Intuitive Itinerary Planner

Development of a Progressive Web App - Itinerary Planner for Travel across Ireland

Agne Postnikova

N00211108

Supervisor: Stefan Paz Berrios

Second Reader: John Montayne

Year 4 2024/25

DL836 BSc (Hons) in Creative Computing

Abstract

This thesis explores the development of "Buckle Up", a Progressive Web Application (PWA) designed to enhance user experience in itinerary planning for travel across Ireland. The primary aim was to create an intuitive travel planning tool that addresses pain points identified in similar applications, including overcomplicated interfaces and long processes. The application was developed using Next.js, integrated with Google Maps API and Firebase, despite a mid-project pivot from the original React Native framework due to technical barriers. SCRUM methodologies were used during implementation, which consisted of five sprints, focusing on core functionalities such as itinerary creation with drag-and-drop capabilities, map integration, and intelligent travel assistance. Two types of testing were conducted: functional testing using Jest and user testing with participants of varied technical backgrounds, which revealed mostly positive usability outcomes. The project demonstrates the importance of adaptability and feasibility during development while emphasising how user-centric design can create an effective application, even when faced with significant technical challenges. Future development opportunities include expanding beyond Ireland, adding booking services, and enhancing offline capabilities.

Acknowledgements

I want to share my sincerest gratitude to my primary supervisor, Stefan Paz Berrios, for not only providing me with weekly feedback and inspiration, but also giving me words of encouragement at times when working on this project became tough. My gratitude also goes to John Montayne, my second supervisor, who pivoted my project in the right direction when I was stuck. Last but certainly not least, I want to thank Mohammed Cherbatji, who took time out of his busy day to attempt to solve some technical difficulties. A lot of blood, sweat and tears went into this project, I want to thank my family and friends for taking care of me and motivating me throughout.

Declaration of Authorship

The incorporation of material without formal and proper acknowledgement (even with no deliberate intent to cheat) can constitute plagiarism.

If you have received significant help with a solution from one or more colleagues, you should document this in your submitted work and if you have any doubt as to what level of discussion/collaboration is acceptable, you should consult your lecturer or the Course Director.

WARNING: Take care when discarding program listings lest they be copied by someone else, which may well bring you under suspicion. Do not to leave copies of your own files on a hard disk where they can be accessed by other. Be aware that removable media, used to transfer work, may also be removed and/or copied by others if left unattended.

Plagiarism is considered to be an act of fraudulence and an offence against Institute discipline.

Alleged plagiarism will be investigated and dealt with appropriately by the Institute. Please refer to the Institute Handbook for further details of penalties.

The following is an extract from the B.Sc. in Creative Computing (Hons) course handbook. Please read carefully and sign the declaration below

Collusion may be defined as more than one person working on an individual assessment. This would include jointly developed solutions as well as one individual giving a solution to another who then makes some changes and hands it up as their own work.

DECLARATION:

I am aware of the Institute's policy on plagiarism and certify that this thesis is my own work.

Student: Agne Postnikova

Signed: 

Failure to complete and submit this form may lead to an investigation into your work.

Table of Contents

Introduction

Buckle Up is an itinerary planner for travel across Ireland, a Progressive Web App. This application was developed with a user-centric design to provide the best possible User Experience (UX) within the project's scope.

This project aims to deliver an application that tackles UX problems, such as overwhelming and confusing User Interfaces. To counter those issues, a simple and effective application has been developed, which allows users to create an itinerary, save and export it, connect it to Google Maps and chat with an intelligent travel assistant.

A wide range of tools and technologies were used to create this application, some new and unfamiliar. This provided the opportunity to expand knowledge and experience, which includes Next.js, the React framework that was used to develop this application, and Google services to implement functionality.

This project offered an opportunity to enhance knowledge and experience, particularly with Next.js, the React framework utilised in developing this application, and Google services for implementing functionality.

The SCRUM methodology was used to manage this project, which consisted of five sprints during the implementation phase. The primary tools used to manage this project were Miro and GitHub. Research and brainstorm sessions were documented in Miro, while the technical development was tracked in a GitHub repository. Weekly meetings (online or in person) were held to review the work and receive feedback from the supervisor, which is a part of the project management process.

Functional and non-functional requirements for the current project are established by reviewing similar applications.

This project includes two design processes: a program design, which looks at the technologies being used, and a User Interface Design, which is first created in Figma by making wireframes, and then implemented into the application.

Implementation was completed during the five sprints. Google services such as Maps and an AI model were integrated. When creating an itinerary, the user can select a location/destination from a dropdown list provided by the Google Places API. This API allows developers to display the places on a map. Additionally, functionality such as downloading and opening the itinerary in Google Maps will be implemented.

Two types of testing will be conducted during this project: functional testing using Jest, a JavaScript testing framework, and user testing with participants. The feedback from these tests will be reviewed and implemented. The participants in the user tests have different technological backgrounds to reduce bias.

Research

Introduction

This section researches the world of user experience (UX) and accessibility, exploring why it is crucial to create user-friendly applications. The meaning of UX will be explained as it often gets used interchangeably with UI (user interface) (Kaplan, 2024), additionally exploring the key principles that guide effective design and examining accessibility. The research shows that while web applications often have better accessibility features than mobile apps, developers can bridge this gap with the right tools and awareness. The original plan was to use React Native for this project because of its cross-platform advantages and accessibility support, but technical barriers (detailed in Appendix 1) led to a switch to Next.js. Throughout this exploration, one theme remains constant: creating technology that is user-friendly and accessible.

User experience and accessibility

What is user experience (UX)?

“User Experience” (UX) is a user's entire experience with a product or service. The spotlight is on the users, so when they have a problem, the UX designers are there to create a product/service to overcome it. The term UX often gets misused; it gets used interchangeably with UI (user Interface), which is incorrect. Norman and Nielsen (1998) explain that UX is about everything the user experiences - before, during and after a product or service. Every UX design process has personas, empathy maps, user journey maps, scenarios, and storyboards. All of these steps are crucial to relate to the users, empathise with them, and understand them since the product is for them. UX is user-centric; every single thing is always related to the user, how they feel, how they would respond, and what they need. It is about usability, accessibility, credibility, satisfaction and usefulness.

Kaplan (2024) says UI is not UX; while they work hand in hand, and they do overlap, they should not be used interchangeably. UI is about the product itself, how it looks and feels, and whether it is easy to learn and use. Don Norman and Jakob Nielsen (the founders of the Nielsen Norman Group[1]) explain the difference as:

“Consider a website with movie reviews. Even if the UI for finding a film is perfect, the UX will be poor for a user who wants information about a small independent release if the underlying database only contains movies from the major studios.”

[1] https://www.nngroup.com/

When it comes to designing and building an application, be it an app or website, it will always be about the experience of a user. The user is the centre of that project. Understanding why the user is so important is key to building a user-friendly application. No app or website will be enjoyable if the user is struggling to complete a task. This is why the knowledge of UX is crucial when it comes to building and designing applications.

What are the principles and concepts of UX?

Viewing “user experience” from an outside perspective is a huge topic. It explores everything the user experiences before, during, and after any interaction with a product or service. It includes feelings, motivations, and expectations. Due to this, Allam and Dahlan (2013) explain why it is important to distinguish between usability and user experience. Usability is the interface the user is interacting with, the effectiveness of the product, and whether it satisfies their needs. Due to the overlap in what UX and usability explore, it often gets misused. There are many concepts and principles in UX, one of them being the ‘User Experience Evaluation Methods’ (UXEMs). The purpose of UXEMs is to support the design process when developing a product, from the very beginning to the very end. This ensures that the developer reflects their work on the end-user and that the product meets the UX target goals. There are three main types of UXEMs: qualitative, quantitative, and comparative. Qualitative methods let researchers learn about users’ behaviours, motivations, and emotions through interviews and usability testing. Quantitative methods allow researchers to gather numerical data through surveys. Comparative methods aim to test alternative designs to determine which one works better, such as A/B testing. In the development phase of UX, the most important decisions are made in the early stages, where the users' experience will be most affected. During this stage, it is important to sketch ideas (low-fidelity prototypes) to find gaps in the design and test throughout the process to maintain good UX and usability. The feedback provided is always integrated into the design process to align with the users’ needs.

In UX, evaluation methods are used to validate, emphasise and highlight the user experience, because based on that, the product concept will be created. Design methods, on the other hand, are the imaginational and inspirational methods where designers generate new ideas rather than analysing existing products, according to Vermeeren (2010). When designing a product (or interface), things like storyboards, user journey maps, and scenarios guide the designers in the right direction. Due to the dynamic nature of UX, it is crucial to achieve the best possible outcome, and this means that all input from users is necessary. Designers should iterate their prototypes based on user feedback.

The user is always at the centre of any product design and development. Without a positive user experience, the product or service will not perform well. Thus, Quaresma, Soares and Correia (2022) explain why applying the concepts to gain the best possible user experience is critical. The main reason to use a user-centric design system is to increase the satisfaction of the user; therefore, it fosters customer loyalty and keeps them coming back, which is the goal for products and services.

Poor usability often leads to frustration, which is why it is necessary to use usability metrics early on in the design stage. By identifying pain points, it reduces the risk of a hard-to-use application. This enforces a user-friendly and intuitive interface, leading to user satisfaction. Getting feedback from users during a test is helpful, as it moves the project forward with fewer design issues to be fixed later in the development process.

Social inclusion is important in products and services, so accessibility is crucial. Merritt and Zhao (2021) state that accessibility prompts social inclusion and helps the organisation. Having this in the product will encourage users with disabilities to return and increase satisfaction. This also makes them feel included. User-centred design is emphasised throughout every stage of the development. This allows the creation of an intuitive and engaging experience for all users, taking their feedback and implementing the changes to avoid poor user experience later.

Why is following principles important when it comes to designing for a good user experience? When a final product is released, the overall goal is for the user to be satisfied. Steps like researching the users, creating realistic personas around the product, user journey maps, prototyping and so much more are essential in the product design phases. Taking into consideration every aspect of the user, their age, background, and education, everything plays into how they will use and interact with the app. Everyone is different, and that is the beauty and challenge of designing a diverse application.

Accessibility in User Experience

Petrie and Bevan (2009) explain accessibility as the design of products, devices, services, or environments for people with disabilities. Given how fast technology is developing in the current world, it is a part of our everyday lives; thus, it is important to make technology accessible to everyone, including those with different capabilities.

Mobile applications are very important in this day and age. Almost everyone uses mobile applications in their day-to-day lives, which means it is crucial to provide a good user experience. In some cases, it is hard to deliver a good UX if the application is complex. Mahmouda (2021) states that many factors influence usability, such as social and cultural differences, user characteristics and context of use; all five variables are shown in Figure 1. All this also adds to the complexity of UX; to work around this, the researchers from Imam Abdulrahman Bin Faisal University conduct interviews with users to collect data on their needs and interactions.

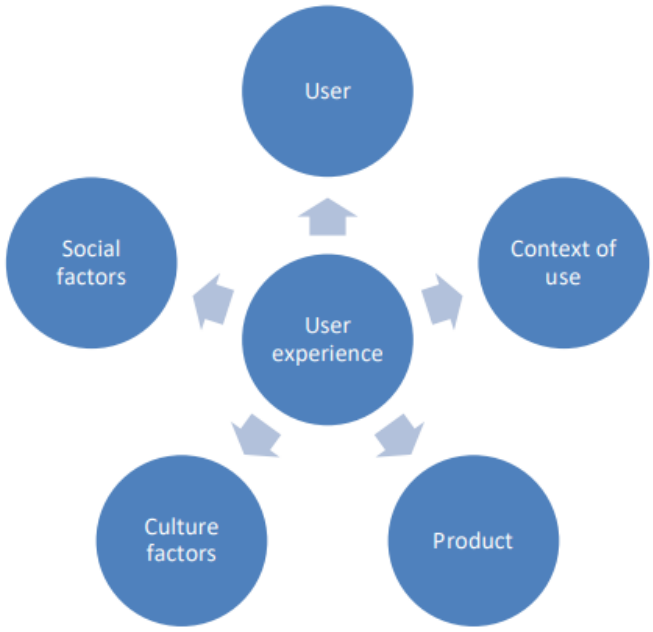


Figure 1 Five Variables of User Experience

According to Ballantyne, Jha, and Jacobsen (2018), most accessibility research is focused on web applications. Thus, there needs to be a specific heuristic evaluation for mobile applications specifically. The lack of attention to accessibility in mobile applications compared to web applications is hindering users with disabilities.

Compared to mobile applications, web applications have a lot more accessibility features, some more than others, but it is a lot more common for web apps to have these features. Mobile applications have a lot fewer accessibility features. This is why it is important to add as many accessibility features as possible. Choosing a programming language (that can provide these features) is important, as this will improve and speed up the development process.

Mobile/Native Applications

Native Applications

Flutter and React Native are open-source, cross-platform SDKs (software development kits). They perform well with a user-friendly development experience. Facebook developed React Native, which is based on the React framework. According to Wu (2018), Google was inspired by React Native and released Flutter for Android and iOS development. Flutter runs on the Dart programming language, while React Native runs on JavaScript.

React Native is praised for the efficiency and speed at which developers can work. According to Gill (2018), the average development time for an application is reduced by 33% by using React Native instead of building a mobile application separately for iOS and Android. React Native is a good choice for developing a cross-platform application.

The ability to create reusable components is encouraged because it also speeds up the development process. Components are files located in a separate location in the development environment; developers can call each component to use the same block of code in different locations throughout the project. This increases efficiency and makes the development process easier and more enjoyable. React Native is also a cost-effective choice for personal projects or small companies. It is an open-source framework which means anyone can use it and build applications on iOS, Android and the Web all on one codebase.

After comparing React Native and Flutter, the optimal choice is React Native. It has a much bigger community since it has been around for longer than Flutter. React Native is also more commonly used than Flutter, and it is based on the React framework. Since it has been around for a longer time, any problems makes it easier to research since the community is bigger. Compared to Flutter, React Native is also the stable option since it is being maintained more. Flutter is newer, and there are less stable versions of it.

Hansson and Vidhall (2016) conducted their research on the usability of React Native for developers. They concluded that React Native provides a good user experience and is recommended for developers. The major finding in this paper is that 75% of the code is the same for iOS and Android. This not only means that the time for development is highly decreased but also that in the future, maintaining the application will be much easier since the codebase is mostly the same across the platform.

Accessibility in React Native

According to Niemelä (2022), the main requirements for accessibility in mobile applications in general are: compliance with the EU directives, following the Web Content Accessibility Guidelines (WCAG), user-centric design and assistive technology compatibility. If everyone followed these requirements, or at least the EU directives, most apps would be accessible to most people with disabilities. In their thesis, Niemelä (2022) also states that at the time there was still a gap in the development of accessibility in applications, and these needed to be addressed. Another issue discussed is the lack of understanding and training among developers about accessibility. This is a huge reason why mobile apps lack accessibility features; this highlights a need for better training about accessibility.

React Native can provide accessibility through its properties, actions and APIs. By default, it has elements for accessibility, such as the “accessible” property that can be set to “true” or “false”, which is shown in Figure 2. React Native allows for the children elements to also use that property, which makes one accessible element. The TouchableOpacity has all the accessibility features; everything inside it also inherits these features.

```
<View style={styles.container}>
  <Text style={styles.header} accessibilityRole={"header"}>
    Testing accessibility properties
  </Text>
  <TouchableOpacity
    accessible={true}
    focusable={true}
    accessibilityLabel="Go back"
    accessibilityHint="Navigates to the previous screen"
    accessibilityRole={"button"}
    onPress={() => Alert.alert("Hello!")}
  >
    <View style={styles.button}>
      <Text>Back</Text>
    </View>
  </TouchableOpacity>
```

Figure 2 React Native accessibility properties

React Native also provides APIs (Application Programming Interfaces) for accessibility features; it makes it a lot easier to develop accessible applications for both iOS and Android in React Native. It provides support for assistive technologies like screen readers and physical keyboard navigation.

There is a good range of accessibility features in React Native, which makes it a good choice for developing an application. It is also cross-platform, which makes it much faster to develop for Android, iOS and the web. As stated before, React Native has a large community, which means there is a high probability that someone has already worked on accessibility features and it will be possible to research solutions.

Conclusion

The possibility to make more diverse applications is there, but a lot more effort needs to be put in. In this paper, it has been discussed how developers need to be trained about accessibility since it has been reported that developers know very little about accessibility and why it is important. Why would they spend extra time on accessibility features when they do not know why it's so important? React Native is a good choice for a start-up project, as it is open source, and the community is very big, which means there are a lot of resources out there for developers to make use of. When combining the understanding of user experience, accessibility and the availability of accessibility features when it comes to application development, it is then possible to create a diverse application that most people can use without difficulties but, most importantly, enjoy using.

In this fast-paced world where technology is developing faster than we can even imagine, it is so important to not forget about the minority that have disabilities/impairments. Technology is unavoidable nowadays, so why not make that same technology accessible to everyone?

Appendix 1 explains why these technologies were not used in this project. The research was done based on the fact that this project was going to be built in React Native. Due to multiple issues, such as versioning and deployment issues, the React Native project was not possible; a tough decision was made to switch to a new and unfamiliar React Framework: Next.js.

Requirements

Introduction

By analysing similar applications, the main functionality of the application will be researched. Functional requirements are necessary; they are the bare bones of how and what the application should be able to do. Non-functional requirements, on the other hand, are not necessary. Based on the time constraints of the project, it will be decided if those features are going to make it into the application. Additionally, personas have been created, which allows for a closer and more focused approach to designing and developing the application since there is a clear end goal, which is the user's problem being solved.

Requirements gathering

In order to be able to gather all the requirements for an application, it is important to know what's out there. Running a comparative analysis on similar applications is a way to gather functional (and non-functional) requirements. Additionally, creating proto-personas (a proto-persona is a fictional character that represents a real-life scenario) is another crucial aspect of collecting functional requirements; it provides a goal that will fix the persona's problem, and it also provides motivation for the developer(s).

Similar applications

For an itinerary planner application, the two similar applications that are being analysed are Wanderlog and Stippl. Both applications allow users to create an itinerary and manage them, but each one has different strengths. Wanderlog focuses more on the planning of a trip and is very map-centric, while Stippl is more about the experiences since it has a social approach to it. It offers a lot of other features such as collaboration, trip sharing, memory preservation and much more. The following sections will outline the advantages and disadvantages of both.

Wanderlog

Figure 3 shows the screenshots of Wanderlog[2]. What this application provides to users is an itinerary planner where they can create their itineraries, track information such as bookings (hotel, car rental, flights), upload related documents, track travel expenses and browse activities. They have a few categories available, such as restaurants, activities, cafes, photo spots and more. Based on the created trip, the app provides personalised recommendations.

[2] https://www.behance.net/gallery/193689559/WanderLog?tracking_source=search_projects|wanderlog&l=1

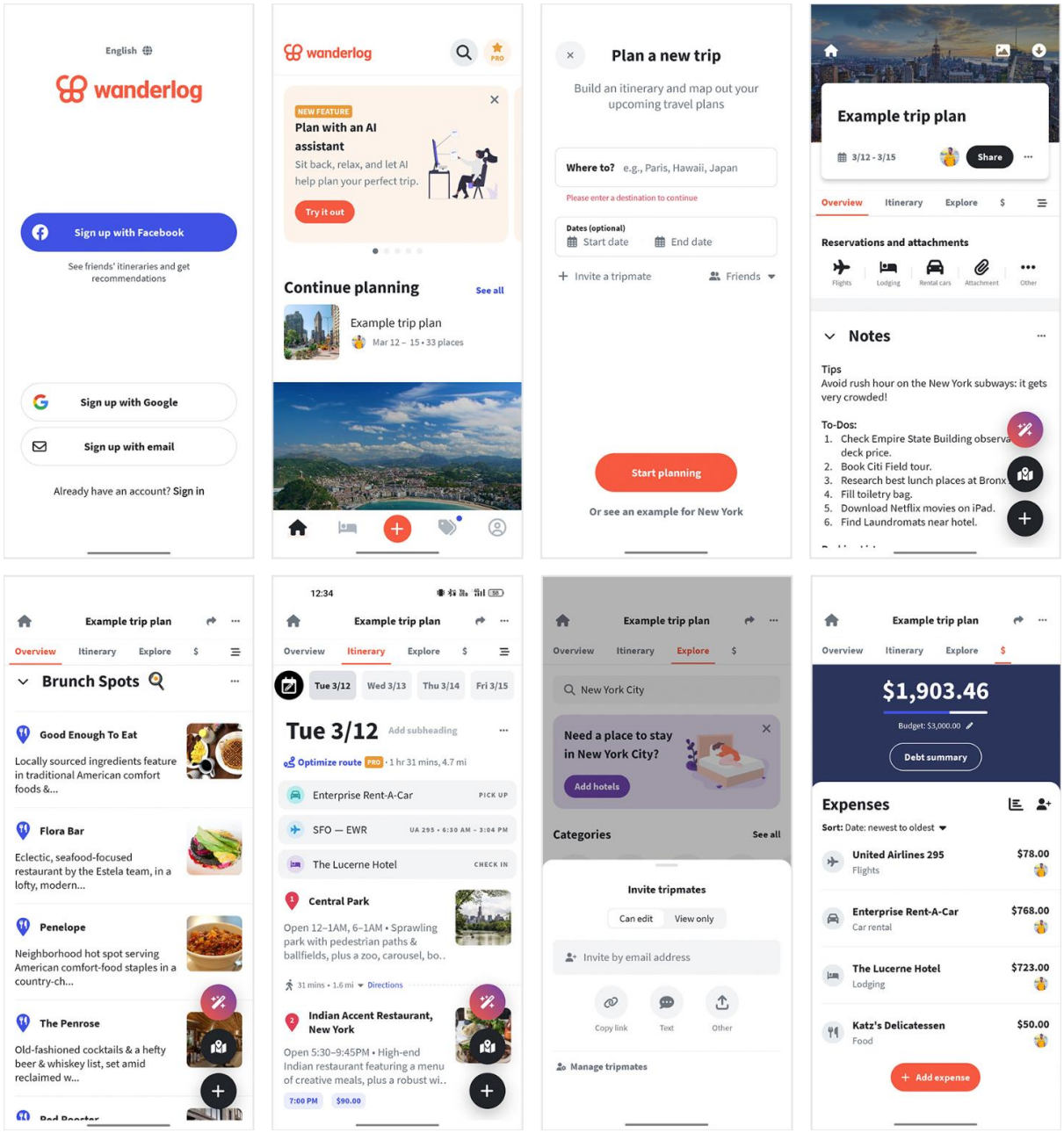


Figure 3 Wanderlog Screenshots

An advantage of Wanderlog is that all the important information can be stored in one place, which then can be viewed offline. Notes are useful when it comes to planning; Wanderlog provides a section for notes related to the trip and the destinations. Having an overview of the finances associated with the trip can help the user make informed decisions based on the expenses; this is a very nice feature, considering the user put in their budget, which shows a bar displaying how much they have already spent. Wanderlog released an AI feature on the 23rd of May 2023 that allows users to chat with the agent that suggests places for their trip, which they can then add to their itinerary directly.

Key advantages

- Trip planning (day-to-day activities)
- Note-taking
- Offline access to existing trips
- Finance overview
- Document upload
- AI Agent

The main disadvantage of this application is the onboarding. It is very long, and some of the questions are not even about the user. Mankulam (2025) ran a UX case study[3] on the onboarding experience of Wanderlog. He concluded that their onboarding had 38 screens, which they claimed made the application more personalised for the user. In the end, the dashboard ended up being very general. The problem with having many features and sections is that the UI gets visually cluttered. As a first-time user, it is hard to know where to look and what to look for.

Key disadvantages

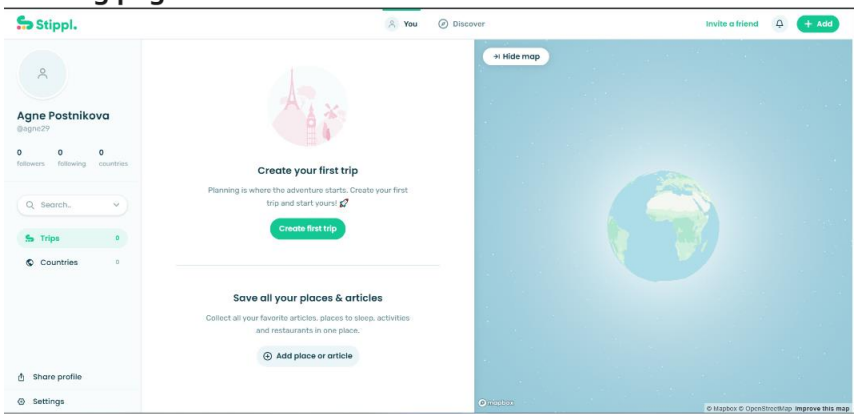
- 38-screen onboarding process (which does not work)
- Cluttered UI design
- Not easy to use

Stippl.

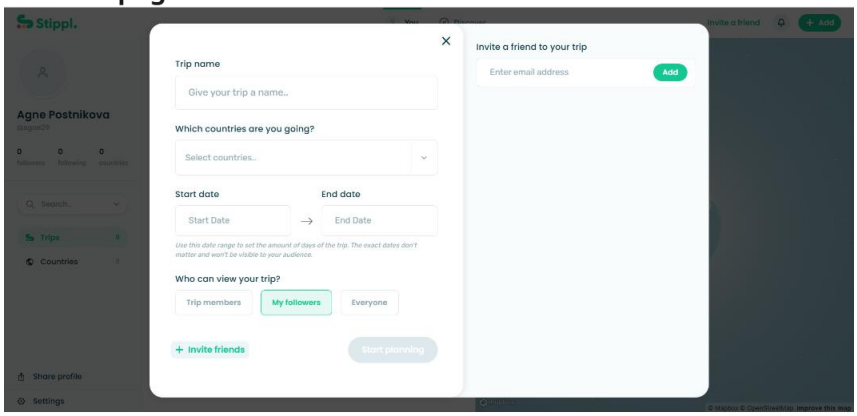
Stippl is another travel planner which focuses more on the experience of the trip and the social functionality, while it also provides users with an itinerary planner. This application provides a wide range of functionality such as planning, budget tracking, journaling and more. This application is a preferrable choice for someone who wants to keep track of their trips and experiences since the journaling aspect allows them to upload images, captions and create “movies” (videos), like on social media.

[3] https://www.behance.net/gallery/218301065/Wanderlog-Onboarding-friction-UX-Case-Study?tracking_source=search_projects|wanderlog&l=0

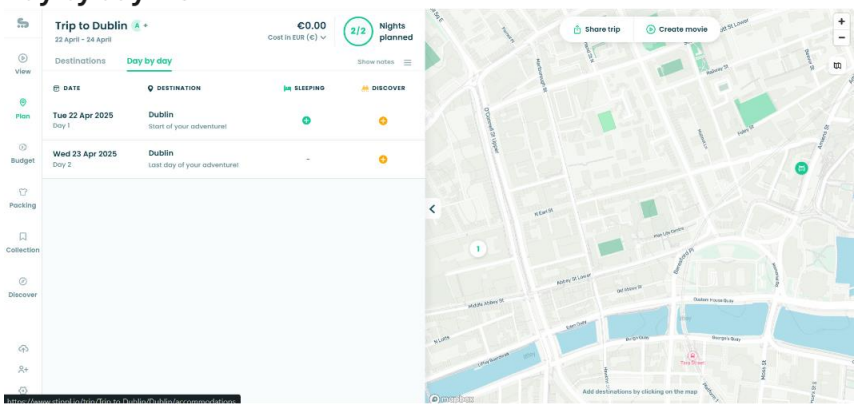
Landing page



"create" page



Day by day view



Other features

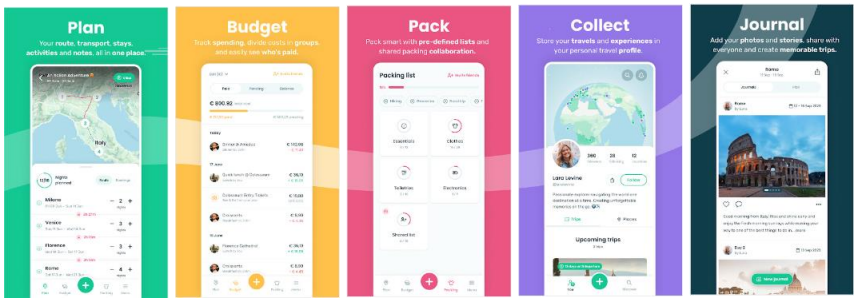


Figure 4 Stippl Application screenshots

Figure 5 Circular Progress bar indicating days

The first three screenshots in Figure 4 show the process of making a trip. The user can create a trip, title it and select dates for the trip, which automatically generates the circular process bar. It indicates to the user how many days the trip is. From there, the user can fill each day with a destination, which fills out the circular progress bar visible in Figure 5.

This application has many advantages, the first one being a very simplistic design – it is clean, all the components are visible, and the interface does not overload the eye. The next thing is the trip planner itself. They have a drag-and-drop feature that allows for easy organisation; each destination is chosen from a drop-down list of places. The planner is quite detailed; each destination can be for multiple days. Maps are integrated into the app, the trip is displayed on the map, which gives a nice overview of the entire trip. This application also provides a budget which the user can keep track of.

Key Advantages

- Clean UI
- Well-designed trip organisation (drag and drop)
- The user chooses destinations from a drop-down list (prevents writing entire words)
- Content creation (reels, journals)
- Discover community itineraries

When I was going through the process of making a trip, it was unclear where to click next. This application has recommendations on what to do in the place selected, but it was not straightforward to figure out how to add my own activities/destinations.

A key disadvantage of this application is its stability. Many users complain in their reviews about the app crashing and being glitchy or lagging, which ruins the experience of using the app. These reviews come from the Google Play Store[4] and Producthunt[5].

[4] https://play.google.com/store/apps/details?id=com.stippl.stippl&hl=en_IE
[5] https://www.producthunt.com/products/stippl

Key Disadvantages

- Poor stability
- Lots of features mean a steeper learning curve
- Hard to navigate as a beginner

Requirements modelling

Proto-personas

Personas are used to create user-centric products. In user experience design, they are used to empathise with the user; they give the designers (and sometimes even developers) a common target, which reduces the risk of miscommunication. Personas have pain points which are trying to be solved with the product being created. Personas are based on real-life people that have been interviewed, while proto-personas are created after doing user research; they are fictional characters that reflect a real-person case and their needs.

Becky Williams

The first persona is Becky, a single mother who wants to go on a road trip through Ireland. Since she is quite occupied with her baby, she barely has time to sit down and browse through the internet and figure out how to use convoluted itinerary planner apps. She needs something simple and effective. Since she's planning to drive across Ireland, she will need something that will connect to maps. Figure 6 shows more details about Becky.

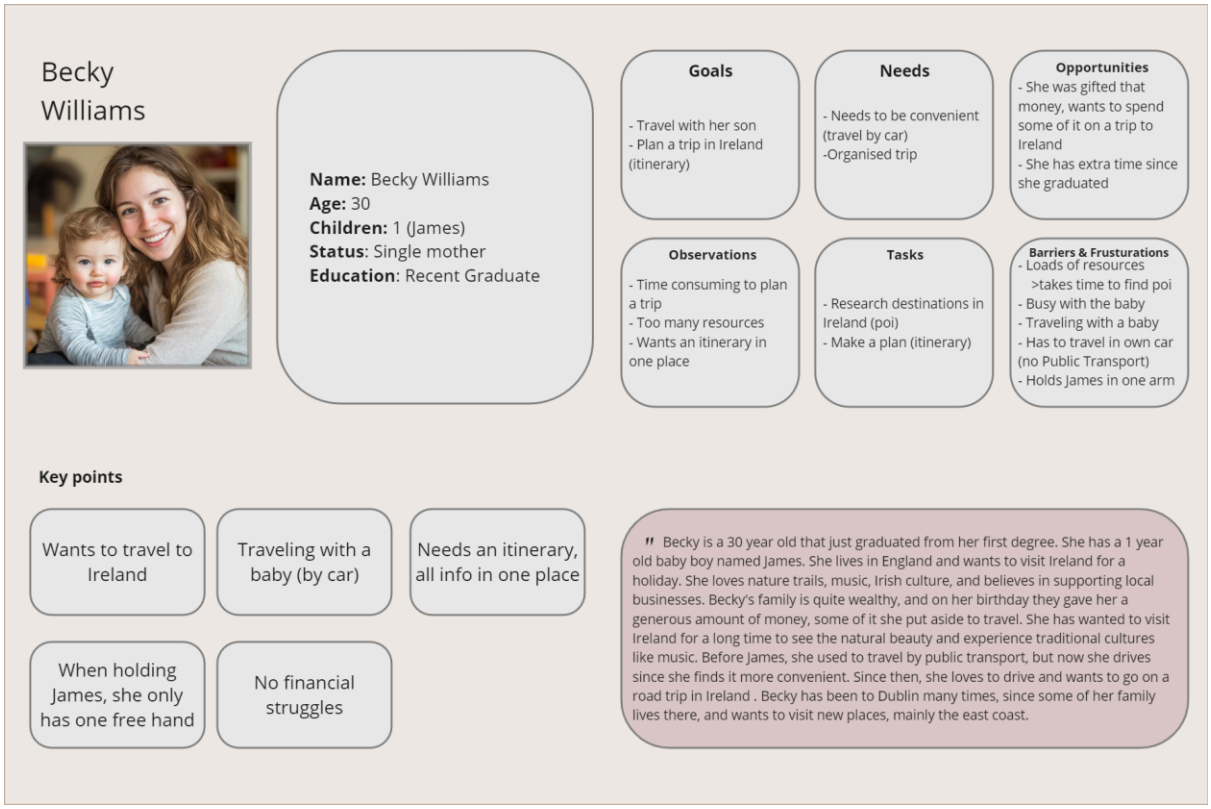


Figure 6 Proto-persona 1: Becky

Jackson O'Shaughnessy

Jackson is the second persona in Figure 7. He is an Irish-American who has not had a chance to visit Ireland. Now that he is retired, he has time to fulfil his dream of visiting Ireland. He needs an application that is simple to understand and that can store his itinerary information. Since he goes on hikes, he might not have an internet connection. This means that it is important to him to have his itinerary information offline or stored safely on his phone.

Figure 7 Proto-persona 2: Jackson

Functional requirements

Functional requirements are the foundational functionalities which are necessary for the application to work or at least do the bare minimum while also solving the user's problems. For this itinerary planner application, the functional requirements are the following:

- User Authentication (to keep user data private)
- Create, read, update and delete (CRUD) itineraries
- Download itinerary information to local device
- Browse through local bed and breakfast locations
- Add BnB locations to the "saved" folder

Non-functional requirements

These are requirements that are not crucial for the application's functionality. They improve the application's performance and usability if applied. The non-functional requirements are as follows:

- Drag and drop feature
- Maps integration displaying itinerary destinations
- An AI travel assistant (chatbot)
- Google Places API for itinerary destinations (dropdown list of places)
- Offline capabilities (PWA)
- Open Itinerary destinations in Google Maps
- Add information about attractions and Irish Music events
    - Save them to a specific itinerary

Use Case Diagrams

The purpose of use cases is to understand how the system can help solve problems for the user. The actors are the users who will be using the application; this way, it is easier to have an overview of the application and understand the core features. It also helps to see if the user's problems will be solved.

A use case diagram is a good way of defining the scope of an application. In this case, the core features of the itinerary planner application are clear. Each actor has different yet similar needs, but both of their needs are being satisfied through the application, which is demonstrated in Figure 8.

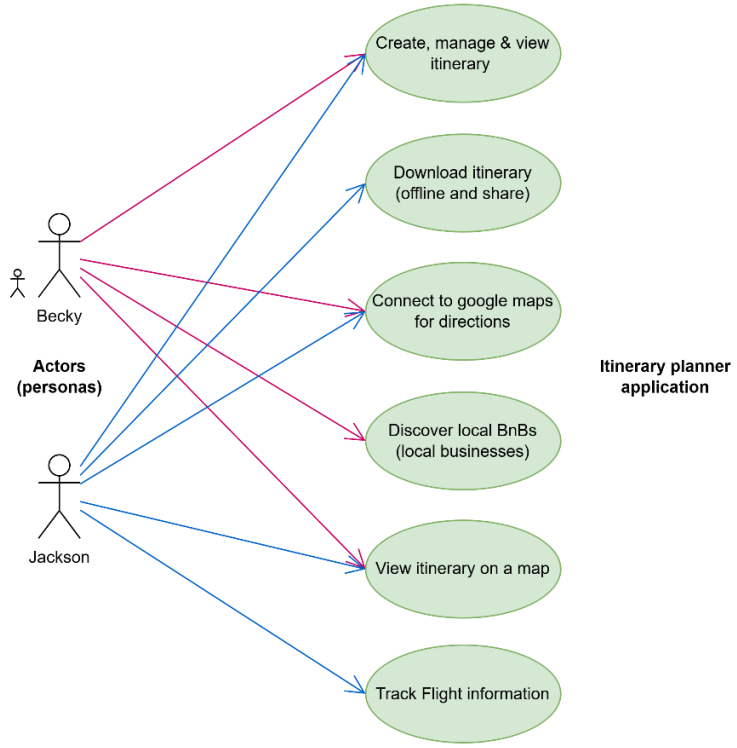


Figure 8 Use Case diagram

Feasibility

Feasibility is knowing and understanding if an idea or plan is possible, and if so, how. The goal of this section is to determine what technology will be used to complete the application. A very common issue is compatibility with different technologies. Since technology develops so fast, it often happens that other technologies take longer to keep up with the current stream, which can lead to issues.

For this project, there are several types of technologies being used, including code, design, research and communication.

Code:

- Next.js
- Firebase (BaaS) – Realtime Database and Authentication
- Google Maps API
- Google AI Studio
- Vercel

Design:

- Figma
- Miro
- ShadCN (Component library)
- Tailwind CSS

Research:

- Miro
- Google Scholar
- Google searches

Communication:

- Teams
- Outlook (backup)

Most of the technologies used to code this application are new to me. I have no experience with Next.js (which is a React framework) or Google tools such as Maps API, Gemini and Firebase Authentication. The lack of experience is a huge challenge, considering the time constraint for this project. The benefit of using Next.js is the file-based routing, which is very similar to React Native. React Native was used originally to create this application (for mobile only). Unfortunately, that did not work out, so a difficult decision was made to switch front-end frameworks to Next.js. Moving to a brand new technology that I was completely unfamiliar with was a huge learning curve. On top of that, I had to do all the research to see how the rest of the technologies would work with Next.js.

Considering all the new technologies, many challenges arise. The only way to work through them is to research the technology and look into other projects that use a combination of technologies to see if they are compatible or not. For example, Next.js and Google Maps API was a huge section to research since this was one of the major reasons for the decision to switch from React Native to Next.js. Looking up YouTube tutorials on how people built projects using the Maps API was helpful. Another tool that was helpful when it came to explaining these technologies was Claude. It is an AI tool that outperforms ChatGPT when it comes to code and reasoning. I was able to get a basic understanding of all the technologies and their compatibility in a short amount of time.

Appendix 1 explains in more detail why the decision was made to switch from React Native to Next.js.

Conclusion

This section covered the outline of what the project will be. Similar applications are Wanderlog and Stippl, which do similar things, but they both have their strengths and weaknesses. A common weakness is that they both have a lot of features, which is good, but it also can be quite overwhelming. The goal is to build a simple yet useful application that will do the job without overwhelming the user. Proto-personas Becky and Jackson were created. From there, the functional and non-functional requirements were outlined. Based on that, a use case was created to help understand what the main functionality will be. Finally, the technologies used in this project were mentioned, and the challenges that could arise throughout the project were discussed.

Design

Introduction

This section will cover the entire design process of the project, from the technical aspects such as code to the UI design of the application, such as prototypes and design systems. After analysing similar applications and tools that will be used to build this application, it is important to outline and shape (design) how to go forward with the design process. This will help later when the building process starts.

Program Design

After researching the technologies, making a plan of how they will work together is important to help speed up the development process. Having the functional requirements helps outline the design of the technical part of the project. This part of the section aims to have a good overview of what the project will consist of and how it will be done.

Technologies

The following technologies are used to create the itinerary planner application:

- Next.js
- Visual Studio Code (VsCode)
- Android Studio
- Google Maps API
- Google AI Studio – Gemini 1.5 Flash API
- Firebase (BaaS) – Realtime Database, Authentication
- ShadCN and Tailwind CSS

These are the core technologies used to build this application. VsCode is an IDE (integrated development environment). I chose it because I have a lot of experience working with it; every project I built was made using this software. Another similar tool is Android Studio, but it is used to develop mobile applications. The benefit of that is that Android Studio has built-in virtual devices. This means that while building a web application, Android Studio allows users to easily display the same Web application on a mobile device. Once the emulator has started, it is possible to open localhost:300 (the application) on both desktop and android web browsers.

Google Maps API and Google AI Studio were used to build up the functionality. Google provides a free tier for both, which comes with limitations. Google also provides Firebase; Realtime Database and Authentication were used. The reason I chose Realtime Database is because I already had experience with it, and it is straightforward to use and work with. On the other hand, authentication is something new that I had to learn and figure out. Google is good at providing lots of documentation for their services.

There are lots of examples of these services online, on YouTube or in articles that provide a breakdown of what it is, how to use it and how to implement it. Since the focus of this project is on the front end and user experience, I decided to use Firebase as my Backend as a Service (BaaS). Considering the time constraints, I wanted to give myself more time to focus on the UX and front end.

ShadCN is a component library. The good thing about it is that once it is installed and set up, to start using their components, you simply run one line of code to add it to the UI folder. ShadCN provides the code to the entire component, which means that the developers can change the way that component looks and behaves since we have the code directly in the project. This was one of the reasons to go for this UI library instead of gluestack-ui or Material Design. The simplistic design of ShadCN also looks clean, and it's easy to use. It is built on top of Tailwind CSS, a very popular and powerful CSS framework. This combination allows for a straightforward design process since it is possible to use both at the same time.

When it came to researching what tools are out there that can provide a map and interact with it, there are a few, such as MapBox, Leaflet, OSM (Open Street Map) and HERE maps. MapBox was the first choice when it came to maps at the very beginning of this project. The only problem, and the reason I did not continue with MapBox, is that they do not provide official support for React. There is a community-supported library that supports MapBox in React Native, but even that failed since the versions of the library were not compatible with React Native. I was in contact with the MapBox support team for a few months in the very early stages of the project to see if they were able to support my project somehow. They were very willing, but because I was working with React, they could not help. They offered lots of other services and solutions, and I was very satisfied with their swift response. This is when Google Maps became the primary tool for the project. React Native did not work out, but Next.js did not disappoint. Google services seamlessly integrate with this React framework.

A Progressive Web App (PWA) is essentially a website with a native 'feel' to the device it is displayed on, as it can be downloaded like an app. NoteboolLM was used to summarise the key findings and benefits of PWAs from the book Progressive Web Apps by Hume (2017). He mentions in his book that Alex Russell, who works for the Google Chrome team, describes PWAs as "just websites that took all the right vitamins". The benefits of a PWA over a regular website are the following features: offline capabilities, installability, push notifications, near-instant load times, an app-like feel and progressive enhancement, to name a few. Progressive enhancement means that if the user's browser does not support PWA features, then the website can still work like a regular website. Due to this, if the PWA fails to work, the website will always work as a backup.

Next.js structure

Next.js is a framework that is built on top of React, which is built on top of JavaScript. Next.js requires Node.js to run the project. It has many features that are accessible to the developers, such as API routes; it is possible to build API endpoints within the application itself, server-side rendering; in Next.js, it is possible to render the code on the server or the client side, which speeds up the loading time for the users, among other features.

Before explaining the folder structure, it is important to clarify that there are two types of routers in Next.js: a Page Router and an App Router. The App Router is used in this project. They are similar, but the major difference is the file conventions; the page router uses "_app.tsx", while the App Router uses "page.tsx" as the endpoint. For example, in this project, when a user wants to get to their itinerary, the file path for that is: "app/itinerary/[id]/page.tsx". The name of the folder is the endpoint for the user; each folder can have a "page.tsx" which the user will be able to access. Figure 9 displays the file conventions in App Router.

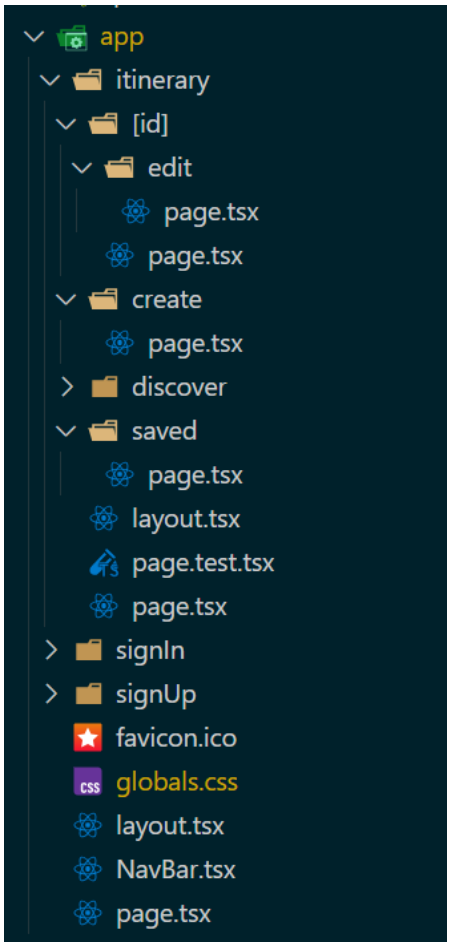


Figure 9 Screenshot of the app folder structure

React emphasises the component folder where reusable components can be stored. Figure 10 shows the components used throughout the project. The folder has two additional folders consisting of UI files, which are ShadCN UI components, and the itinerary folder, which has the reusable components from the main pages of the application.

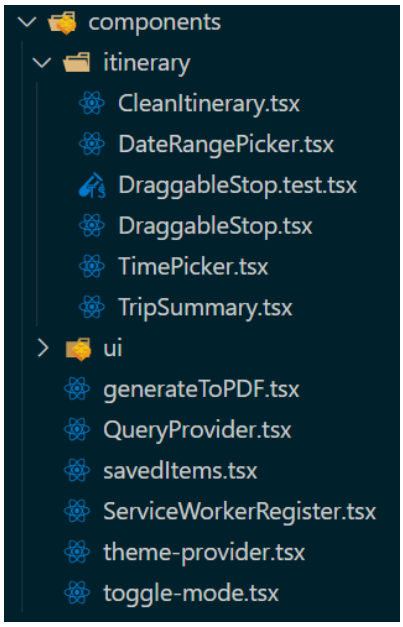


Figure 10 Components folder

The public folder is where the static assets are for the application. Things like logos and service workers go in there. A service worker is a JavaScript script that runs in the background of the website, it enables functionality like push notifications and offline capabilities. Figure 11 displays the public folder from this project. The three folders, “illustrations”, “bnbs”, and “landscapes”, are images used in the project. Next are the different size logos and “screenshots” (while the app is installing on a device, these screenshots are displayed in the background). Finally, the manifest, service worker and Workbox are files for the PWA. Later in the sprints, the purpose of those files will be explained.

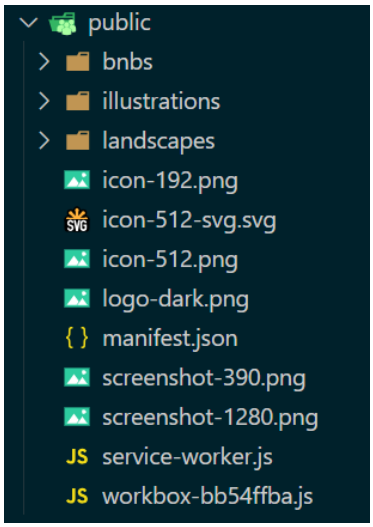


Figure 11 Public folder with assets and images

Next.js Design Patterns

Next.js 13+ introduced Server and Client Components (Next.js 15 is used in this project). This allows developers to declare which side the components will be rendered on, the client side or server side. They provide documentation that explains in which cases to use the client or server to render the components, displayed in Figure 12.

What do you need to do?	Server Component	Client Component
Fetch data	✓	✗
Access backend resources (directly)	✓	✗
Keep sensitive information on the server (access tokens, API keys, etc)	✓	✗
Keep large dependencies on the server / Reduce client-side JavaScript	✓	✗
Add interactivity and event listeners (`onClick()`, `onChange()`, etc)	✗	✓
Use State and Lifecycle Effects (`useState()`, `useReducer()`, `useEffect()`, etc)	✗	✓
Use browser-only APIs	✗	✓
Use custom hooks that depend on state, effects, or browser-only APIs	✗	✓
Use React Class components ↗	✗	✓

Figure 12 Screenshot of Next.js Documentation: Server and Client Composition Patterns

Server rendering is used to complete tasks such as fetching data, accessing databases and any backend services. While event handling and most other tasks can be done on the client side. The benefit of this is that the JavaScript sent to the client is reduced, data can be accessed without API calls, and the code organisation is improved since there's a separation between the interactive (“onClick” and “useState”) and non-interactive (API calls, accessing tokens) code.

Application architecture

The point of an application architecture is to visually represent how the tools/software fit together. This helps to plan the design of the system and how the tools will interact. Figure 13 illustrates how the front-end software communicates with the Google services. The user begins by logging in/registering; this communicates with Firebase Authentication. After authentication, the user can create an itinerary. Upon typing in the destination for each stop, Google Places API presents the user with a drop-down to autocomplete the destination. Once that itinerary is saved in Firebase, the user can view it. Since Places API provides latitude and longitude coordinates, which are saved with each stop in the itinerary, the user can view their destinations on a map. The coordinates are checked when the map is loaded.

If they are provided, everything is displayed on the map, otherwise, the user is shown a “No location data available for this itinerary”. Gemini 1.5 Flash is the model used to create the chatbot to assist the user if they have any questions about the itinerary. The chatbot is placed in the itinerary view page to be able to provide the model with the information from the specific itinerary.

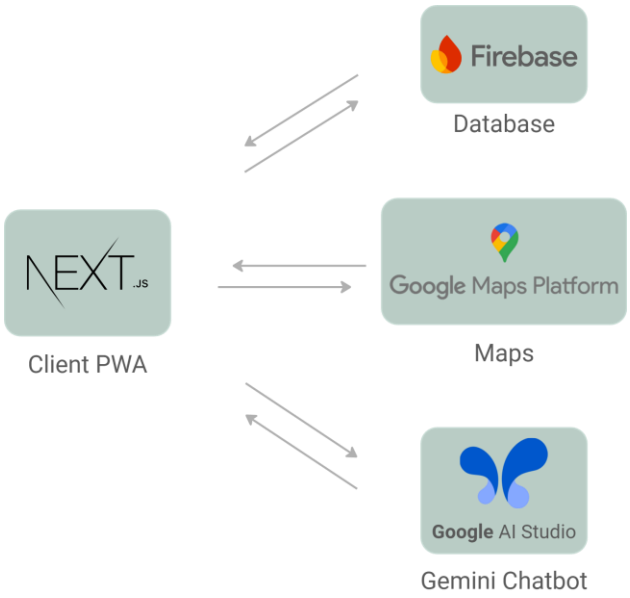


Figure 13 Application Architecture Diagram

Database design

Before explaining the data structure, it is important to know how the Firebase Realtime Database stores data. It is a NoSQL (non-relational) database. The data is stored in JSON format, which is then synchronised across all of Firebase’s clients. This is how the user data is protected.

```
{
  "users": {
    "alovelace": {
      "name": "Ada Lovelace",
      "contacts": { "ghopper": true },
    },
    "ghopper": { "..." },
    "eclarke": { "..." }
  }
}
```

Figure 14 Screenshot of Firebase RTD data Structure

Figure 14 is an example of how Firebase stores the data in JSON format; the user has their own object where their data is stored. If the database rules are set up correctly, only the authenticated user can access their data; everyone else will be denied.

```
"rules": {
  "User": {
    "$userId": {
      // grants write access to the owner of this user account
      // whose uid must exactly match the key ($userId)
      ".write": "auth !== null && auth.uid === $userId",
        ".read": "auth !== null && auth.uid === $userId"
    }
```

Figure 15 Screenshot of the Firebase RTD Rules

Figure 15 shows the rules assigned to the database. What these rules are saying is that if the user is not authenticated, restrict access; if the authenticated user is not the same user, restrict access. Essentially, these rules are protecting the user from anyone else accessing their data.

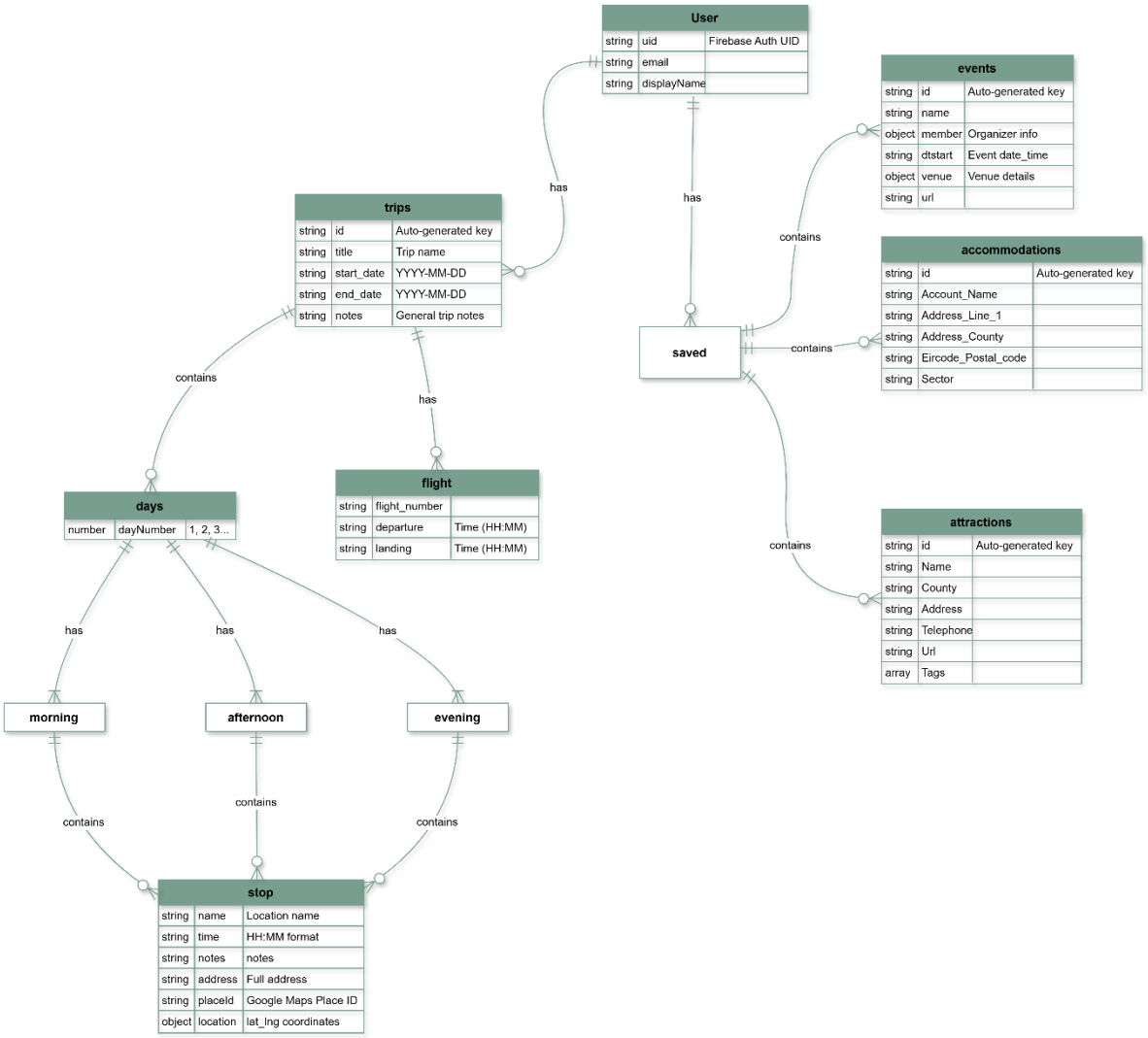


Figure 16 Entity Relationship Diagram of Data

For a better understanding of the data structure, an ERD (Entity Relationship Diagram) is used to visualise how the data is structured. In reality, as mentioned earlier, this data is not relational (NoSQL). For a better user experience, most of the input fields are optional, except the days table.

When the user selects dates for their trip, that dynamically generates the number of days the trip will be; thus, that part is not optional. Figure 16 is a detailed ERD which provides an overview of the application skeleton.

Process design

The design chapter covers the user flow diagrams, which were used to design the process of this application. The main reasons for this were to have an overview of the entire application and to help with the development planning. This chart helped plan out the logic of the application and figure out the multi-step itinerary creation. Instead of making one long/scrollable page, create a multi-section page that the user can click between. Since the itinerary can become quite long anyway, a summary was created later that lets the user quickly view what was in the previous days in the itinerary, which also improves user experience by removing the need to scroll all the way up and down through the itinerary.

User interface design

This section will cover the user interface design for this project. The UX nature of this project has led to covering a lot of UX steps, such as studying design thinking, creating user storyboards, empathy maps and journey maps. All of this is to get a better understanding of what the user needs from the application and which problems need to be solved.

Design Thinking

Design Thinking is a system which helps designers keep track of the priorities of the project, the user and their needs/problems that need to be solved. There are 6 stages in design thinking. They are important to identify the problem, come up with solutions, experiment to find out what works best and create a final product. The 6 stages (Figure 17) allow the designers to make the user the centre of the project, starting with empathising with the user and ending up with a solution to their problem. The whole point of the design stage is to be able to create a solution for an existing problem, hence fixing the problem of the persona (user).

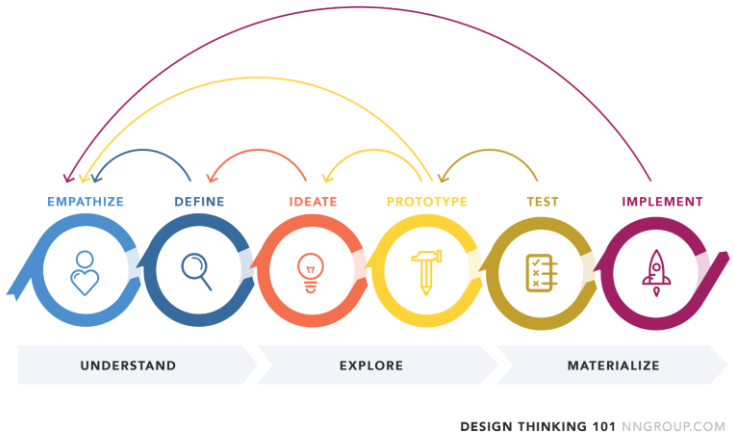


Figure 17 Design Thinking visualised diagram

Figure 17 shows the 6 steps of design thinking. Every stage relates back to the user we are empathising with. Each stage goes in a circle, which can lead back to the previous stage. Sometimes, ideas go forward that need to be improved or completely changed, and taking a step back is the step forward. Figure 18 explains each step in more detail.

Empathising with the user is important because it creates this connection, creating a sense of wanting to help the user with a problem they are facing.

Defining and trying to understand their problem helps focus the goal and solution. Pinpoint the users needs and problems.

Ideating is coming up with a few solutions for the problem, then narrowing down on the best one(s). Like the Double Diamond theory.

Prototyping is important to pinpoint what aspects of the solutions work, and which do not.

Testing the prototype helps identify flaws, bumps and imperfections in the ideas/solutions. Test if the prototype solves the initial problems of the user.

Implementation is bringing ideas into life and improving the life of the user by solving their problem.

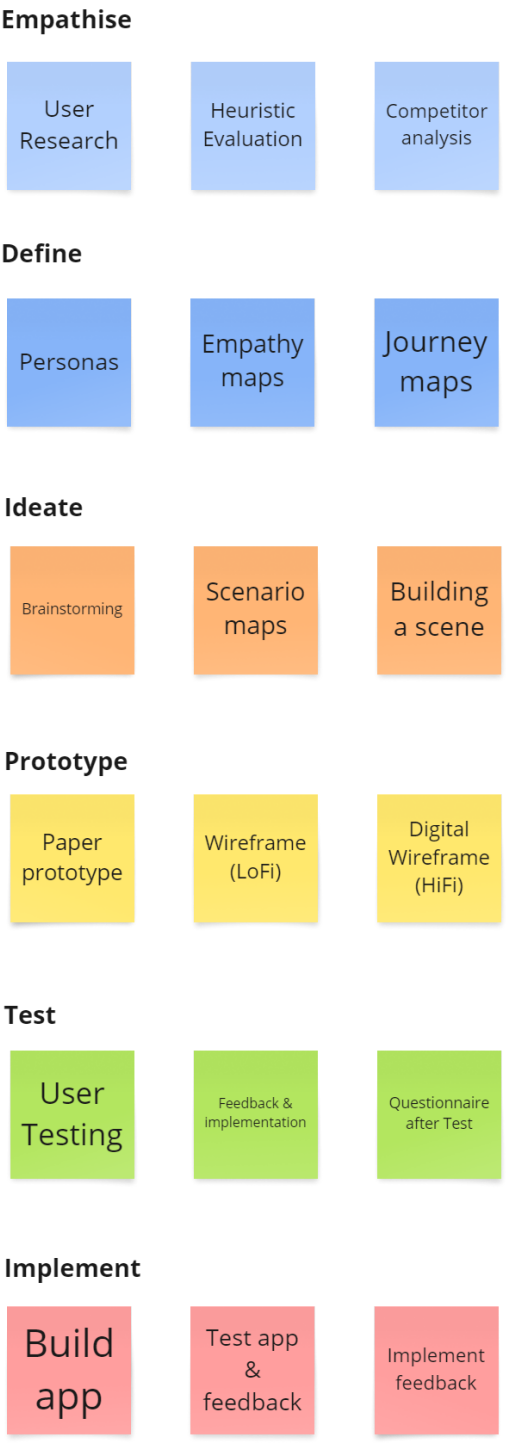


Figure 18 Design Thinking - Stages Explained

Storyboards

A storyboard is a board that tells a story of the user; most of the time, it is about their journey in which they use the product that has solved their problem. The point of storyboards is to help visualise the solution to the problem. The user uses the product and reaches their goal without problems with the help of the product. It helps to visualise the end product.

These storyboards were made in Canva. It is a very helpful tool that allows anyone to plan and design everything in visual communication. They provide a lot of free templates, which are being used here. Storyboards can be hand-drawn, drawn digitally, or anything really, as long as the point of the story is clear. Due to time constraints of the project, I chose to find any relevant images in Canva and use them in my storyboard. This is why the images are of different people, but the actions are what matter.

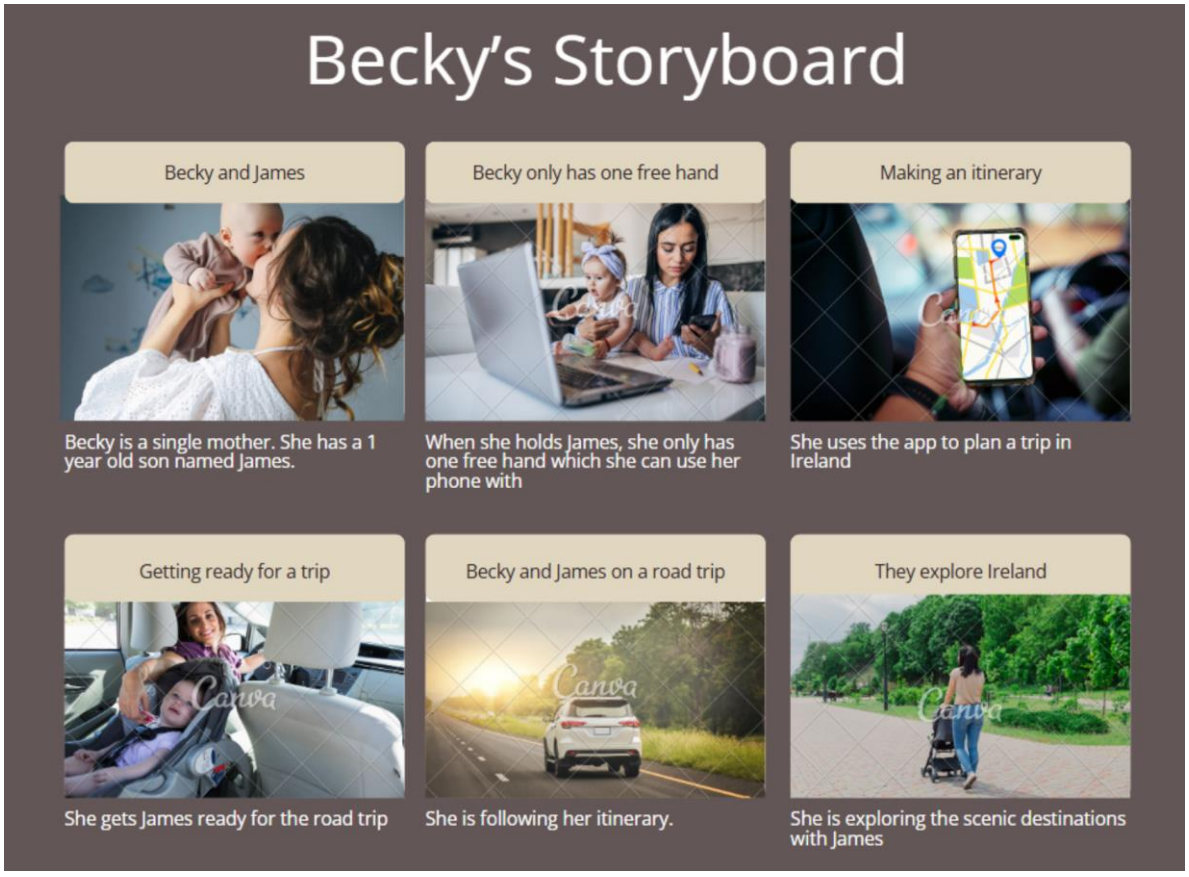


Figure 19 Storyboard - Becky Williams

This board (Figure 19) tells a story of Becky going on her trip to Ireland, where she is stress-free and has her itinerary at the tip of her fingers. She can view her trip in Google Maps, which she can follow on the road. She is happy with the app since it helped relieve the stress of travelling with a baby. She could plan every step of the way and have it connected to Google Maps.

Figure 20 Storyboard - Jackson O'Shaughnessy

Jackson's story (Figure 20) is that he finally makes it to Ireland. After searching the internet for things to do in Ireland, he has made himself an itinerary. Since he loves to hike, he has downloaded his itinerary to have it offline before he goes off into the forest. After his hike, he goes into an Irish pub where he can listen to some traditional Irish music that he found through the app he used.

Empathy Maps

The purpose of "empathy maps" is to map personal information about the user. It can also be used to identify gaps in the products. Being able to see exactly what the user thinks, says, feels and does; helps designers empathise with them and drive toward a better solution. The target group has overlapping problems since the solution is designed to fix their problems. After researching Microsoft Inclusive Design[6] I decided that one of the proto personas (Becky) would have some impairment to make my target audience a bit more realistic and relatable. This is why Becky is a single mother.

[6] https://inclusive.microsoft.design/tools-and-activities/Inclusive101Guidebook.pdf

An empathy map for both personas was created to help relate to the user. The personal information and the pain points are there to keep the project goal on track. Figures 21 and 22 show the created empathy maps.

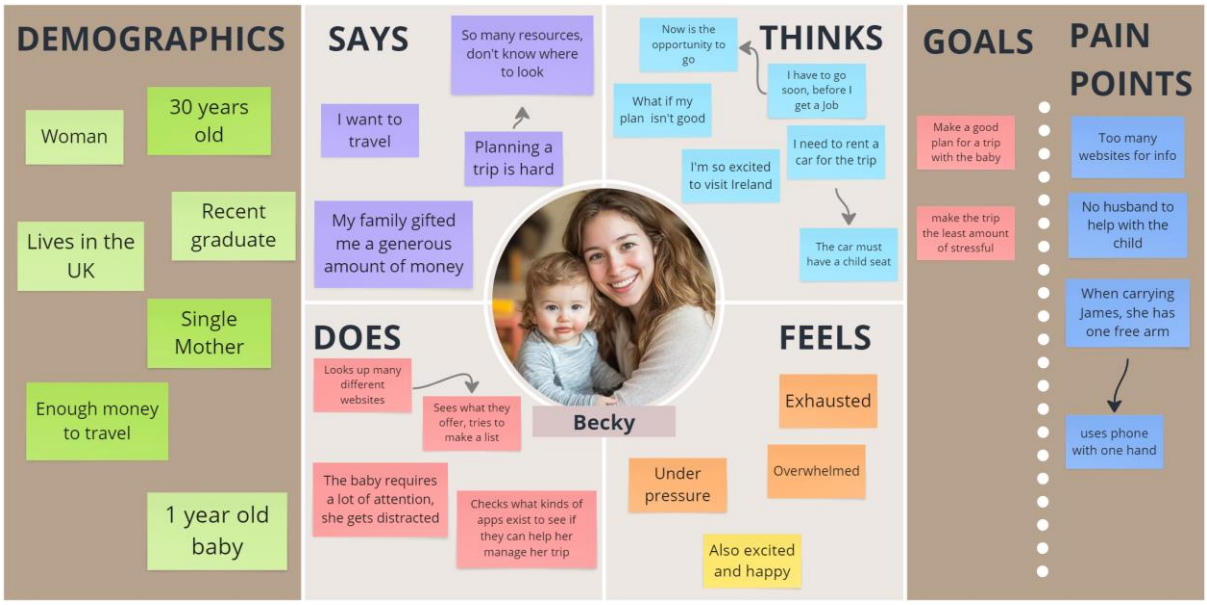


Figure 21 Empathy map – Becky Williams

In summary, Becky's empathy map in figure 21 points out she wants to travel and is happy to finish her studies – which means she has time. Simultaneously, being a single mother is hard, especially when it comes to travelling. She finds driving a lot more convenient than public transport because of her son, so Google Maps is necessary for a road trip. Finances are not a problem for her, luckily, since her family gifted her with money. She needs a place where she can plan each and every step since the baby requires a lot of care.

Figure 22 Empathy map - Jackson O'Shaughnessy

Summarising Jackson's story (Figure 22), he has wanted to go to Ireland for a very long time. Retirement presented the opportunity to go. The lack of travelling in the past has made trip-planning quite the task. He is excited but also overwhelmed by all the things he can do when he visits Ireland. His Donegal roots make him feel connected to the country, which fuels the excitement. He loves hiking, nature and music. His goal is to make an itinerary that he can save to his phone or access offline, since forests do not always have a great connection.

Journey Maps

Completing a task is a process. Journey maps are a way of showing this process. By viewing the process and having an overview of everything, it is helpful to spot common pain points and find solutions for them. Being able to analyse step by step what the user is doing also helps put the designers in their shoes, which creates a greater connection with the persona. Understanding what the user is going through helps motivate for the best result to solve the problem.

Usually, user journey maps begin with a bad experience. It is that experience that is aiming to be fixed. Both Figures 23 and 24 have a similar journey since the same problem is aiming to be fixed with an itinerary builder application. The emotions are a good overview of the entire journey since they give a straightforward understanding of how their journey went overall. Towards the end, the line increases, and the persona is happy with the result of the application they found as it solved their problems.

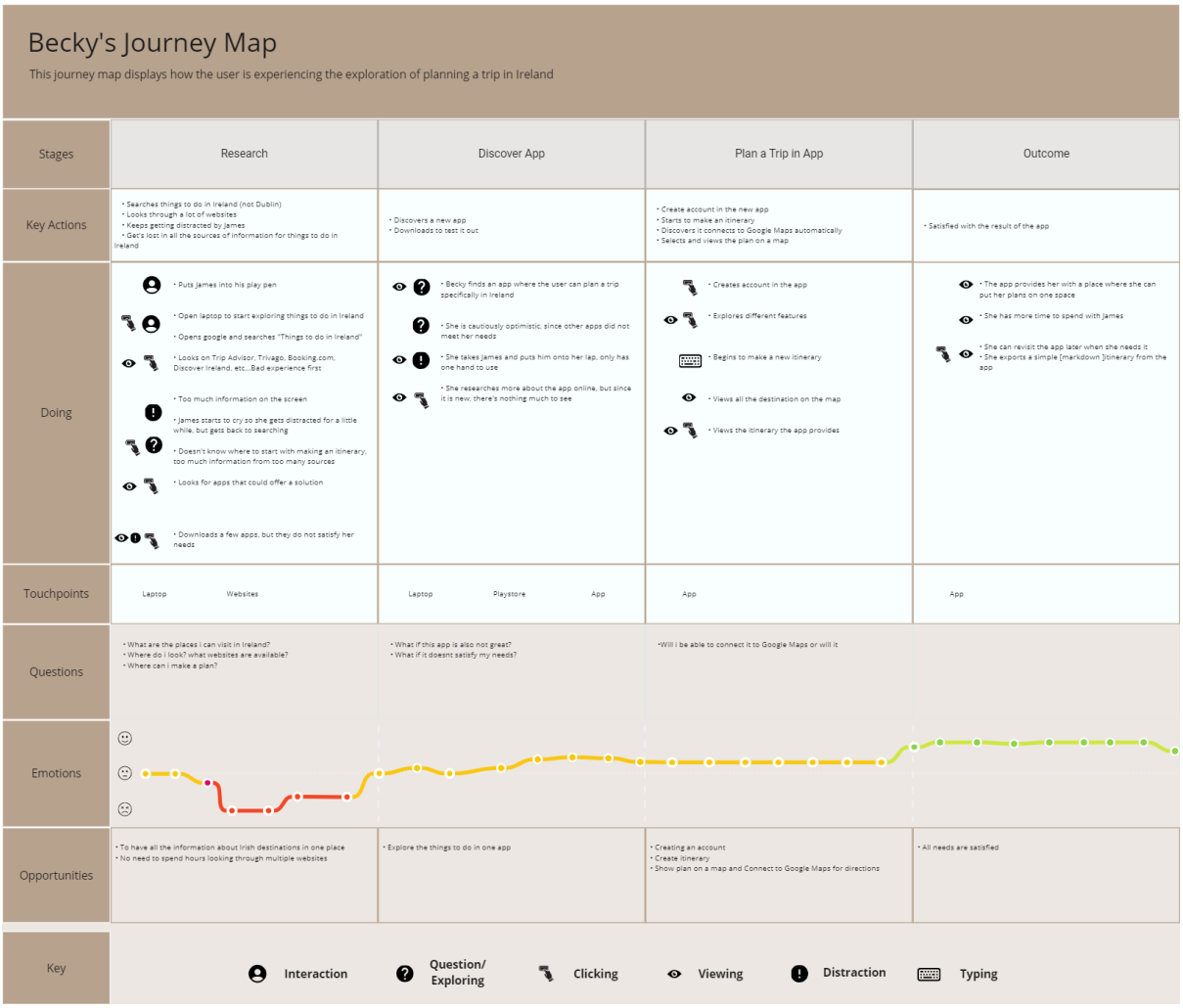


Figure 23 Becky's Journey Map

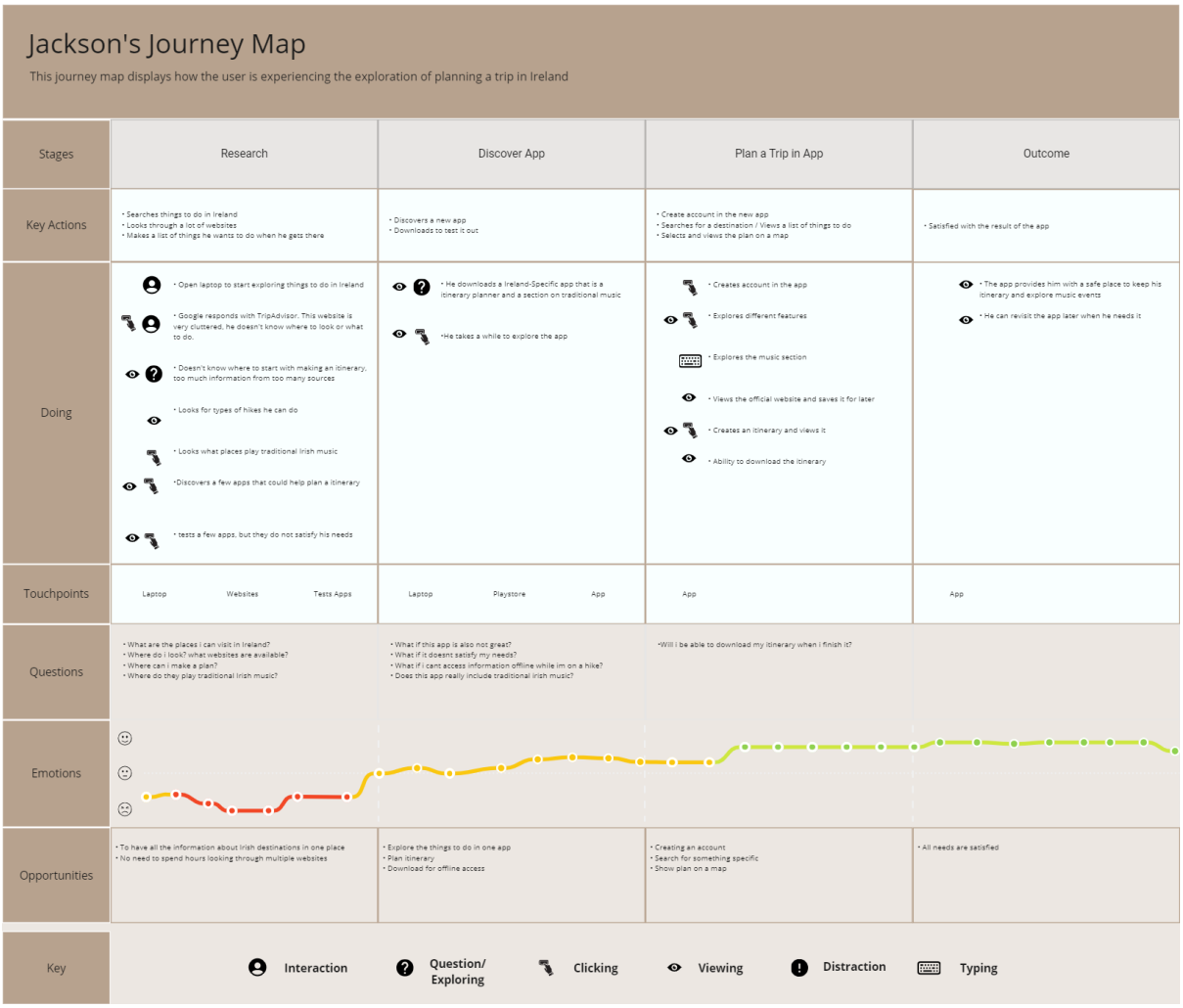


Figure 24 Jackson's Journey Map

Wireframes

Wireframes were created to visualise the product. Figma (education plan) was used to create them. The Figures in this sub-section will display the visual layout of the application. Wireframes not only help to visualise the design, but also help with testing the usability of the app and overall user experience.

The wireframes are completed for the entire application, but for this report, only the main aspects will be displayed.

Due to the rushed development phase, only a desktop version of this application was created, but the actual application works on both desktop and mobile phones.

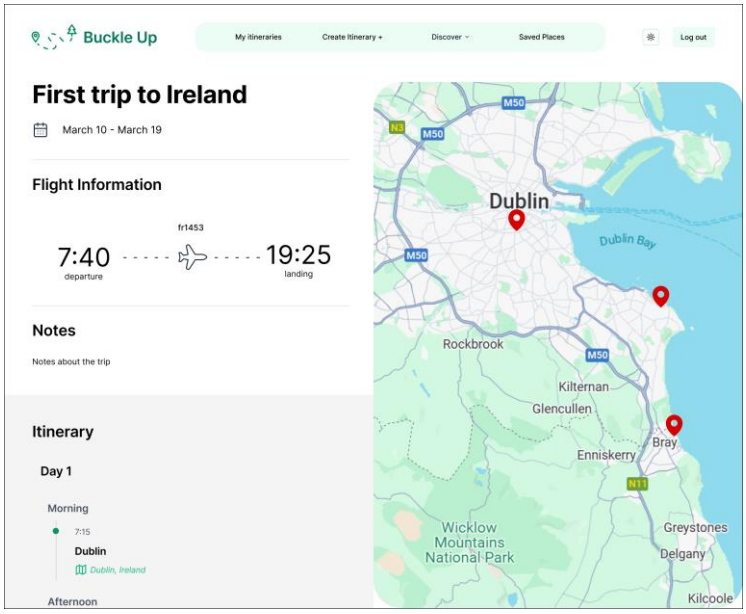


Figure 25 Itinerary View Page with Map Display

Figure 25 shows the itinerary on the left side and the map on the right-hand side. The locations in each period are displayed on the map; they can also be opened in Google Maps by clicking on the green link under the name of the location.

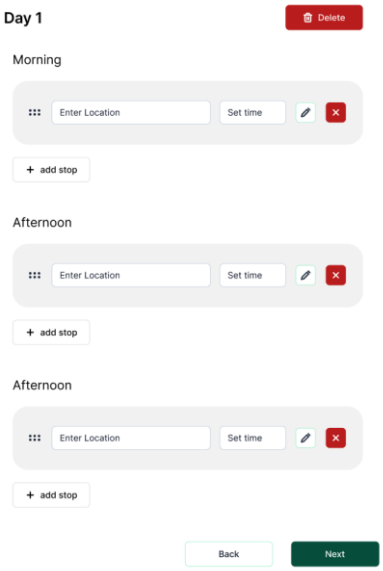


Figure 26 Creating itinerary destinations

When the user begins to type the name of a place, Google Places API autocompletes, and the user can select whichever place they need. It is also possible to add a time for the destination, add notes, or remove/add another stop displayed in Figure 26. When adding more stops, it is possible to reorder them with the drag handle.

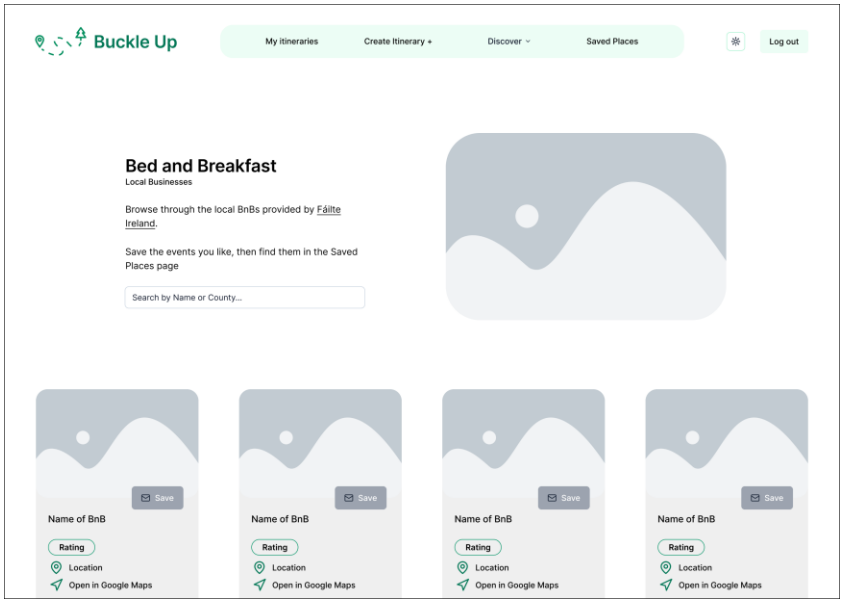


Figure 27 Local BnBs to browse and save

It is possible to browse through BnBs, attractions and music events. The general layout for these discovery pages is displayed in Figure 27. Each card can be saved to the user's account. In the future, it would be possible to add these things directly to the itinerary, but due to the time constraints, only saving it to the account was possible.

User Flow Diagram

These diagrams visualise all the different paths the user can take throughout the application. Another way of looking at this is like a blueprint of the application. Figure 28 shows the user flow for this application.

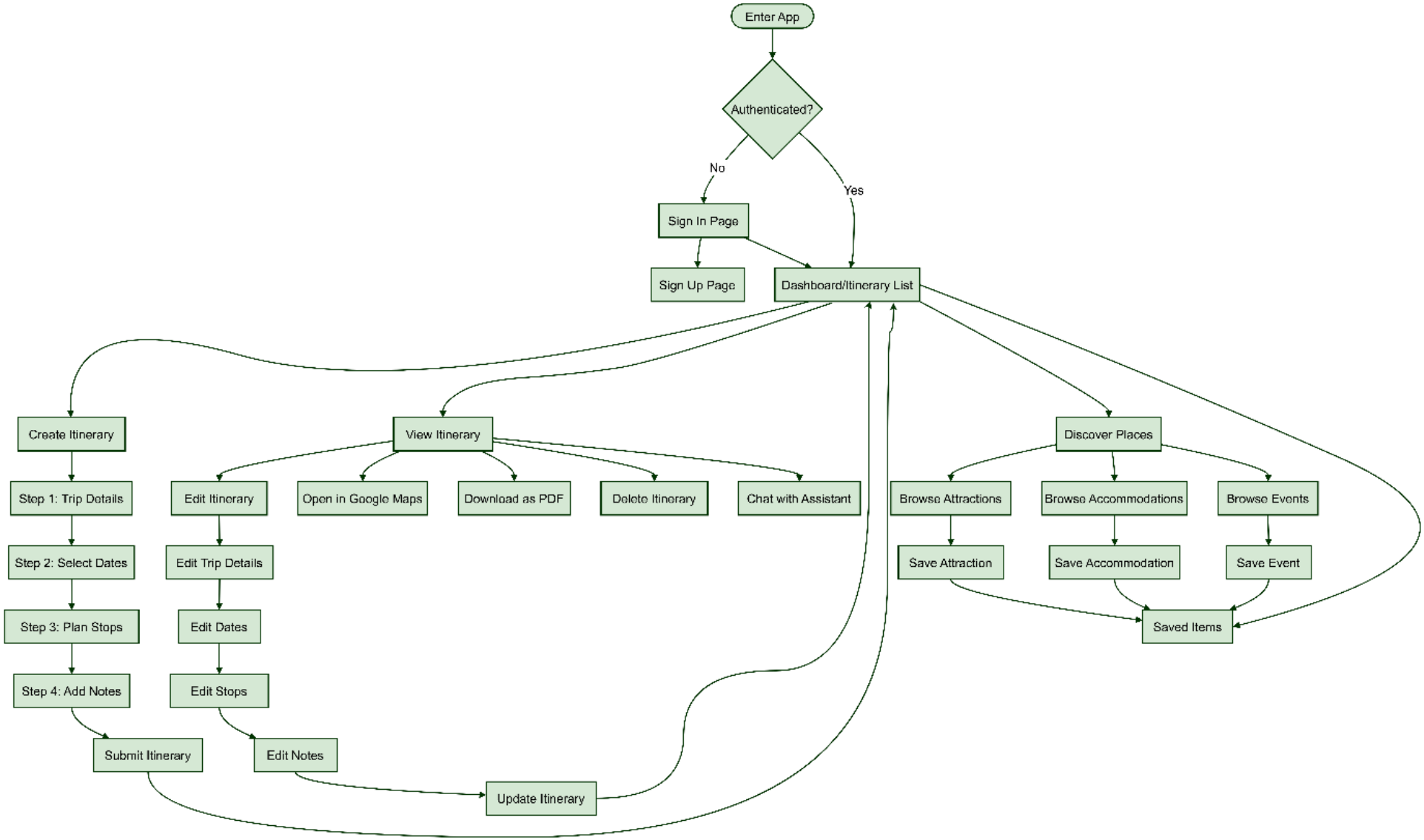


Figure 28 User flow diagram

Style guide

This application focuses on Ireland, so the only logical choice was emerald green. Since Ireland is an evergreen country, emerald green is the primary colour, to make this application have a calm atmosphere to it, soft green and light colours were chosen to complement the emerald green. Figures 29, 30 and 31 show sections of the application and how the colour themes are being used.

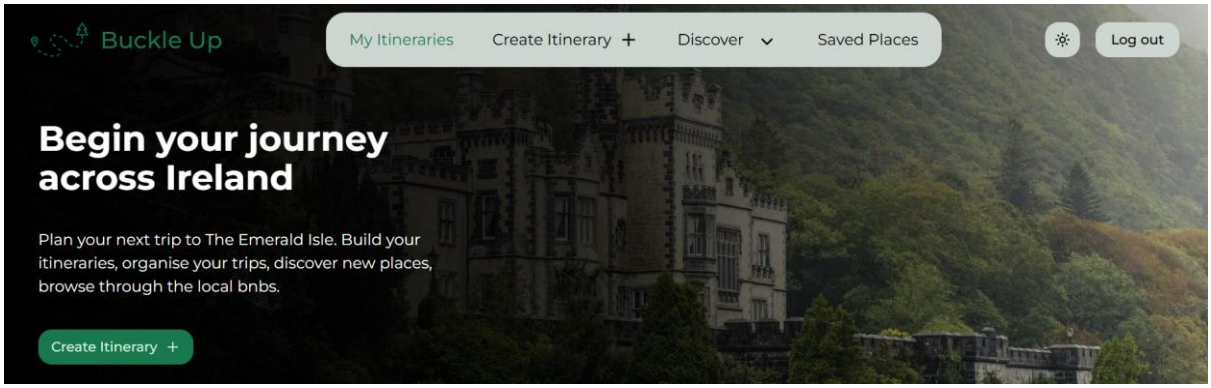


Figure 29 Colour theme - Hero and Navigation bar

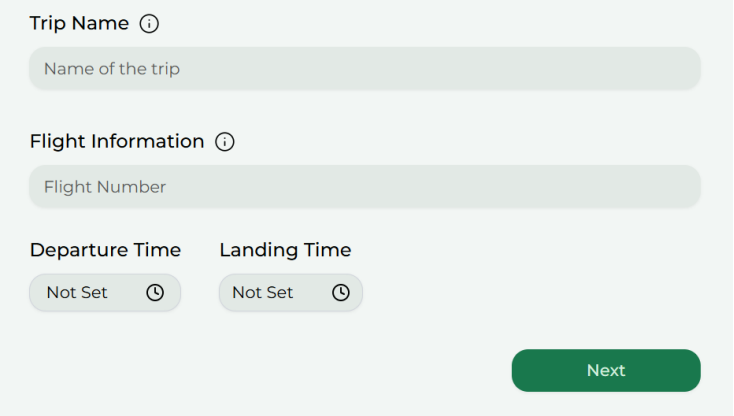


Figure 30 Colour theme - Create page section

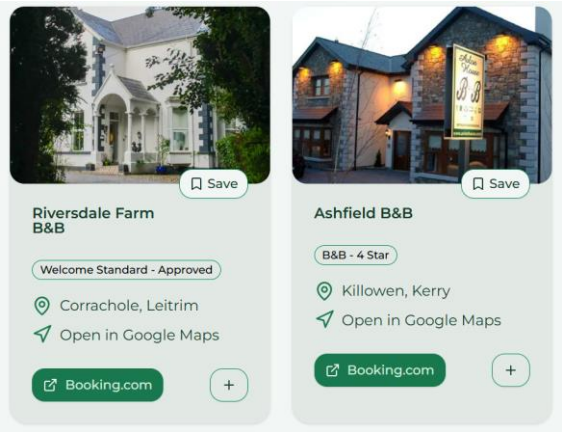


Figure 31 Colour theme - BnBs card

ShadCN is a component library that was used throughout the entire application to design its look and feel. The components they provide are ready-to-use, so not much was changed when using them, except for the main cards that display the users' itineraries. ShadCN uses a different colour type called OKLCH. De Kort (2023) defines the OKLCH CSS colours as;

> *"OKLCH is a color space that also includes P3 colors. This allows you to use a broader range of colors than regular RGB does. To display those colors users need a wide-gamut monitor."*

It was interesting to learn of a new way that CSS can be defined, to convert colours, the OKLCH[7] website was used to. The way the colours are presented on that website is displayed in Figure 32.

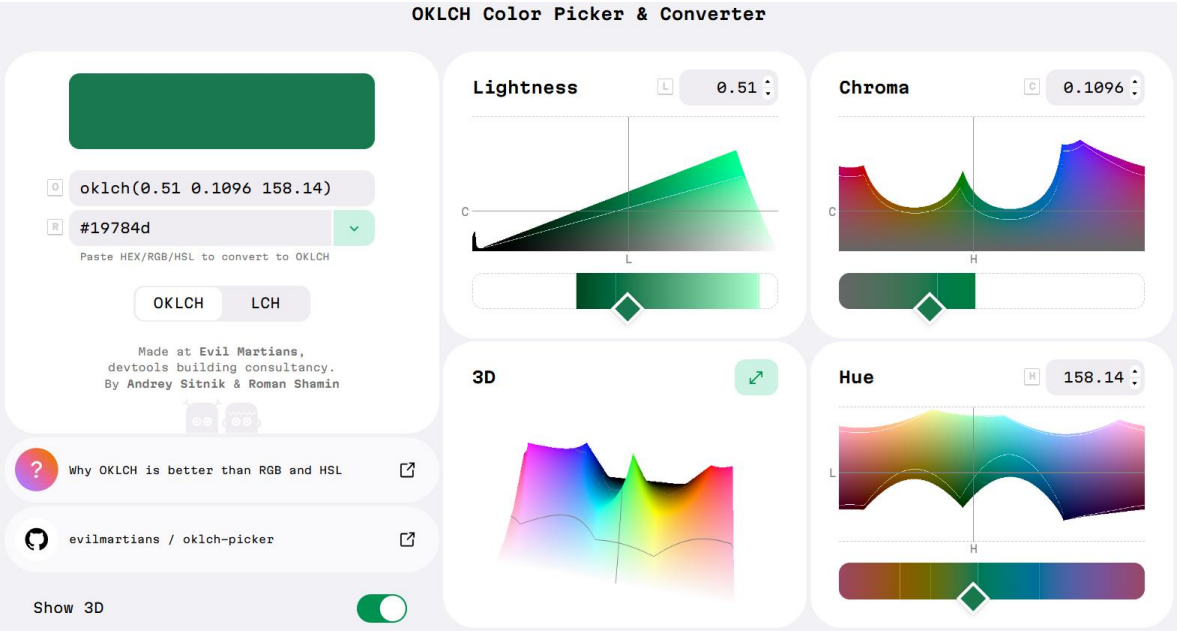


Figure 32 OKLCH CSS colour

This application uses the Montserrat font family. It was chosen due to its soft, rounded curves, which are easy on the eyes. One of the personas is not young anymore. Michael is an older gentleman, and the last thing he needs is a font that is hard to read. Norman and Nielsen (1998) ran a study[8] which resulted in Montserrat being the preferred font for older users. This does not mean it is any less preferred for younger users, but it was chosen since it is more accessible to everyone, as the study noted,

> *"picking the wrong font penalizes older users more than young ones."*

[7] https://OKLCH.com/#0.51,0.1096,158.14,100
[8] https://www.nngroup.com/articles/best-font-for-online-reading/

Conclusion

The design section covered a lot, from the tools being used to create this application to the design of the application itself. The main applications that were used are Next.js, Google APIs, Firebase and Figma. Figma was used to create wireframes of the application, from which the application then started in the implementation phase. To relate to the user and truly understand them both, empathy and journey maps were created, along with storyboards.

To begin the development phase, research was done into the Firebase Realtime Database, and an architecture application was created to visualise what and how the applications would work together. ShadCN is the component library that was used to design the application. In Sprint 5, the design followed the wireframes covered in this section.

Implementation

This section is going to cover, in detail, the implementation of this project. The Scrum methodology was used, and there were five one-week sprints due to the reasons explained in Appendix 1.

Figure 33 is a Gantt chart that displays an overview of the development events that took place throughout this project. While the original project was scoped for development using React Native, significant technical barriers (discussed in Appendix 1) required a sudden mid-project switch to Next.js. The project management section reflects both the challenges and learning outcomes associated with this shift.

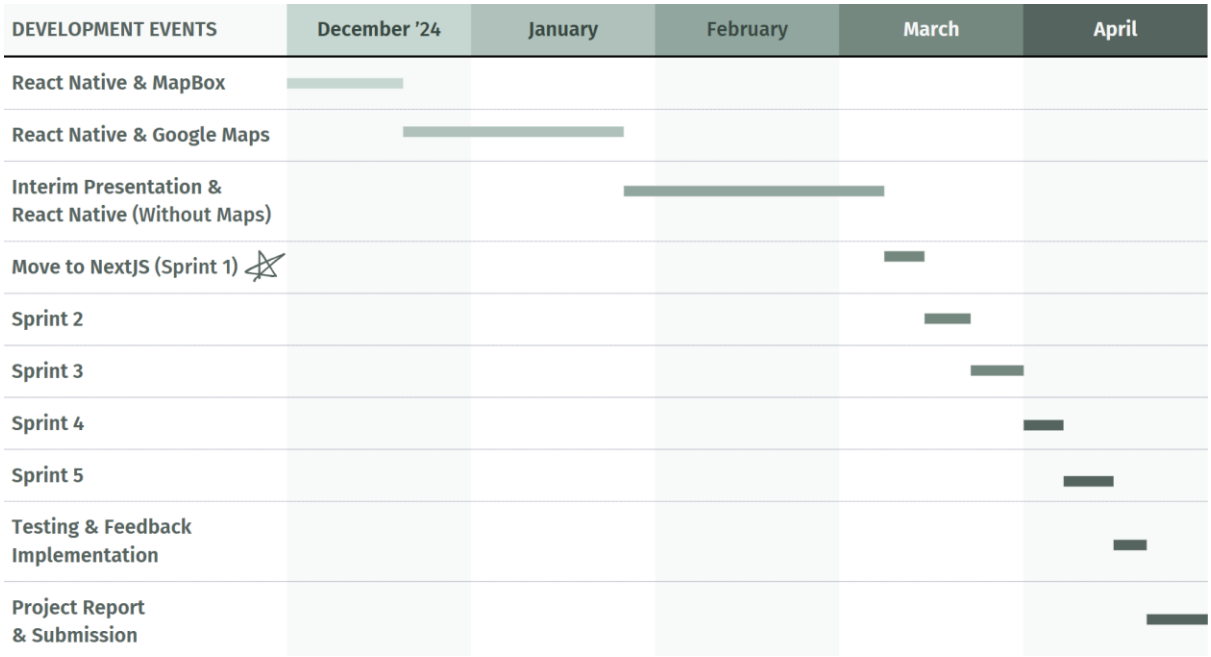


Figure 33 Gantt Chart of Project Phases

Introduction

This chapter covers the implementation of the application. The following technologies were used to build the application:

- **Next.js**
  Front-end React framework used to build the face of the application
- **Firebase RTD (Realtime Database)**
  The BaaS used to store data
- **Firebase Authentication**
  Used to authenticate the user. Registration and logging in/out of the application
- **Google Places API**
  Google API service that autocompletes text based on the input (like in Google Maps)

- **Google Maps JavaScript API**
  Used to load and display the map on the screen
- **Google AI Studio (Gemini 1.5 Flash)**
  Gemini 1.5 Flash is the model used in the Chatbot
- **Tailwind CSS**
  A very popular, Open-source CSS framework that provides utilities which can be used across the entire application
- **ShadCN**
  Component library built on top of CSS, used to style the application alongside Tailwind CSS

Once the implementation is finished, this application will be an itinerary planner PWA with a focus on user experience. This project aims to assist users by providing them with an app that allows them to plan an itinerary and open that trip in Google Maps. The user will also be able to save that trip as a PDF that they can keep and share with others. This application has a 'Discover' section that allows the user to browse through attractions and BnBs provided by Fáilte Ireland and Irish music events provided by TheSession. They are able to save items to their list for later. Additionally, there is a chatbot in the view page of each itinerary that the user can interact with.

Scrum Methodology

During the implementation phase of this project, the Scrum methodology was used. It is a project management methodology, usually applied when there is a large team working on a project. It aims to help the team manage the workload and keep track of what goals need to be achieved in a specified timeframe. "Sprints" are usually a week or two during which the team has a goal that needs to be reached. As mentioned before, there are five sprints in the implementation period. Each sprint lasted a week.

The benefit of using this in this project is that each week throughout this entire project, a meeting was held with the main supervisor. The progress was reviewed with the supervisor, and feedback was provided based on the week of work that was done. Figure 34 displays a simple scrum methodology, although in this project it was for the author and main supervisor.

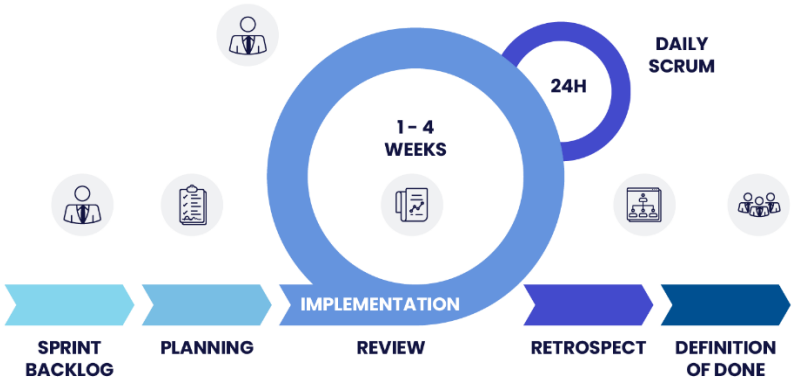


Figure 34 SCRUM Methodology diagram

When the sprints began, some of the basic code was moved from React Native, which increased the speed of development. In sprint one, it will be explained in more detail. As each sprint went by, a weekly scrum meeting was held.

Development environment

As mentioned previously, Visual Studio Code is my IDE. It provides a lot of plugins, such as Prettier, which formats code to make it more readable, and ES7+ React/Redux/React-Native snippets, which complete small code snippets. Both are used throughout the implementation.

For the project's version control, GitHub was used. After major implementations, the code was pushed to the repository.

After the application was deployed on Vercel, it was tested on Android, desktop and iOS. The installation of the application has been successful on all three devices and works as intended. The only inconvenience spotted on iOS is the gradient used in the hero which is the solid background colour for the text, does not show up on iOS devices.

Sprint 1

Goal

The main goal of this sprint is to build an authentication system with Firebase Authentication, which allows users to register and log in/out of the application. Next, moving the authentication context from the React-Native project was important since changes had to be made to make it work in Next.js, which requires debugging. Additionally, setting up the realtime database was an important goal for this spring. Finally, setting up basic PWA configurations to be able to install the application on devices.

Next.js and TypeScript

The first step was to set up the development environment. TypeScript was used; it is JavaScript that has strict type rules. The benefit of using TypeScript is that it makes the project a bit more secure by catching type errors early on, thus reducing errors (related to variable types).

To create the project, the following command was run in the terminal of a folder:
npx create-next-app@latest

This command created a starter Next.js project with TypeScript pre-installed and using App Router. The project contained files/folders such as the app folder (where the app itself lives), next.config.js file, tsconfig.js (TypeScript configuration file), package.json, .gitignore (where files/folders are stated to be ignored when pushing to GitHub) and the public folder, to name a few.

Firebase Database and Authentication

Firebase provides a universal firebaseConfig.js code snippet, which configures Firebase in the project. Figure 35 shows the configuration details; lines 2 and 4 are the services imported that will be used throughout the project. The “process.env.NEXT_PUBLIC_FIREBASE_[key name]” are the keys that are safely stored in the .env.local file. This is to prevent those keys from being pushed to GitHub, where the keys are unprotected.

```
JS firebaseConfig.js > ...
1   import { initializeApp } from 'firebase/app';
2   import { getAuth } from "firebase/auth";
3
4   import {getDatabase} from 'firebase/database';
5
6   // initialise Firebase
7   const firebaseConfig = {
8     apiKey: process.env.NEXT_PUBLIC_FIREBASE_API_KEY,
9     authDomain: process.env.NEXT_PUBLIC_FIREBASE_AUTH_DOMAIN,
10    databaseURL: process.env.NEXT_PUBLIC_FIREBASE_DATABASE_URL,
11    projectId: process.env.NEXT_PUBLIC_FIREBASE_PROJECT_ID,
12    storageBucket: process.env.NEXT_PUBLIC_FIREBASE_STORAGE_BUCKET,
13    messagingSenderId: process.env.NEXT_PUBLIC_FIREBASE_MESSAGING_SENDER_ID,
14    appId: process.env.NEXT_PUBLIC_FIREBASE_APP_ID,
15    measurementId: process.env.NEXT_PUBLIC_FIREBASE_MEASUREMENT_ID
16  };
17
18
19  const app = initializeApp(firebaseConfig);
20  export const firebase_auth = getAuth(app);
21
22  export const db = getDatabase(app);
```

Figure 35 firebaseConfig.js File

Lines 20 and 22 export the variables that allow the use of Firebase services elsewhere in the code. Figure 36 is an example of how to import the variable.

```
5   // FIREBASE //
6   import { db } from '@/firebaseConfig'
```

Figure 36 Importing database to use inside the file

The AuthContext.js file was moved from the old React Native project. Originally, that code was taken from @aaronksaunders on GitHub[9] and later modified to work in Next.js. This file provides a context for the application, acting as some sort of shield to allow through authenticated users and keep out everyone else. The reason I chose this repo is that all the functions I needed for my application, for example, the log in, log out, and register functions, were already there. Aaronksaunders provides good comments throughout the code, which helped me to understand what was going on. Throughout the project, some of the code had to be changed to work in Next.js, which means that the comments do not exactly fit the actual code.

[9] https://github.com/aaronksaunders/firebase-exporouter-app/tree/main

Figure 37 shows the code that renders the context for the project.

```
225     return (
226         <AuthContext.Provider
227             value={{
228                 signIn: handleSignIn,
229                 signUp: handleSignUp,
230                 signOut: handleSignOut,
231                 user,
232                 isLoading,
233             }}
234         >
235             {props.children}
236         </AuthContext.Provider>
237     );
```

Figure 37 Rendering AuthContext functions

The values are the functions that can be used within the AuthContext provider inside the project. These functions can be imported and used anywhere within the application as long as that file is within the <AuthContext.Provider>. This context is exported as SessionProvider. In most cases, it is placed at the very root of the app files; in this case, it is in the layout.tsx, which is the RootLayout of the project. It encapsulates the entire project. Figure 38 is how the SessionProvider wraps the application {children}. The children will be able to access the functions shown in Figure 37.

```
48                 <SessionProvider>
49                   <ServiceWorkerRegister />
50                   <ThemeProvider
51                     attribute="class"
52                     defaultTheme="light"
53                     enableSystem
54                     disableTransitionOnChange
55                   >
56                     {children}
57                   </ThemeProvider>
58                 </SessionProvider>
```

Figure 38 Authentication protection for the entire project

Next was the database implementation. Since the initialisation was completed, it was only a matter of importing the db variable (line 22 in the fireConfig.js). Figure 39 is how the application talks to the database.

```
54          try {
55              await signIn(user.email, user.password);
56              router.push("/itinerary");
57          } catch (err: any) {···
75          } finally {
76              setIsSubmitting(false);
77          }
```

Figure 39 User Sign In code snippet

This try/catch is inside the handleSubmit function, which the user triggers once they click the “Sign in” button. It is an asynchronous function, which is why there is an “await” on line 55. The “signIn” following the await is the function imported from the AuthContect.tsx, the two values are the credentials used to sign the user into the application.

Next, the catch is catching any errors, which are being converted into user-readable text, and finally, setIsSubmitting is being set to false. This changes the Boolean of that variable which indicates to the user that they are signed in. When that Boolean is set to true, the user sees a “Logging in…” text.

This project has quite a lot of input fields. Figure 40 shows how those input fields are being set.

```
101                 <Input
102                     placeholder="Email address"
103                     value={user.email}
104                     onChange={handleChange}
105                     className={` ${error && !user.email ? 'border-red-500
                        focus-visible:ring-red-500' : ''}`}
106                     name="email"
107                     type="email"
108                     autoComplete="email"
109                     disabled={isSubmitting}
110                 />
```

Figure 40 Email Input Field in signIn.tsx

The Input component is the ShadCN component; later in the sprints, this will be explained. The value parameter is where the actual value of that input field is set. onChange is where the value of the input field becomes the value parameter. First, onChange is triggered, then the value becomes user.email. The last parameter is disabled={isSubmitting}; when this is activated, the user sees the “Logging in…” text.

Figure 41 is the handleChange code that sets the value the user put in.

```
21  const handleChange = (e: { target: { name: string; value: string; }; }) => {
22      const { name, value } = e.target;
23      setUser(prevState => ({
24          ...prevState,
25          [name]: value
26      }));
27
28      if (error) setError("")
29  }
```

Figure 41 handleChange function across the entire application

Firstly, the e.target is an event target of a user’s interaction with the event handler function. Target is the input element that has been changed by the user; when they type in the email field, target becomes email. The same goes for the password and every other input field. The target was defined as an object that contains a name and value, which are both string variables (this is TypeScript, setting a type for each variable). In line 22 (Figure 41), the name and value become the e.target, this allows for an easier understanding of what is happening with each variable when setting the setUser. prevState is taking the previous state and setting the name of that targeted variable and changing it to value. This way, for each field, the value is dynamically changed.

PWA configuration

To create a Progressive Web App, a manifest file is required.

```
{
    "name": "Buckle Up",
    "short_name": "Buckle Up",
    "description": "An Itinerary management app built in NextJs",
    "start_url": "/",
    "display": "standalone",
    "icons": [ ···
    ],
    "screenshots": [ ···
    ],
    "form_factor": "wide",
    "theme_color": "#008d5c",
    "background_color": "#e5eee9",
    "prefer_related_applications": false
}
```

Figure 42 manifest.json file for PWA configuration

This manifest.json file (Figure 42) contains the basic configuration details of the app. The JSON contains keys and values, such as the name of the app, description, start_url (location of where the app will initially open), icons and screenshots (displays the favicon and loading screen when the application is being installed), theme_color and many more. Next.js provides a package called “next-pwa”. After it was installed, it was configured in the next.config.js file, as demonstrated in Figure 43.

```
const withPWA = require("next-pwa")({
    dest: "public",
    register: true,
    skipWaiting: true,
    scope: "/app",
    sw: "service-worker.js",
    disable: process.env.NODE_ENV === "development",
    runtimeCaching: [
        { ···
        },
        { ···
        }
    ]
});
```

Figure 43 next-pwa setting up pwa in next.config.js

The code in Figure 43 sets up the application to become a PWA. Once the project runs in build mode, it generates a service-worker.js file. That file handles the precaching of essential assets, runtime caching with specialised strategies (caching Firebase and Google Maps data) and the handling of network requests (to Firebase and Google Maps).

```
14      runtimeCaching: [
15          {
16              // Cache Firebase data for offline use
17              urlPattern: firebaseUrlPattern,
18              handler: "NetworkFirst",
19              options: {
20                  cacheName: "firebase-data-cache",
21                  expiration: {
22                      maxEntries: 50,
23                      maxAgeSeconds: 7 * 24 * 60 * 60 // 7 days
24                  }
25              }
26          },
27          {
28              // Google Maps resources -  limited offline capabilities
29              urlPattern: /^https:\/\/maps\.googleapis\.com\/.*/,
30              handler: "StaleWhileRevalidate",
31              options: {
32                  cacheName: "google-maps-cache",
33                  expiration: {
34                      maxEntries: 50,
35                      maxAgeSeconds: 3 * 24 * 60 * 60 // 1 day
36                  }
37              }
38          }
39      ]
```

Figure 44 Caching Firebase and Google Maps data

runtimeCaching (as shown in Figure 44) displays how and for how long the data is being stored. The urlPattern sets the URL where the data comes from. Handler is a caching strategy. There are a few types: NetworkOnly, NetworkFirst, StaleWhileRevalidate, CacheFirst and CacheOnly, to list a few. NetworkFirst is used for the Firebase data, which means that it will only use the cache if the app is offline. StaleWhileRevalidate is set for maps, meaning that it displays the cached data while it fetches the up-to-date data.

Figure 45 shows the manifest options displayed in the Application tab, which can be viewed in the developer tools.

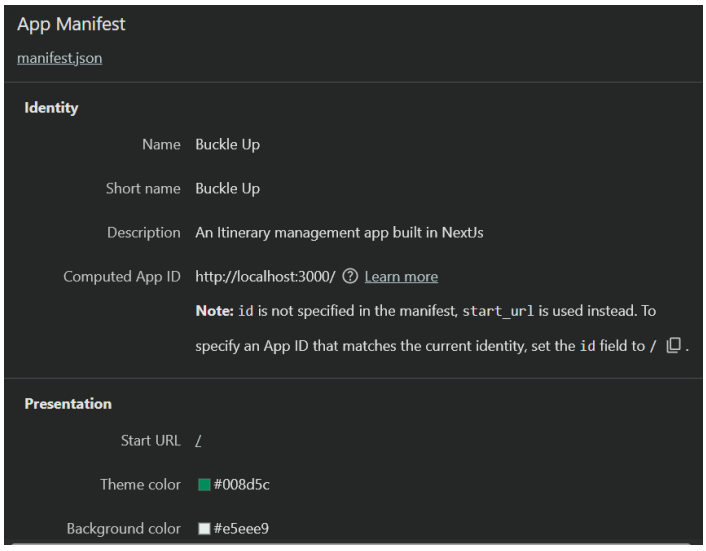


Figure 45 Application tab in DevTools - Manifest file

Sprint 2

Goal

During the second week, the goal was to build up the CRUD functionality, create, read, update and delete. Most of the code was transferred from React Native, but it had to be adjusted to make it work in this application. Some items, such as the draggable stops (when creating an itinerary), are repeated, so creating them into a component was a part of the task.

Create and update

The itinerary creation was designed to be intuitive, which comes with its own challenges. Creating all the functionality included;

- splitting the itinerary into 4 steps
- generating itinerary days for the selected date range
- draggable items (which will be explained in the next step),
- adding/removing stops and days,
- Instead of having one long page the user needs to scroll through, it was broken up into 4 steps: title and flights, selecting dates, itinerary planning and notes. A step state was created to handle which step the user was on. Figure 46 displays how that state was created and used. The step is defaulted to 1 as the user will start there. nextStep and prevStep are activated when the user clicks the Next or Back buttons, which are displayed in Figure 47.

```
50      const [step, setStep] = useState(1);
51      const nextStep = () => setStep(step + 1);
52      const prevStep = () => setStep(step - 1);
```

Figure 46 Step States

```
<div className='ms-auto'>
    <Button onClick={nextStep} className='w-40'>Next</Button>
</div>
```

Figure 47 Example of the "Next" button

If the user wants to add flight information, they are offered to insert a time. For this, a library called react-datepicker [10]was installed. They provide all sorts of functionality for times and calendars, but only the time elements were used. They have a lot of demonstrations and examples of how to use their library, which was very helpful.

```
<DatePicker
    selected={new Date()}
    onChange={onChangeTime}
    showTimeSelect
    showTimeSelectOnly
    timeIntervals={15}
    timeCaption="Time"
    dateFormat="h:mm aa"
    inline
    minTime={getTimeRange(selectedSlot).minTime}
    maxTime={getTimeRange(selectedSlot).maxTime}
/>
```

Figure 48 Time selection box

The code in figure 48 shows how time selection boxes are displayed. The last two lines set a range that the time can be selected between, which is used in the itinerary creation. The morning period is from 12 am to 12 pm, the afternoon is until 6 pm, and the evening is till midnight. When the user selects a time, the onChangeTime is triggered, and that value is inserted into the itinerary day-period-stop-time slot. This works the same for the flight time information.

The next step was to generate the itinerary dynamically when the user selects a range of dates. A component was created for the calendar since it is used in both create and edit. ShadCN has a calendar component which is built on top of react-day-picker. The DateRangePicker.tsx file handles the calendar view and the dates selected. During development, this calendar ended up having problems, which will be explained later in the paper. Figure 49 is how the component was imported into the create and edit pages.

```
<DateRangePicker
    dateRange={dateRange}
    onDateRangeChange={handleDateRangeChange}
/>
```

Figure 49 Calendar component

A generateDays function was created along with the calendar to automatically generate the amount of days for the itinerary. Two values (dates) are received; a calculation is made to get the number of days, after which that number of days is generated in the itinerary. Figure 50 shows how that function works.

[10] https://reactdatepicker.com/

```
94  // Function to generate itinerary days
95  export const generateDays = (start: string, end: string): Record<number, any> => {
96      const startDate = new Date(start);
97      const endDate = new Date(end);
98      const numDays = Math.ceil((endDate.getTime() - startDate.getTime()) / (1000 * 60 * 60 * 24)) + 1;
99
100     const newDays: Record<number, any> = {};
101     for (let i = 1; i <= numDays; i++) {
102         newDays[i] = {
103             morning: [{ name: "", time: "", notes: "" }],
104             afternoon: [{ name: "", time: "", notes: "" }],
105             evening: [{ name: "", time: "", notes: "" }],
106         };
107     }
108     return newDays;
109 };
```

Figure 50 generateDays function

The number of days is calculated on line 98, which is looped in line 101. newDays is then returned with the itinerary containing the three periods, which contain a name, time and notes for each period. When the user goes to the next step, the itinerary is generated using newDays.

The following step is to make the itinerary functionality. A file called itineraryUtils.ts was created that contains the adding and removing of stops and days from the itinerary and updating the stop location for the draggable stop functionality. This was all put into one file since the code is identical for both create and edit. All these functions have one thing in common: they change the values/structure of the itinerary. The user can add and remove stops, move the stop (draggable) and remove entire days. All of these make changes to the itinerary, which is why these functions were grouped. An example of one of these is the moveStop. It was the trickiest since the item (stop) can be moved around within its array, and the index changes. The JS “splice” method was used to achieve this functionality, which is displayed in Figure 51.

```
26  // Move a stop within the same day and period
27  export const moveStop = (
28    itinerary: TripType,
29    day: number,
30    period: keyof DayType,
31    fromIndex: number,
32    toIndex: number
33  ): TripType => {
34    const stops = [...itinerary.days[day][period]];
35    const [movedItem] = stops.splice(fromIndex, 1);
36    stops.splice(toIndex, 0, movedItem);
37
38    return {
39      ...itinerary,
40      days: {
41        ...itinerary.days,
42        [day]: {
43          ...itinerary.days[day],
44          [period]: stops,
45        },
46      },
47    };
48  };
```

Figure 51 moveStop function in itineraryUtils.ts

Lines 34 through 36 take the stop, locate the original index, remove it from there, and then replace it with the new index (which is the one the user is dragging over). Once that is done, the itinerary is returned with the updated stops. The rest of the functions within the itineraryUtils.ts file perform similar functionality.

Most functionality can be moved to the edit page; the only difference is populating the existing fields with data coming from Firebase. It was a challenge to populate the calendar with existing dates from the database. This was a confusing task since there are 2 types of dates, the ones in the database (start_date and end_date) and the ones used in the functions (startDate and endDate). Since both need to be integrated into a prefilled edit form, they had to be set at the very beginning of the page when the data was just being called. This is shown in Figure 52.

```
85      // Setup date states
86      if (tripData.start_date && tripData.end_date) {
87          const startDate = new Date(tripData.start_date);
88          const endDate = new Date(tripData.end_date);
89
90          setDateRange({
91              from: startDate,
92              to: endDate
93          });
94
95          setMarkedDates(generateMarkedDates(
96              tripData.start_date,
97              tripData.end_date
98          ));
99      }
```

Figure 52 Setting dates/days for the calendar/generateDates

Lines 87 to 93 set the dates that will be used to generate the days for the itinerary; the rest of the lines set the date ranges that will display on the calendar to mark the date range.

If there were more time, a lot of cleanup could be completed in these files, since they are big. More functions could be moved to smaller files.

DraggableStop.tsx

This component function is how the user can drag and drop each stop within a time period. For example, if a user has 5 stops during the morning period, they can drag and drop it around within the morning period, but they cannot drop it outside of that time period.

A library called “react-dnd”, a React drag and drop library. Figure 53 shows how to import the hooks that the library provides.

```
2   import { useDrag, useDrop } from 'react-dnd';
```

Figure 53 Importing drag and drop functions from react-dnd

This library provides good documentation[11], which contains many examples, including a GitHub repository with examples. After reviewing the documentation and examples, it was time to implement. Since there are many props being passed through into the function, each one is assigned a type. Figure 54 is a screenshot of the DraggableStopProps.

```
8   type DraggableStopProps = {
9     day: number;
10    period: "morning" | "afternoon" | "evening";
11    index: number;
12    stop: StopType;
13    updateStop: (day: number, period: any, index: number, field: keyof StopType, value: any) => void;
14    removeStop: (day: number, period: any, index: number) => void;
15    toggleNotes: (day: number, period: any, index: number) => void;
16    showNotes: Record<string, boolean>;
17    setSelectedDay: (day: number | null) => void;
18    setSelectedSlot: (slot: "morning" | "afternoon" | "evening" | null) => void;
19    setSelectedEntryIndex: (index: number | null) => void;
20    setShowTimePicker: (show: boolean) => void;
21    moveStop: (day: number, period: any, fromIndex: number, toIndex: number) => void;
22    updateStopLocation: (day: number, period: any, index: number, placeData: any) => void;
23  };
```

Figure 54 Types for the DraggableStop props

Each prop is assigned a type, which means the value for that prop must be that exact type. For example, “period” must be one of the strings on line 10. Other props can contain a few items, such as updateStop on line 13, each one having a type. Some of the props return a type “void”, which means they do not return anything.

Next, the actual component. Figure 55 shows the component being initialised and exported.

```
25    // Draggable Stop component
26  > const DraggableStop: React.FC<DraggableStopProps> = ({ ···
160   }
161   export default DraggableStop;
```

Figure 55 Initialising and Exporting DraggableStop function

React.FC is the ‘React Function Component’. It is a TypeScript type which is built into React itself. What React.FC does is take the type of another prop as its type. In this case, it takes DraggableStopProps as its type.

[11] https://react-dnd.github.io/react-dnd/docs/api/hooks-overview

Onto the drag and drop functionality. useDrag and useDrop are hooks which are used to create the functionality. Figure 56 shows the implementation for the drag functionality.

```
46  // Setup drag and drop
47  const [{ isDragging }, drag] = useDrag<DragItem, unknown, { isDragging: boolean }>({
48    type: ItemTypes.STOP,
49    item: { day, period, index },
50    collect: (monitor) => ({
51      isDragging: monitor.isDragging(),
52    }),
53  });
```

Figure 56 useDrag functionality in DraggableStop.tsx

In this snippet, DragItem and ItemTypes are imported from the types-file, which contains types for each variable throughout the project. isDragging and drag are the return values of useDrag. Drag is used to target the DOM element; in this case, it is the specific stop the user is aiming to drag. For a better user experience, isDragging is used to display to the user which item is being dragged (it is used for styling). Type, item and collect are the three specification object members being used. Together, they take the data from the current stop and set the isDragging to true until the item is dropped.

```
55  const [{ isOver }, drop] = useDrop({
56    accept: ItemTypes.STOP,
57    hover(item: DragItem) {
58      if (!ref.current) return;
59
60      // Only handle items from the same day and period
61      if (item.day !== day) return;
62
63      const dragIndex = item.index;
64      const hoverIndex = index;
65
66      // Don't replace items with themselves
67      if (dragIndex === hoverIndex) return;
68
69      // Call moveStop to handle the actual array movement
70      moveStop(day, period, dragIndex, hoverIndex);
71
72      // Update the index for future drag operations
73      item.index = hoverIndex;
74    },
75    collect: (monitor) => ({
76      isOver: !!monitor.isOver(),
77    }),
78  });
```

Figure 57 useDrop functionality in DraggableStop.tsx

Figure 57 shows the useDrop function, which works in a similar style to the useDrag. The first line is the return items for useDrop. Next, only items of type “STOP” will be accepted. Line 57 is the callback function when the item is hovering over another item, this is where items are restricted to be dropped in one area (within the same period, line 61). Collect updates the state of the entire component, which determines whether an item is being dragged over another item.

'Refs' are used to get the reference of the exact item that is currently being dragged. useRef is a React hook to be able to get the DOM element of the current item, which is then used in the DnD functionality. Figure 58 shows the ref being initialised, which is then utilised in the div (Figure 59).

```
42    const ref = useRef(null);
```

Figure 58 ref initialisation

```
87    <div
88      ref={ref}
89      className={`mb-2 p-2 relative ${isDragging ? 'rounded-xl  border border-muted' : ''} ${isOver ? 'rounded-xl bg-muted  ' : ''}`}
90    >
```

Figure 59 ref being utilised in the main div element

Next, this component is imported into the create and edit pages, where it gets looped. All the necessary parameters are passed where the functionality is carried out. The screenshot in Figure 60 shows the looping, draggable stop in those pages. The first line is a map function which loops through an array of the time period (morning, afternoon and evening), and for each of those, the draggableStop is created. The component receives all the necessary prop values (lines 537 to 551). Earlier in the sprint, it was explained what each one does.

```
535       {data[period as keyof DayType].map((stop, stopIndex) => (
536           <DraggableStop
537               key={stopIndex}
538               day={Number(day)}
539               period={period as keyof DayType}
540               index={stopIndex}
541               stop={stop}
542               updateStop={updateStop}
543               removeStop={removeStop}
544               toggleNotes={toggleNotes}
545               showNotes={showNotes}
546               setSelectedDay={setSelectedDay}
547               setSelectedSlot={setSelectedSlot}
548               setSelectedEntryIndex={setSelectedEntryIndex}
549               setShowTimePicker={setShowTimePicker}
550               moveStop={moveStop}
551               updateStopLocation={updateStopLocation}
552           />
553       ))}
```

Figure 60 Looping the DraggableStop component in Create and Edit pages

Other components were created that work similarly to this, which can be viewed in the GitHub repository, such as CleanItinerary.tsx, DateRangePicker.tsx and ItinerarySummary.tsx.

While working on the calendar in the DateRangePicker, I ran into issues with the version incompatibility. Originally, when creating this functionality, all was well. Throughout the development of the rest of the application, there were issues with the calendar. The ShadCN calendar is built on top of react-day-picker (a React library), and the latest version (9.6.3) is not entirely compatible with the latest version of React (version 19). It is not completely broken; the functionality of the calendar works, but the design of the calendar has changed, and it cannot be changed at all. I tried to find out how it stopped working, but there is no new information on this. I chose to leave the calendar the way it is because it still works, it just does not look the way I intended.

Sprint 3

Goal

The goal for this sprint was to get the Google APIs to work. This included the Places API and a map to display. The Places API provides all the names and locations of places. These details will be used to mark the stops on a map in the ItineraryMapView.tsx. To display a map, it must be loaded first; for that, a script is required to load the map once across the entire application.

Loading Google Maps

Google Maps services provide a JavaScript API, which is being used here to load the map in the application. Figure 61 is the code that checks if a map is already loaded in the application.

Maps > TS loadMapsAPI.ts > loadMapsAPI

```
1  export const isGoogleMapsLoaded = (): boolean => {
2      return (
3          typeof window !== "undefined" &&
4          window.google !== undefined &&
5          window.google.maps !== undefined
6      );
7  };
```

Figure 61 Function to check if map is already loaded

This code snippet returns a Boolean value (True or False) after checking if a map is loaded. These conditions check if the map's API is already loaded; if so, do not load it again.

Next is a Promise, which is an asynchronous operation that will eventually complete (either successfully or with an error). In this case, it will store a reference to the loading process, which means that it can keep track of whether the API is currently being loaded, return the same Promise to multiple callers (preventing duplicate loading) and reset this reference if loading fails. Figure 62 shows the loadPromise being initialised.

```
9  let loadPromise: Promise<void> | null = null;
```

Figure 62 loadPromise initialisation

Finally, the actual loadMapsAPI function is shown in Figure 63. Line 11 in this Figure shows the function that returns a Promise.

```
11  export const loadMapsAPI = (): Promise<void> => {
12
13      // return existing promise if its already loading
14      if (loadPromise) return loadPromise;
15
16      // return immediately if already loaded
17      if (isGoogleMapsLoaded()) {
18          return Promise.resolve();
19      }
20
21      loadPromise = new Promise((resolve, reject) => {
22
23          // script element
24          const script = document.createElement("script");
25          script.src = `https://maps.googleapis.com/maps/api/js?key=${process.env.
            NEXT_PUBLIC_GOOGLE_MAPS_API_KEY}&libraries=places,marker`;
26          script.async = true;
27          script.defer = true;
28          script.id = "google-maps-script";
29
30          // loading events
31          script.onload = () => resolve();
32          script.onerror = () => {
33              loadPromise = null;
34              reject(new Error("Google Maps API failed to load"));
35          };
36
37          document.head.appendChild(script);
38      });
39
40      return loadPromise;
41  };
```

Figure 63 loadMapsAPI function in loadMapsAPI.tsx

The comments explain what the first two conditions do (lines 13 – 19). Next, the actual Promise is made to handle the asynchronous loading process. A script is made to handle complex features of the application (the map). Lines 24-28 create the script that initialises the map by getting the URL (including the private key and libraries: places and markers for the map), async and defer improve performance (better UX), and lastly, the id is necessary to be able to reference this script elsewhere. Following are the event handlers on lines 31 – 35. Onload resolves the Promise, onerror resets loadPromise to null and rejects the Promise with an error. Finally, line 40 adds the script element to the document's head, which starts loading the JavaScript API.

Places API

PlacesAutocomplete.tsx is the file that autocompletes the input of the user whenever they enter a destination. First, the function checks if a map is already loaded or not; when it is loaded, the Places API is initialised, which is shown in Figure 64.

```
45      try {
46          const autocomplete = new google.maps.places.Autocomplete(inputRef.
            current, {
47              fields: ['formatted_address', 'geometry', 'name', 'place_id'],
48              componentRestrictions: { country: 'ie' },
49          });
50
51          autocomplete.addListener('place_changed', () => {
52
53              const place = autocomplete.getPlace();
54              console.log("Selected place:", place);
55
56              if (place && place.geometry && place.geometry.location) {
57                  const placeData = {
58                      name: place.name,
59                      address: place.formatted_address,
60                      location: {
61                          lat: place.geometry.location.lat(),
62                          lng: place.geometry.location.lng()
63                      },
64                      placeId: place.place_id
65                  };
66
67                  if (onChange) {
68                      onChange(place.name || '');
69                  }
70
71                  console.log("Formatted place data:", placeData);
72                  onPlaceSelect(placeData);
73              } else {···
75              }
76          });
```

Figure 64 Places API initialisation in PlacesAutocomplete.tsx

This “try statement” runs the Autocomplete function. It takes in the input from the user and once the user selects one of the options, the data is returned, such as name, address, location and placeId (lines 58 to 64).

Next, this component is imported into DraggableStop.tsx since this Autocomplete input is in every stop. Figure 65 displays how this component is being used.

```
96  <PlacesAutocomplete
97    value={stop.name}
98    onChange={(value: any) => updateStop(day, period, index, 'name', value)}
99    onPlaceSelect={handlePlaceSelect}
100   placeholder="Enter location"
101 />
```

Figure 65 PlacesAutocomplete Component in DraggableStop.tsx

Once the value changes, the user is presented with a drop-down list of places, shown in Figure 66.

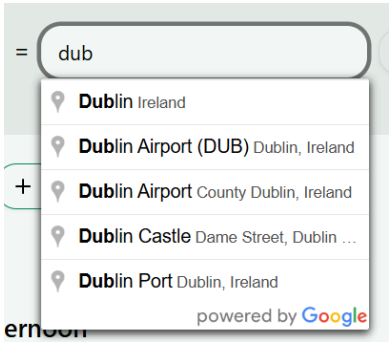


Figure 66 Example of Places API autocompleting text

Itinerary Map View

This component displays the map with all the markers on it, which comes from the database when the user has created an itinerary (provided they use the autocompleted places; otherwise, it will not work since there are no location coordinates to be placed on the map).

Figure 67 shows the marker.AdvancedMarkerElement, which is used to create markers on the map. It is a constructor that is provided in the library.

```
const newMarkers = allStops.map((stop, index) => {
    if (!stop.location) return null;

    // Create marker
    const marker = new google.maps.marker.AdvancedMarkerElement({
        position: stop.location,
        map: map,
        title: stop.name
    });

    // Add info window to show details when clicked
    const infoWindow = new google.maps.InfoWindow({
        content:
            `
            <div>
            <strong>${stop.name}</strong>
            <br>Day ${stop.day}, ${stop.period}
            ${stop.time ? `<br>Time: ${stop.time}` : ''}
            ${stop.notes ? `<br>Notes: ${stop.notes}` : ''}
            </div>
        `
    });

    marker.addListener('click', () => {
        infoWindow.open(map, marker);
    });

    return marker;
}).filter(Boolean) as google.maps.marker.AdvancedMarkerElement[];
```

Figure 67 Initialising marker for the map

While first creating these markers, google.maps.Marker[] was used, but as of February 21st 2024, it was deprecated. It was recommended to use the up-to-date google.maps.marker.AdvancedMarkerElement, but at this point, the map still did not work. The error said to include the MapId otherwise, the AdvancedMarkerElement will not work. After finding the mapId in the Google Maps console and adding it to my code (Figure 68), this fixed the problem, and the makers worked perfectly on the map.

```
mapId: process.env.NEXT_PUBLIC_GOOGLE_MAP_ID || undefined
```

Figure 68 MapId required for Map to work

Sprint 4

Goal

This week's sprint focused on functionality, such as being able to open the itinerary in Google Maps and download the itinerary to PDF. Another major goal is to create a chatbot which allows the user to communicate with it about the trip.

Chatbot

services.ts integrates with Google's Gemini AI model (1.5 Flash). This file creates the prompts for the chat. One of the most important functions is to be able to send a message. This function is displayed in Figure 69.

```
83  // https://ai.google.dev/gemini-api/docs/text-generation?lang=node#generate-text-from-text
84  // Send a message to the Gemini model and get a response
85  export async function sendMessage(message: string, tripContext: TripContext) {
86      try {
87          const systemPrompt = generateSystemPrompt(tripContext);
88          const model = genAI.getGenerativeModel({ model: "gemini-1.5-flash" });
89
90          const result = await model.generateContent([systemPrompt, message]);
91          const response = result.response;
92          // console.log(result.response.text());
93
94          return response.text();
95      } catch (error) {
96          console.error("Error sending message to Gemini:", error);
97          throw error;
98      }
```

Figure 69 SendMessage function in services.ts

This function receives a message and the trip context. The trip context goes through the generateSystemPrompt function, which returns the context of the chat for that exact trip. Next is the model variable that is calling the model in line 88. That model is then used to create a response, which is stored in response, which is returned to the user as response.text().

Two other functions generateSystemPrompt and getChatSession are created to make the functionality. ChatMessage, TripContext and sendMessage are used in the ChatContext.tsx file.

Google AI Studio provides models that are free to use with certain limitations. The Gemini 1.5 Flash latest release was in September 2024, which is recent enough. It is stable and is able to complete all the necessary tasks for this project. Figure 70 is a screenshot from ai.google.dev [12] of the capabilities of this model.

[12] https://ai.google.dev/gemini-api/docs/models#gemini-1.5-flash

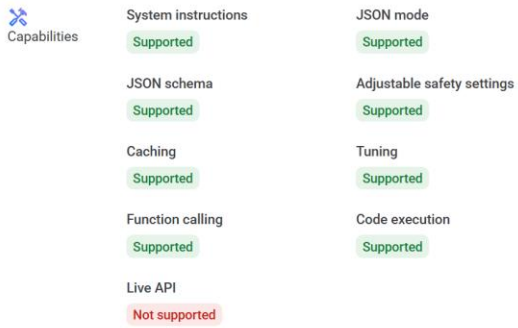


Figure 70 Google Gemini 1.5 Flash capabilities

The more recent models are still unstable and experimental. This is why the older, more stable model was chosen for these tasks. The user should be able to ask basic questions about the trip, such as “What are the top 3 things to do in Dublin?” and things like that.

To start the process, a chat context is required, which wraps around the actual chat component. ChatContext.tsx holds the ChatProvider and useChat functions.

ChatProvider is a function that provides all the parts to the chat, such as messages, tripContext and more shown in Figure 71. These states and functions are needed to display the starter messages and the context of the chat to the user when they open the chat.

```
89      const contextValue = {
90          messages,
91          isLoading,
92          tripContext,
93          setTripContext,
94          sendUserMessage,
95          clearMessages
96      };
97
98      return (
99          <ChatContext.Provider value={contextValue}>
100             {children}
101         </ChatContext.Provider>
102     );
```

Figure 71 ChatProvider return values

A hook is created to be able to access the context quickly, displayed in Figure 72.

```
105 export function useChat() {
106     const context = useContext(ChatContext);
107     if (context === undefined) {
108         throw new Error('useChat must be used within a ChatProvider');
109     }
110     return context;
111 }
```

Figure 72 useChat Hook

This hook returns the context, if the useChat is not wrapped in the context, it will throw an error on line 108.

The ChatComponent.tsx is where all the elements come together. The ChatComponent and the button to open that chat are located here. sendUserMessage is where the message of the user gets sent; when a response returns, it gets displayed on the screen.

```
326                 <ChatButton trip={trip} />
```

Figure 73 ChatButton in the itinerary view page

Figure 73 shows how the actual chat button component is used. The entire trip data gets sent to be able to give all the context possible to the model, such as the title, stops, locations of the stops and more.

Open in Google Maps

The first step is to check if there are any locations to be able to input into the URL.

```
11  // Check if there are any stops with location data
12  const hasLocations = Object.values(trip.days || {}).some(day => {
13    return ['morning', 'afternoon', 'evening'].some(period => {
14      const stops = day[period as keyof typeof day];
15      return stops && Array.isArray(stops) &&
16        stops.some(stop => stop.location !== undefined);
17    });
18  });
19
20  if (!hasLocations) {
21    return null; // Don't show the button if there are no locations
22  }
```

Figure 74 Checking for locations in the trip

The code snippet in Figure 74 is looping through all the days inside the trip; it is looking for the stops in each period of the day and is getting the location of each stop, since these will be used later for the URL.

Next is the URL, which will determine what the user will land on when clicking the button. Figure 75 is the code displaying the Google Maps URL and how it is handled if there is only one destination.

```
69    // Create Google Maps URL with waypoints
70    let mapUrl = "https://www.google.com/maps/dir/?api=1";
71
72    // If there's only one location, open it as a destination
73    if (stopsWithLocation.length === 1) {
74      const stop = stopsWithLocation[0];
75      mapUrl += `&destination=${stop.location?.lat},${stop.location?.lng}`;
```

Figure 75 mapURL and setting a destination

More likely, the user will have multiple destinations in their itinerary, which means all of them will go into the link. The limitation of the free version is that only 10 waypoints are allowed (including the origin and final destination). Figure 76 shows how the first 10 destinations into the URL, which the user will be able to open in Google Maps.

```
99        // Add waypoints (Google Maps limit is 10 waypoints in free version)
100       if (stopsWithLocation.length > 2) {
101         const waypoints = stopsWithLocation.slice(1, -1)
102           .slice(0, 8) // Limiting to 8 waypoints (plus origin and destination
              makes 10 total)
103           .map(stop => `${stop.location?.lat},${stop.location?.lng}`)
104           .join('|');
105
106         mapUrl += `&waypoints=${encodeURIComponent(waypoints)}`;
107       }
```

Figure 76 Adding a max of 10 waypoints to url

Convert to PDF

To be able to convert to a PDF, the jsPDF library was used. Figure 77 shows a small code snippet that shows how the PDF can be created. Line 44 creates a new instance of jsPDF, It is then possible to add all the information and styling to that PDF by using methods such as setTextColour and other similar methods. Due to the huge variety of methods (the jspdf file has 1460 lines) and the time constraints of the project, Claude (AI model) was used to create a basic design for the PDF; it helped get the names of the methods to design the PDF file and sped up the process of creating this functionality.

```
43      // Create the PDF document with custom styling
44      const pdf = new jsPDF();
45      const pageWidth = pdf.internal.pageSize.getWidth();
46      const pageHeight = pdf.internal.pageSize.getHeight();
47
48      // Add header with dark emerald green background
49      pdf.setFillColor(5, 102, 54); // #056636 - Dark emerald green
50      pdf.rect(0, 0, pageWidth, 30, 'F');
51
52      // Add title
53      pdf.setTextColor(255, 255, 255); // White text
54      pdf.setFont('helvetica', 'bold');
55      pdf.setFontSize(24);
56      pdf.text(title, 20, 20); // Left aligned
```

Figure 77 Creating and designing itinerary PDF in generateToPDF.tsx

The way the trip information is accessed is with document.getElementById. The element which contains the actual trip in the itinerary view page has an ID of "*itinerary-container*". This allows access to the information inside that element, which is used to create the PDF. Figure 78 displays how that element is accessed.

```
6       // Get the trip data from the DOM
7       const element = document.getElementById('itinerary-container');
8       if (!element) {
9           console.error('Itinerary container element not found');
10          return;
11      }
```

Figure 78 Accessing the element from the itinerary view page

The id is set like this (Figure 79).

```
<div
    id='itinerary-container'
    className="px-6 md:w-1/2 overflow-y-auto
">
```

Figure 79 Setting the element ID

Sprint 5

Goal

This week's goal was to finish up the functionality and design of the application. Another important goal is deployment with Vercel.

ShadCN implementation

This open-source component library is built using Tailwind CSS. ShadCN enables developers to use Tailwind on top of their components to restyle them. Alternatively, it is possible to redesign the component itself. This creates a very good user experience for the developer.

ShadCN provides excellent documentation for their components. They provide seamless installation with Next.js, there is also an option for manual installation (with Tailwind CSS), which can be used in other projects (like React Native, in the original project). To begin using a component, it needs to be installed in the terminal, for which they provide the command. Figure 80 is an example of how they provide the installation command for any of their components.

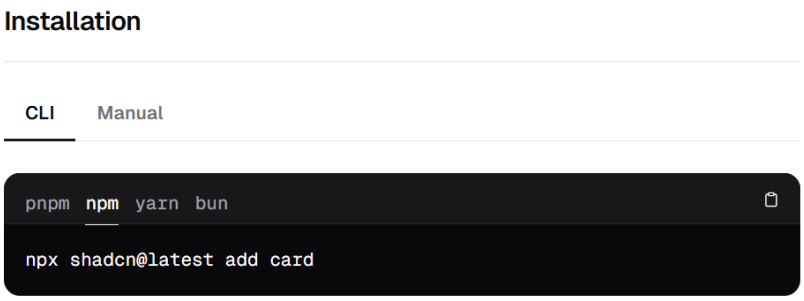


Figure 80 ShadCN component installation command

After running this command in the terminal of the project itself, it initially creates a UI folder, which will contain all the components. ShadCN provides the source code for every component it provides. This means that developers are free to redesign the components.

This component is very powerful because it is possible to add classes on top of the component itself. What is meant by this is displayed in Figure 81.

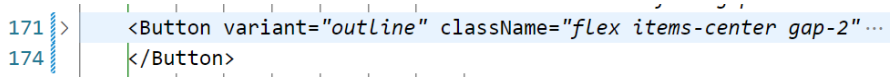


Figure 81 ShadCN Button component - restyled

That is a very simple button component. Adding that className prop to it allows developers to restyle it directly with Tailwind CSS. This principle applies to all the components.

```
<Card
    key={item.id}
    className='rounded-none -my-[8px] border-none bg-transparent shadow-none flex flex-col md:flex-row w-full'
>
    <Link ···
    </Link>
    <CardContent className='flex items-center gap-3 w-full md:w-1/4'>
        <CalendarDays className='w-6 h-6 text-primary' />
        <div>
            <p> {startDate} - {endDate} </p>
        </div>
    </CardContent>
    <CardContent className='w-full md:w-1/4'> ···
    </CardContent>
    <CardFooter className='w-full md:w-1/4'>
        <Button
            variant="default"
            className='md:ms-auto'
            onClick={() => { router.push(`/itinerary/${item.id}`) }}>
            View Itinerary
        </Button>
    </CardFooter>
</Card>
```

Figure 82 Redesigned ShadCN card component in the Home page

The code in Figure 82 is redesigning the card to look like the card in Figure 83.

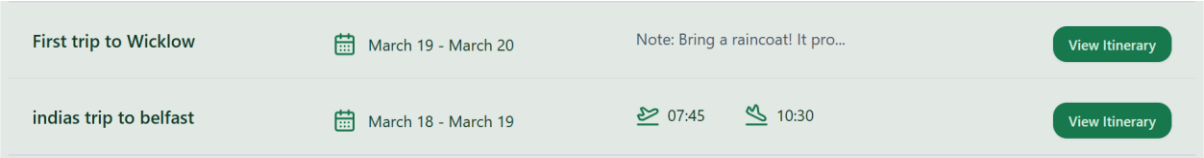


Figure 83 Cards displaying created itineraries

Since this application works on both mobile and desktop, the design process took a while due to the responsive design.

ShadCN use the OKLCH format for their theming. When I was working with jsPDF, there was an error due to the OKLCH format not being recognised, which is why it was necessary to create the design for the PDF from scratch.

Deployment with Vercel

Since this was not the first deployment to Vercel, it was a straightforward process. The project deploys using the GitHub repository, which means that any time there is a new push to GitHub, there will be a new deployment built automatically by Vercel. The link to the application [13] is available publicly.

[13] https://buckle-up.vercel.app/itinerary

Conclusion

This section has covered most of the building process for the application. Due to the time constraints of this project, the sprints were very intense. As mentioned in Sprint 2, most of the CRUD functionality was possible to move from the original project, which helped speed up the development process.

Each sprint covered a large section of the application. The first sprint set up the authentication and the backend services with Firebase. The context provider allows access to the pre-defined functions, such as login and logout. By the end of the sprint, it was possible to register, log in and log out of the application. The connections to the Firebase Realtime Database were also established, which led to the next sprint.

Sprint 2 focused on the CRUD functionality, which went swiftly since the code was moved from the original project. The create and update functionality contains a lot of states since there are a lot of different types of values being moved around, such as the time pickers (flight information and each stop in the itinerary), date pickers (automatically generates the days for the itinerary), and the itinerary can contain an unlimited amount of stops. Each stop has a name (Places API), time and note. Those stops can be dragged/reordered around within the same period (morning, afternoon and evening).

Sprints 3 and 4 contained the additional APIs from Google, such as Maps and Gemini. It was a learning curve trying to set up the map since it is not just inserting a map into the application. The map needs to be loaded (only once); if the map is loaded multiple times, it causes the map to be infinitely rendered. That is why the loadMapsAPI file has a Promise to check if the map is already loaded. The Gemini model integrated quite well, but again, due to the time constraints, I did not get the chance to explore more functionality for the chat. It would be nice to structure the responses better, for example, make bullet points if there is a list.

The final sprint was focused on redesigning the entire application. The responsive design took longer than expected, but considering the short sprint, I am pleased with the final results. Deployment with Vercel was swift. They provide good documentation as well, but due to using their services a few months prior, I remembered the steps to deploy the application.

To conclude the sprints, this application was completed quite quickly. A lot was learnt with the new technologies of Next.js, Google Maps, Google Gemini and many libraries, for example, react-dnd and ShadCN.

Testing

Introduction

The tests that were undertaken for this application consist of 2 types: functional and user testing.

Functional testing is testing the actual code. A framework called Jest was used. It is a JavaScript testing framework, which is maintained by Meta (formerly known as Facebook). Jest is one of the most popular testing frameworks, especially for React applications, due to its compatibility. It is also popular because it is easy to use and set up. Jest tests if the given input for the tests comes out with the expected output.

User testing, on the other hand, tests the usability. The point of this kind of test is to see what improvements need to be made to make the user experience as seamless and enjoyable as possible. The user will be able to point out any negative experiences or confusing interactions. Based on the results, improvements are made. At the end of the day, it is about the user since the application is for them.

Functional Testing

The functional tests that were carried out were the following:

- Authentication
- Draggable Component
- Map integration
- Itinerary display
- Chat context

The tests use Jest's mock capabilities to simulate dependencies like Firebase services and Google Maps, allowing isolated testing of component logic. The tests follow the black box testing approach by focusing on inputs and expected outputs rather than internal implementation details. Since Jest is used to test JavaScript, it does not provide support for testing DOM elements for interaction such as button clicks and component testing. For this reason, @testing-library/dom has to be installed to be able to perform those tests.

Authentication

Authentication is one of the core features since the user's data is protected. The main functionality is being tested here: log in, log out, registration and the status of each. The status is crucial to have since it communicates to the user what the problem is. This contributes to the user experience overall. If the user is not able to successfully complete a task without knowing the problem, they get frustrated and can give up on trying, which is a bad user experience.

Test No	Description of test case	Input	Expected Output	Actual Output
1	Login Success	Valid email and password	Object with authentication info	Object with authentication info
2	Login Failure	Invalid credentials	Error: "Invalid email/password"	Error: "Invalid email/password"
3	Logout Success	User session	Void (successful logout)	Void (successful logout)
4	Logout Failure	Error during logout	Error: "Logout failed"	Error: "Logout failed"
5	Registration Success	New user email, password, full name	New user object	New user object
6	Registration Failure	Existing email	Error: "Email already in use"	Error: "Email already in use"

Draggable Component

The draggable stop component is one of the key features of the itinerary creation process. That component contains a few buttons, two of which only contain icons (notes and delete). The responsiveness of those components is being tested. They are being tested to validate the functionality and mimic how the user would interact with them. Another reason to test these is that when clicking these buttons, it correctly triggers the appropriate callback functions with the right parameters (day, period, and index).

Test No	Description of test case	Input	Expected Output	Actual Output
1	Delete Button	Click on the delete button	removeStop function is called with the correct parameters	removeStop function is called with the correct parameters
2	Notes Button	Click on the notes button	toggleNotes function is called with the correct parameters	toggleNotes function is called with the correct parameters
3	Time Button	Click on the time button	Time picker setup functions are called	Time picker setup functions are called

Chat Context

The chat functionality is wrapped in a chat context; without it, it will not work. Here, the context setting, message handling, and error handling.

Test No	Description of test case	Input	Expected Output	Actual Output
1	Initialisation	New ChatProvider	Empty messages, no context	Empty messages, no context
2	Setting Context	Trip context data	Welcome message with user name and trip	Welcome message with user name and trip
3	Sending Message	User message text	AI response message	AI response message
4	Error Handling	Network error	Error message displayed	Error message displayed

Map integration

The map is another core feature of the app. Since there were issues with the map functionality before, I found it important to test it properly. Loading of the map is crucial, and multiple maps loading is problematic. This tests if the map is already loaded or if it needs to be loaded.

Test No	Description of test case	Input	Expected Output	Actual Output
1	Already Loaded	Maps API is already loaded	Immediate resolution	Immediate resolution
2	Loading	Maps API not loaded	Script element created with correct attributes	Script element created with correct attributes

Itinerary display

This section tests the different states of the data loading from Firebase. Like, if the user is authenticated or not and if there is data to be fetched or not. This is also important for user experience since the user gets clear feedback of the situation, if they are unauthorised or there are no itineraries to be displayed.

Test No	Description of test case	Input	Expected Output	Actual Output
1	No User	Unauthenticated state	Redirect to sign-in page	Redirect to sign-in page
2	Loading	Authentication in progress	Loading indicator	Loading indicator
3	Trip Data	Authenticated user	Trips are displayed with correct information	Trips are displayed with correct information
4	No Trips	No trip data	"No Trips Found" message	"No Trips Found" message

Results Overview

The test results for all of these tests have been successful, due to their simple nature. Mainly the functionality that interacts with the user has been tested, to give users accurate feedback to help better understand what problem they have ran in to (if any).

If there was more time, I would have run more tests, such as:

- End-to-end testing of the creation of an itinerary. Creating an entire itinerary and submitting it to a test database.
- Drag and drop functionality. Test that the drag-and-drop reordering of stops correctly updates the Firebase database.
- PWA offline functionality. Test to see that the application works correctly when offline and syncs when the connection is restored.

User Testing

User testing was completed once the application was ready to be tested. Several tests were carried out to see what improvements needed to be made. The following tasks were carried out:

- Registration
- Creating a short Itinerary (2 days)
  - Reorder stops within the itinerary

- View itinerary
  - Open Stops in Google Maps
  - Download itinerary
- Navigate to the BnBs page and save one
  - Open the official website of that BnB
- Navigate to liked items
- Download the application
- Open the downloaded application

The two users who completed testing for the application had different backgrounds, this was intentional to have the least amount of bias during testing and to get the most accurate results possible. Both users want to stay anonymous, so they will be named User 1 and User 2.

User 1

The first user is tech-savvy and enjoys trip planning. They have tried other apps to complete itinerary planning, but always ended up making a document or notes with their plans. After they found out what kind of test would be done, they were excited to try it out. Before the test began, the user was informed of the tasks they were about to complete. Notes were taken throughout the test to later make appropriate changes to the application.

Tasks

User 1 began the tasks with no problems, and creating the account was very straightforward. Landing on the itineraries page, the user took a second to scan the page. They noticed that the landing page had two different buttons for the same thing, a “create itinerary +” in the navigation bar and a “+ new itinerary” in the Hero. After navigating to the create page, they began to fill in the details for the trip. I noticed that they did not take the time to read the helper text in the box at the top of every create page. This is something to think about after the tests. They proceeded to select 2 days in the calendar, which automatically generated 2 itinerary days for them to fill in. The user took a second to analyse the buttons on each stop and tested out the notes button. They continued to add the stops and commented that it was nice to have the drop-down of places they could select from.

Halfway through the creation, they decided to click on the eye icon on the top right to see the information they had put in already. They added a few stops to their afternoon on the second day and reordered their days without any problems. After submitting the itinerary, they immediately located the itinerary they had just created and clicked on it. They double-checked the information and clicked on the markers on the map; they were happy to see the information displayed on the marker. They also stated that more data could be added to it, such as a link to Google Maps, to be able to navigate to it. They then clicked on the “Open in Google Maps” button. They pointed out that the button is only visible when you scroll to the bottom of the map; make it stick at the same level (as the chat bubble). They continued to

download the itinerary and opened it to see what information was there. After navigating back into the application, they clicked the “Discover” dropdown menu and selected “BnBs”. The user successfully completed the tasks by opening the link and saving it. They said that there probably should be a link to the “Saved Places” in the description of the page. Despite that, they navigated to the Saved Places page and reviewed the items. The user has had previous experience with PWAs, so they had no difficulty downloading the application. The PWA opened instantly without additional clicks. The user finished the tasks without major issues.

Conclusion

The user’s feedback was helpful and implemented after all testing was complete. The fact that this user has had experience with these things helped him point out extra little bits that can overall improve user experience, such as adding extra links throughout the application and point out the out of view button (even though I saw that the user did analyse the Itinerary View page first, so they did see the button there, but they chose to point it out anyway. I found this very helpful).

User 2

The second user was someone who did not have experience with these types of applications. Before the test began, they were explained what this application was for and that the point of the interview was to point out anything they found difficult, confusing, or any negative experience (because they had not done this type of test before). They had the list of tasks in front of them so they could follow at their own pace. Another important thing I mentioned to them is that there was no timer, so they were free to browse and look around, but the tasks were a guide and were advised to complete if possible.

Tasks

Same as the first user, registration was no problem. They examined the page and proceeded to click the create button in the Hero after reading the text. Unlike the first user, this user read everything on the card, which helped them proceed with the task. Only the title of the trip was filled in; they did not enter flight information. When creating the itinerary, this user had some trouble reordering the stops; they tried to put it in a different period of the day instead of the same period.

An important aspect the user pointed out is that the colour of the delete buttons did not look red (like most destructive actions/buttons), so they had to look twice to realise that those buttons would delete the day/stop. After they added a few more stops to the same period; they successfully reordered them. This showed that there should be a clear visual cue to show the user they cannot place it there. This user knew that not all the information needed to be complete to be able to submit it, since they are free to come back later and edit it. They found the itinerary they just created and clicked the “View itinerary” button. After scrolling through the itinerary, they opened it in Google Maps (they did not check the markers like User 1 did).

They checked the list of tasks and started looking for the “Download” button at the top of the itinerary, so this user needed help to find the button at the bottom of the itinerary itself. After they navigated back to the home page and looked for the BnBs section, they seemed to use the process of elimination, I watched their mouse move from one navigation item to another, then they clicked the discover item. They did not have any problems navigating through the BnBs page. They opened the “Booking.com” button and saved it for later. They found the Saved Places page successfully. When it came to downloading the application, they needed some help finding the button, which was outside of the application itself. This user was given an Android phone with the application already opened and logged in (using existing credentials), this was to test the installation process from a mobile phone, since a big “Install” drawer shows up when using it. They easily installed it then.

Conclusion

This user had a few extra struggles throughout the application, but again, they were helpful since the issues were fixed and the application became more user-friendly. The user stated after the test that the application was easier to navigate than a lot of other websites they have tried to use, and they ended up happy with participating.

Conclusion

In the best case scenario, testing 4-8 people would have been ideal, but time only allowed for two tests. Both users completed all the tasks that were given to them. The first user observantly flew through the tests since they have a lot of experience with technology and have an interest in planning. This user was able to point out minor but important details that were fixed once all results are analysed properly. The second user was more of a “control” test since they had never used similar technology before, so their feedback was crucial. The following improvements were made:

- Different create buttons – Navigation and Hero sections
- Placement of the "Open in Google Maps" button – make it stick like the chat button
- Add a link to Google Maps within map markers for direct navigation
- Add a link to "Saved Places" in the Discover pages
- Improve visual feedback for reordering stops
- Make destructive buttons red
- Make the Download button more discoverable

One of the most important things about this application is the simplicity of creating an itinerary. Both users had a good overall experience and provided constructive feedback, which is aimed at improving the overall user experience.

Project Management

Introduction

Throughout this section, the management of the entire project will be discussed. The phases begin from the project proposal to the requirements, design, implementation and testing. The SCRUM methodology will be discussed based on the project phases. Alongside that, the management tools will be discussed and how they were used within the project. The reflection will highlight the difficult challenges faced and what was learnt from them.

Project Phases

The project phases will be discussed in detail. As mentioned earlier in the Implementation section, the original project was intended to be built in React Native before the switch to Next.js. The original project[14] is half built and most of the functional requirements are completed. The proposal was created for the original project, but the final product does not match the proposal. This will be discussed throughout the next few sections.

Proposal

In total, two proposals created for this project, one at the beginning of Semester 1 and an updated proposal in Semester 2.

The first proposal was intended to build a mobile application in React Native using MapBox. Throughout the months of November and December 2024, multiple attempts were made to get MapBox to work as intended, but since MapBox stopped official support for React Native applications (it became a community-supported library), the map did not work. MapBox support was contacted, and many back-and-forth emails have been sent. The staff were very willing to help in any way they could, but it was not needed since Google Maps was the next solution.

The second proposal was adjusted after the interim presentation, which took place at the end of January with both supervisors. That proposal aimed to build a React Native application which allows users to create an itinerary and save it to the local device. As the development began for this project and the map integration kept failing, the decision was made to switch to Next.js mid-March. The basic idea of the project stayed the same; the user gets the same application, but the build of the application has changed.

[14] https://github.com/AgnePK/major-project

Requirements

Research was done to collect the requirements for the project. Similar applications were studied to get an understanding of what functionality is available for users. The reviews for those apps were also reviewed to assess the user experience of the application. Personas were another important step in creating the functional and non-functional requirements. Becky and Jackson (proto-personas) played a huge role in the creation of these requirements since the goal was to fix their problems. Once all the requirements were established, a use-case diagram was created to visualise how the personas would use these features. Feasibility was the last step before the design phase began. Numerous libraries claimed to be compatible with React Native, so the design phase commenced.

Looking back on this part of the project, more research should have gone into the compatibility research of the Expo framework and React Native libraries. While building the original project, an ejection (this means the application was no longer using the Expo framework and was bare React Native) was done to test if that was the problem; unfortunately, this did not help.

Design

The design phase consisted of 2 parts: Program design and User Interface design. The tools planned to be used in the project were listed and organised, database designs were created, and a general process was designed to get a better understanding of how the application will work (from a user's perspective).

Multiple LinkedIn courses were completed on paper prototyping, displayed in Figure 84.

Figure 84 LinkedIn courses on Paper Prototyping

These courses helped me better understand the benefits of creating paper prototyping when developing/designing an application. Paper prototypes were made for the original project, displayed in Figure 85. These were not in the User Interface Design section because these wireframes do not accurately represent the final result of the project.

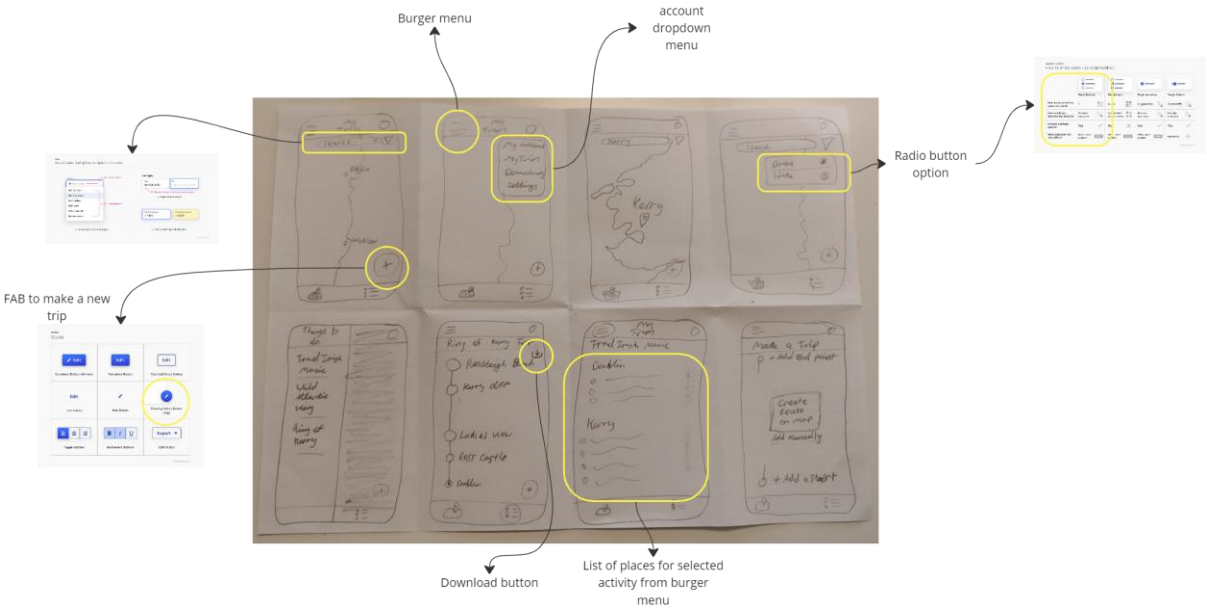


Figure 85 Paper Prototype for the original project

Based on the personas, empathy maps, journey maps, and storyboards were created. A user flow diagram was created to mimic how the personas/users would use the application. This helped during the implementation phase, which was coming up next. Being able to look at a blueprint of the system and what the steps were was beneficial in planning out the sprints.

Wireframes were created both for the original project and the current one. Since the first one was only for mobile phones, wireframes were created for mobile only. Once the switch was made to Next.js and the application became both desktop and mobile, a desktop version was created, which is covered in the UI design section. The design has changed over time with the new project implementation, which is why the mobile wireframes are not included in this paper, but they can be found in Figma.

ShadCN is a component library that was used for the application. They provide a generous list of components that developers are free to redesign if needed, since they provide the code. The colour theme is possible to change across the entire application through the global.css file. A green theme was chosen because it represents Ireland's eternally green landscape. Tailwind CSS is a utility-based design framework which provides sizing, colours, and anything that is needed to design an application. Instead of creating custom grids and spacing, I used the utilities from Tailwind CSS.

Implementation

There were two implementations, as shown in Figure 85. Once the switch was made to Next.js, there was a lot of time pressure due to the short amount of time left. The development was split into five sprints, in which all the logic and functionality were planned to be finished and later tested. Moving the CRUD functionality and authentication from the React Native project to this one did not take too long. Finding replacement libraries was the tricky part.

The drag-and-drop library was one of the more difficult implementations, but the documentation and their GitHub repo, which had examples, helped develop the functionality, along with a Medium tutorial[15]. The issue which arose with the calendar during development was not resolved. Research went into the versioning and compatibility with React, which showed that the versions are not compatible. To see if overriding the styling would solve the problem, I ended up using Claude to help solve the problem. I uploaded the file and explained the situation, after which it provided alternative ways to solve the problem. Through global.css or directly in the Calendar component, but none of which were successful. These attempts can be viewed in the GitHub repository.

The Google services (Maps and Gemini) that were used were implemented during the sprints. The chatbot was a non-functional requirement, but since there was momentum with the development and time allowed for another major feature, the Google Gemini AI was implemented. At the beginning, the documentation for Google services is quite overwhelming, and it is difficult to navigate, but the more I used it, the easier it became to navigate. Considering the difficulties during the development of the original project, because this one (most of the time) went smoothly, it was rewarding to get almost everything to work, especially within those 5 weeks (sprints).

Testing

The testing that was completed on this application resulted in success. The functional tests tested things like Contexts, buttons and functions (such as log in and register) were successful. The results were saved to the “test-results.txt” file, which can be reviewed line by line. Figure 86 reveals the results of the tests.

```
429   Test Suites: 6 passed, 6 total
430   Tests:       28 passed, 28 total
431   Snapshots:   0 total
432   Time:        4.944 s
433   Ran all test suites.
```

Figure 86 Functional test results

As discussed in the Functional Testing section, it would have been ideal to run more unit tests, but as already explained, time constraints only allowed so much. Jest and a React DOM testing library were used to complete all these tests.

User testing was conducted on two users. The results of those two tests were immensely helpful. For this type of project, it was beneficial to have two different types of users test the application to get fair and accurate results about the usability.

[15] https://medium.com/@liadshiran92/easy-drag-and-drop-in-react-22778b30ba37

It shows that if someone who has never used something similar can use it (for the most part), and that the application is successfully simple and works the way it was intended to. The results of these tests were listed in the User Testing section and were implemented afterwards. Before the user testing, a list was created based on the use case created earlier in the project. This way, the functionality of the actual application is being tested against the initial use case. The tests resulted in positive and constructive feedback.

SCRUM Methodology

This method of managing a project was applied during the second implementation phase for the actual application. The five sprints that divided the workload for the development were helpful and rewarding. Originally, this project was meant to have seven sprints, but due to the technical barriers and the move to Next.js, the sprints were reduced to five weeks. The supervisor provided helpful feedback every week.

This method works well when it comes to practical projects like this. Having a list of things to do each week was helpful and motivational. Due to the previous technical issues in the project phases, any bugs that were encountered were fixed, but in cases like the calendar with the version issue, that was left untouched since the functionality works well.

Project Management Tools

Many tools were used to manage and track the progress of this project. Each major tool, such as Miro and GitHub, will be discussed and explained in detail. Due to the project's scope, various tools were needed to complete it. Each tool had a different purpose and was used to complete a specific task; for example, Figma was used for design and prototyping.

Miro[16]

Miro is a digital whiteboard. It is a collaboration platform designed for teams to communicate and manage projects together. Some of the functionality they provide is sticky notes, coloured pens, pencils, highlighters, shapes and countless templates for anything a team needs for a project. The user experience, actions and objects mimic a real-life whiteboard, which is known as skeuomorphism. The Interaction Design Foundation[17] explain it as:

> *"Skeuomorphism is a term most often used in graphical user interface design to describe interface objects that mimic their real-world counterparts in how they appear and/or how the user can interact with them. A well-known example is the recycle bin icon used for discarding files"*

[16] https://miro.com/app/board/uXjVLRp2OqI=/?share_link_id=842883340057
[17] https://www.interaction-design.org/

It has been the primary management tool throughout the entire process. All of the research, ideas, and brainstorming went into this board. My supervisor has access to it, so during the weekly meetings, he has been able to open it, review the work and provide feedback and suggestions. Figure 87 is a screenshot of this Miro board, which is an overview of all the research that has been done. The link is in the footnote linked to this subsection title.

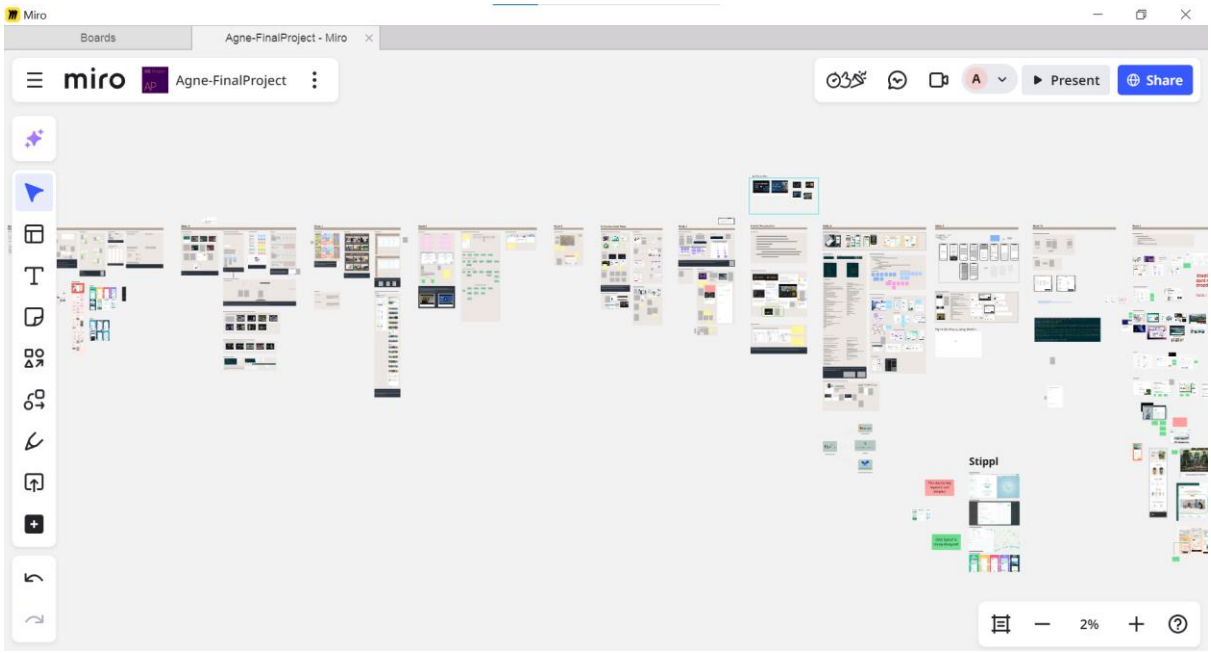


Figure 87 Screenshot of this project's Miro Board

This tool has been extremely helpful since everything can be saved in here. During each week where important research has been done, it has been put into a section labelled with the week it was done in.

Figma

Figma is a prototyping tool which lets designers visualise the end product. It is possible to create every detail of an application and make it into a fully functioning prototype, which means that it looks like it works, without it being a fully developed application. This is necessary before developing the actual application since these are used to test the app first, fix any UX/UI issues, and then, if the application works as intended, the development phase can commence.

This tool was used to design a prototype of the original idea (a mobile app), which ended up changing to a desktop and mobile app for itinerary creation. A desktop prototype has been created for the desktop version of the current project, but due to the time constraints of this project, illustrated in the Gantt chart (Figure 33), a mobile prototype was not created. Ideally, both would have been created and tested before implementation, but only a desktop design and prototype have been created. Figure 88 is a screenshot of Figma's interface and the prototype made for this application.

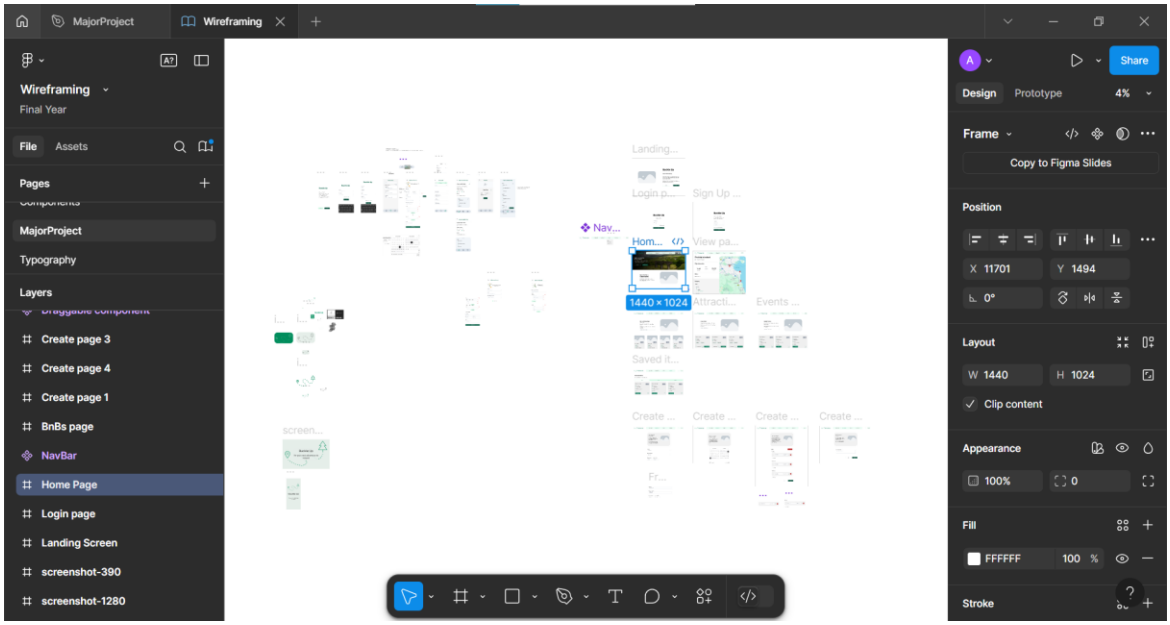


Figure 88 Screenshot of the Figma design file

As mentioned in the previous sections, ShadCN was the component library used during development. They also provide a public Figma component file, which contains all the components that can be used to design an application. These were used to design the prototypes[18].

GitHub

GitHub was used to manage the code since I have used it for all my coding projects. It was used for version control throughout the project development phase. Since there are two versions of this application, two GitHub repositories have been made. The old repository was used as a reference for the current project during development. As changes are made to the code, it is possible to push it to the repository, which enforces version control. If things go wrong, it is possible to track those changes and change to an older version. Figure 89 shows the repository for this project.

[18] https://www.figma.com/community/file/1203061493325953101

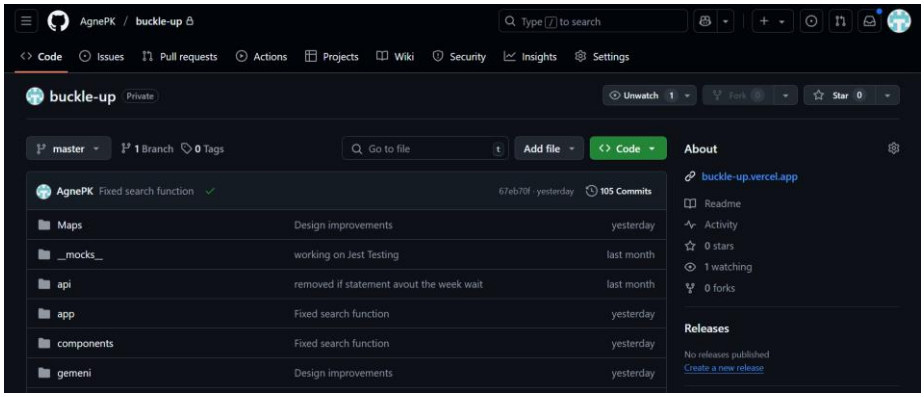


Figure 89 Screenshot of GitHub Repository

Reflection

Your views on the project

Working on this project has taught me one of the most important lessons, which can be used in any part of life where things become difficult. It encompasses taking a step back and looking at the situation from a broader perspective, can help make the right decisions. This has happened twice, once at the beginning when Maps was the main feature of the application, then in the middle of the development phase when the switch to Next.js occurred. These events are discussed in Appendix 1.

Now that the final project is mostly complete, I am very happy and proud of the decisions I have made. If I decided to continue in React Native, it would not have worked out as well as this one did. The difficult decisions that have been made have been worth it because a lot more has become possible in Next.js, and many non-functional requirements have been implemented.

I am very satisfied with the final application that has been completed in a small amount of time, and a great deal has been learnt from it. I am very grateful for the experiences during this major project since I will be taking those lessons with me into my future, both professional and personal life.

Another thing that I have learnt is to better manage my projects. Although it all worked out in the end, I believe that taking a step back sooner and not purely focusing on a mobile application with a map system as the core functionality would have given me more time to make the application better and run more tests.

These difficulties have become a strength that powered the progress and motivation to finish this project and do.

Completing a large software development project

Throughout this course, a lot of projects have been completed. Learning to use different tools and applications, and towards the end of each year, making them all work together. This project has put everything that has been learnt over those years into one application. This has taught me how important proper management is and how well the SCRUM methodology can work. Completing this large application, from start to finish, without having weekly lectures on how to do things, has been challenging but eye-opening.

Software development is a field that will always be changing; the learning will never stop, as technology will continuously evolve. This has already been experienced when researching which framework to use to build this application in. Bare React has been used in previous years, but now it is advised to use the Next.js framework (as of 2025), instead of starting a bare React project. This has shown how fast technology is growing and changing, and we developers must change with it.

Working on a huge project has been challenging but rewarding. Researching what technology to use for which purpose has been difficult since there are a lot of options out there. The real trick is to find ones that all work together with the least amount of problems as possible. Completing that project is the next step; depending on how long that takes, those technologies (for example, libraries and frameworks) may have already changed versions, which can cause other problems (such as the calendar issue). A part of the learning process is dealing with these issues and inconveniences.

Working with a supervisor

Having a supervisor who cares about the work and provides constructive feedback was an important aspect of this project. I had a very positive experience, and I was lucky to have the supervisor I had. The weekly (sometimes even twice a week) meetings have been of great help, as they gave me a sense of direction for the week ahead.

During the difficult times of the development stages, my supervisor believed I would break through and provided words of encouragement. These have helped more than I thought they could have. During the first interim presentation, while I was still stuck on the original idea for the project, my second supervisor provided me with advice that brought this entire application to life. He helped me take a step back and change the course of the original project to this one, which I am grateful for.

This project would not have turned out so well without the guidance and advice of both supervisors. When it comes to large-scale (or any) projects like this, it is always encouraged to ask for help if needed. At the beginning of this project, it was said that “This major project is a solo project, you will not be told what to do. It is for you to manage and direct it. But if you need help, reach out to any of us”, which is exactly what I did.

Technical skills

The idea of building another CRUD application did not appeal to me, but as I continued to develop it, adding features and functions, I began to enjoy it. Working with different libraries, such as “react-dnd”, and Google services such as Maps and Gemini has been a huge learning aspect. After numerous failed attempts at using Google Maps, trying again for a final time has widened my technical skills after things got working. The motivation to complete a project using tools and frameworks which I had never used before has encouraged me to be less nervous about failing, while trying new things. As I well know by now that failing and trying again will never be a bad thing since there is always something to learn from that experience.

Future opportunities

When comparing the final product to the original plan, it has become clear that there is space for this application to grow in the future, should it be continued. Many things can be added and improved. Some of those things include expanding outside of Ireland, adding functional/direct booking services, adding accurate weather forecasts, adding the discover items (BnBs, events and attractions) straight to an itinerary, collaborative itineraries, full offline capabilities, adding relevant documents (for example; hotel, flight, rental bookings/reservations), and many more features. This application has a lot of potential to become a full-scale application that can help many people who struggle when it comes to trips by creating a step-by-step plan for a trip.

Conclusion

To conclude this section, a wide range of skills has been learnt, from technical and management skills (and why those are important), life skills and perseverance. Building a large-scale project has been a great challenge, considering all the new technologies being used, such as Next.js, Google Maps, Google AI Studio, and Firebase Authentication, just to name a few. Many challenges have been faced during the development of this project, but the result is something to be proud of.

An itinerary planner with an integrated map system has been developed from start to finish. It exports the itinerary to Google Maps itself for a better user experience, and users can download it and share it with friends or family. The process of creating an itinerary has been made as simple as possible, with full control over what the user wants to add or not. It is a dynamic application and works similarly to Notion, where the process of creating something does not have to be entirely completed, which lets the user stay flexible.

Tools such as Miro, Figma and GitHub have been important factors in this project’s management. All the research and brainstorming have been done in Miro, prototyping has been done in Figma, and version control was monitored in GitHub.

Conclusion

My passion for travelling has inspired the topic of this project. Many things have changed throughout the process, but the theme of travel has stayed consistent. Planning a trip can be quite stressful, and the tools out there that are meant to help that process are not as straightforward as they could be. The main goal of this application was to build a good and simple user experience and a helpful tool to plan journeys across Ireland, without extra functionality that overwhelms the user.

The technologies used to complete this project are Next.js (a framework that supports full-stack development, but in this project, it was only used for the front-end), Google Maps, Google Gemini 1.5 Flash, and VSCode for development. Additionally, Android Studio was used to display the application on an emulator. This was during the last sprint, where the mobile view was worked on. Working with all of these new technologies has taught me a lot about learning new things; it can be overwhelming, but it's not new forever. The more I practised, the more confident I got. This important lesson will help in all aspects of life.

A wide range of research has gone into this project. Due to the original plans of this project, a lot of research went into user experience and accessibility, but also into the technical aspect, which was React Native. Back then, research was done on both Flutter and React Native since one was going to be chosen to develop this project, and React Native ended up being slightly better. This research was included in this paper because it is a part of this project, even though the final product does not include those technologies. Research was done before the project began, which ended up slightly changing throughout the development process. Immense research has gone into user experience, and throughout the Empathise and Define stages (in the design process stage) a lot of user research has been done, including the creation of personas who have helped create this application.

Paper-prototypes, wireframes and prototypes have been made for this project. Due to the time constraints of this project, instead of creating components and building them from scratch, a component library was used called ShadCN. They provide ready-to-use components that can be easily placed in a project.

The implementation was completed over five sprints due to the barriers faced halfway through the original development. The SCRUM methodology was used, which helped break down the large chunk of development into smaller, manageable pieces over five weeks.

Functional and user testing were completed. The results of the functional requirements were positive, and all tests passed. As mentioned before, if time allowed, more concrete tests would have been done. Also, the time this project was built in also did not allow for a wider set of tests. The user testing was also received positively. There have been no major issues reported by the users, but the things they have brought up have been implemented and fixed.

The result of this project has successfully targeted the goal, which was to create an application that allows users to create an itinerary, save it to their phone, connect it to Google Maps and have it all in one place. A lot of time has been lost due to the technical difficulties, but the result ended up just as good as originally intended, if not better. The additional AI feature was a bold but rewarding step for the application, since another unknown service was brought into the development, even though time was of the essence.

Miro was the primary tool for project management. Each week before development began, all the research, brainstorming and plans were done in the Miro board. For the development of the project, SCRUM methodology was used to organise the things that had to be done, which took place over the course of five sprints (one week in duration). Each week, a meeting was held with the supervisor to discuss the progress, review the work and decide on a plan of action.

Not only were technical skills learnt during this project, but also important life skills that will be used in a professional career setting and outside of that. I learnt how important it is to take a step back if things get difficult, reassess and try another solution, even if it may seem overwhelming. The difficulties have become a strength that has led to a breakthrough. When it comes to technical skills, spending more time researching the compatibility of libraries, frameworks, and versions has been a lesson learnt the hard way. I have learnt to use Google Services like the Maps and AI Studio, the documentation it provides is good and helped with the setup. If there were more time, I would have explored more of what Google AI Studio offers and how the chat functionality can be personalised.

To tie everything up, the results of this full-scale software development are satisfactory considering all the barriers faced. This application has a future, and many things can be improved and scaled out. Many lessons and skills have been learnt which will come in handy outside of this project. Working with a supervisor has been a great experience, and receiving guidance and feedback has helped this project move forward. This process was not easy; a lot was lost and gained, but there are no regrets regarding the decisions that have been made for the greater good of this project.

References

Allam, A. H., & Dahlan, H. M. (2013). User experience: challenges and opportunities. *Journal of Information Systems Research and Innovation, 3*(1), 28–36.

Ballantyne, M., Jha, A., Jacobsen, A., Hawker, J. S., & El-Glaly, Y. N. (2018, November). Study of accessibility guidelines of mobile applications. In *Proceedings of the 17th International Conference on Mobile and Ubiquitous Multimedia* (pp. 305–315).

De Kort, R. (2023, May 30). Using OKLCH colors in Tailwind CSS. *Studio 1902.* Retrieved from https://1902.studio/en/journal/using-OKLCH-colors-in-tailwind-css

Gill, O. (2018). Using React Native for mobile software development.

Hansson, N., & Vidhall, T. (2016). Effects on performance and usability for cross-platform application development using React Native.

Hume, D. A. (2017). *Progressive web apps.* Manning.

Kaplan, K. (2024, November 15). What is user experience (and what is it not)? *Nielsen Norman Group.* Retrieved from https://www.nngroup.com/articles/what-is-user-experience/

Mahmouda, M. A., Badawib, U. A., Faragc, T., Hassand, W., Alomarie, Y. M., & Alghamdif, F. A. (2021). Evaluation of user experience in mobile applications. *Evaluation, 15*(7).

Mankulam, S. (2025, February 2). Wanderlog: Onboarding friction - UX case study. Retrieved from https://www.behance.net/gallery/218301065/Wanderlog-Onboarding-friction-UX-Case-Study?tracking_source=search_projects|wanderlog&l=0

Merritt, K., & Zhao, S. (2021). An innovative reflection based on critically applying UX design principles. *Journal of Open Innovation: Technology, Market, and Complexity, 7*(2), 129.

Niemelä, E. (2022). How to improve accessibility in React Native mobile applications.

Norman, D., & Nielsen, J. (1998, August 8). The definition of user experience (UX). *Nielsen Norman Group.* Retrieved from https://www.nngroup.com/articles/definition-user-experience/

Petrie, H., & Bevan, N. (2009). The evaluation of accessibility, usability, and user experience. In *The Universal Access Handbook* (Vol. 1, pp. 1–16).

Quaresma, M., Soares, M. M., & Correia, M. (2022). UX concepts and perspectives – From usability to user-experience design. In *Handbook of Usability and User-Experience* (pp. 3–16). CRC Press.

Ramos, M. O. (2023). Accessibility guidelines proposal for the interaction design of mobile applications: Creating a more inclusive user experience.

Vermeeren, A. P., Law, E. L. C., Roto, V., Obrist, M., Hoonhout, J., & Väänänen-Vainio-Mattila, K. (2010, October). User experience evaluation methods: current state and development needs. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries* (pp. 521–530).

Wu, W. (2018). React Native vs Flutter: Cross-platforms mobile application frameworks.

Appendix 1

The reasons this project took a turn

This appendix will cover the reasons why this project switched from React Native to Next.js, what decisions were made, and the overall learning outcomes.

Before using React Native and Google Maps, the plan was to learn Flutter and use MapBox to display and work with the map. After doing some research, it turned out that React Native has better performance for accessibility and a better development experience. Then I began testing MapBox and React Native, which ended up failing due to MapBox not supporting React Native[19], and the community[20] library was not enough to make the maps work. The entire process was documented in the Miro board. The next step was to move to Google Maps, and many more test projects were made to experiment with Google Maps in React Native, but none of which succeeded. For example, the Places API was tested, simply to display places that the user is typing in, but they were not showing up. I researched what the issue could be; for example, the box could be rendering outside of the viewport. All sorts of issues were addressed (and documented in Miro), but unfortunately none of these solutions worked.

There were too many issues with React Native for the project to succeed, otherwise, it would have hit a dead end. Issues such as Google Maps API, versions of Kotlin and multiple libraries, limitations of Expo Go, deployment issues, and many more. These issues will be explained later, but because of this, I had to make a very difficult decision to switch front-end frameworks to Next.js, especially considering it was already March 7th. This left me with very little time to learn a new Framework, find libraries to substitute the ones from React Native (since those libraries do not work in Next.js), and migrate my entire project to Next.js. Most JavaScript functions were able to be moved, but everything else had to be rewritten or debugged to make it work in Next.js. Most of the functional requirements work in the React Native version, such as CRUD, dragging stops, browsing BnBs, attractions and events, and downloading the itinerary.

Expo is an open-source platform for building native applications for Android, iOS and even web. Expo Go is their sandbox environment where the code runs, so the developer has a preview of what they are building. Expo is built on top of React Native, which is supposed to make the development process faster since it is only one codebase for Android, iOS and web. This came with limitations on the Expo Go side. To be able to display each stop on a map, the stops must have coordinates (latitude and longitude).

[19] https://docs.mapbox.com/help/glossary/maps-sdk-for-react-native/
[20] https://github.com/rnmapbox/maps

For that, Google Places API was used so that the user can select their stops, which saved those coordinates from that stop. This Google Maps API service did not work in Expo. Usually, there are errors that show the reason something is not working, but with this, there were no errors. One of the possible problems was that the box displaying all the autocompleted places was somewhere outside of the screen. After many attempts of fixing the display location, it still did not work. If there were no coordinates available for a map to display a location, a map was redundant. Expo Maps worked, but it just displayed the map. Since it is a new feature at the time of writing this report, there are many things that did not work. This is when I tried to work with Google Maps instead, but that failed since the errors displayed problems with the API key.

Some of the libraries I was using, such as react-native-maps, react-native-directions, and expo-google-places-autocomplete, did not work; Figure 90 displays the error. Some of these libraries were not compatible with Expo Go, and others (when attempting to run the build command) threw a lot of errors about the libraries being used. One of the possible problems was the location of my project; I had it in the OneDrive that my institute provided (to have it safely backed up on the cloud). I was advised to move it to my desktop since OneDrive can hinder some of the functionality due to syncing issues. This, unfortunately, did not help. After removing the libraries (to test if the application could be deployed at all), the final issue surfaced.

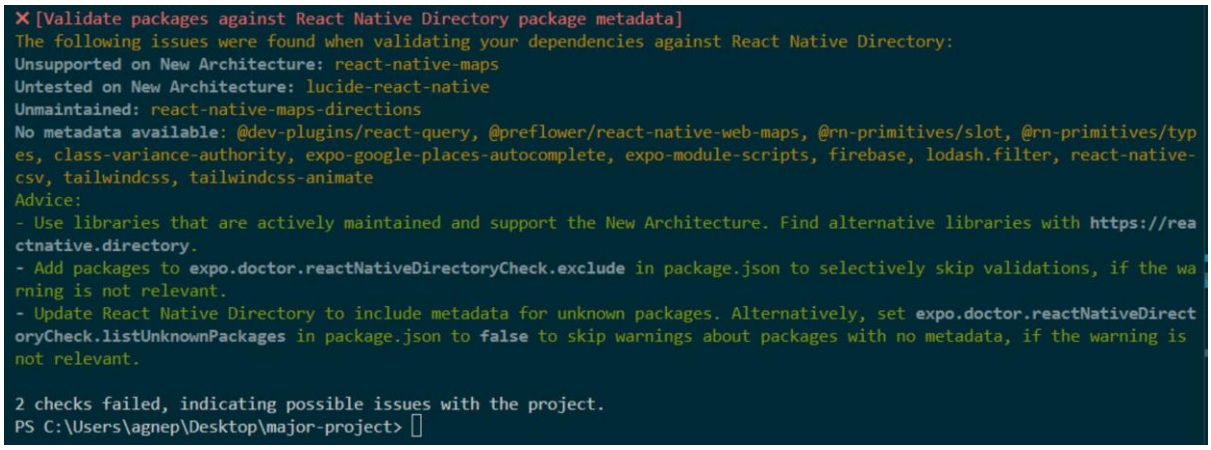


Figure 90 Error about packages in React Native

Finally, the ultimate problem that triggered the decision to move away from React Native was the Kotlin version issues. While attempting to deploy the project to EAS (Expo Application Services, allows to deploy React Native Applications on their cloud), the build kept failing. I went online to research the error I was getting, and I came across "*Execution failed for task ':expo-modules-core:compileReleaseKotlin'. #32844*".

I wrote my comment explaining my issue and all the attempts I've tried to make to fix the issue without any luck[21]. Taking into account the limited amount of time I had left to save my project, I made the difficult decision to discontinue my React Native project and move as much as I could to another front-end framework, which ended up being Next.js after researching it.

This experience has taught me a valuable lesson. I spent a lot of time trying to make this work, without taking a step back to consider other options or realise that there was no future if I were to continue with React Native. Once I took that step back to see the project as it is and how things were going, it became clear that moving away from React Native was the only real option if I wanted to create something I genuinely enjoyed. This lesson does not only apply to a career but also to life. It made me realise that taking a step back when life gets stressful can be very beneficial. Reflecting on the time when this project was at its lowest, when all the problems started to surface, compared to what I have managed to build since I moved to Next.js, I am quite satisfied with the progress and what I have learnt from it. That is the best and most important decision I have had to make during this project.

[21] https://github.com/expo/expo/issues/32844#issuecomment-2704094564