Software Nanodegree Group Project: Travel App

https://github.com/chanyl-nicole/CFG_Nano_Project_Travel

Amelia Bourton, Dumisile Mbuthuma, Nicole Chan, Rosalind Grubin and Teresa Diaz Calvo

1.1 Introduction

1.1.1 Aims & Objectives of the Project

The aim of our project was to create a travel 'search engine app' which helps users plan ahead for their holiday, especially with reference to Covid-19 travel restrictions. The main idea of the app was to allow users to input their preferred holiday destination and get back information on the current Covid restrictions for that destination. It should give real time updates on Covid travel restrictions for the location of choice. Since these restrictions are changing all the time, it is important for travellers to be able to keep up to date with this information and not have to visit multiple websites to access it.

Main use and advantages of using the app:

- Provides real time information on Covid restrictions for selected destination/s.
- Reduces complexity of having to search multiple government websites
- Helps plan the trip by also providing weather info and packing list suggestions according to the weather
- Reduces stress during holiday-planning

1.1.2 Roadmap of the Report

1.2 Background

TravelApp is an application developed to help the user plan their summer holidays. Initially it asks the user in which summer month (June, July, August or September) they would like to travel in. Then the app provides the top 8 European holiday destinations and the expected weather for those cities for the selected month. With this information, the user can then enter their choice of city. After this, the app allows the user to input up to ten personal items to bring on their trip that will be stored in a personal 'reminder list'. The app returns a list of essential suggested items for the user to bring on their holidays (according to the expected weather on the destination of choice) and it also shows the personal items previously added by the user to the reminder list. Finally the app retrieves current Covid-19 travel restrictions including testing policies, internal travel restrictions, restrictions on gatherings and contact tracing for the destination they plan to travel to.

1.2.1 Design Conditions

- City limited to top 8 destinations in Europe: Paris, London, Rome, Florence, Barcelona, Swiss Alps, Amsterdam or Santorini.
- Month of travel limited to 4 summer months: June, July, August or September
- The app considers warm weather to be an average monthly maximum temperature of greater than or equal to 15 Celsius and cold weather to be an average monthly maximum temperature of less than 15 Celsius.
- The app considers wet weather to be an average monthly average rainfall of greater than or equal to 50mm and dry weather to be an average monthly average rainfall of less than 50mm.
- This information was obtained manually from Google weather searches and was used to create the classification information in the database, for the weather of each city per month (dry-warm, wet-warm, dry-cold or wet-cold).

1.3 Specifications and Design

1.3.1 Requirements

In our initial discussions we hoped that the app would have a number of parameters for the user to define/input regarding their desired holiday including;

- Origin
- Destination
- Dates of trip
- Willing to quarantine in the selected holiday destination? (Yes/No)

It was also hoped that the app would then output a number of parameters:

- Whether you can go to the destination based on current covid restrictions
- If not, it may suggest other countries (with no/less Covid restrictions)
  What the Covid-related restrictions are
- Does the user need a vaccine? When does the user need to get their vaccination completed by? (e.g. 2 weeks before departure date)
- Isolation period required
- Is hotel quarantine required?
- Are tests required? How many and on which day of arriving/returning? What kind of test?
- Nice to have features if time allows;
  Weather forecast for that destination
    - Weather
    - Suggested packing list with essential items according to weather

1.3.2 Design and Architecture

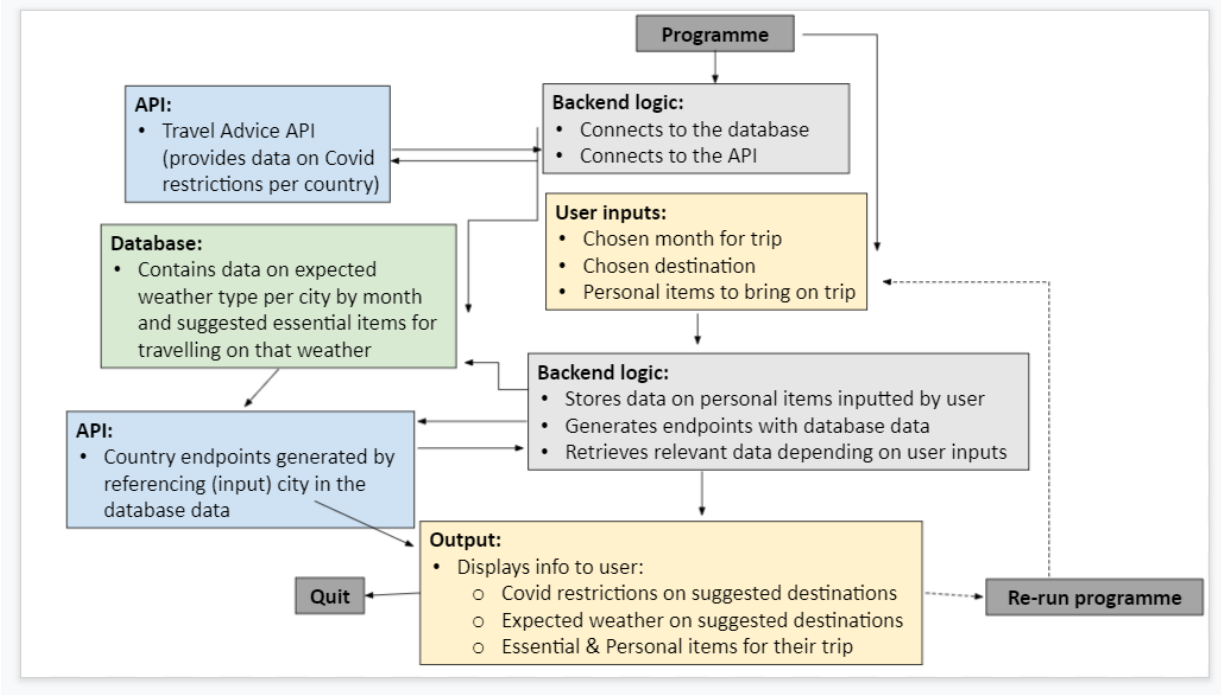

Diagram 1.3.2: System architecture diagram which shows the flow of between the inputs, backend logic, API, database and user outputs.

1.4 Implementation and Execution

1.4.1 Development Approach

In our first meeting as a group we outlined how much time each of us would have to commit to the project each week, so that we could allocate tasks accordingly;

- Nicole - 1 to 2 days, mainly weekends
- Amelia - 1 days, weekends
- Rosalind - 2 days - weekdays
- Teresa - 1 to 2 days - weekdays
- Sanele - 3 to 4 days - weekdays

We decided to work on small sprints (short-term goals) and divide the work between us for each sprint. We also used a Kanban board to help us allocate tasks and check progress. The overall timeline for the project was devised by breaking each week into a sprint.

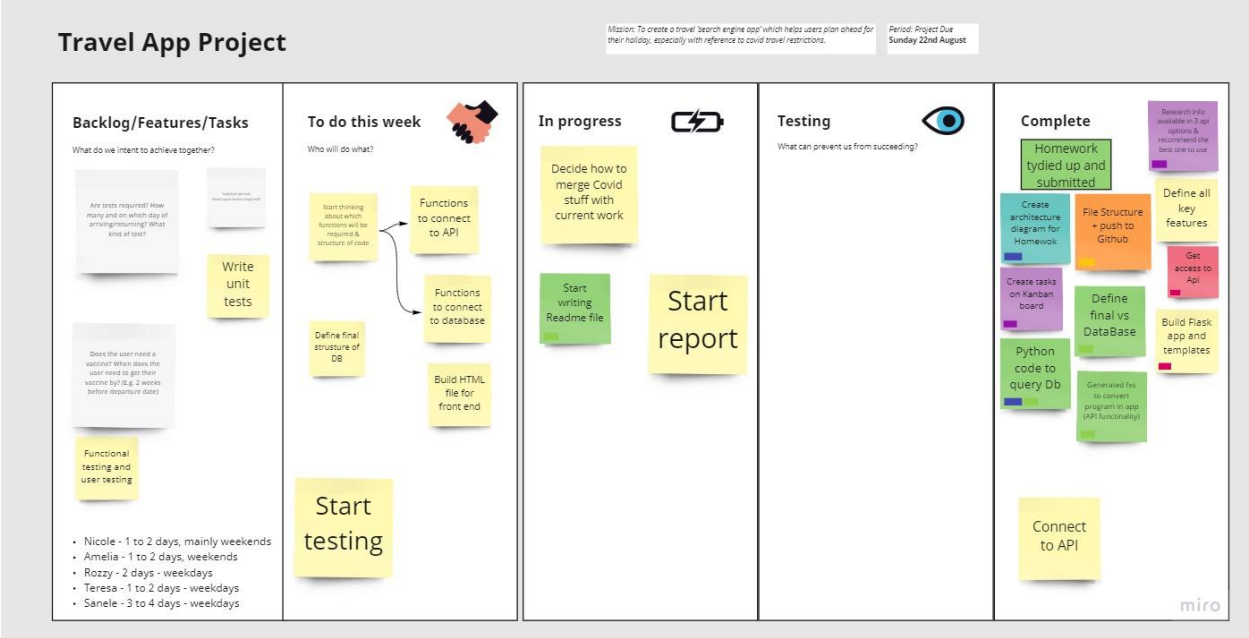

Diagram 1.4.1: Example of our Kanban board used throughout the project. As post it tasks were assigned a colour coded tag with the assignee's name was added. Tasks that would be dependent on other tasks were linked with arrows.

1.4.1.1 Week 1: Complete Planning

- Project conceptualization (all team members)
- Define all key features (all team members)
- Tidy up homework to submit (Teresa)
- Rough draft of database (Teresa)
- Architecture diagram (Rosalind)
- Get access to the API (Nicole)
- Complete Kanban board (Amelia)
- File structure + push to Github repo (Sanele)

1.4.1.2 Week 2: Development

- Build database - Teresa
- Connection to API - Amelia
- Testing connection to API -Dumisile
- Define functions - Rosalind

1.4.1.3 Week 3: Development

- Features to develop
- OOP - Dumisile
- HTML + Flask (good to have is time allows) - Nicole
- Testing on database connection and features - Teresa

1.4.1.4 Week 4: Testing + Documentation

- Unit tests - Rosalind
- Bug fixing - Dumisile
- Report (PDF) -Amelia
- Read Me documentation - Teresa
- Powerpoint - Nicole

As well as maintaining the kanban board (Diagram 1.4.1) and messaging in our slack group we also tried to hold a scrum meeting as a team once or twice a week to discuss progress and any challenges that we were currently facing that may hold up the project. Due to everyone's different commitments outside the Nanodegree (and time zone differences) it was sometimes difficult to find a time that we could all attend to do this. When not everyone could attend, notes of the progress and actions were taken and shared in our slack group.

Whilst we tried to stick to the plan as best as possible there were a few challenges that threw the project slightly off course. Gathering information from the Covid API took us longer than expected due to the lack of information available and cost requirements (see **1.4.2.1.1** in challenges for more details). This meant that we weren't sure which fields would be available for us to access or use to link to the database. We therefore had to start creating the database without knowing which countries we would have access to. During this period, additional features and requirements were also added,

including functionality for creating a personal packing list and retrieving defined recommendations dependent on weather.

Each week tasks were divided out depending on how much time each member had to offer the project and at what point in the week they were available e.g. weekdays or weekends. Broadly speaking Rosalind and Teresa worked on the database and functions required for accessing and updating it. Meanwhile Dumisile and Amelia worked on accessing data from the covid api. Nicole worked on the front end flask elements that we were unfortunately unable to incorporate in the end due to time and function incompatibility, which is covered in more detail in **1.4.2.1.3**. Where possible we tried to review each other's code. In the final week everyone was involved in testing and refactoring the code so that it was consistent and ran smoothly.

1.4.2 Implementation Process

1.4.2.1 Challenges

1.4.2.1.1 API

Our first challenge arose whilst looking for an API that could provide COVID travel restrictions at no cost and that was accessible to non- business users. The first API that we identified provided by Sherpa had a sufficient level of granularity on Covid restriction data and good supporting documentation, but access was only available at a cost to business customers. Our next option was Travel Perk; we quickly found that the data in the free sandbox environment was extremely limited. For example data could only be accessed for the origin country of Germany for one destination of Spain. The data was also not updated and was only available for 15/10/2020, which would have made our objective of real time information unachievable. We then identified a third possible API from Rapid API which provided details on EU travel restrictions. Whilst it covered most locations in the EU, the covid restriction data was very limited eg ‘Travel Restrictions for the country’ without specifying what the restrictions were. Finally we found a fourth API option from Travel Advice. Whilst the requests are limited to 100 per api key, this api does provide a wealth of information on all aspects of travel restrictions. Manipulating the json data was at first tricky with dictionaries inside lists, but we did eventually get it to work and decided on this option for future scalability.

1.4.2.1.2 Git

Working collaboratively on Git was also a challenge. For the first week or two of the project, new branches and pull requests were being created all over the place which made it hard to keep track of which files were the most up to date files. As a result we probably lost a bit of time with potentially duplicated efforts and needed to refactor our

code in order to merge it together.Throughout the project our communication as a team definitely improved.

1.4.2.1.3 Frontend vs Backend

The initial idea was to have a front end HTML app that was connected using Flask. This would allow the user to more easily interact with the functionalities of the app itself. For example, being able to create an input (origin and destination country) and see the output defined in a clear and user friendly way.

We created a virtual environment where the flask packages and the rest of all the packages can reside. This meant that the app was easier to collaborate on and team members wouldn't have to separately download packages for the app to run properly. Then a file structure consisting of a 'static' and 'templates' file was created to support the HTML files that would form the front end. These files utilised Jinja syntax, which essentially act as an 'HTML wireframe' that is connected to an apptest.py file where the separate functions for communicating with the database and api would run. However, due to the changes in some of the app's direction/functionality, this led to increased complexities in integrating the backend functions to the front end functions of the app. This meant that there were now more inputs that were dependent on each other: destination location, travel month, packing list items. This also meant that there was an increase in the amount of outputs: destination weather, user's specified packing list, packing list recommendations that were dependent on the weather, covid restrictions. Time allowing, we would like to finish creating a web app that could act as a travel dashboard/guide to increase usability for end-users. (The initial front- end work can be viewed on the flask_connection branch)

1.4.2.2 Changes along the way

We defined a design specification for the app (see section **1.3.1.1**) in our initial discussions however, pretty early on we realised that all of these features would not be achievable in the time frame that we had, so opted for a minimum viable product approach. As a result of this we decided to limit the destinations to only the top 8 European cities; Paris, London, Rome, Florence, Barcelona, Swiss Alps, Amsterdam or Santorini. This meant that we did not need to create data for additional countries in the database and also meant that there was not an endless amount of combinations to test. In addition for the same reasons, we also limited the month of travel to the summer months (June, July, August or September) and categorised the weather into 4 groups (dry-warm, wet-warm, dry-cold,wet-cold).

The inputs were refined to the month of travel, destination city and an option to add items you don't want to forget to a personal packing list. The travel app outputs covid related restrictions as we hoped, although a future development of the project would be to be able to choose a new destination city if your original destination choice was too

restricted for you to be able to travel. The willingness to quarantine field (input) would have tied in nicely with this. It may also be beneficial for the essential and personal packing lists to only be returned if the covid restrictions are below a certain level. We also did not have time for additional features such as factoring your earliest travel date based on the quarantine and vaccination restrictions. This would also be a beneficial future development so that the user can plan their entire trip. Unfortunately we did not have time to create a front end interface for the project as outlined in section **1.4.2.1.3**, however this was a nice feature to have anyway, depending on the available time.

1.5 Testing and Evaluation

1.5.1 Testing Strategy

Throughout each stage of the project manual testing took place to ensure that the database could be connected to and the covid api data could be received and manipulated to return the fields we required. This took place on both a unit and functional level when the elements were brought together. During the testing process we did identify some regression issues whereby changes to some of the functions affected the ability to access data from the covid api, however these were solved as part of our final bug fixing. The main focus of our project plan for Week 4 was testing.

1.5.2 Functional and User testing

User input testing was carried out for each of the functions in the travel app to check what impact spelling errors, capital letters and punctuation would have on each of the variables. This was key as if the month or city for example is entered incorrectly then the rest of the functions will not return correctly as the variable will not be able to be matched against the fields in the database or covid api. To minimise frustrations for the user of inputting variables incorrectly (as they obviously don’t know in which format it should be entered), most inputs have been refactored to remove extra spacing, punctuation, numbers at the end of strings and capitalise the first letter of each word. This reduces the error rate of the inputs although obviously doesn’t account for misspellings. In the case of error through misspellings, exception handling has been put in place for most functions in combination with for and while loops to allow the user to re-input the required field. As a future development piece it could be built on so that only a certain number of attempts are permitted before the programme is exited. Exception handling was also implemented to handle cases where no personal items are added to the list (in which case only essential items are returned).

1.5.3 Unit testing

We aimed to write unit tests for all of our main functions. As our functions use both self-defined API endpoints and query the database, mocking was used as it enabled us

to test functions without calling the actual API or the database. This meant that tests would not be affected by external dependencies, and the results would be based solely on the code in each function. The patch decorator was used as this allows for the target, in this case the endpoint or database, to be patched with the new object, as defined by us.

There are further unit tests which could be written, however due time limitations it was not possible to write these tests. These include unit tests which test the main() function (e.g. functions are called in the correct order), that only 10 items could be added to the personal items list in the add_personal_items() function and tests to ensure the user could not add the same item to their personal list. Our tests focused on functions, however if the project was to be developed further, then unit tests for each class would also need to be written. Finally, if we had more time then more advanced testing of the database connection and api connections through mocking would be advantageous.

1.6 Conclusion

Overall we are very pleased with the outcomes of our project. We believe we have achieved what we set out to do, by creating a fully functioning travel app which incorporates the main code requirements. Our travel app solves a very relevant and current issue whereby travellers need to know the most up to date Covid-19 travel restrictions including testing, quarantine and mask wearing. We have also made the app a place where the user can plan their entire trip, by including average weather for the month of travel and both essential and personal packing list items. This is a scalable project with an array of possible areas for future development. Whilst it has been stressful at times, overall we have enjoyed putting our skills learnt throughout the Nanodegree to the test with this project. We feel we have worked well as a team and have been able to practice areas of coding that individually may not have originally been our strength or preferred topic.