GoPacking

INTRODUCTION

GoPacking is a web application that uses user-provided data on upcoming holidays and external weather forecasts to provide users with custom travel packing lists.

Aims

1. To accept user input on destination for holiday, length of stay, type of activity and type of clothing.
2. To use user input on destination and dates of travel to gather relevant weather information from Visual Crossing API.
3. To provide the user with a customised packing list which factors in the length of the trip, the weather at the final destination, the type of activity the user will be doing as well as the gender/style of their clothes.

Contents

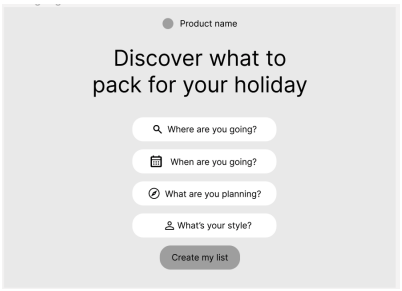
BACKGROUND

The app takes input from the user on a travel destination, dates, activities and style to return the list. Using the date and location input, the app retrieves via API call the weather forecast for the specified time. The activities input relates to certain types of holiday that require speciality clothing, for example, a beach trip would require a towel and a swimsuit and a ski trip would require ski gloves. The ‘style’ input allows the user to input a gender for their clothing or to choose not to have a gendered packing list. Using this input the app follows the below rules:

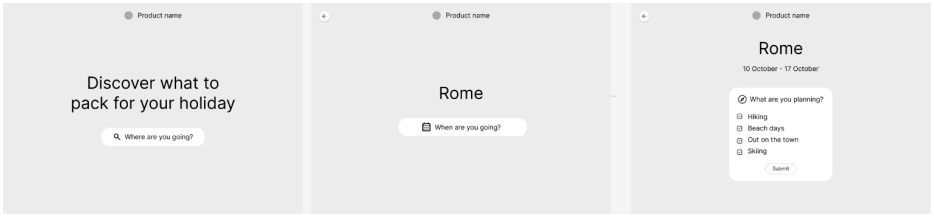
- Items of clothing have temperature ranges stored in the database which specify how low and how high the temperature would be for someone to pick that item of clothing. For example, a ‘light jumper’ has a temperature range of 10 - 20 degrees celsius.
    - The app uses the temperature in the forecast generated by the user input to filter out any clothes outside of the temperature range.

- Items of clothing have other properties: activity and style. The app uses user input to filter out items of clothing based on these principles too.
    - The app can also use a forecast of rain to recommend items with a ‘rain’ property, such as an umbrella, ‘snow’ for a hat or ‘clear’ that will recommend for example sunglasses.
- Items of clothing have ‘quantity per week’ values, which specify how many of something a user would need per week. For example, socks have a ‘quantity-per-week’ value of 7, because the user will change socks every day.
    - The app calculates the length of stay from the calendar input and then uses that information to calculate the quantity needed for each item.
- This information is then returned to the user in the form of a list with checkboxes, so that they can check off each item as it is packed.
- There is an option to email the list to the user (we are still working on this feature!)

SPECIFICATIONS AND DESIGN



Packing for a holiday can be a stressful task, so we have tried to make our design as simple as possible, as we do not want to overload the user with too many steps or too much information. There are 4 user inputs on our landing page: city, date, activity, style. After some experiments (illustrated right), we felt the clearest way to display these was in a single line. This is a common style used by some major travel/booking websites.



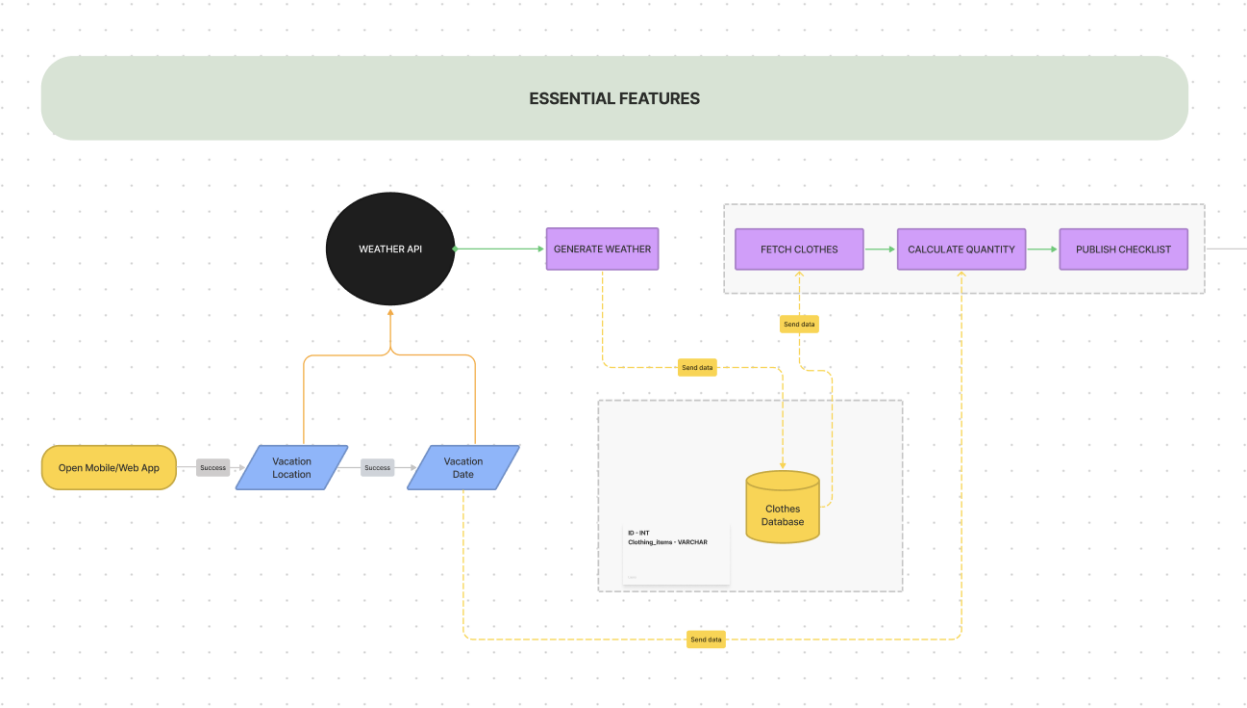
Competitor research also encouraged us to use travel photography in our design, as we had seen it successfully used by companies like airbnb and SkyScanner.

We decided upon the name ‘GoPacking’ as we felt this highlights the main purpose of our website (creating a packing list for travel). We incorporated a suitcase into our logo (to strengthen the message portrayed by the name), and showed this moving around a globe.

Our final design, illustrated right, earlier iterations illustrated above.

Technical Requirements

The app uses a MySQL database to hold all information about clothes. It then uses Flask and helper functions to calculate what types of clothes are needed. The front end is a React Web App which uses external API calls and calls to the backend to generate the final pages.



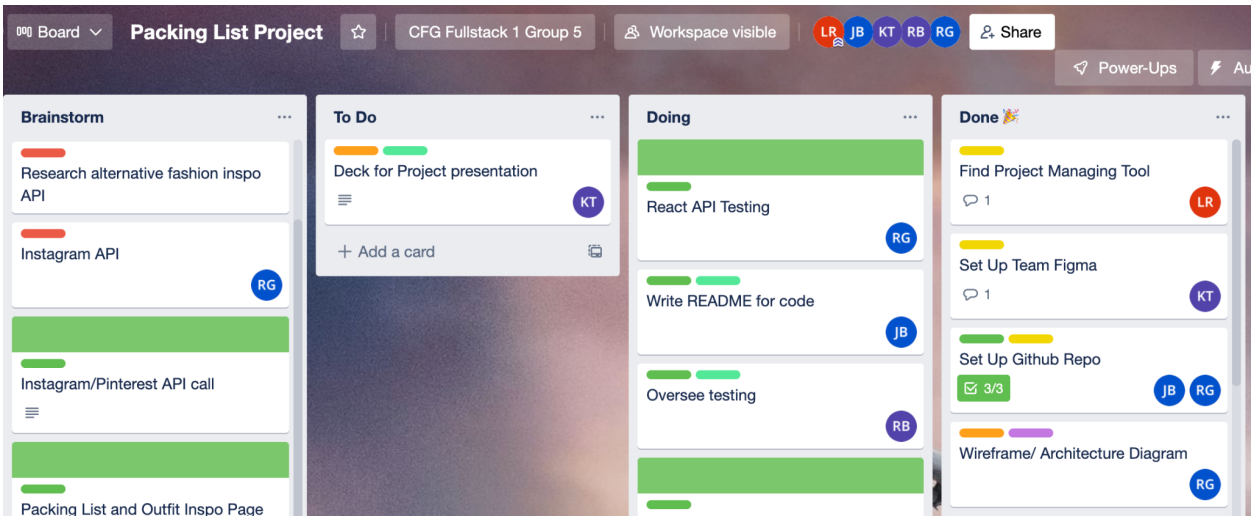
A diagram of our technical requirements

IMPLEMENTATION AND EXECUTION

Project management

Taking inspiration from Agile methods that we have been learning about in class, we decided to complete work in week-long stretches (inspired by ‘sprints’). Meeting formally weekly, we would recap tasks that had been done and discuss what worked and what we should either let go of or improve. We would plan and distribute future work for the week and also discuss general blockers which often also involved sharing learning strategies and difficulties we were having on the course more broadly. We would try to iterate on previous stretches of work, implementing changes and learnings from these meetings.

Our implementation strategy was planned using a colour-coded Trello board updated weekly.



A view of our Trello Board showing how team members are assigned different tasks

Teamwork

Through our weekly meetings and constant communication through slack, our team was able to help each other on difficult points, but the key responsibilities for each team member are below.

Backend

Rachel B took on all database-related tasks: designing the database structure, creating tables, writing queries and stored procedures.

Joanna B took care of our Python logic, first building the Flask app based on a static json file representing mock content coming from the frontend, connecting it with database, Flask endpoints and displaying results in a template React page. Then she worked on connecting our React app to our backend with the real data.

Frontend

Rachel G researched and wrote the React code for our two external APIs, turning our user input into API calls and then feeding that back into dynamic React components and to Joanna to incorporate into the backend. She has also been working on the front end more generally, using React libraries to build inputs.

Kavita T led on design, creating our wireframes and high-resolution prototypes and incorporating our learning on design principles and our brainstorms on user journeys into a seamless and intuitive app. She then helped to build that app in React, focusing on component-building and styling.

Laura R began with more technical design, planning the technical stack for the app and planning the team approach and roadmap, and then leading weekly meetings to ensure the team covered the necessary tasks and could adapt or change when blockers arose. She has been working with Kavita on the frontend to build some components and deliver styling.

Tools and libraries

In addition to React, a JavaScript library, we used the below tools and libraries to create our app:

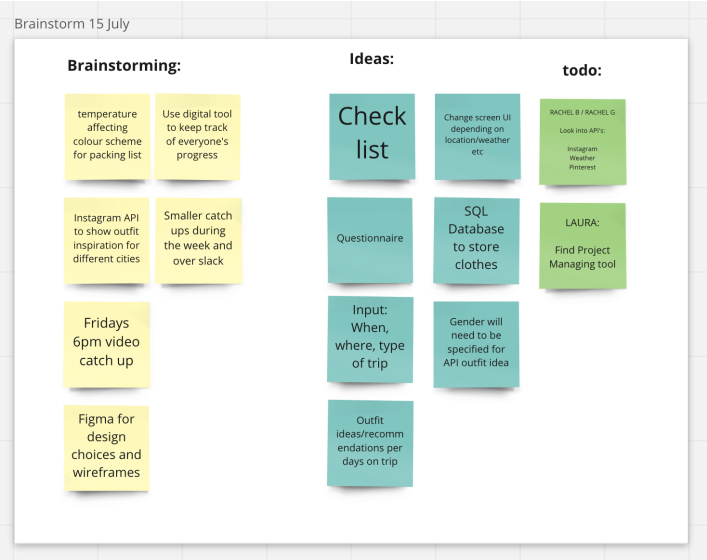
- Flask - for Python backend
- React-Select - to create some of our input fields
- AsyncPaginate - to create an interactive auto-compelte input field
- React-Date-Picker - to facilitate our date input
- Bootstrap - for some form styling
- Flask-cors
- PyMySQL
- Pytest
- Jest and React-Testing-Library
- Material icons

Successes

- Accessing two external APIs which can gather a great deal of information about different locations and their weather forecasts.
- Combining our tables of clothing with certain logic relating to temperatures, activities and length of stay to return well-thought-out, clear and useful packing lists.
- Good, functional and attractive design.

Challenges

- Ambitious scope - the program itself is quite complicated so this combined with designing a system that was considerably outside of our knowledge area at the beginning of the project has led to some challenging moments.
- Testing - React testing has been complicated, and only becomes more complicated as additional tools are added to the app.
- Time-management - some team members have been completing the project alongside full time jobs or family commitments as well as completing the CFG program, assessments and homeworks.



A Miro board from our first brainstorming session.

<u>Decisions to change/ Challenges that could not be overcome</u>

- Inspiration pages - our initial plan involved using an Instagram or Pinterest API to provide 'fashion inspo' according to the user's results. After much research and a few attempts, we found that the reliance on Instagram and Pinterest accounts would make it too unwieldy for users. Because of this, we had to abandon the feature.

Our user testing showed some bugs that we had not caught earlier on. As this was so close to the final deadline, we were unable to solve them but know we would need to work on them if we continued on with the project. These are:

- If the user selects the city, clicks out of the field, and then clicks back on the city again, the city entered stays in the form field and it's not possible to delete it (have to be overwritten by starting to type again).
- If only one date field is selected, the dates on the Packing page display as two, but the request going to the backend has only one date, making for a weird clothes calculation

TESTING AND EVALUATION

Our overall testing strategy was to test each part of the code individually and as we went along. We would then run a series of unit tests and functional user tests to catch any bugs that need fixing. As we went along with the project it became evident that fixes would need to be made when the code evolved or changed, for example, styling would become an issue at certain points when code had been changed. We overcame these minor changes by communicating daily in our slack group chat and also our weekly zoom call.

<u>Database testing</u>

Since system testing on the database is not within the scope of our technical ability we had to come up with a way of evaluating the database. The basic principle of evaluating the database is checking that the output resulting from a test case is compared to the expected result. We can achieve this by running select statement queries on the stored procedure, the data and also the tables themselves. The following checks were made:

- Tables- logical, complete, named correctly, primary keys match, contains unique information and no unnecessary columns.
- Columns- correct data type, character size, named correctly, constraints, relationships and null values where applicable.
- Stored procedure – correct logic and outputs, is it fast?, does it work?

Software unit testing

We took into consideration the following order for unit tests for the software we used- React, Python and API's.

- Identifying a testing framework
- Identifying chunks of code which affect the behaviour of the application
- Making test case and scripts
- Testing the code.

User testing

We gave family members and friends access to the application to test the functionality of the user interface, API's, Database, client/server communication and based the tests on a simple form to ensure consistency in testing.

- Mainline functions: Testing the main functions of the application
- Basic Usability: checking to confirm whether a user can freely navigate through the pages without any difficulties.
- Accessibility: Checks the accessibility of the system for the user
- Error Conditions: Usage of testing techniques to check for error conditions. checking whether suitable error messages are displayed.

CONCLUSION

The web app that we have created achieves all of our key aims, to use user input to create weather-appropriate, customised packing lists. This means that it can provide a useful service to our users – making the trip-preparation process less stressful by automating decisions to be made around packing. We can attribute the success of the app to some of the most ambitious elements of the project, the use of external APIs and the program written to provide the app with the final packing list. Some of the greatest challenges have emerged from these parts too, but we have found that only some of our 'nice to have' features had to be cut in order for us to successfully complete the project, with the core functions all completed as planned. There are still some areas in which we could develop the project with time allowing; we could find a solution to our 'outfit inspiration' feature which did not require the user to log into external accounts, and we could fine-tune a few bugs that arose during user testing. But all in all, GoPacking is a well-designed, functional and useful product brought to life through cooperation, teamwork, and a lot of dedicated research time.