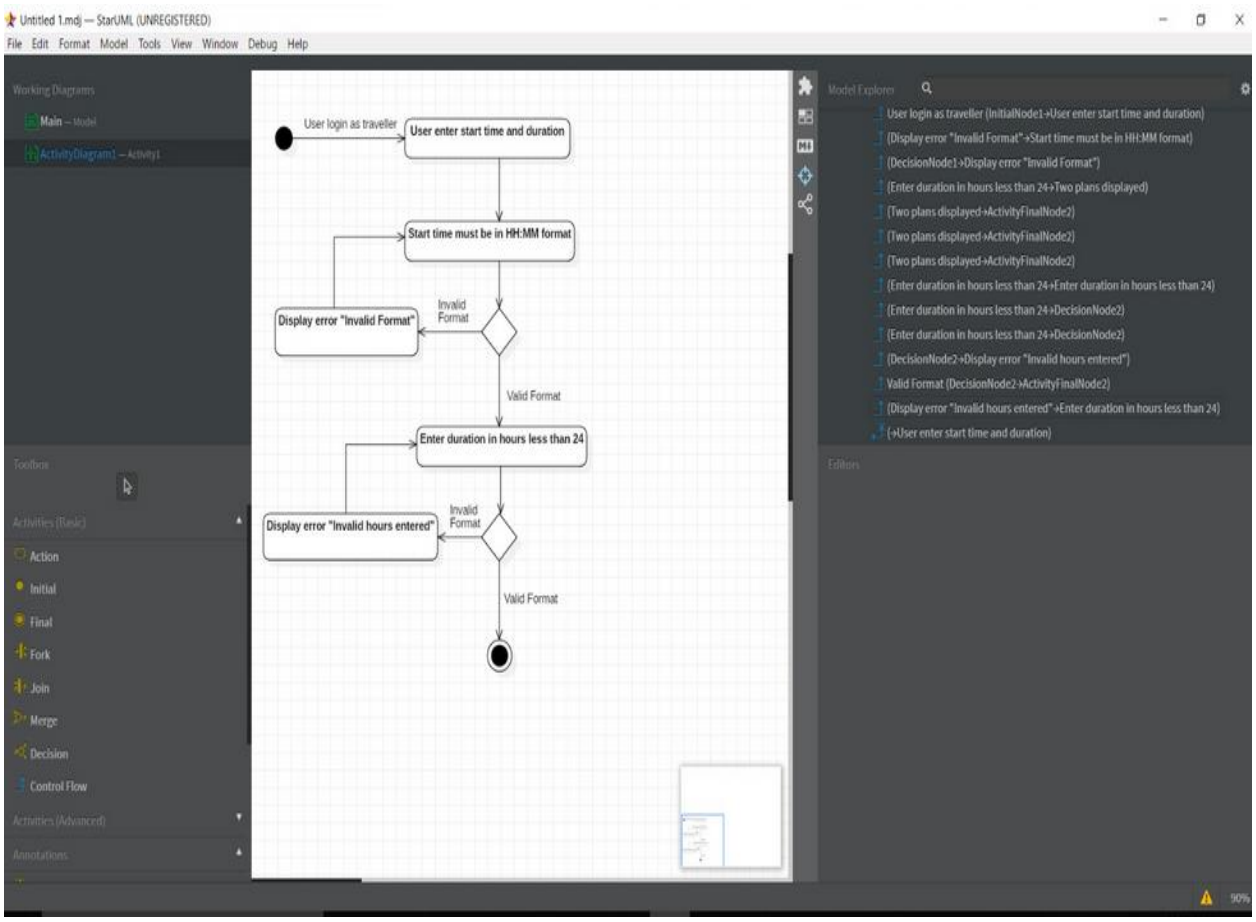

today.mdj — StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ActivityDiagram1 — Activity1
Toolbox
Merge
Decision
Control Flow
Activities (Advanced)
Annotations
Text
Note
Note Link
Hyperlink
Rectangle
Rounded Rectangle
Ellipse
User login as traveler
User enter location
Location spelling must be valid
System display error "Place not valid"
Invalid
Valid
User enter their likes (not mandatory)
Likes enterted
Multiple likes can be entered separated by comma
Likes not entered
Likes automatically set as top picks

View Plan 1 or 2 Activity Diagram



InitialNode1
ActivityFinalNode1
DecisionNode1
DecisionNode2
(InitialNode1→Places displayed on MapBox denoted by markers)
(Places displayed on MapBox denoted by markers→User click on the marke
(User click on the marker→Place name gets displayed)
(Place name gets displayed→DecisionNode1)
(DecisionNode1→Map zoomed out)
(DecisionNode1→Map zoomed in)
(Map zoomed in→DecisionNode2)
(Map zoomed out→DecisionNode2)
(DecisionNode2→ActivityFinalNode1)

Enter start time and duration activity diagram


Untitled 1.mdj — StarUML (UNREGISTERED)
File Edit Format Model Tools View Window Debug Help
Working Diagrams
Main — Model
ActivityDiagram1 — Activity1
Toolbox
Activities (Basic)
Action
Initial
Final
Fork
Join
Merge
Decision
Control Flow
Activities (Advanced)
Annotations
User login as traveller
User enter start time and duration
Start time must be in HH:MM format
Display error "Invalid Format"
Invalid Format
Valid Format
Enter duration in hours less than 24
Display error "Invalid hours entered"
Invalid Format
Valid Format
Model Explorer
Editors
90%

User login as traveller (InitialNode1→User enter start time and duration)
(Display error "Invalid Format"→Start time must be in HH:MM format)
(DecisionNode1→Display error "Invalid Format")
(Enter duration in hours less than 24→Two plans displayed)
(Two plans displayed→ActivityFinalNode2)
(Two plans displayed→ActivityFinalNode2)
(Two plans displayed→ActivityFinalNode2)
(Enter duration in hours less than 24→Enter duration in hours less than 24)
(Enter duration in hours less than 24→DecisionNode2)
(Enter duration in hours less than 24→DecisionNode2)
(DecisionNode2→Display error "Invalid hours entered")
Valid Format (DecisionNode2→ActivityFinalNode2)
(Display error "Invalid hours entered"→Enter duration in hours less than 24)
(→User enter start time and duration)