Holiday Helper App Report

Created by Samin Nazarpouri, Maneh Seiranian
Magdalena Sosinska & Mary Ruth Bongon

Introduction

Aim and Objectives

Our aim is to help travellers find their way around in new places, discover the best activities and events, communicate with locals in their native language, keep track of finances by converting their costs into the currency they are familiar with and help them connect with other travellers.

In order to achieve our goals we will build a web app that will give a user an interactive map, easily find attractions and events in the area, use a currency exchange calculator, check for translations and the weather - all the essential information a traveller needs in one place!

Roadmap of the report

1. **Background**
2. **Specifications and Design**
3. **Implementation and Execution**
4. **Testing Approach**
5. **Conclusion**

Background

Since the end of strict Covid-19 lockdowns in most areas of the world, world tourism rates have increased by 172% between 2022 and 2021 and with the ongoing proliferation of low-cost airlines you can now travel from London to Barcelona for as low as £16![1] This has unlocked a new level of demand for travelling and resulted in the need to reduce the information asymmetry between a traveller and the experience offered by the destination country.

Within the sea of applications available for tourists, such as GMaps and Airbnb, we haven't come across an app that offers all the necessary travel information in one place and therefore decided to make an app which after some iterations can become the ultimate Holiday Helper.

Walkthrough of the app

A menu bar allows a user to choose between some options, including "Destinations" and "Contact" tabs. By navigating to the "Destinations" tab a user can see the destinations that our Holiday Helper offers to help with. For the first release of our app, we are only focusing on 4 cities: *Prague, London, Budapest and Barcelona*. We have chosen these cities based on our

[1] *International Tourism Back to 60% of Pre-Pandemic Levels in January-July 2022* (2022) *World Tourism Organization*. Available at: https://www.unwto.org/news/international-tourism-back-to-60-of-pre-pandemic-levels-in-january-july-2022#:~:text=According%20to%20the%20latest%20UNWTO,%25%20of%20pre%2Dpandemic%20levels (Accessed: November 4, 2022).

familiarity with them and time constraints. However, given our program design, discussed below, it would be easy to scale our app up to make it functional for almost any destination. In order to unlock most of the features of the app, a user has to sign up using their full name, unique email and password. An existing user can simply log into their account with their email and password.

After authenticating, users will be able to see their profile in the app, where they will be able to navigate to the same "Contact" and "About us" tabs as before, however after clicking on the "Destinations" tab they will be able to click on one of our focus destinations to get the essential city-specific information. The following features are available for each city:

- Map with pins on landmark locations
- Currency converter (user gets equivalent of local currency in their home currency)
- Current weather in the location
- A translator, allowing a user to translate some phrases into the local language
- A list of events taking place in the destination
- A list of the main attractions in the destination

Specifications and Design

Technical and non-technical requirements

During the initial planning stage of making Holiday Helper, we came up with a list of primary, secondary and tertiary user requirements based on our experience with using other apps and travel, which we summarised and arranged in a prioritised on a Trello board. To summarise, the following are the non-technical requirements of our app, complemented by the technical requirements which would help with releasing some of the non-technical requirements:

Non-technical Requirements	**Corresponding Technical Requirements**
Primary features (critical for the first release):	
Account creation/Signup page: The feature should be on the landing page for a user to sign up if they don't already have an account in order to unlock most of the features. The account creation requirements include name, email and a password. A user should get an error message if there is an account associated with the email. Passwords of minimum 8 characters should be chosen, otherwise be flashed with an error.	*Front-end:* A popup window enabled by Bootstrap, allowing a user to input their name, email and password *Back-end:* The flask and flask_login libraries were used in Python in order to ensure the back-end was receiving the user signup details input into the screen, doing the necessary checks (e.g. checking that the email doesn't already exist). These also allow provisioning of the directories for all the pages and connection to the frontend using render-template. *Database:* A MySQL database is connected to the back end allowing to check if an email already exists and store a new user's details in the users table.
Login page: The login option should require a user to input their email	*Front-end:* A popup window enabled by Bootstrap, allowing a user to input their login credentials

and correct password in order to authenticate onto the app. The user's email and password should be checked against the database and an error should be flashed if they are incorrect.	*Back-end:* The Python flask and flask_login libraries were used to ensure the back-end was receiving the user details from the app, doing necessary checks (e.g. that email and password are correct). *Database:* A user's email and password are checked against the MySQL database entries to make sure they exist and that the correct password is input for the respective email.
Profile page: After authenticating, users will be able to see their profile in the app, showing their name and email (in the next release an option will also be added to upload a user's profile picture)	*Front-end:* Bootstrap is used to show some basic information about a specific user. *Back-end:* The flask and flask_login libraries were used in Python in order to ensure that the pages shown to the user after authentication are only possible if a matching id is found for the user. *Database:* A MySQL database connection enables to check that the user id corresponding to the email is in the table. The authenticated version is available through the reconciliation with the id in the table.
List of destinations to choose from: Although the pre-authentication page also showed destinations, after authenticating a user is able to choose from our focus cities to travel to and get redirected to their respective pages	*Front-end:* Bootstrap is used to show our focus on 4 cities, including an image for each. A hyperlink for each of the cities allows a user to click on them. *Back-end:* The flask and flask_login libraries were used in order to allow redirection to city-specific directories and connect with the front-end. *Database:* A cities table is included in the MySQL database, which is used in the city-specific features.
Map of city with pins on its main attractions: A user is able to see a city-specific method with pins on the main attractions on a city-specific page	*Front-end:* the map of a city is included on a page, which uses Bootstrap to split the screen into sections for each of the app features, including the map. Folium was used to render the map. *Back-end:* Geocoding function of google maps api was used to retrieve the geographic location of the cities, and Places function to retrieve touristic information within a certain radius of a city.
<u>Secondary features (should have in first release):</u>	
Pictures of main attractions: A user is able to see a list of the main attraction in the city that they can visit at the bottom of the screen	*Front-end:* Bootstrap was used to render the images and some details of the tourist places *Back-end:* Places API of google maps api was used to retrieve the name, images, and address of attractions
Translator: A user is able to input a phrase in their desired language and get a live translation in the local language, so that they can use it to speak to locals if necessary.	*Front-end:* HTML was used to create and structure the front-end for the function and CSS for styling it and add details to its design. *Back-end:* The Google Translate API is used and connected to the backend in order to retrieve translation from any language to a target language. Google language detection is used in this

	function in order to recognize the input language. Therefore, the user is not required to choose their input language.
Currency Converter: A user chooses from the currencies of one of the target cities and converts it into a currency familiar to them (for the first release of the app we have Pounds, Hungarian Forint, Euro and Czech Koruna to convert into one another. More currencies will be added to the system in the future).	*Front-end:* HTML and CSS languages were used to create, structure and design a user-friendly front-end for this function *Back-end:* The Frankfurter API was used to get live data about current exchange rates, connected to the backend to retrieve the current rates and use them to convert from a chosen to a target currency.
<u>Tertiary features (could have in the first release)</u>	
Weather in chosen destination: A user can see current temperature in the chosen destination, together with a description and a changing image showing e.g. clouds for cloudy weather.	*Front-end:* Weather is displayed on a city page with other features *Back-end:* An OpenWeather API is used to retrieve live information about a specific city's weather *Database:* City coordinates (latitude & longitude) retrieved from the database and passed to API requests to get city results.
Local Events: A user is able to see some of the most popular events in the destination on the city page to make an informed decision on how to split their time	*Front-end:* Events are displayed on a city page with other features like map, weather, attractions, etc *Back-end:* An Event API (Serpapi-Google Events API) is used to retrieve live information about events happening in the near future in a specified location.

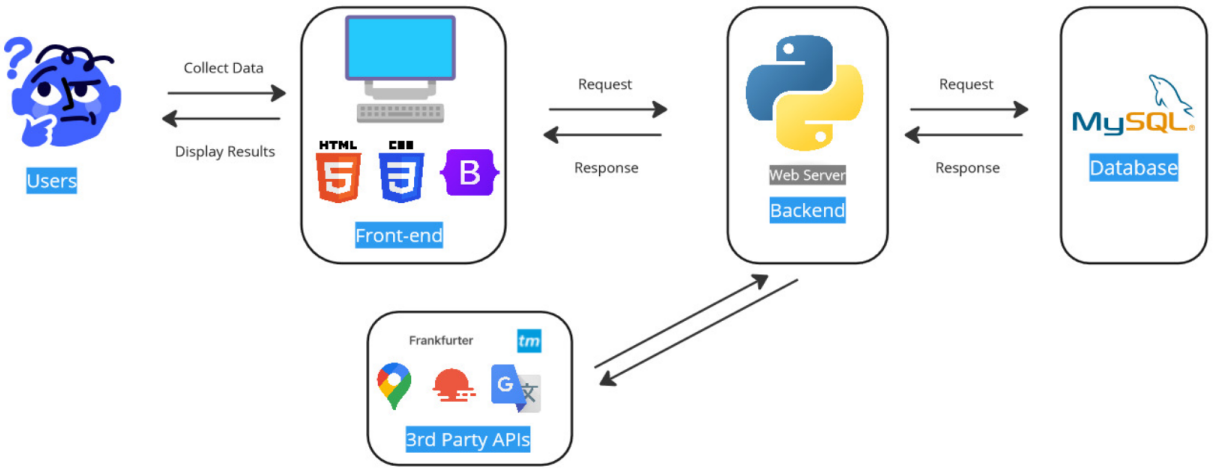
There are a number of features that we would like to implement in future sprints. Although we have already made the login, signup and profile options for users, this personalised approach will become very useful in future sprints, as we incorporate the following features:

- Ability to click going to an event and see anyone else who has chosen to do so
- Ability to message users by clicking on their picture (e.g. if I see a user interested in attending the same event as me and, I could message them to go with them)
- Once logged in a user should be prompted with a questionnaire, indicating interests and budget such that a personalised itinerary is created for them. A Shortest Path Graph could be used to incorporate activity distances in calculating the ideal plan
- The option to be redirected to Google Maps when choosing a location a user wanted to go to, such that they can search how to get there on Google Maps
- Make the app available for more destinations

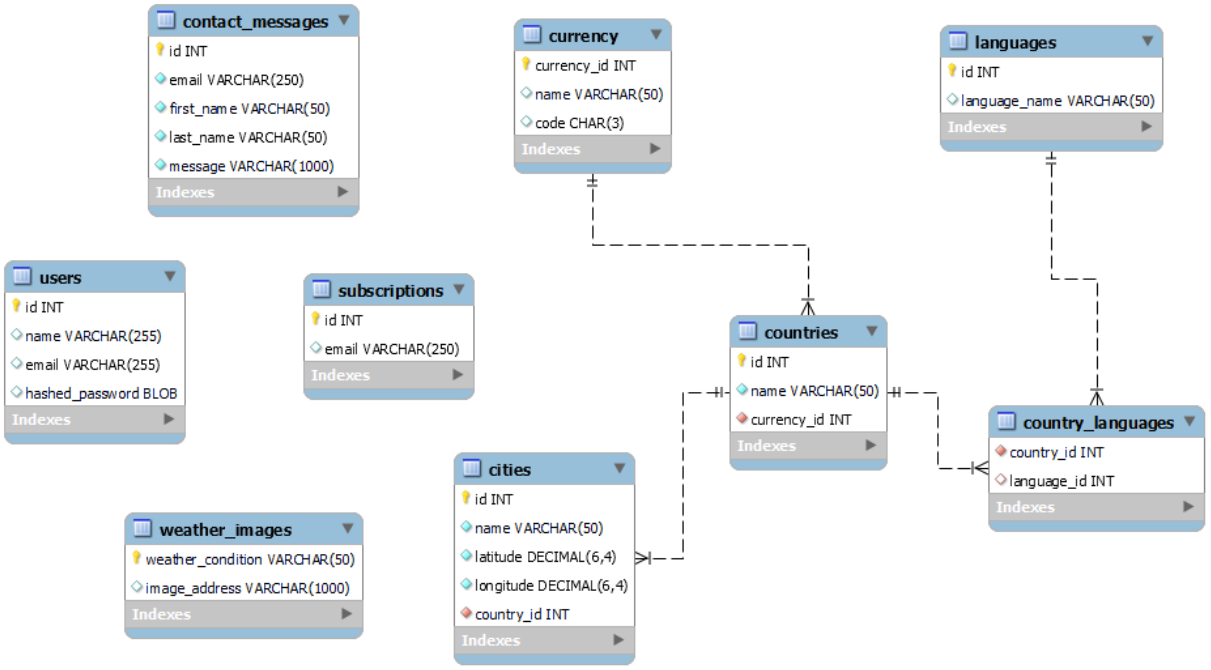
We would prioritise which features to implement based on feedback from users on what they find useful on the current version and what they would like to see the most in the future.

Design and Architecture

A summary of the design and architecture required to bring into fruition the non-technical and technical requirements is shown below:



We designed our database using MySQL based on the required features:



Implementation and Execution

Project development approach and use of the Agile methodology

The team used elements of both Agile and Waterfall methodology in our approach. Since each team member had different schedules (some of us had full time jobs / were full time students) daily standups were not possible, instead most of the work was done over weekends. We started off the planning stage by brainstorming ideas, which were then transformed into user stories and specific action points (see our actions Trello board) that were continuously iterated. Based

on our interests and strengths we divided up the requirements between us and after working on them for some time updated the backlog based on any observed difficulty and feasibility. Our team members had the following responsibilities:

- Magdalena Sosinska - GitHub organisation creation, Database setup, implementation, front-end and testing of Weather feature, Events feature
- Maneh Seiranian - Project management (Slack channel setup, meeting arrangements), implementation and testing of Login, Signup and Profile pages
- Mary Ruth Bongon - Designing the app on Figma, writing the front-end in HTML and CSS, implementation and testing and front-end of Maps and Attractions features
- Samin Nazarpouri - Implementation, front-end and testing of Currency converter and Translator features

We had stand-up-like frequent calls, where we discussed our achievements, challenges and set specific objectives by the next call. We also had ad-hoc calls for problem solving.

We used iteration throughout our project, where we would revisit the design decisions that had been made after starting to code, for example our translation function was initially constit of a class with various methods(there was a unique function for every chosen language to be converted to other 3 languages), however, while designing the front-end for the function, we had a chance to improve the code. Therefore, one function was written which could cover the translation for all the chosen languages.

We refactored our code throughout the process. For example after building a working version of our app, we went back and implemented OOP in our Weather and Events code, by introducing classes and hence making the code more sustainable for future iterations and use if need be.

We set up a GitHub organisation for uploading code, where we created branches for individual team members and would upload any code to our personal branch, have it reviewed by at least one other person and then merge it to the main branch.

Languages, Tools and Libraries

Pycharm CE was used as our IDE for our code and the following tools and libraries listed below were installed to extend the capabilities of our product:
Languages: *Python, MySql, HTML, CSS (Bootstrap framework)*
Tools: *Trello, Miro, Figma, GitHub*
Libraries: *Mysql-connector-python, Bcrypt, Flask, Flask-login, Googlemaps, Folium, Requests, Google Cloud Translation API wrapper*

Implementation Limitations and Challenges

In order to implement the features noted in the Technical and non-technical requirements section above we stumbled upon and overcame some challenges, which included:

1) Time management - some of our teammates are full time employees / students, so we had to be efficient with our time. The work we did over the weekend and late evenings made the app a reality.
2) Front-end - since none of our team members had extensive front-end experience, some of the features, especially the Translation and Currency Converter were difficult to implement dynamically. We initially thought that Javascript was our only option and not only that was beyond our knowledge and capabilities but, also because all of our front-end was done by raw HTML and CSS, we wanted to use the same languages for these features too. We overcame this challenge by spending a lot of time researching and learning HTML and CSS specifically for the features that we required for our design.
3) Private API keys - since this was our first use of private API keys and doing a project on GitHub, we initially stored our keys on our public page, which led to red flag alerts from our API owners. We then overcame this by putting all the private keys in a private configuration document, which we shared internally and did not push to GitHub

Although any password input by users is only stored in the database after getting hashed and salted, going forward the password complexity requirements could be increased for users in order to improve security, such as requiring a combination of numeric characters and letters and not allowing users to use common passwords.

Through iteration we came up with requirements beyond our initial list of requiremnets, such as the Contact and Subscription page, which we thereafter implemented so that users could report bugs and stay tuned for future updates.

Testing and Evaluation

Since we were mostly following an Agile development methodology we started our testing procedures early on in the process. Since each one of us were close to the code that we decided to each test the respective code that we have written.

Before moving a change into the virtual repository we would run the code to make sure it is working as expected on the app. We carried out functional testing using Unit Tests. Where possible we did negative and positive tests to cover different scenarios and stored them in a “Tests” directory. We asked some of our friends to do user testing, who clicked and navigated through the app. We then implemented some of the feedback they gave. The following is a list of feedback that we got towards the end of the project and would include in our future releases:

1. Left top corner logo looks clickable but it does nothing.

2. Not Logged in. On Destinations, I clicked on a city picture and it took me to the main screen without any message.
3. On Contact, Social Media icons don't work.
4. When logged in, it would be useful to see the profile name
5. Sign in should be named Log in
6. When you sign in, it takes you to the profile screen, but you are not able to do anything there.
7. When signing out, there should be a confirmation message
8. When filling in the Sign up form and inputting a password that is less than 8 characters, all fields are cleared
9. City page – map does not fill entire box and strange description for Attractions
10. When successfully created a new profile, I need to login in again

Testing limitations and challenges

We used a few APIs where output is a live changing data, to test our code we had to mock the API output. It was quite tricky for us as we are still beginners, but with an extended research we managed to implement that.

In order to test the user management file, dependent on database connection, we explored multiple options. Due to time limitations, instead of creating a mock database, we added a test user into it and wrote code that removes the user each time before running the tests. A challenge, which we overcame was retrieving hashed and salted passwords so that they can be compared with the input password in tests. We did this by rerunning select commands in tests.

Conclusion

We find that overall our project was a success, as we were able to meet our objective of provisioning an effective holiday helper app for travellers, which has features such as an interactive map, attractions, events, weather, translator and currency converter in one place. Implementing these many features, given our time constraints, was possible through our effective teamwork, where we played on our strengths and helped each other out. By working on some of the features we had planned for future sprints, such as a personalised itinerary planner and events match-up feature, we will be able to offer a truly unique and useful app.