Department of Informatics, University of Leicester CO7501 Individual Project.

City Tour Planner App

Happiness Nwosu Chidiebere
hcn6@student.le.ac.uk
Student Number: 179049490

Dissertation

Project Supervisor: Dimitrova, Rayna (Dr.)
Second marker: Prof. Reiko Heckel
The date of submission: 13-02-2020

ACKNOWLEDGEMENTS

My sincere gratitude to my supervisors Dimitrova, Rayna (Dr.) and Prof. Reiko Heckel, who gave valuable feedback throughout the period of this thesis.I also am grateful to the university of Leicester for this opportunity to learn here and to work on this project. I appreciate my family and friends for their support in different ways. I am grateful for having this opportunity. It has been both a challenging and interesting journey for me and I appreciate every bit of it. Thank you.

ABSTRACT

Tourists face different challenges when exploring their destinations of choice. These challenges include planning their tour in order for them to get the best experiences in these destinations, this in turn poses another challenge as to how they can create a personalized tour for themselves that helps them explore their interests in different locations.

The internet hosts a rich and varied set of platforms, services and applications that promise to help tourists get the best out of their tours in desired destinations.

While these varied set of tools exist online, tourists still consult local and personal tour planners to help them create personalized tour plans, that fits their interests and help them best utilize their time by doing what they enjoy on their tour.

This thesis focuses on using open APIs available on the internet and user-expressed preferences to create personalized itinerary for their tourists.
The approach employed by this thesis is allowing users to express their interests using an android application, which then creates an editable tour which tourists can further select what they want and then get a final itenary.

This will eliminate the need for tourists to hire personal tour planners thereby eliminating the extra layer of human communication for tourists when creating their tours.

The results obtained from this thesis will enable users to create a personalized tour itenary by themselves while also eliminating the need for offline consultations or the hiring of a personal tour planner which will in turn reduce time spent planning tours themselves.

Keywords: Personalized tour itenary, open APIs, exploring destinations.

DECLARATION

All sentences or passages quoted in this report, or computer code of any form whatsoever used and/or submitted at any stages, which are taken from other people's work have been specifically acknowledged by clear citation of the source, specifying author, work, date and page(s).

Any part of my own written work, or software coding, which is substantially based upon other people's work, is duly accompanied by a clear citation of the source, specifying author, work, date and page(s). I understand that failure to do this amounts to plagiarism and will be considered grounds for failure in this module and the degree examination as a whole.

Name: Nwosu Happiness Chidiebere

Date: 13-02-2020

Table of Contents

1. Introduction

1.1 Background

The internet has revolutionized the way people experience different locations.
There are applications and platforms available that offer various solutions
in regards to activities, tours and attractions for different locations.

A lot of work has been done in the area of tourism and travelling to help people best utilize their time and have meaningful interesting experiences as they travel to different locations. Travellers are often presented with a curated list activities to perform, sights to see,
meals to try out, places to shop while on their tour.

Though a lot has been done in the area of tourism and the internet has rich resources and applications to help users make best use of their time during their explorations of different locations, many users still consult private tour planners to help create personal itinerary for them. One of the major reasons behind this is that, while there are many platforms, applications and resources online that present users with varied options on how to best utilize their time and enjoy their tours in different locations, itineraries and recommendations these online resources present are not personalized to the needs of the users.

To address this challenge of providing personalized itineraries to users, there are a number of online platforms and applications that connect travellers with specialist tour planners and natives to create itineraries with user expressed preferences so they can better enjoy tours with itinerary planned to meet their needs.

While this solves the problem of providing personalized itinerary to users, it poses another challenge of dependency on the human factor. Having a human tour planner can cause the following challenges:

- Time: Increase in time needed to plan a tour. Many of these platforms work by users expressing their preferences, the tour planner examining these and reverting, which usually takes a few days. Also, due to this turnaround time, a user cannot quickly and conveniently consult a tour planner on a day and expect

to receive feedback on that same day, this has to be planned and discussed in advance with the tour planner.
- Misunderstanding between the tour planner and the user, which might come up due to difference in language of communication, and also misuse of words.
- Increased cost. The rates charged by these tour planners differ between destinations, and that cost overhead can sometimes be the deciding factor as to if the user gets this customized plan or not.

This project will eliminate the need for human specialist tour planner.
The users will express their interest and preferences into the application, and with this information, the application will plan and output a custom 24 hour itinerary for the user.

The application will give users a tour itinerary that is based on their expressed preferences and interests as opposed to presenting users with an auto-generated tour plan based on metrics unknown to them

This system will take away the human layer that comes with personal tour planners and in doing so it:
- Reduce feedback time between the planner and the tour as the application will respond in an automated manner
- Reduce cost for the tourist as they dont pay to hire a personal planner
- Make changes to their itenary in a timely manner as the plans offered by the system are editable.

In all, it will save time and money for the tourist while offering them a personalized itinerary that meets their needs on their trip.

These preferences can also be re-used by the user for different locations or they can choose to create a new one for each tour.

1.2 Motivation

People go on tours to have experiences outside their normal day-to-day lives, meet new people, learn about different cultures and destinations, to do business amongst others.

Touring, though it has its benefits, often requires a lot of planning to ensure they get the best of it and enjoy their time. This plan entails tourists take into consideration how much time they have planned, their budget, their personal preferences and other details that make the tour enjoyable.

There are platforms and services available on the internet to automate this planning process for the tourist. These solutions range from booking flights, hotels, planning itenary to as far as suggesting locations and spots to tourists in their city or location of interest.

In the area of touring and exploring different locations, a lot has also been done as we have various applications and platforms that work to provide a curated list of activities to the user to explore and experience while on their tour.

While these solutions are helpful, they leave out a very important aspect of tourism, which is customization and user preference.
Showing most popular activities to users based on location alone, does not take into account the users preferences, their budget neither does it consider the time they have planned for their visit.
These platforms do not consider what is important to the user while showing the listings and activities in their location of interest.
This lack of personalized listings is the primary reason experts believe that almost half of all travel booksing still come through offline channels.

Using this available data on open APIs and getting more details about the tourist, City tour planner can create a personalized one-day tour for the tourists. This tour will meet the restrictions defined by user profile, start and end locations, time, their dietary preferences, showing them what they can eat where while on their tour.

The application built from this project will enable users to plan their tours and eliminate the need for offline consultations or the hiring of a personal tour planner or spend time spend time planning personalized tours themselves.

1.3 Aims and Objectives

The aim of this project is to develop an Android application that will enable a person to easily plan a one-day city tour, that allows her to best utilise her time, according to her interests.

For the application to achieve this, it should be able to do the following:

- Allow a user to sign up/login
- Allow a user to update their profile with personal details like gender, age etc
- Allow the user to create a new tour
    - The user needs to enter location to create a tour or allow the app to auto-detect location by enabling it in their system preferences (if it is not enabled already)
    - Ask the user specific questions to understand their preferences for this tour. These questions allows the user to express their preferences for the tour
    - If this is not the first tour of the user, the application should provide user with the option to use or update previous preferences for a past tour or decide to answer the questions for this specific tour

- Retrieve data from open APIs using the normalized user information
- Analyze this data from APIs using time and the proximity constraint to create an itinerary for the user tour
- Return this itenary to the user
    - Itenary should include start location and time
    - Distance between each venue in the itinerary
- Allow user to select from a set of options generated using their preferences, which will then be used to create the final itinerary
- Allow user to view this itenary on a map or a list
- Store the user and trip data to provide better experience and recommendations for future tours
- Store the user tour data to provide better experience for next tour using recommendations and learning from this tour
- Allow user share their tour itenary as a URL link
- The tours must be bound in time to 24 hours (A Day)
- Output the plan to the user

1.4 Scope

The scope of the work being carried by this project is limited to building an android application creates a 24-hour city tour plan for a user using their preferences, time.

This project does not cater for group tours, where more than one person is touring a location. It can be extended in the future for this behavior but that is not in the current scope.

This project employs open and free APIs for its data collection, and the language of content and communication on the app will be the English language.

While applications should be built for accessibility and wide user reach, the scope of this project does not allow for it to cater to people with certain disabilities such as blindness, as it does not have the resources needed for users with such conditions to enter or read information

1.5 Overview of Dissertation

A brief introduction to the content of other chapters of this dissertation is defined below:

Chapter Two talks about related work. It talks about existing applications and platforms that try to offer users varying solutions on how they can best utilize their time while touring different locations, it also talks about the challenges these solutions present and how this project will try to overcome such challenges.

Chapter Three talks about the design of the project. The architecture, models, views and controllers, database designs, REST API routes and external APIs used are discussed in this chapter.

Chapter Four discusses implementation. These implementation details discuss the backend API, the endpoints available on the server side, the Android application (client), the various screens on the application client and actions performed on those screens. It also provides some code snippets and instruction on how to run the application.

Chapter Five touches on testing the application. The server side unit tests and why they are important, the client side manual test cases are also discussed. This chapter ends with a summary, limitations and potential improvements for the application developed in this project.

2. Related Work

This chapter describes existing work in the area of tourism as it relates to
tourists and travellers exploring destinations and having interesting experiences
in different locations with the aid of technology, both offline and online.

RoadTrippers, a web and mobile application, is a road trip planning application that helps
users dscovers items such as accommodations, restaurants, attractions and culture, outdoor and recreation activities, entertainment and nightlife, shopping, sports and motoring
along the route of their road trip. It does this in the following steps:
- users select start and stop locations for their trip
- the application generates a map showing the users items such as , that is available on the route of their trip from start to stop locations
- users can select any of the items, adjust the budget setting (if it applies to that item category)
and add to their trip.
- users can then save trip by creating an account and it is the trip is available on their profile

RoadTrippers boasts database includes millions of the world’s most fascinating places, making planning the unexpected easier for its users.

RoadTrippers has its own set of places APIs, Boone (available for use at a price), that it uses to serve the application Backend. some of the endpoints are:
- Search: The Search endpoint powers discovery of the most interesting travel and tourism places.
It takes a coordinate center and a search radius, as well as a list of optional tags and returns a list of relevant places.
- Place Details: Return name, address, hours, photos, reviews, tags, and more about any place in the Boone database

With these endpoints built around the database, RoadTrippers returns items along a route for a start and stop location within a specified radius to its users.

Tripit, a web application, plans the itenary for the user with their booking details.
tells them where to be when, alerts them on next moves, shows nearby places etc.

Users can book their trip on TripIt supported booking sites and share the booking details with Tripit. Tripit then plans an itenary for them for when they arrive at the airport to their final destination.
Tripit also alerts users on time to their next action on the itenary and for discounts for nearby events or things to do.

Musement, an online platform offers to help travellers get the "best" from destinations by providing a great choice of
local tours and attractions bookable on multiple devices.
As seen on the website, they offer a range of services that includes temporary, exclusive, hidden-gem and even free activities for multiple locations. Some of its service offerings include:
- book guided tours online
- buy entrance tickets
- register to join bus tours
- day trips and excursions etc.

Blink Travel, also an online platform for tourists, offers the users curated cards of "best" attractions and experiences.
The user can schedule things to see or do by viewing the card and making their schedules. Its card categories include: top attractions, what to eat, the best things to see and the best things to do amongst other things

While there are multiple platforms on the internet with automated solutions to help users get the best experience
on their trips, travels and tours, there are also applications that offer physical consultation and experts online
to guide the tourists and travellers in making the best from their trips as well.

The **Personal-Trip-Planning-Service** on **Adventurelink** is another service where tours are planned personally for the tourist.
Tourists answer questions that help the platform create a personal itenary for them to use on their trip.
The questions asked include:
What's the purpose of your trip?
What's your preferred trip?
What would you like to do?
And the aim for these questions is to get more details about the tourists preferences and needs for their tour.

Journy, is an online application that helps travellers plan their trips by connecting them with
a personal trip designer.
It works in the following steps:
- Users share their travel preferences by filling out a questionnaire about how they like to travel and
things they would like to do on their trip.
these includes questions about travellers budget, dietary restrictions, interests etc
- Users get paired with a personal trip designer that works with them to review and plan their trip via email and chat on the application.
- users receive a customized itinerary with activity bookings and restaurant reservations, which they can access by the application
or by web browser.

With platforms like Musement, TripIt etc, it is a wonder why the likes of GoJourny still have millions of user base.

As a user on GoJourny put it:
“I found Journy after already getting an itinerary from a typical travel agency. Simply put, Journy’s recommendations blew theirs out of the water. Every concierge we worked with really listened to what we wanted as travelers and led us to the off-the-beaten-path sights we were looking for. Plus, the process is super smooth and easy!”

The key words to note in the user review are: "worked with them" and "listened to what they wanted ". This is one of the key selling features of the Journy platform.

While there are multiple platforms on the internet with solutions around how a tourist can enjoy their time in a location or have a planned itinerary, the recommendations from these platforms are not based on personal preference of the tourists.

People want to be listened to and want their input to be taken into consideration when a tour is being planned, but the existing platforms do not offer a solution for this challenge.

This creates the need to have an online platform application that works with tourists/users to get the profile, know their needs and discuss their preferences for their tours.

City Tour Planner gives users an itinerary and tour plan that is based on what they like to do and how they enjoy spending their time.
It does not offer an auto generated itenary to the users based on metrics unknown to them.

This system will take away the human layer that comes with personal tour planners and in doing so it:
Reduce feedback time between the planner and the tour as the application will respond in an automated manner
Reduce cost for the tourist as they dont pay to hire a personal planner
Make changes to their itenary as the plans offered by the system are editable.

In all, it will save time and money for the tourist while offering them itinerary and plans that are customized to their needs on their trip.

The need for a personal tour plan is one of the major reasons why offline travel agency and tour planners still have thousands of users.

This project will also develop to learn more about the user, based on their interaction with the initial provided tours, thereby using their data as they go on multiple tours to plan tours for them in future.

3. Design

To achieve its objectives, the system will be implemented using a set of technologies. This section discusses how this project will be developed and the various components built to achieve the set project objectives.

3.1 System Functionality specification

Approach

From the project objectives, the application will generate a tour itinerary for the user to help them utilize their time in a city doing what they enjoy. Thus, the application needs to enable the user to express these preferences.
The application follows these steps for a user to generate an itinerary:

- User signs up or logs into account if already registered
- User selects a city: The following cities are presented to the user as options:
    - Munich, Germany
    - Hamburg, Germany
    - Frankfurt, Germany
    - Leipzig, Germany
    - London, UK

  These city options are fixed choices hardcoded in the application code.
- Selects the meals they will like to have on the tour. Options include: breakfast, lunch and dinner.
- Users select categories of interest as preferences of what they will like to do on their tour. Categories and sub-categories presented to the user in this project include:
    - Museum
        - Art Museum
        - History Museum
        - Science Museum
    - Entertainment
        - Art Gallery
        - Casino
        - Movie theatre
        - Performing arts

    - Food
        - American restaurant
        - Asian restaurant
        - Indonesian restaurant
        - Japanese restaurant
        - Korean restaurant
    - Outdoor and recreation
        - Beach
        - Hill
        - Lake
        - Mountain
        - Parks and gardens
    - Shopping
        - Fabric shop
        - Cosmetic shop
        - Comic shop
        - Clothing store
        - Arts and Craft store
        - Automotive shop
    - Performing arts
        - Dance studio
        - Indie theatre
        - Opera house
    - Buildings
        - College art building
        - City hall
        - Historic site
        - Monument/Landmark

- A tour draft is generated for the user using the city, meal options and the categories selected. This tour draft contains five options for each the meals and sub-categories selected by the user. This is to show the user multiple options for their desired categories.
- Time selection for tour: At this step, the user selects how long they will like to spend at each location of the tour. The options here are 1, 1hour30 and 2 hours respectively. The app has different configuration these time options.
  Because this project aims to allow users tour the city for a day, the time options for these app is as follows:
  Total tour time: 6 AM to 12 AM (18 hours)

- **1 hour:** Here the user will select 18 spots from the tour draft generated. This includes the meals. So if the user selects this option and also wants to have breakfast, lunch and dinner on the tour, they will end up selecting 15 spots and 3 spots for the meals. So also they will select 17 spots if they only want to have one meal on the tour.
  The formula for this can be said to be:
  total spots = spots for time selected - meal options selected.

- **1:30 hours:** here, the user can select 12 spots in total, which also includes their meal of choice as explained above
- **2 hours:** Here the user can select upto 10 spots.
  These time selections also have index positions for where the meals will be placed in the tour itinerary. The aim of this configuration is to try to enable the user to have meals on time while on their tour. The assumption is that breakfast is taken between 9 and 10 AM, lunch between 1 and 2 PM and dinner between 6 and 7 PM, thus depending on the time option they pick, the application tries to add these meals to index positions to match these times as much as possible.

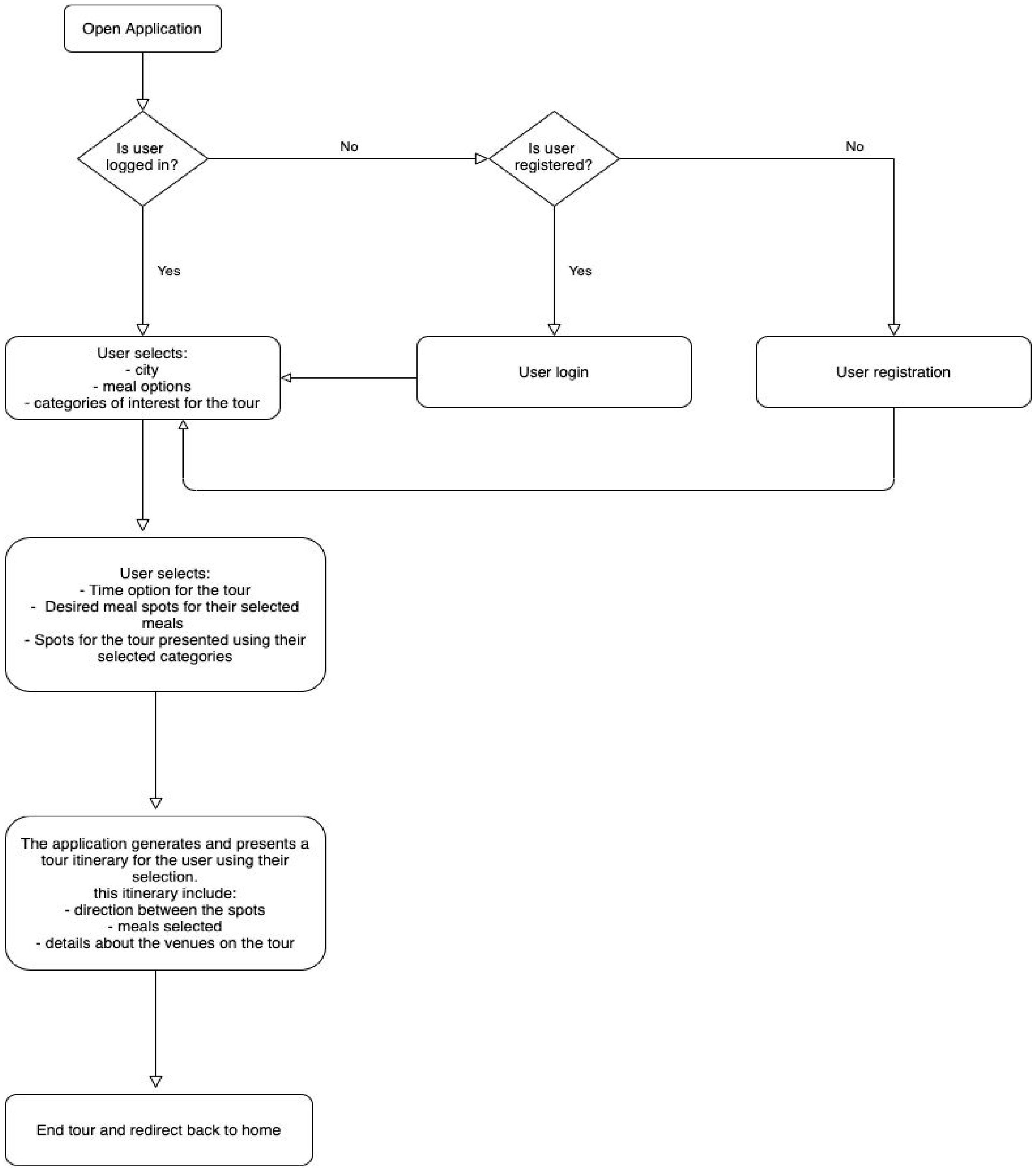
  This is a JSON representation of the configuration for time selection for the application:

```
[ { name: '1 hour', value: '1', spotsCount: 18, mealIndexes: {
breakfast: 3, lunch: 7, dinner: 12 } }, { name: '1 hour 30', value:
'1:30', spotsCount: 12, mealIndexes: { breakfast: 3, lunch: 6,
dinner: 10 } }, { name: '2 hours', value: '2', spotsCount: 8,
mealIndexes: { breakfast: 0, lunch: 4, dinner: 7 } }, ];
```

- After the user selects the meals and spots from the draft generated, the applications gives the user an itinerary for their tour, directions between the spots and option for the user to view detail for each spot on the tour.

Flow Chart

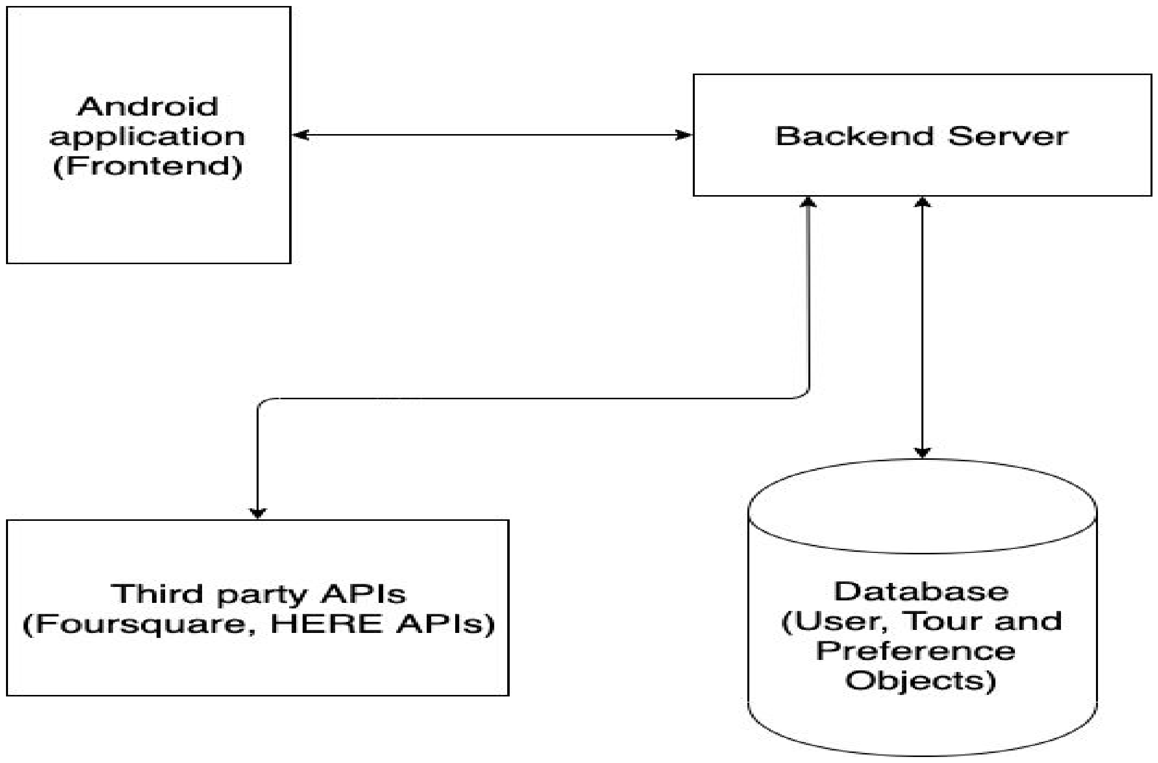
The flowchart to illustrate the approach above can be seen as:


Open Application
Is user logged in?
No
Is user registered?
No
Yes
Yes
User selects:
- city
- meal options
- categories of interest for the tour
User login
User registration
User selects:
- Time option for the tour
- Desired meal spots for their selected meals
- Spots for the tour presented using their selected categories
The application generates and presents a tour itinerary for the user using their selection.
this itinerary include:
- direction between the spots
- meals selected
- details about the venues on the tour
End tour and redirect back to home

3.2 Architecture

This project is built with the client-server architecture. The android application is the client and the server consists of a NodeJS application built to expose REST APIs for client-server communication over http. User data is stored in a PostgresQL database on the server side and Sequelize Javascript framework is used for Object relational mapping of the database objects.

Architecture diagram


Android application (Frontend)
Backend Server
Third party APIs (Foursquare, HERE APIs)
Database (User, Tour and Preference Objects)

From the architecture diagram above it can be seen that the client side (Android application) is communicating with the backend server APIs via http protocols.
The backend server inturn queries the database for data regarding users and tours. The backend server also communicates with third-party APIs to fetch data for user itinerary.

The communication between the application components are asynchronous, a response is awaited for each request before execution can continue. The server side uses the Nodejs “promisify” utility to make the request module asynchronous.

The application architecture has the following components:

- Backend
- Frontend
- External APIs

The Backend

The server side of the project is built following the Model-View-Controller software design pattern also known as MVC. The routes communicate with database objects using the Models and the Sequelize ORM for data operations and the results from these operations as presented to the user as JSON objects through REST API endpoints

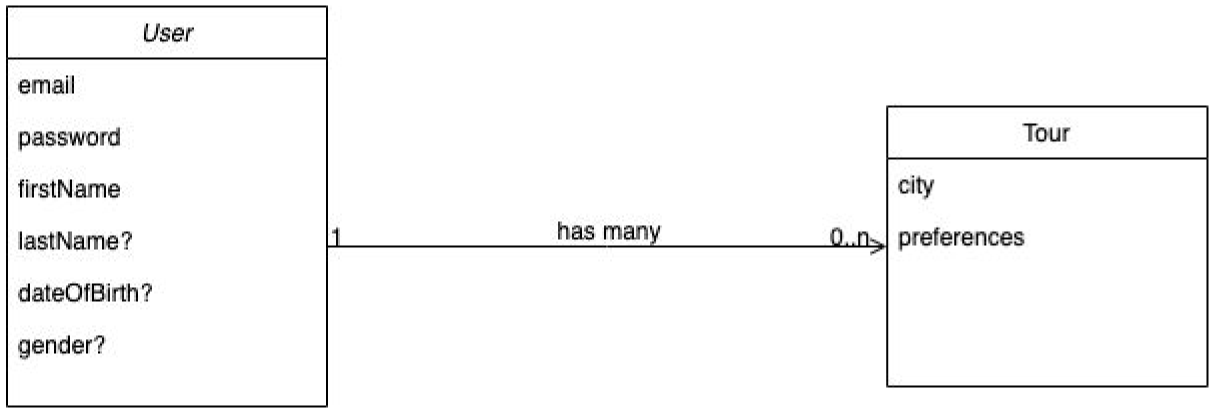
The Models

The models are defined as classes that map to objects using the sequelize ORM to create, update, get or delete database objects.
The application has the following models on the server:

- User: This model as defined in https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/models/user.js creates a "users" table that will be used to store User data objects. Data creation, retrieval, update and deletion for a user will happen with this table.
- Tour: This model as defined in https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/models/tour.js creates a "tours" table that stores user related tour data objects.

  The User model has a one-to-many relationship with the Tour model, this allows a user to create multiple tours using the same account. The tour preferences are also stored on this table as JSON string.
  The preference is saved as a JSON string to avoid over-complication of the database structure as the preference from the user can be dynamic, and ach preference is mapped to a tour, thereby eliminating the need to have a separate table for preferences.The JSON string can be parsed into a key value and reused by user to create future tours

Entity relationship diagram of the models:



The User model has a one-to-many relationship with the Tour model and while the fields in the Tour model are all required, the lastName, dateOfBirth and gender fields in the User model are not required to create a new User record in the database.

With the sequelize ORM, the “hasMany” relationship is expressed on the user model as seen here:
https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/models/user.js
When the migrations for these models are run, the “tours” table will have a “userId” attribute that maps a foreign key on the “users” table for that specific user that created the tour.
The migration files are also generated with the sequelize ORM and can be found here:
https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/migrations/

With Node Package Manager(npm) scripts are created that can be executed on the server to create and drop database, run and undo migrations using the database configurations specified in
https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/config/config.js for different development environments.
For this project backend server, the following scripts exist in the package.json file:

```
"createDb": "node_modules/.bin/sequelize db:create", "dropDb":
"node_modules/.bin/sequelize db:drop", "migrate":
"node_modules/.bin/sequelize db:migrate", "migrate:undo":
"node_modules/.bin/sequelize db:migrate:undo"
```

The Controllers:
The server is built to expose REST APIs for client-server communication and data exchanges hence the server side logic is expressed in routes.

The server has these routes exposed:
User routes:
This is defined here
https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/routes/user.js and defines these endpoints

- `/users/`: A POST endpoint used to create users. It expects the following payload structure to create the user successfully:
  ```
  {
  	"firstName": "John5",
  	"email": "john5@demo.com",
  	"password": "john5@pass"
  }
  ```
  On successfully user creation it returns the user object that has attributes defined in the User model. The model executes a “beforeCreate" hook, this is an action to be performed on a new data record before it is inserted into the database, to hash the user password. This is a security measure as it is against development standard to store passwords as plain text in the database.

  This is what a hashed password will look like before it is persisted in the database:
  Before: `"john5@pass"`
  After: `$2a$10$cy2pVuF.GN2cLCaMN4Qpd.FzKAHli6SfytkHh9kyBaq258EL9Vh4W`

- `/users/login`: A POST endpoint used to log the user in. It expects the following payload structure to create the user successfully:
  ```
  {
  	"email": "john5@demo.com",
  	"password": "john5@pass"
  }
  ```

  It logs in the user using the “bcrypt-nodejs” javascript package to validate if the user encrypted password matches what is sent in the request payload. When there is a match, an authentication token is generated and returned by this endpoint.

This token will be used to authenticate the user on the server for further communication, the token is also valid for one week, after which it expires and the user has to login again.

- `/users/profile`: A GET endpoint that returns the authenticated user profile.This endpoint requires authentication.
- `/users/:id`: A PUT endpoint that is used to update the user profile. The payload will contain the fields that need to be updated for the user. This endpoint also requires user authentication.

Tour routes

This is defined here https://campus.cs.le.ac.uk/svn/hcn6/code/branches/app/server/routes/tour.js and defines these endpoints

- `/tour/`: A POST endpoint used to retrieve and return venues to the user with their preferences. This endpoint requires authentication and its payload includes an array of strings for user selected categories, an array for the meals selected and a city selected by the user.
- Sample payload to this endpoint:

```
{
      city: "Hamburg",
      meals: ["breakfast", "lunch"],
      Interests:["4bf58dd8d48988d18f941735",
      "4bf58dd8d48988d190941735"]
}
```

Introduction Using these Ids for the venues and the name of meals the user selected, the server side will make a request to external API to get results that match these. While the results are returned to the client as a JSON response, the interests are also saved to the user account while the tour is created.

The Interests aray in the payload will be stringified and saved as preference for the tour, so when this is retrieved in the future it can be parsed back to an array and used as needed.

Example of the stringifying the interest:
Before: `["4bf58dd8d48988d18f941735", "4bf58dd8d48988d190941735"]`
After: `"["4bf58dd8d48988d18f941735","4bf58dd8d48988d190941735"]"`

The Client (Frontend)

The android application which is the client side is built with ionic framework. Ionic is an open-source software development kit (SDK) for hybrid mobile app development. Using ionic hybrid mobile applications can be developed with javascript web programming frameworks/libraries such as Angular, React and Vue.
These applications can then be packaged into different platforms for Android and iOS. For this project, the android mobile app is developed with the ionic-angular framework.

Angular is a TypeScript-based open-source web application framework led by the Angular Team at Google and by a community of individuals and corporations. It is a MVC framework used for web and hybrid mobile application development.

The MVC architecture in Angular supports both two-way and one-way data binding between the models and the controllers.

Two-way data binding in Angular will help to exchange data from the component to view and from view to the component. It helps to establish bi-directionally communication between the models, views and the controllers.

One-way data binding will bind the data from the component to the view (DOM) or from view to the component. One-way data binding is unidirectional. You can only bind the data from component to the view or from view to the component and not both at the same time.

The Frontend of the application has the following groups of components:

- Public: these are the components that users who are not logged can access. It includes the these components:
    - Login: Here an already existing user can login with their email address and password. This makes a POST request to backend API and gets an auth token for the user. This auth token is stored on local storage and is used to make further requests for that user until the token expires, in which case the user has to login again.

    - Register: This component creates a new user and after successful creation, calls the login endpoint and gets an auth token for the user.

- Secured: These are the components for authenticated users. The components here are:
    - Profile: This is the component for the user profile. Here the user can view and edit their profile
    - Create-tour: this is where the tour itinerary creation is initiated. The user selects city, interests and meals and the backend API is queried with these selections.
    - Tour-itinerary: This holds the final itinerary for the user. Users can get directions between spots on the tour, can view spot details and can also end the tour. Ending the tour redirects to the home page where users can initiate another tour creation with a button click.
    - Logout: This component logs the user out of the application and deletes the auth token from local storage.

The categories presented to the user for selection options are derived from external API categories and stored as configuration constants for easy retrieval and use in the application. The constants can be found here

External API

The Foursquare Venues API is used for retrieving venues for the tour itinerary generation. This external API was selected for use on this project because Foursquare boasts of 900+ Venue Categories in 190+ Countries and 50 Territories with 30+ Attribute Fields. Also because it is free and does not require credit card details for account creation or project setup on the platform.

3.3 Challenges faced

Over the course of the implementation of the project's objectives, the following are the challenges that were faced:

- **Lack Categories endpoints grouped by locations:** FourSquare API used for development currently does not provide a category of venues for selected locations. Its categories API provides all the categories in hierarchies. HERE API provides this API endpoint to query venue categories by location, but they have a different categories naming.

  To overcome this challenge a object mapping saved as a JSON object is used to store these categories for easy retrieval, query and use in the application. While this poses another challenge of having to update this file anytime there is need for change in these categories, for the scope and time of the project, this seems a feasible solution for this challenge.

- **Internationalization:** People of different languages will use this application in locations with different locales this makes it difficult to match user input text to specific languages on the Foursquare and Backend API's.

  A robust solution to this challenge will be to create translations for locales of interest and allow users to select their preferred locale configuration to use on the app. But due to time and scope of the project, text input from users is limited to bare minimum and users instead express their choices and preferences by choosing from a finite set of options available on the app and these options are expressed in English language.

- **Rate Limit on the Foursquare API for free account:** There are limits to attributes returned on the free account, and also maximum number of requests per hour. This makes it difficult to query for when spots will be open, get ratings for spots amongst other challenges.

4. Implementation

This chapter shows the results of using the city tour planner android application. The project contains the Android application and the backend API server.

4.1 Backend API Server (REST)

The backend server exposes REST API endpoints for the Android application to communicate and query with the server. Using an API on the backend server helps to develop a client side that is decoupled from the server and therefore development and deployment can be managed independently for the server and client sides.
This also makes it easier for the client side and server side to be developed using different programming languages or frameworks if needed as they are both independent systems.

The endpoints are grouped into two:

- /users/: These are the endpoints used for creating, updating and querying the User database objects.
- /tours/: These are the endpoints used to create a tour for a user. This involves the server making requests to external API (Foursquare) to get meal and venue recommendations using the expressed preferences by the user.

The sample below shows the API request and response for creating a new user:
URL: /users/
Request method: POST
Request body:
```
{
        "firstName": "John5",
        "email": "john5@demo.com",
        "password": "john5@pass"
}
```

Sample response (in JSON format)

```
{ "response": { "id": 11, "firstName": "John5", "email":
"john5@demo.com", "password":
"$2a$10$cy2pVuF.GN2cLCaMN4Qpd.FzKAHli6SfytkHh9kyBaq258EL9Vh4W",
"updatedAt": "2019-11-18T00:37:00.386Z", "createdAt":
"2019-11-18T00:37:00.386Z", "lastName": null, "dateOfBirth":
null, "gender": null }, "status": 200 }
```

The table below shows the available endpoints on the backend server, their request method, payloads and query parameters.

URL	Details	Request Method	Payload/Parameters	Output
/users/	This is the endpoint used to create users on the backend server.	POST	- firstName - Email - Password	Id, email, hashed password, irstName, lastName, gender, dateOfBirth, createdAt, updatedAt
/users/login	This is the endpoint that is used to log users into the backend server.	POST	- Email - password	JWT token for the authenticated user. The JWT token has an time-to-live TTL of one week after which users has to re-authenticate
/users/profile	This endpoint returns the user profile. This is an authenticated route, meaning it must have the headers that contains the JWT token	GET		Id, email, hashed password, irstName, lastName, gender, dateOfBirth, createdAt, updatedAt

/users/<id>	This endpoint allows for users to be updated. It is also an authenticated endpoint	PUT	Any of the user fields provided as payload will be persisted for the user as an update.	Id, email, hashed password, irstName, lastName, gender, dateOfBirth, createdAt, updatedAt
/tour/	An authenticated endpoint used to create a tour for the user	POST	- City - Meals - Interests	A JSON response from the external API containing 5 venues each for each categoryId provided for interests and meals of choice

Table 1: Table showing the REST API endpoints exposed by the backend API server

4.2 Android Application (Client)

The Android application is the means by which the user interacts with the city tour planner application. This is the process flow for users to successfully create a tour itinerary using the Android application:

- User logs into the account if they are an existing user or creates a new account if none exists
- User is redirected to the home page when login is successful
- User clicks call-to-action button to create a tour
- User is taken to the “create tour” page where they select city, meals to include and interests for the tour.
- User gets redirected to a tour draft page where they see five results for each of the selections made in the previous page
- User then selects a time option, that is how long they would like to spend on a spot during the tour. The application presents option for 1, 1:30 and 2 hours
- After selecting the time option, the user is required to select meal and venues from the API options retried.
- After successfully selecting, user is then redirected to the itinerary page, here they see the tour, where it start, where it ends, the meals they selected are

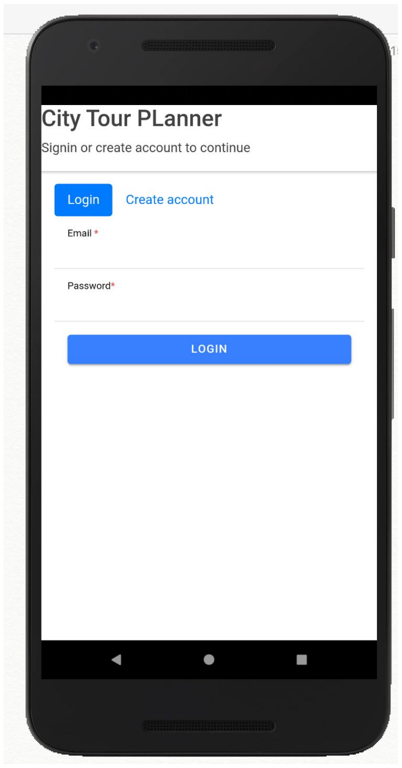
inserted in the right index positions that match their time selection and they can click a button to end the tour and go back to the home page.

The android application has the following page components:

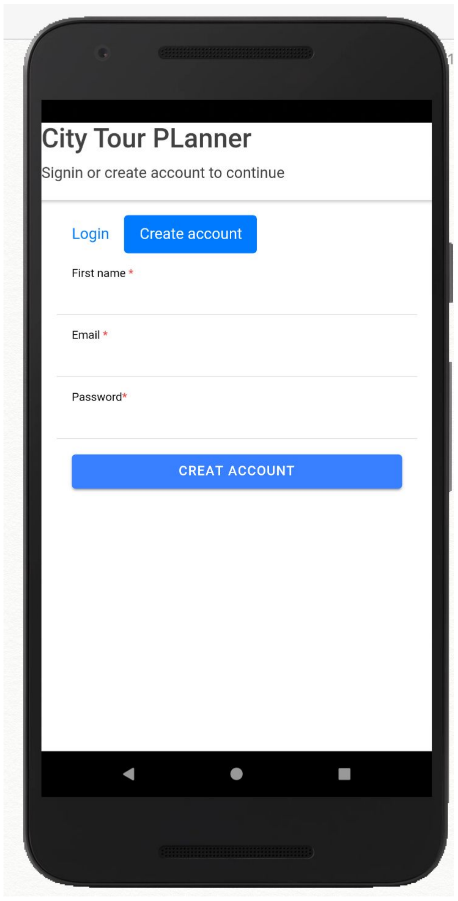
- **Register/login:** This is a page where a user lands when they launch the app. There is the option for existing users to login and new users to create an account.
  If it is a new user, they enter their email, password and firstname, then the front end makes a POST method API to /users/ on the backend server. If the user is successfully created, they are logged in and are redirected to the home page.

  The same process applies for user login, email and password are entered and the user is logged in, the JWT token for the logged in user is stored on local storage and user is redirected to home page

  The login screen image

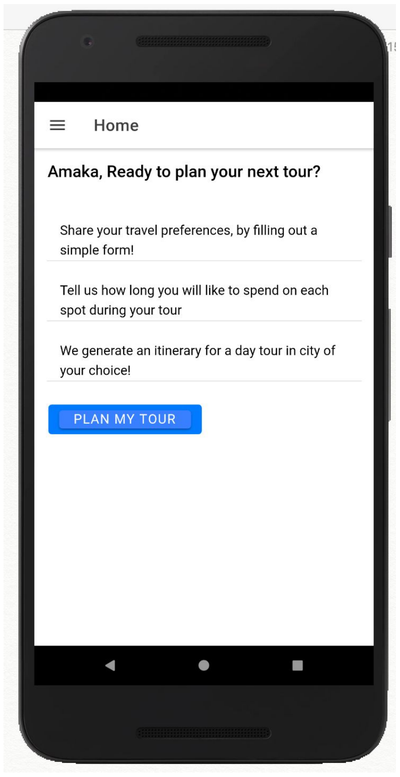


The registration screen image



- **Home**
  This is the page where the user is redirected after successful login. It has a short text and a call-to-action button for the user to begin creating a tour.

Home page screen image



Amaka, Ready to plan your next tour?

Share your travel preferences, by filling out a simple form!

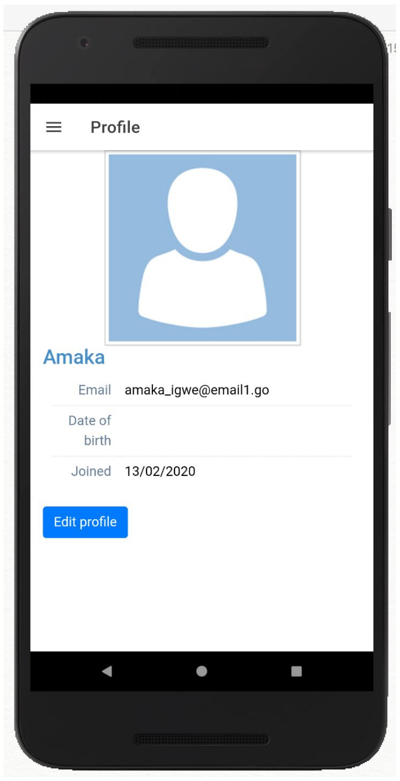
Tell us how long you will like to spend on each spot during your tour

We generate an itinerary for a day tour in city of your choice!

- **View/update profile**
  This page displays the profile of the logged in user. These profile details are derived by making an API call to /profile endpoint on the backend server. The user can also update their profile on this page.

  Clicking the “edit profile” button opens a modal window with all the user fields, and when submitted, makes a PUT request to /users/<id> endpoint to update user profile.

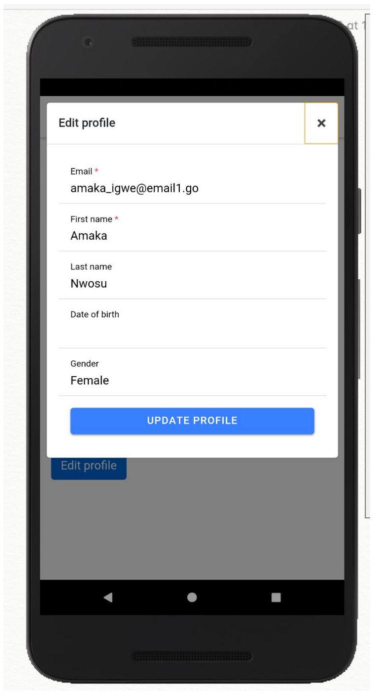
User profile screen page



Amaka

Email	amaka_igwe@email1.go
Date of birth	
Joined	13/02/2020

Edit profile screen page



Email *
amaka_igwe@email1.go

First name *
Amaka

Last name
Nwosu

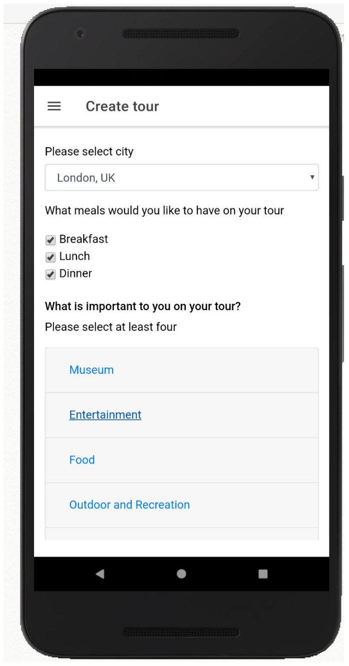
Date of birth

Gender
Female

- **Create tour**
  This is the page where the user starts the tour creation process. It presents the user with options to select a city from a finite number of options, select the meals they want to include on the tour, and asks questions about what activities they would like to do on their tour.

  With these details it makes a POST request to “/tour/” endpoint and returns the response. This response is presented to the user, upon which the selects a time option, different time options have different number of activities a user can select for their tour. After these selections are complete for the time selection, meal and spots, the itinerary is created and presented to the user.

  Create tour screen image:



Create tour

Please select city

London, UK

What meals would you like to have on your tour

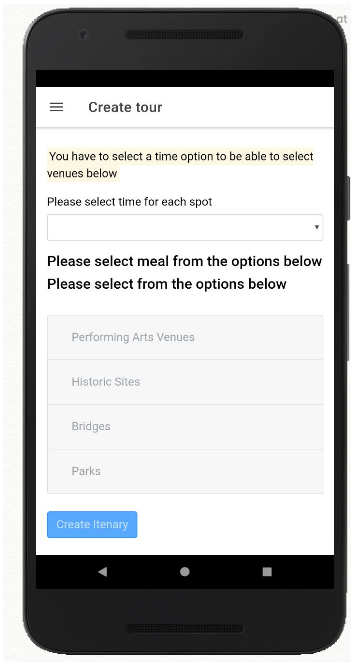
- [✓] Breakfast
- [✓] Lunch
- [✓] Dinner

What is important to you on your tour?

Please select at least four

- Museum
- Entertainment
- Food
- Outdoor and Recreation

Sample tour itinerary draft screen image
Users have to select the time option first to activate the panels and select from the options available.


Create tour

You have to select a time option to be able to select venues below

Please select time for each spot

Please select meal from the options below

Please select from the options below

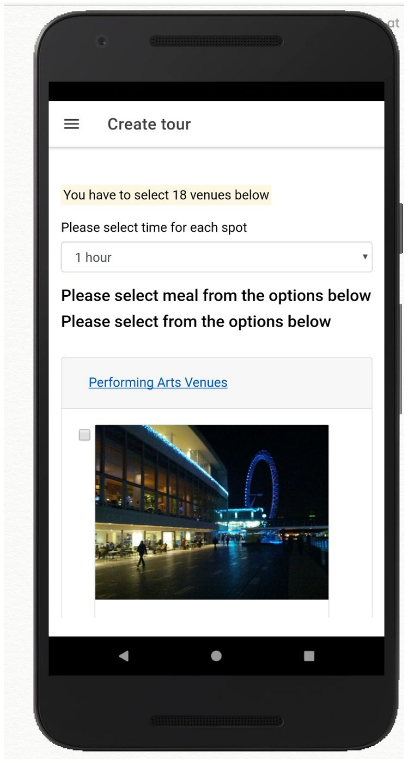
Performing Arts Venues

Historic Sites

Bridges

Parks

Sample screen image after time option is selected:
User has to select the required number of venues and also meal options to activate the "create itinerary" button


Create tour

You have to select 18 venues below

Please select time for each spot

1 hour

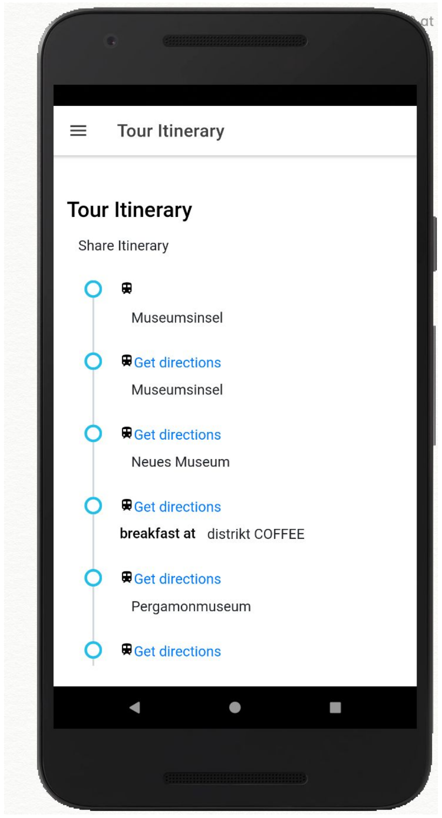
Please select meal from the options below

Please select from the options below

Performing Arts Venues

Tour itinerary page image (After user has completed the required selections)
Clicking on the spot name opens a modal window with the details of the venue, clicking on “get directions” opens a Google map with the start and destination prefilled.

Clicking “End tour” marks the end of this tour and redirects users to the home page.



Tour Itinerary

Share Itinerary

Museumsinsel

Get directions
Museumsinsel

Get directions
Neues Museum

Get directions
breakfast at distrikt COFFEE

Get directions
Pergamonmuseum

Get directions

Because angular is an MVC framework, each component has a model, view and a controller and in this application uses a two-way data binding between the models,views and the controllers.
The view are html templates that render the markup and data associated with it.

4.3 Code snippets

Android Application (Client)

Code snippet 1: Authentication service

```
getUserProfile() {
   const user = JSON.parse(localStorage.getItem('user'))
   const authHeader = {
     headers: {
       'Content-Type': 'application/json',
       'Access-Control-Allow-Origin': '*',
       'Authorization': user.token
     }
   };

   return this.http.get<any>(`${API_URL}/users/profile `, authHeader).toPromise()
 }

 loginUser(loginCredentials: any): Promise<any> {
   return this.http.post<any>(`${API_URL}/users/login `, loginCredentials,
httpOptions).toPromise()
     .then(response => {
       if (response.success) {
         localStorage.setItem("user", JSON.stringify(response));
         return true
       }
     })
     .catch(error => {
       console.log(error, 'error logging user in')
       return false;
     })
 }

 createUser(userCredentials: any): Promise<any> {
   return this.http.post<any>(`${API_URL}/users/ `, userCredentials,
httpOptions).toPromise()
     .then(response => {
       if (response.status == 200 && response.response.id) {
         return this.loginUser({ email: userCredentials.email, password:
userCredentials.password })
       }
     })
     .catch(error => {
       console.log(error, 'error creating user')
       return false;
     })
 }

 updateUserProfile(userData: any): Promise<any> {
   const user = JSON.parse(localStorage.getItem('user'))
   const authHeader = {
     headers: {
       'Content-Type': 'application/json',
       'Access-Control-Allow-Origin': '*',
       'Authorization': user.token
     }
   };
   return this.http.put<any>(`${API_URL}/users/${userData.id} `, userData,
authHeader).toPromise();
 }
```

This is a utility service on the frontend of the application that handles authentication for

- Creating user
- Logging user in
- Updating user profile

It creates a service that can be used in any other component in the application.
This is useful to eliminate code duplication throughout the code for frequently occurring actions and with the dependency injection in Angular, it is easy to inject this service and initiate it through the component constructor

Code snippet 2: Create tour component

```
selectMeal(meal) {
   if (meal.selected && this.selectedMealOptions.indexOf(meal.value) == -1) {
     this.selectedMealOptions.push(meal.value);
   }
   if (!meal.selected && this.selectedMealOptions.indexOf(meal.value) != -1) {
     this.selectedMealOptions.splice(this.selectedMealOptions.indexOf(meal.value), 1);
   }
 }

 selectMealOption(event, meal, key) {
   const spotsCount = this.noOfTourItemsToSelect + this.totalMeal;
   const mealOption = this.timeOptions.find(option => option.spotsCount ==
spotsCount);

   meal['index'] = mealOption.mealIndexes[key];
   meal['label'] = key;

   if (meal.selected && event.target.checked) {
     this.selectedMeals[key] = meal;
   }
 }

addItemToInterests(item) {
   if (this.formData['interests'] && this.formData['interests'].indexOf(item) == -1) {
     this.formData['interests'].push(item)
   } else {
     this.formData['interests'] = [item]
   }
 }

updateTotalSelected(item, category) {
   if (item.selected) {
     this.totalSelected++
     this.addItemToInterests(item.value);
   } else {
     this.totalSelected--;
     this.removeItemFromInterests(item.value);
   }
```

```
  this.updateButtonStatus();
}

submitPreference(formData) {
  if (!this.city) {
    this.showWarning = true;
  } else {
    this.showWarning = false;
    this.apiService.createTour({
      city: this.city,
      interests: formData.interests,
      meals: this.selectedMealOptions
    })
      .then(responses => {
        this.itenarnaryDraftGenerated = true;
        this.itenarnaryDraft = responses.response;
        this.meals = responses.meals;
        this.totalMeal = this.meals.length;
      })
      .catch(error => {
        console.log(error, 'API error')
      })
  }
}
```

This component performs major weight lifting in the tour itinerary creation.
The values imported from “./constants” contain JSON configurations derived from Foursquare API to help a user select what they want to do.

The cities and museumOptions can be seen below:

```
const CITIES = [
 { name: 'Berlin, Germany', value: 'Berlin' },
 { name: 'Munich, Germany', value: 'Munich' },
 { name: 'Hamburg, Germany', value: 'Hamburg' },
 { name: 'Frankfurt, Germany', value: 'Frankfurt' },
 { name: 'Leipzig, Germany', value: 'Leipzig' },
 { name: 'London, UK', value: 'London' },
]

const museumOptions = [
 { name: 'Art Museum', value: '4bf58dd8d48988d18f941735', selected: false },
 { name: 'History Museum', value: '4bf58dd8d48988d190941735', selected: false },
 { name: 'Science Museum', value: '4bf58dd8d48988d191941735', selected: false }
]
```

With these values, user interests when selected are being mapped to the category IDs, which are then sent to the API to fetch corresponding responses for these interests.

Some of the methods in this component are explained below:

SelectMeal: This is a method that binds the meal option selected by the user on the view. It checks if that option is selected, by checking that the model value "selected" is true, then it also checks that the value does not already exist in the array, this is necessary to avoid duplication of same values in the selectedMealOptions.

Whenever a user selects or de-selects a meal option, this method ensures that it is added or removed accordingly. This is where the "meals" object in the payload sent to the /tour/ API endpoint is created.

addItemToInterests: This method adds user selections to the formData object. It using the key "interests" to store the selected categories. There is a corresponding "removeItemFromInterests" that removes items from the interests. The model is bound to the item.selected property that is either true or false depending on if it is selected or not. Angular provides a "change" html extension markup, with the an input will be observed for changes and a function can be executed when this change is emitted.

Example:
```
<div class="form-check">
             <input class="form-check-input" type="checkbox"
value="museumOption.value" id="museumCheckbox"
               (change)="updateTotalSelected(museumOption, 'museum')"
[(ngModel)]="museumOption.selected">
             <label class="form-check-label" for="museumCheckbox">
               {{museumOption.name}}
             </label>
           </div>
```

This is a sample HTML markup in the create-tour view, the change HTML markup extension from Angular makes it possible to listen for when this input is clicked. This click event can be a select or a de-select, and that value is bound by the ngModel markup extension, also provided by Angular.

So whenever this checkbox is clicked, “updateTotalSelected” will be executed with the data model musuemOption.selected, which is either true or false if the checkbox was checked or unchecked.

submitPreference: This method uses the API service to make an API call to the /tour/ endpoint, to create a tour using the selections made by the user.

When the response is returned from the backend API server, the response data is bound to “itenarnaryDraft” in the model. Also, the “meals” and “totalMeal” are derived from this API response.

selectMealOption: this method word with the meals from the API response. It adds an index position to each selected meal option to match the time selection of the user.

updateSelectedDraftItems: this method is called whenever a user makes a spot selection on the view. It checks that the item selected does not already exist in the model and updates accordingly. If the items exist and the user deselects it from the view, that item is removed from “selectedDraftItems” data binding object, if it does not exist and the user selects it, it is added to “selectedDraftItems”.

When a user has selected the items required for the itinerary to be created, the “create itinerary” button is activated in the view, and when clicked executed “createTourItinerary” method.

createTourItinerary: this method goes through user selected items, sorts them by distance starting with the first item in the array. It uses the TurfJs (A javascript GeoSpatial package for nodeJs) to compute the distance between points and then sorts the items based on this distance. This ensures that the items in the itinerary are sorted by their proximity to one another, so users don't spend much travelling between locations.

When this method is executed successfully, user is redirected to itinerary page, where they can

- See all the spots for their tour
- Click on each spot to view its details
- See when to take each meal on their tour
- Get directions between spots on their tour by clicking on the link.

Backend API Server

Code snippet 3: User login

```
router.post('/login', async (req, res) => {

 const email = req.body.email;
 const password = req.body.password;

 const user = await User.findOne({
   where: { email: email },
   raw: true
 }).catch(error => {
   res.send({ error: error, status: res.statusCode })
 })


 const isMatch = await promisify(bcrypt.compare)(password, user.password)

 if (isMatch) {
   const token = jwt.sign(user, dbSecret, {
     expiresIn: 604800 // 1 week
   });

```

```
    // Don't include the password in the returned user object
    return res.send({
      success: true,
      token: 'JWT ' + token,
      user: {
        id: user._id,
        name: user.name,
        email: user.email,
      }
    });
  }
  else {
    return res.status(400).json({ success: false, msg: 'Wrong password.' });
  }
});
```

This is the endpoint that implements user login on the server side. It gets the user object from the database using the sequelize ORM, compares the hashed password retrieved from the database to the hash of the one in the request payload, when there is a match, a JSON web token, JWT is generated and returned as response.

4.4 Running the application

The application is developed using the client-server architecture, with a hybrid app development framework, so it is available to bothe view in the browser on the web and can be built to run as an android application.

To run the app, the following steps should be taken:

- Ensure PostgresQL is up and running
- Cd into the server folder
- execute “npm install” in the command line
- execute “npm run createDb” to create the postgresQL database using specified configuration in the app
- execute “npm run migrate” to run the migration using the generated migrations files.
- To view the application on the web, cd into client folder and execute “ionic serve”
- To run the android application, an android emulator has to be up and running, then execute “ionic cordova run android” this runs the application on the running Android emulator

5. Testing and Evaluation

5.1 Testing

5.1.1 Backend API server (REST)

The backend of the application has unit testing written using the following javascript tools:

- Chai: This is a Behavioral Driven Development (BDD) / Test driven Development (TDD) assertion library for node and the browser. It is an assertion library that can be used to test expectations on NodeJs applications.
- Mocha: Mocha is a Javascript test runner, it runs tests serially, allowing for flexible and accurate reporting, while mapping uncaught exceptions to the correct test cases.

The server side uses Mocha as the test runner and Chai for testing expectations with assertions. To run these unit tests on the server side, execute the command `"npm run test:mocha"`

Some of the tests implemented on the server are to check that important routes in the application exist, and return a 200 status code when called, and also that for routes that don't exist, 404 is returned.

One of the important unit tests in the application is the test for the `“/health”` route.
It is good practice to have this health route implemented and tested on REST APIs as they are a way to test that the server is up and running which is a very important requirement for the application to be used.

The unit tests on the server side are located in the test folder

Some of the unit tests on the server side are:

```
it('should return a 404 for routes that don\'t exist', (done) => {
     chai.request(server)
     .get(`/test`)
     .end(function (err, res) {
       if (err) {
         console.log(err)
       }
       expect(404)
       done()
     })
   });

   it('should test the health check route', (done) => {
     chai.request(server)
     .get(`/_health`)
     .end(function (err, res) {
       if (err) {
         console.log(err)
       }
       expect(200)
       expect(res.body).to.be.an('object')
       expect(res.body.message).to.equal('ok')
       done()
     })
   });
```

The snippet above shows two unit tests that exist on the server side of the application. The first test is checking that the application returns a 404 for routes that are not implemented, this is an important test because it helps any client calling the server to know that the route they are trying to reach does not exist on the application and so it is a useful test to have.

The second test is for checking that the server is up and running by testing the health route and checking what it returns. In an automated test system this health test is run before applications are deployed, because they help detect if something is wrong with the server or if it is up and running.

It can also be used by clients calling the API in writing integration tests, they can invoke _health endpoint to see if the server they are trying to reach is running without calling any routes that executes a business logic.

5.1.2 Android Application (Client)

The Android application is tested manually for some test cases to check that functionality implemented meets the application requirement.

The following test cases were carried out on the Android application:

- User registration
- User login
- User update profile
- User create tour itinerary for 1 hour at each spot time option and want to have breakfast, lunch and dinner on their tour
- User create tour itinerary for 1:30 hour at each spot time option and want to have breakfast and lunch on their tour
- User create tour itinerary for 2 hours at each spot time option and want to have lunch and dinner on their tour
- User end tour
- User logout

User registration test case

Title: Registration page - Create a new user account successfully
Description: A new user should be able to register when the app is launched
Precondition: User has an email, first name and a desired password for their account
Assumption: The user has successfully launched the android application
Test steps:

1. Launch the android application
2. Select the “Create account” tab
3. In the “first name”, “email” and “password” input fields, enter the user firstname email address and the password for the account
4. Click “CREATE ACCOUNT” button

Expected result:
The user should be redirected to the home page where they see a heading with their first name and a welcome message. They also see some paragraphs of text and a call-to-action button to start tour creation

User login test case User view and update profile

Title: Login page - authenticate a user successfully
Description: A registered user should be able to login successfully with their email and password
Precondition: the user must already be registered with an email address and password.
Assumption: The user has successfully launched the android application
Test steps:

1. Launch the android application
2. Launch the android application
3. Select the “Login” tab
4. Enter the email and password of the registered user in the email and password input fields respectively
5. Click “LOGIN” button

Expected result:
The user should be redirected to the home page where they see a heading with their first name and a welcome message. They also see some paragraphs of text and a call-to-action button to start tour creation

User update profile test case

Title: Profile page - logged in user should be able to update their profile
Description: A logged in user should be able to update their profile details and see the changes reflected on the profile page
Precondition: the user must already be logged into the application
Assumption: The user has successfully launched and logged in to the android application
Test steps:

1. Click on the menu icon on the application screen
2. Select the “profile” menu link
3. Click the “Edit profile” button
4. Enter the details that you will like to update in the modal window that shows up
5. Click the “UPDATE PROFILE” button

Expected result:
The page should be refreshed and the user should see their updated profile with the new changes.

User create a tour itinerary for 1 hour at each spot time option test case

Title: Create itinerary page - A logged in user should be able to create a tour itinerary where they select to spend 1 hour at each spot and also have breakfast, lunch and dinner on the tour.
Description: A logged in user should be able to create a tour itinerary where they select 1 hour time option (time to spend on each spot as one hour) and they select breakfast, lunch and dinner for meal options
Precondition: The user must already be logged into the application
Assumption: The user has successfully launched and logged in to the android application
Test steps:

1. Click on the menu icon on the application screen
2. Select the “Create Tour” menu link
3. Select a city for the tour
4. Select “breakfast”, “lunch” and “dinner” meal options
5. Select the options for things to do by expanding each panel and selecting desired preferences
6. Click the “Create Tour” button when done selecting preferences
7. On the new page, select “1 hour” for time option
8. The number of spots to be selected will be displayed on the page, In this case it should be 15, select the meals and the spots of choice
9. Click the “Create Itinerary” button when it is activated as this shows the number of expected spots have been selected

Expected result:
The user should see a tour itinerary that has 18 spots. The breakfast spot should be the 4th item, lunch should be the 8th item and dinner should be the 13th item respectively. The user should also see a “Get directions” link between the spots on the list and clicking each one should open Google maps with start and stop locations already entered.

User create a tour itinerary for 1:30 hour at each spot time option test case

Title: Create itinerary page - A logged in user should be able to create a tour itinerary where they select to spend 1 hour at each spot and also have breakfast and lunch on the tour.

Description: A logged in user should be able to create a tour itinerary where they select 1:30 hour time option (time to spend on each spot as one hour) and they select breakfast and lunch for meal options

Precondition: the user must already be logged into the application

Assumption: The user has successfully launched and logged in to the android application

Test steps:

1. Click on the menu icon on the application screen
2. Select the “Create Tour” menu link
3. Select a city for the tour
4. Select “breakfast” and “lunch” meal options
5. Select the options for things to do by expanding each panel and selecting desired preferences
6. Click the “Create tour” button when done selecting preferences
7. On the new page, select “1 hour 30” for time option
8. The number of spots to be selected will be displayed on the page, In this case it should be 10, select the meals and the spots of choice
9. Click the “Create Itinerary” button when it is activated as this shows the number of expected spots have been selected

Expected result:

The user should see a tour itinerary that has 12 spots. The breakfast spot should be the 4th item and lunch should be the 7th item respectively.

The user should also see a “Get directions” link between the spots on the list and clicking each one should open Google maps with start and stop locations already entered.

User create a tour itinerary for a 2 hour at each spot time option test case

Title: Create itinerary page - A logged in user should be able to create a tour itinerary where they select to spend 2 hours at each spot and also have lunch and dinner on the tour.

Description: A logged in user should be able to create a tour itinerary where they select 2 hours time option (time to spend on each spot as one hour) and they select lunch and dinner for meal options

Precondition: the user must already be logged into the application

Assumption: The user has successfully launched and logged in to the android application

Test steps:

1. Click on the menu icon on the application screen
2. Select the “Create Tour” menu link
3. Select a city for the tour
4. Select “lunch” and “dinner” meal options
5. Select the options for things to do by expanding each panel and selecting desired preferences
6. Click the “Create tour” button when done selecting preferences
7. On the new page, select “2 hours” for time option
8. The number of spots to be selected will be displayed on the page, In this case it should be 6, select the meals and the spots of choice
9. Click the “Create Itinerary” button when it is activated as this shows the number of expected spots have been selected

Expected result:

The user should see a tour itinerary that has 8 spots. The lunch spot should be the 5th item and dinner should be the 8th item respectively.

The user should also see a “Get directions” link between the spots on the list and clicking each one should open Google maps with start and stop locations already entered.

User end tour test case

Title: Itinerary page - A logged in user that has created a tour should successfully end the tour

Description: A logged in user that has successfully created a itinerary for their city tour should be able to end the itinerary successfully

Precondition: the user must already be logged into the application and have created a tour and can see the itinerary generated on the itinerary page

Assumption: The user has successfully launched and logged in to the android application

Test steps:

1. Click the “End Tour” link nn the itinerary page where the user can see the itinerary generated.

Expected result:

The user should be redirected to the home page.

User logout test case

Title: Logout user - logged in user should be able to logout of the application
Description: A logged in user should be able to log out of the application successfully
Precondition: the user must already be logged into the application
Assumption: The user has successfully launched and logged in to the android application
Test steps:

1. Click on the menu icon on the application screen
2. Select the “logout” menu link

Expected result:

The user should be logged out of the application and redirected to login screen.

5.2 Evaluation

Summary

The project took a look at existing platforms and solutions that offer different ways for users to tour a location and developed a solution to help a user plan a day tour in a city by expressing their preferences.

While applications like RoadTrippers, Tripit, Musement and Blink Travel offer different solutions to help users make the best use of their time while on their trip, none of them offer users the option to select what they would really like to do when they are in their tour destinations. The itinerary these applications output to the user are not personalized, the user provides no input as regards to their preferences on these applications.

While Journy platform provides a solution for users to get personalized itinerary on their travels and tours by pairing users with human tour planner, this introduces an extra layer to the tour planning process, increases the time for user to get their itinerary, makes changing plans on the fly impossible and also cost more to hire the tour planner.

With the application from this project, users do not have an extra human layer to deal with, therefore they can plan their tour by themselves whenever they want. They can easily change their mind on their expressed preference or even start a whole new tour creation with no time delay or extra layer of communication. Also, using an application costs less than hiring a human to plan the tour.

Some challenges were also encountered during the development of this project.
One of such challenges is creating an deciding how to take the amount of time users will like to spend on the tour into consideration while generating the itinerary. The approach used to tackle this challenge is to create a configuration object, such that when a user selects a time, it maps that selection to a configuration that includes the total number of attractions and meal position indexes for that time selection.

While testing out the application it was also observed that allowing users to enter text might be a challenge for the external API due to locale differences that might exist between the user and the location they want to generate a tour for. To tackle this, the application enables users to express their preferences through selection from available options instead of them to enter free text. Also, the application is expressed entirely in English language, thus eliminating the confusion of locales between the user and the tour city.

While the project achieves its key objectives, these are some features that are missing from the implementation:

- Users cannot enter a location to start the tour with and the application does not implement device location pickup. This was not implemented for some reasons:
    - User input as text will need to be geocoded before it can be useful in the tour planning, but the APIs that offer such geocoding come at a fee.
    - Allow users to share itinerary: This was listed as one of the features to implement, but during development this proved to be more challenging than anticipated and as it is not a core feature of the project, it was dropped. But it can be developed as future improvements for the application.

Limitations

While this project achieves its key objects, there are however some limitations:

- Users cannot specify start and stop locations for the tour: the solutions that would allow users to enter an address and get that address geocoded is provided by Google Maps, but it comes at a fee and thus could not be used in this project.
- Users cannot use the app in languages other than english. This is because due to time and scope of the project, internationalization was not implemented.
- Users cannot edit an already generated itinerary, they have to start all over with the process of creating a tour.
- There are no default options, while this project allows the users to express their preferences and plans a personalized tour for them in a city, it would be nice for users to have default options. This is because sometimes users are confused and don't really know what they want and so it would be a great feature for the application to have default settings for such scenarios.
- Not accounting for the travel time between attractions. The application does not take into account or consider the commute time between the attractions when the user is on the tour. This feature if implemented will help the user have a better overview of the time from the start to end of the tour.

Potential Improvements

The application presented in this project meets the key objectives of the thesis and achieves the desired results but with room for improvements.

- The application should provide a way for users to share their tour and enable live location tracking for whoever they share with. This feature will help keep their loved ones fully informed of their location at any point in the tour.
- The application should allow users to use their geolocation and also provide means for them to enter start and stop locations by text if they don’t want to use their device geolocation.
- The applications should also improve on the design and presentation, as now it focuses more on feature than design.
- There should be automated tests on the server and client side of the applications so that builds, packaging and deployments will be done with a more confident level.
- Users should be able to import a shared itinerary from other users and be able to use this imported data to plan a tour. This will be a useful scenario for when a

user does not know what they want to do in a place, but have a friend who knows the place and knows what they like, these friends can create an itinerary using the app, export or share it with the user and they can easily import and use it.

Bibliography:

1. Wiki tourism definition (2019) available at https://en.wikipedia.org/wiki/Tourism
2. The World Tourism Organization definition of tourism https://en.wikipedia.org/wiki/World_Tourism_Organization
3. Zach Samantha. Testimonial for gojourny application Available at: https://gojourny.com/#testimonials (Accessed: November 2019).
4. Roadtrippers Application website: https://roadtrippers.com/
5. TripIt Website: https://tripit.com/web
6. Musement App website: https://www.musement.com/us/
7. Blink Travel Website: https://www.blinktravel.guide/
8. Adventurelink: http://www.adventurelink.com/
9. Here API documentation: https://developer.here.com/documentation
10. Foursquare API: https://developer.foursquare.com/
11. Yallaling Goudar. (2019) *One-way and Two-way Data Binding in Angular*. Available at: https://www.pluralsight.com/guides/one-and-two-way-data-binding-angular (Accessed: February 2020).