Trippr

Group Report

Jae Kim - Frontend
Reuben Morris - Frontend
Ethan Lee - Frontend
Josh Nazareth - Frontend
Zane Ayris - Backend
Michael Gouldstone - Backend

Team
Caffeine
Team 23

Executive Summary

Our motivation and purpose

Trippr, a New Zealand-based initiative, is revolutionising the carpooling landscape, steering it towards heightened user engagement, reliability, and eco-consciousness. With a sharp focus on fostering a sense of community and leveraging technological advancements, Trippr is committed to delivering a seamless and enriched carpooling experience tailor-made for Kiwis. Our application has been meticulously designed to cater to the nuanced needs of drivers and passengers, focusing on creating an enriched user experience.

At its core, Trippr fosters real-time communication amongst users, allowing for seamless coordination of trips. Users can effortlessly connect, share details, and plan their journeys with our intuitive messaging feature, ensuring that each ride is as smooth as possible. To bolster confidence and trust within our community, a robust rating system has been integrated, paired with CarJam, to authenticate and validate the credibility of each driver, thereby enhancing the overall safety and reliability of each trip.

We have emphasised personalization within the application, allowing drivers to customise settings to their preferences, enhancing usability and satisfaction. The system is also designed to be economically viable, promoting carpooling as a cost-effective alternative to traditional transportation modes. Furthermore, our application underscores the importance of sustainability by facilitating carpooling, a more environmentally friendly approach to commuting.

In terms of features, our platform stands out due to its interactive capabilities. Users are encouraged to engage more deeply with the community by sharing and creating music playlists via Spotify integration, further enriching the journey experience. We have also integrated a group chat feature, allowing users to stay connected and communicate effectively, ensuring that the carpooling process is as seamless and enjoyable as possible.

Going forward, Trippr is committed to continuous improvement and innovation, with plans to implement enhanced payment options, increase robustness, and introduce features that emphasise eco-friendliness and user interactivity. By doing so, Trippr aims to remain at the forefront of carpooling solutions, delivering an exceptional and unmatched service to all users.

Integrated a group chat feature, allowing users to stay connected and communicate effectively, ensuring that the carpooling process is as seamless and enjoyable as possible.


“Step into the future with Trippr, where saving money and the planet unite through unforgettable journeys.”

Table of Contents

50 pages

Introduction

Welcome

Our project, centred around the concept of a Carpool mobile application, was devised with the intention of addressing and mitigating the escalating issues of traffic congestion, environmental pollution, and the spiralling costs of transportation in New Zealand. The aim was to craft a user-centric application that optimises carpooling options, enhancing the commuting experience while also contributing positively to the environment and community.

Aims and Objectives

The key objectives of this initiative were multifaceted. Primarily, we aimed to reduce traffic congestion, particularly in bustling urban centres, by promoting ride-sharing. Concurrently, this effort would catalyse a reduction in greenhouse gas emissions, aligning with New Zealand's commitment to environmental sustainability. Moreover, through this carpooling application, we sought to provide a cost-effective commuting alternative, enabling users to save on substantial commuting expenses. The application was also conceptualised to foster a sense of community and social interaction among users.

Target Audience

Our application is particularly targeted towards daily commuters, travellers, and essentially anyone seeking a more cost-effective, eco-friendly, and sociable means of transportation. The inclusive design makes it applicable and beneficial to a broad spectrum of New Zealand's populace.

Scope of the Project

The project encompassed the design and development of a carpooling application, taking into consideration usability, security, location services, and user engagement. Given New Zealand's unique geographic and demographic factors, the application was meticulously tailored to meet the specific needs and travel patterns of the local populace.

Approach

We adopted an agile project management methodology, utilising Scrum to maintain flexibility, foster team collaboration, and ensure that the project evolved in alignment with user needs and feedback. Regular meetings and iterations allowed us to adapt and refine the application to meet our objectives effectively.

Important Outcomes

The outcome of our project is a robust and user-friendly carpooling application that not only provides an efficient and viable alternative to conventional commuting methods but also promotes sustainability and community engagement. Through effective ride-sharing, the application aims to significantly lessen traffic congestion, reduce environmental impact, and offer a more economical and sociable commuting experience.

Background

Where it all began

Our project seeks to blend technological innovation with a community-focused vision to reimagine commuting in New Zealand. We aim to provide a platform that facilitates not only shared commuting options but also fosters a spirit of community and shared environmental responsibility, contributing towards a more sustainable and connected society.

Preliminary Background

New Zealand's urban centres like Auckland and Wellington are grappling with growing traffic congestion, prolonging commuting times and increasing pollution levels. This situation, fueled predominantly by a reliance on personal vehicles, exacerbates environmental challenges, conflicting with New Zealand's renowned clean and green image. The country's unique geography and population distribution further complicate transportation, necessitating innovative solutions to improve commuting efficiency and sustainability.

Existing Solutions

Several carpooling applications, such as Uber and BlaBlaCar, already exist in the market. These platforms facilitate ride-sharing but are often driven by profit motives and individual conveniences rather than prioritising environmental sustainability and community well-being. Furthermore, public transportation enhancements have been initiated, yet these haven't fully addressed the flexibility and coverage required, especially in more remote or rural areas of New Zealand.

Value of Our Work

Our project aspires to breathe fresh air into the commuting landscape of New Zealand. By developing a carpooling application focused not just on facilitating shared rides but also nurturing a sense of community and shared responsibility towards environmental sustainability, we intend to differentiate our approach. The application is envisioned to be a platform where commuters can find cost-effective, convenient, and environmentally friendly transportation options, thus reducing the number of single-occupancy vehicles on the roads and subsequently decreasing overall traffic and pollution.

Methods and Tools

Developing a seamless and effective carpooling application required the right blend of technologies. We utilised React Native to ensure the application is mobile responsive, providing a smooth user experience across different devices. Node.js and Express.js were used to manage the backend processes, ensuring the application runs efficiently and reliably. PostgreSQL was our chosen database, keeping data organised and secure. For testing and managing APIs, POSTMAN was an invaluable tool, ensuring that our application's functionalities run smoothly.

Hosting services were managed on AWS, providing reliability and scalability to our application, ensuring it performs optimally under different loads. Lastly, PM2 helped in managing the application's processes, ensuring uptime and facilitating application management and monitoring.

Specification & Design 

Frontend

User requirements specification

Trippr is a unique travel solution intricately crafted to seamlessly connect drivers and passengers embarking on journeys across the stunning landscapes of New Zealand. Designed with meticulous attention to usability and functionality, Trippr acts as a network, aligning the pathways of travellers whose interests, destinations, or routes intertwine.

Trippr is streamlined to offer a rich user interface providing a multitude of functionalities

Profile and vehicle registration

Users can create and manage profiles, and drivers have the added functionality of registering their vehicles.

Listing rides

Drivers can readily list their upcoming journeys, providing necessary details such as origin, destination, departure time, available seats and preferences.

Searching rides

Passengers can effortlessly search for rides, tailoring their searches to meet specific needs, schedules, and destinations.

Enquiring about rides

A built-in enquiry feature allows passengers to engage with drivers, seeking further clarity or negotiating journey terms.

Messaging

A robust messaging platform enables fluid communication between drivers and passengers, fostering a sense of connection and trust.

Review system

Users can leave or view reviews, leveraging past experiences to make informed decisions regarding future journeys.

User requirements

For our users

Passenger and driver account

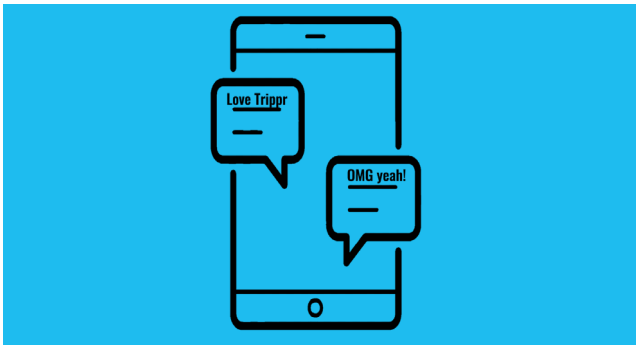
Users should be able to create and manage their profiles, customising their information and preferences to enhance the user experience.

Ride listing and search

Drivers should be able to list their rides comprehensively, and passengers should be able to search for these rides effectively.

Enquiries and messaging

A secure messaging system allows users to communicate, ask questions, and confirm details seamlessly.



Review and rating system

The application provides a system where users can leave and check reviews for drivers and their journeys.

Use cases

Caleb Johnson

Caleb's Journey from Tauranga to Raglan for New Year's Celebration

Caleb is tech-savvy. He is comfortable using mobile applications, understands basic features and functionalities, but isn't highly experienced in navigating through more advanced or intricate tech features. He uses an iPhone.

Caleb has been invited to spend new years in Raglan with his mates. Unfortunately he is the only one of his mates that lives in Tauranga and does not have a private vehicle to get himself there and there are no other options to get to by using bus companies or flying. His friends have suggested checking out Trippr to see if there is an available ride to get him there.

Caleb's experience with Trippr

As Caleb is a new user with Trippr, he must first create an account. After creating an account he decides he doesn't want to register as a driver and is immediately in the app.

In the app he navigates to search for rides that are going from Tauranga to Raglan. He finds that there is a route going from Mt Maunganui to Raglan around the time when he wants to depart and enquires about it.

Caleb is accepted into the ride and receives a message from the Driver letting him know that he will be picking him up a bit later than advertised because of the slight detour he needs to take to pick up Caleb.
On the day of the journey Caleb and the driver use the messaging feature of the app to arrange a successful on-time pickup and drop-off

Jeffery Chen

Jeffrey's Adventure from

Auckland to Coromandel

Jeffery is a frequent traveller, always using his iPhone to check and make bookings, staying in touch with family and friends back home.

Jeffrey is on holiday in New Zealand. His friends back home have recommended that he goes to check out Cathedral Cove. Being a traveller he had previously used BlaBlaCar in Europe and wondered if there was a similar carpooling app in New Zealand.

Jeffreys's experience with Trippr

Jeffery searches the App store for a carpooling app similar to BlaBlaCar for the New Zealand market and finds Trippr. He downloads the app and makes an account.

Jeffery has a browse around the app and he finds it familiar and is able to easily find a ride leaving tomorrow morning and enquiries.

He instantly gets a message back from the driver and they arrange a pickup time.
The driver is running late but lets Jeffery know using the in-app messaging.
Jeffery completes the ride and along the way gets a few recommendations of other good spots to check out around the Coromandel region.
Very happy with his experience Jeffery leaves a 5-star review for his driver.

Sophie Tucker

Sophie's Journey from

Whangarei to Wellington

Sophie is not very technologically savvy. She has an older Android device that she uses for social media and keeping track of events. She has previously used Trippr with little hassle.

Sophie is planning to drive down to Wellington from Whangarei for her first year of uni. However, she has never driven such a long distance alone and would like someone to share the cost of petrol with. She has previously signed up and used Trippr as a passenger and having had previous good experiences, she decides that she would like to use Trippr as a driver to find a travel partner.

Sophie's experience with Trippr

Having already used Trippr, Sophie simply needs to open the app. She tries to list a ride but sees that she first needs to sign up as a driver. She finds this easy to do using the car registration autofill.

She is then able to list a ride. Again she finds this process to be very easy using the google autofill.

Over the next few days Sophie receives a handful of responses but because it is such a long trip and it is her first time using the app as a driver she wants to check the users reviews and chat to them before accepting them onto the ride.

She is able to do all of this using the in-app messaging and reviews system and finds a suitable travel partner who is the same age, enjoys the same music and is also moving down to Wellington to study.

She accepts the passenger and arranges to pick up her travel partner from Auckland on her way through. The two bond over the long journey and become friends throughout their time at Victoria University.

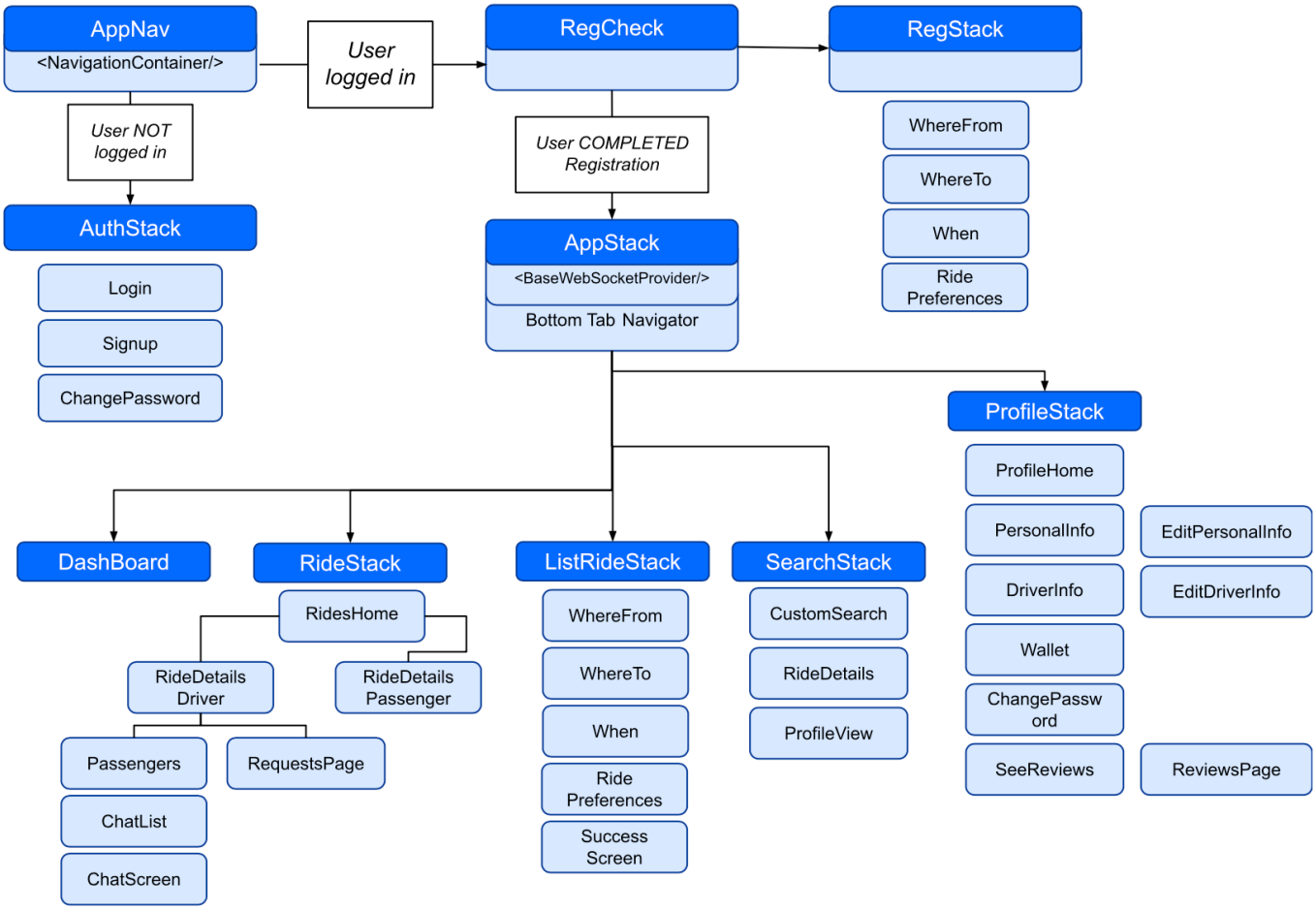
Design

The frontend architecture of Trippr

The frontend architecture of Trippr begins with the NavStack. In this module, we check to see if there is currently a userToken stored in async-storage from a previous session. In the event that there is no userToken present, the AuthStack is shown to the user (Fig 1.0).

In the AuthStack, users are able to login or register as either a passenger by entering some basic details and setting a password. When a user is able to successfully login or register, their unique userToken is stored in async-storage meaning that the user is now shown the RegStack. RegStack is a necessary section of our app because in order to sign up as a driver, a userToken must be provided.

The RegStack will display the option to register as a driver as part of the onboarding process. Once they have registered as a driver or dismissed this page the user will never be shown this page again and the user is taken to the core of our app, the AppStack. AppStack utilises a Bottom Tab Navigator which allows the user to switch quickly between the five main modules of our Trippr - the Dashboard, My Rides, List-a-Ride,



Core modules

Authentication

The Authentication module serves as the entry point to our app. The login screen in Fig 1.1 is clean, simple and familiar to users. Off of this page, the user can tap “Register Here” if they are a new user to be navigated to the registration process or tap the “Forgot Password?” to reset their password (not implemented in the current version).

Fig 1.2 shows the Sign-up process. On the first screen, the user inputs all the information required to create a basic “passenger” account. Error checking is employed throughout this page to ensure that the user is inputting valid credentials and will not be able to submit unless they have completed all fields correctly. Users require a secure password with at least eight characters and one symbol and number. With this, a userToken can now be saved to the account, allowing access to the main app. After this, the user is given a personal welcome page. The decision to personalise the welcoming screen with the user’s first name was essential so that the user would immediately feel welcomed and connected to Trippr as if it was their own personal place rather than a generic app.

Tapping on “No, Continue to Trippr” will immediately navigate to the main page. Otherwise, the user will be navigated to the Sign-Up car details screen. The user inputs their driver’s licence and car registration number on this screen. Thanks to the CarJam API, users can tap “Get Car Details” and have their car details filled in automatically. Users can edit these as they see fit and enter the number of seats available and their Smoking, Pets and Food preferences before finally completing the sign-up and entering the app.

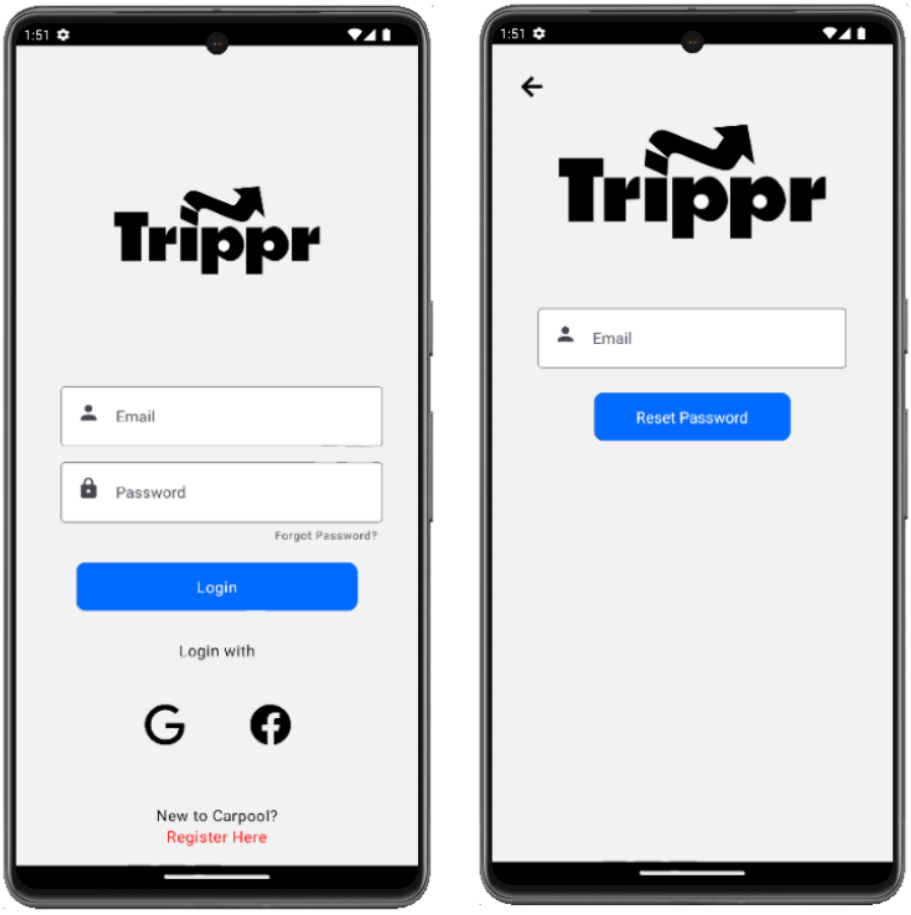


Fig 1.1 - Login Screen

In future iterations of this module, we would like to add more verification of specific details such as email address and driver's licence number for passenger peace of mind and safety.

From the back-end side, we had the means to implement login with Facebook or Google, but other features within the app were prioritised ahead of this.

Lastly, if Trippr were to be released to the public, we would also need to consider a "terms of use" section as part of the sign-up process that removed liability from incidents during trips taken with Trippr.

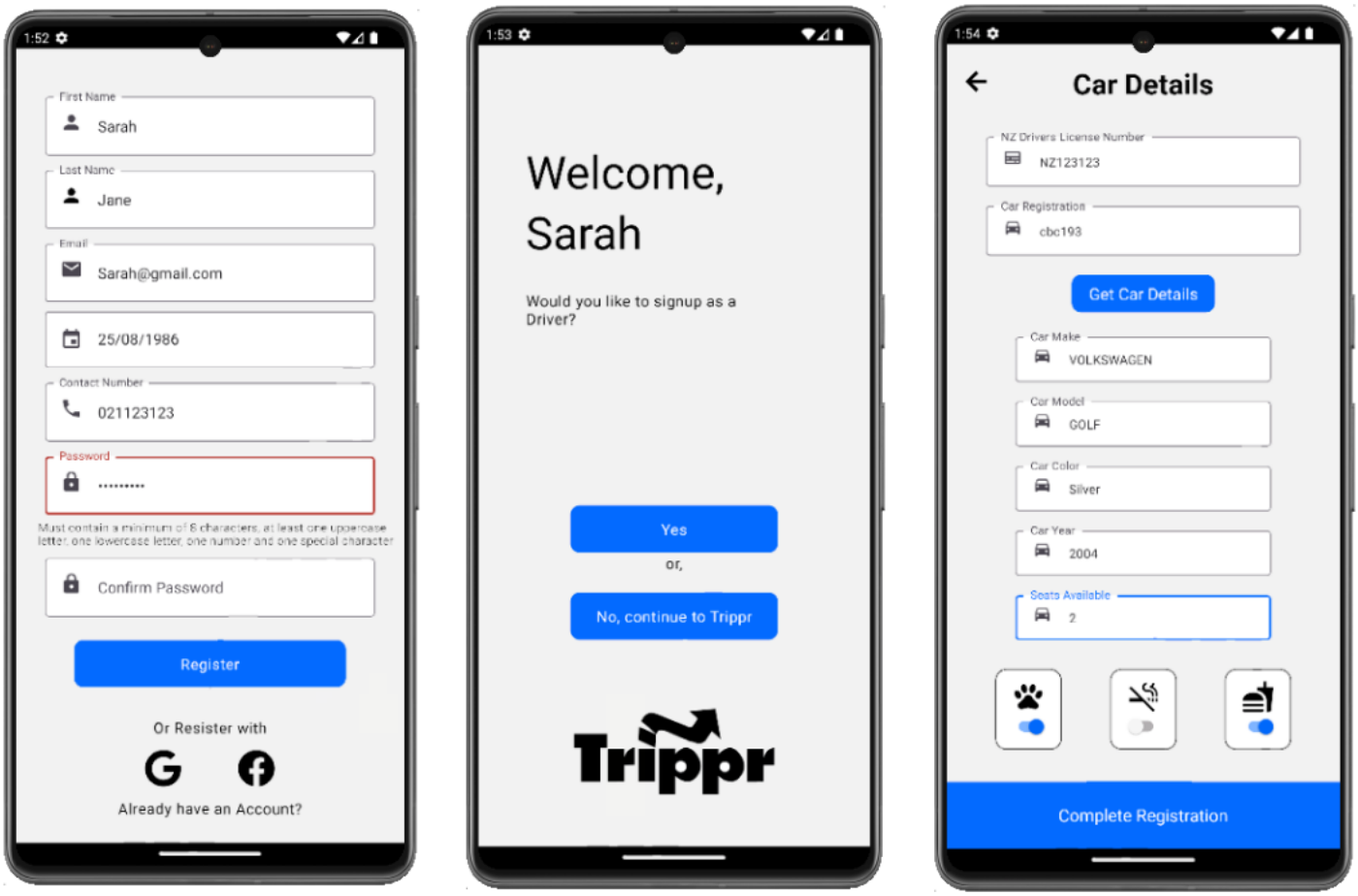


Fig 1.2 - Sign up

Dashboard

Where magic begins

The Dashboard is the core screen of Trippr. It offers the user a “Place-To-Be” in the app, where they can quickly see all the information and options available. In Fig 1.1, you can see that the user can see their bookmarked trips and quick buttons to List a Ride and Search for a ride. As you would expect, tapping on the user’s profile picture will navigate you to the profile module of our app.

Trippr relies heavily on Drivers listing rides to get users around New Zealand; because of this, the team decided to show the “List a Ride” option to all users at all times, regardless of whether the active user is registered as a driver. By doing this, users will constantly be reminded/encouraged to sign up and list a ride.

In future versions of Trippr, we would like to enhance this page to add more features to encourage users to hang around in the app. On top of the features we would implement would be “New Messages” and “Upcoming Trips” widgets as well as “Recommended Trips”, which would recommend trips to the user based on their previous searches and rides.

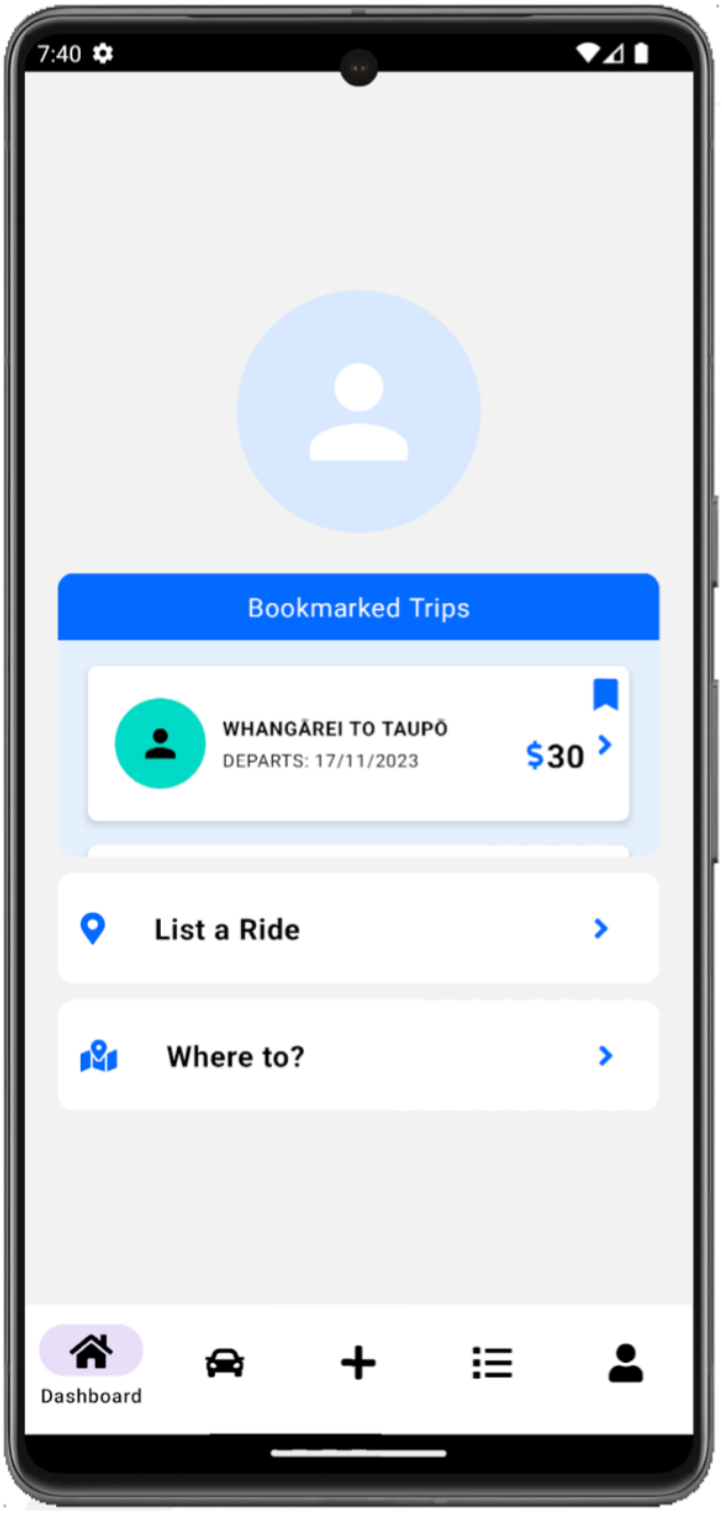


Fig 1.1 - Dashboard

My Rides

Another essential element of Trippr

The My Rides tab of our app is another essential element of Trippr. The My Rides Tab allows users to manage all the rides they are involved in in one place.

At the top of the screen, there is a simple switch button that allows users to switch between showing rides that they are a Passenger and rides that they are a Driver for. The driver tab will be blank for users not registered as a driver, displaying only the message “Not a Driver”.

Each card on the pages will show quick information about the ride, similar to other places on the app, including the driver’s profile picture, start and end-points, departure date, the number of seats available and the price.

Tapping on one of these cards will take the user to the ride details screen, where the user sees a beautiful, full-page view of the route with a card overlaying it with all the trip details. The ride details card will change dynamically depending on the users’ registration status (Driver/Passenger) and the ride status (Scheduled, Enquired, Accepted, Started, Finished).

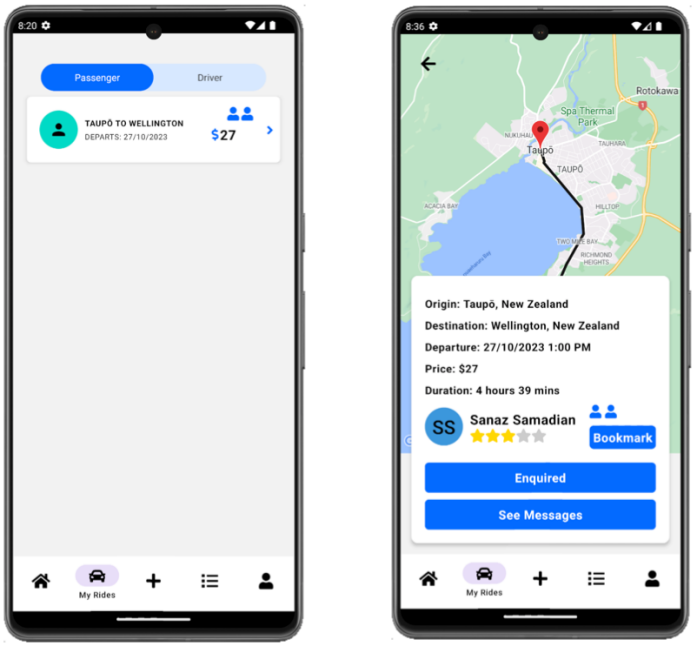


Fig 1.21 - My Rides (Passenger)

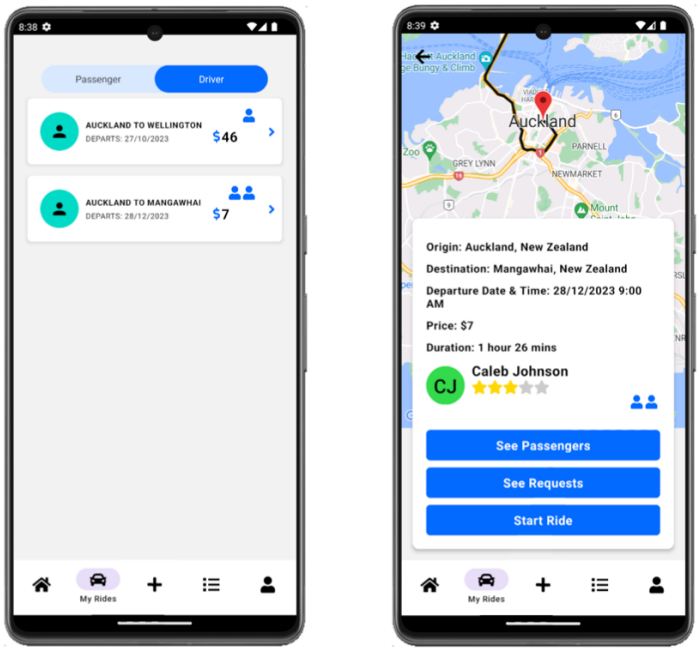


Fig 1.22 - My Rides

For example, on the ride details screen, the “Enquired” button in Fig. 1.21 will dynamically change between Enquire, Scheduled and Completed.
On the other hand, a driver will be able to use buttons to navigate to see passengers that have been accepted onto the ride (Fig 1.23) and a button to see any requests for the ride (Fig 1.24).

Here, they will be able to accept or decline the request. A decision verification pop-up is shown to the user to help prevent accidentally declining or accepting a request. From the See Passengers screen, the user is then able to message their Passenger using our custom instant messaging app (Fig. 1.25)

Different statuses

To accept, check and communicate

This section of the app's development was incredibly complex to bring together all of the different statuses to ensure the correct options were displayed to the user. In future versions of Trippr, we would look into using different colours for the ride status, providing more details on Passengers and providing the Passenger to initiate messaging with the driver once they are accepted onto the trip.

We initially decided that when requesting a ride, only the driver could begin messaging so that drivers could further vet their potential passengers before accepting them. Unfortunately, due to our time constraints, we could not implement this.

We would also like to implement more options for drivers to change ride details and remove passengers. Adding notifications to keep both Passengers and Drivers up to date with trip updates would be a high-priority feature to add to future iterations.

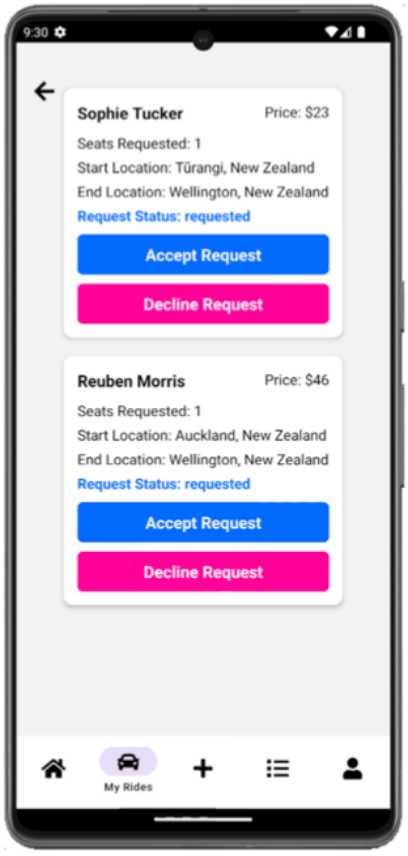


Fig 1.23 - See Requests

Fig 1.24 - See Passengers

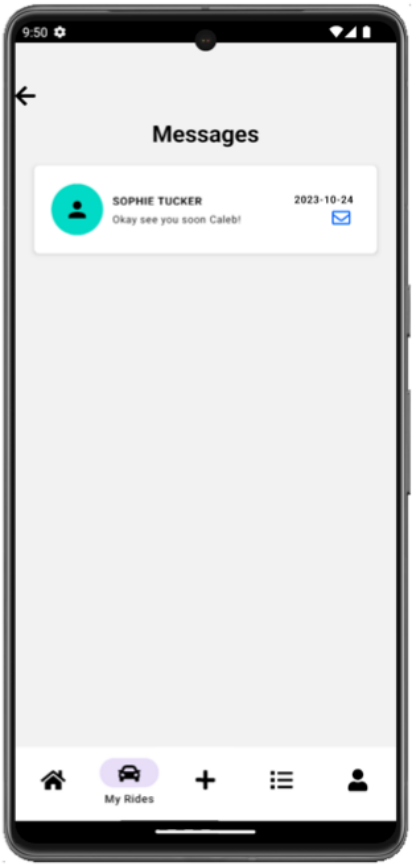


Fig 1.25 - Messaging 1

Fig 1.25 - Messaging 2

List a ride

Lets Drivers list their rides effortlessly

Fig 1.31 - List a Ride (Passenger)

The List a Ride module lets Drivers list their rides on Trippr. Similar to showing the List a Ride button on the dashboard, this module is still available to Passengers but will show the screen shown in Fig 1.31 to encourage them to sign up as a driver.

Fig 1.32 shows the List a Ride follows for a user who is registered as a driver. All of the screens in this process are kept minimal to make this process as seamless as possible for the user, as was requested by our client. To aid this ease of use, we have used Google Places Auto-complete on the “Where To?” and “Where From?” screens to not only allow our users to enter the data quickly but also to ensure that the data that is input is accurate and in the correct formatting for use by the backend.

Moving to the “When?” screen, this ease of input is continued through calendar and clock inputs so the user can easily select a date and see which day of the week it is on. The Time and Date inputs help to reduce user errors. The minimum date input is set as the current date so that users cannot set trips that ‘started’ in the past.

The following preferences screen is auto-filled based on the users’ preferences set during the signup phase or as edited within the profile module. Users are still able to make changes for this specific ride. Lastly, a confirmation screen summarises the users’ inputs so they can double-check before listing the ride. Once the user taps the “Confirm” button, a popup alert is displayed to verify that the listing has succeeded.

Not many changes need to be made to this current List a Ride module version. One of the changes we would like to make would be to provide edit buttons on the confirmation page so that the user could quickly make changes to their ride.

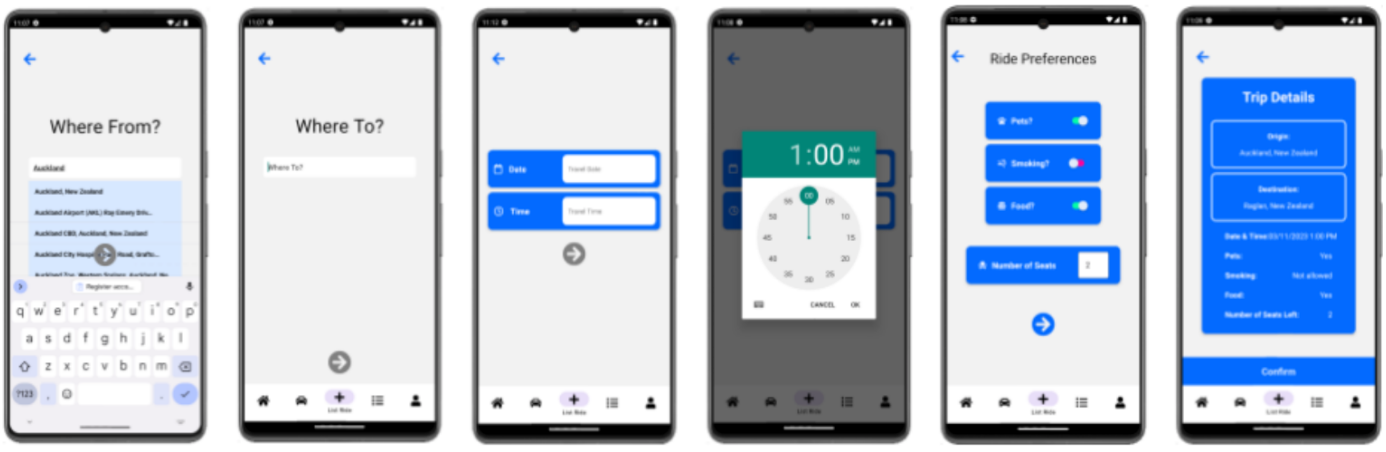


Fig 1.32 - Listing a Ride

Ride searching

With ability to filter rides by locations

Unlike most of the other modules, the ride searching tab will appear the same for both drivers and passengers. The only difference being that drivers will not be shown their own rides.

In the current version of Trippr, the Ride Searching module shows all of the currently listed rides on the entire app in a scrollable list. Clicking on any of these available rides will take the user to the ride details page where they can see an interactive map of the route as well as more details about the trip including the number of seat available, the driver and their rating as well as a button to quickly enquire for the ride or bookmark it for later. As a user would expect from experience using other apps, tapping on the drivers profile will navigate to a more detailed profile view to see the drivers bio and experience.

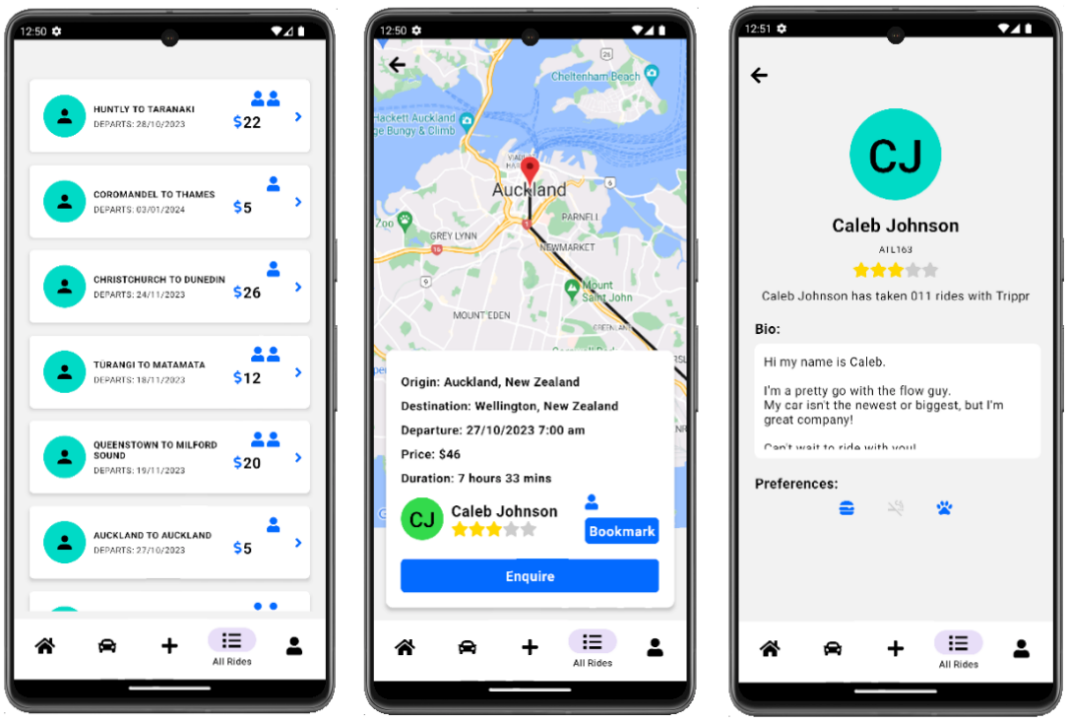


Fig 1.4 - Ride Searching

When enquiring for a ride, a modal window will pop-up over the screen and the user is prompted to input the number of seats they would like to book as well as their desired pickup and dropoff location so that the driver can make an informed decision about whether or not to accept the passenger. Again, these inputs use Google Places Autofill to ensure error free input from the user. Once the enquiry is sent the ride will then appear in the “My Rides” module and the driver will then have access to view the request.

A high priority feature we would like to add to future iterations of the app is the ability to filter rides by starting location, ending location, locations passed through as well as date filtering. We think this functionality would be essential to the user experience in being able to find the rides they desire with greater ease and efficiency.

Profile

Another major module of our app

The user Profile section is another major module of our app. Within the profile section users will be able to see and make changes to everything associated with the users personal details. The home pages for the profile view are similar between drivers and passengers except the passenger will not be able to see the Driver Details Button and instead is shown an option to register as a driver (Fig 1.71).

Going into the Personal Info screen (Fig 1.73) the user has options to view and edit their bio, see reviews that other users have left for them and change password. They also have the option to edit their personal details (Fig 1.74).

When editing the users personal details we wanted to make sure that users could not accidentally make changes so we made use of a pop-up modal to confirm changes. Along with this, we also required users to enter their password when changing their email. This decision was prompted by the fact that the user's email was crucial to their userToken, and changing it would cause their current token to expire. Therefore when changing the user's email or password we refresh the userToken.

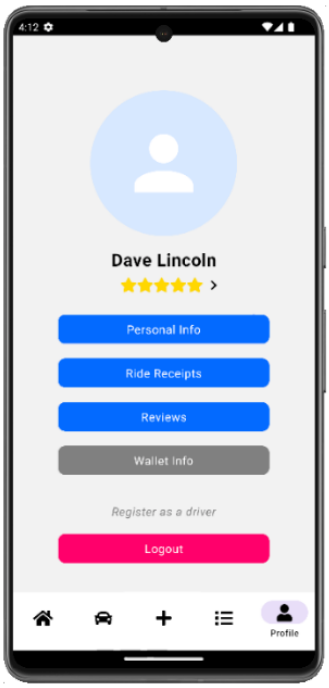


Fig 1.71 - Profile View (Passenger)

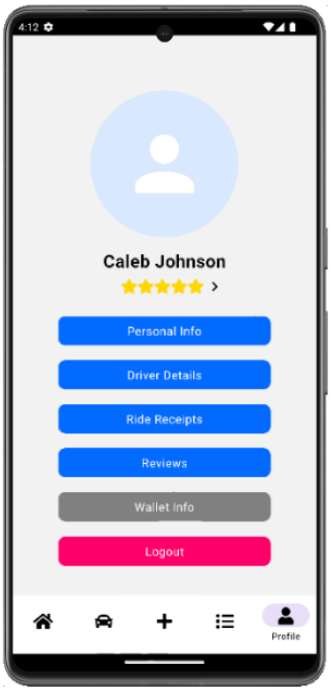


Fig 1.72 - Profile View (Driver)

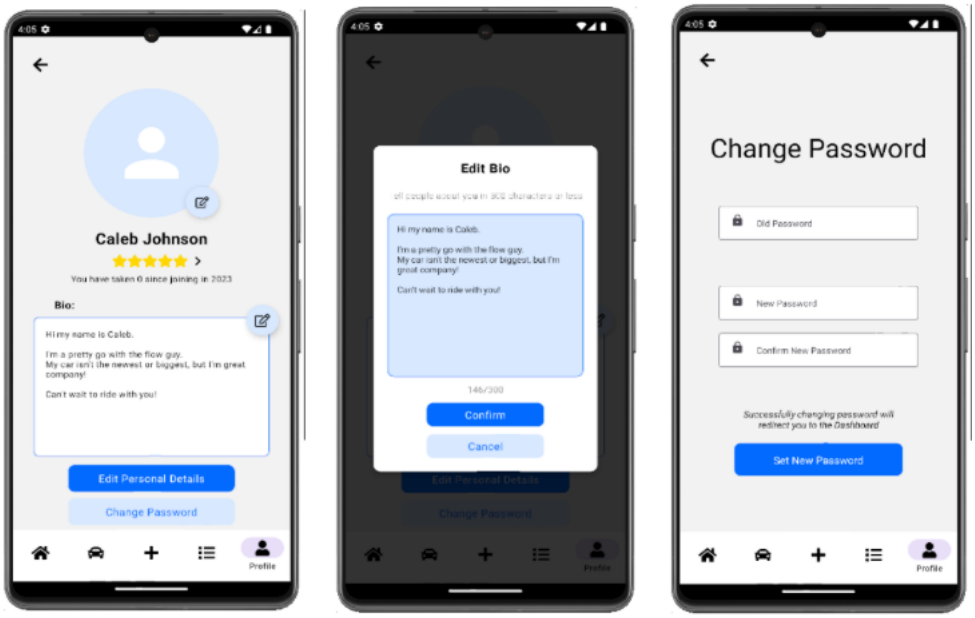


Fig 1.73 - Edit Bio and Change Password

Profile (cont.)

Another major module of our app

The user Profile section is another major module of our app. Within the profile section users will be able to see and make changes to everything associated with the users personal details. The home pages for the profile view are similar between drivers and passengers except the passenger will not be able to see the Driver Details Button and instead is shown an option to register as a driver (Fig 1.71).

Going into the Personal Info screen (Fig 1.73) the user has options to view and edit their bio, see reviews that other users have left for them and change password. They also have the option to edit their personal details (Fig 1.74).

When editing the users personal details we wanted to make sure that users could not accidentally make changes so we made use of a pop-up modal to confirm changes. Along with this, we also required users to enter their password when changing their email. This decision was prompted by the fact that the user's email was crucial to their userToken, and changing it would cause their current token to expire. Therefore when changing the user's email or password we refresh the userToken.

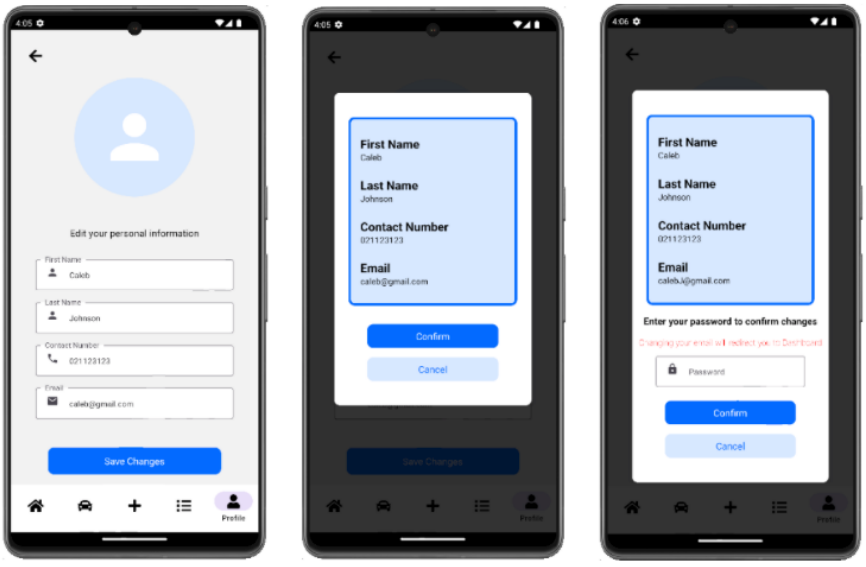


Fig 1.74 - Edit Personal Info

In the Driver Details section, Drivers are able to see and edit their own details using a very similar user interface to that from the registration (Fig 1.75). The ride receipts page (Fig 1.76) shows a simple view of the previous rides that the user has taken so that they can review how much they have spent. Finally, tapping on either the stars or the “Reviews” button elegantly shows the user all of the reviews left for them either as a driver or a passenger (Fig 1.77).

Reviews are listed in a compact form and if the user wants to see further details on the review a simple tap will show them an expanded form as signified by the downward pointing arrow on the compact card.

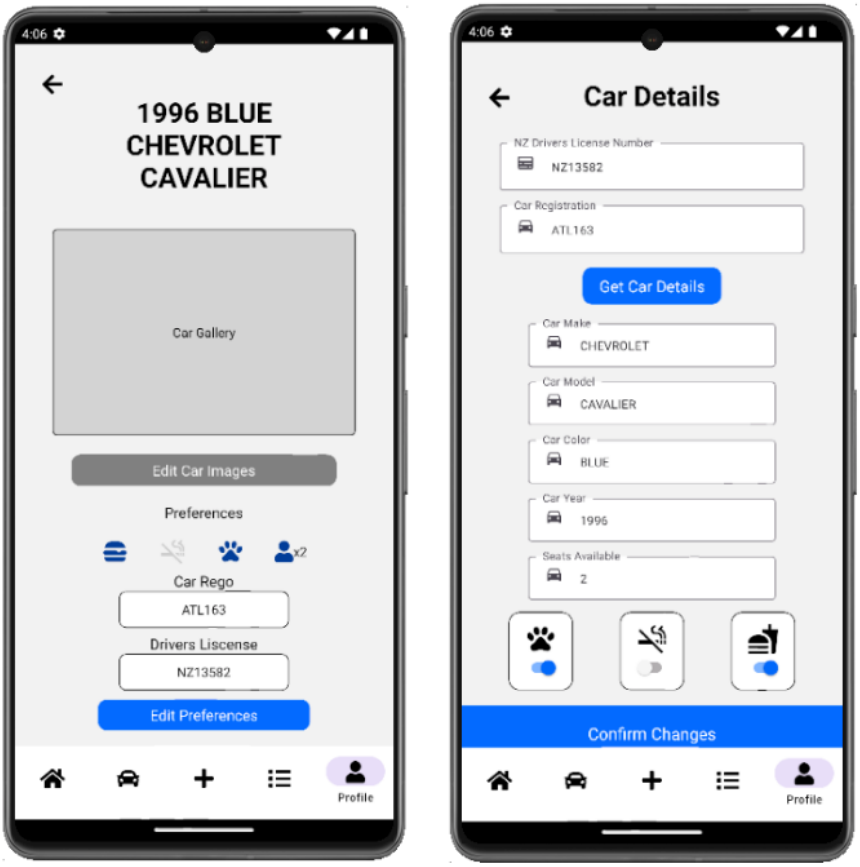


Fig 1.75 - Driver Details

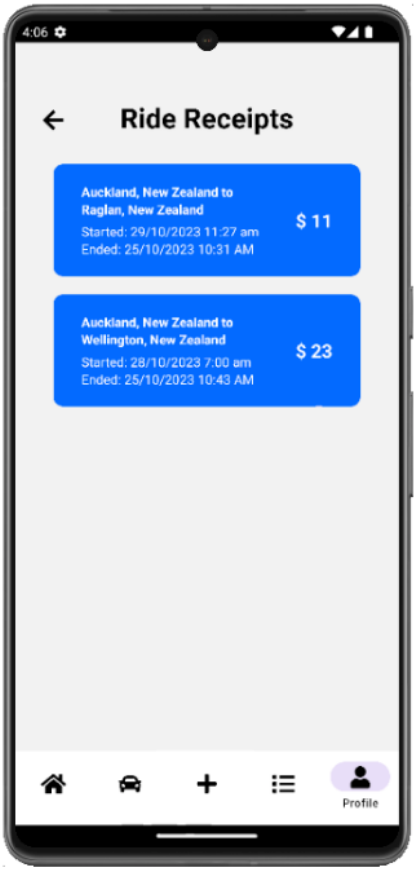


Fig 1.76 - Ride Receipts

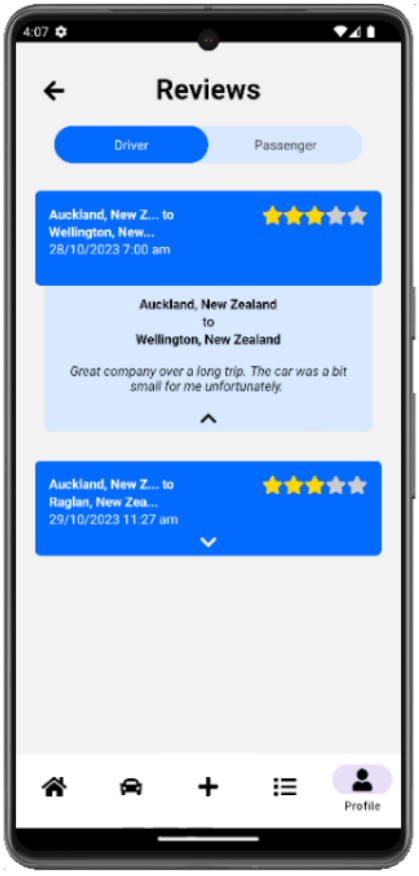


Fig 1.77 - Ride Reviews

Design features

Components

For all of our buttons, we set a radius of 10 for consistency throughout the app. We found that this was less jarring than a full “bubble”. We trailed using elevation and shadows on different elements of our app.

After looking at other app designs we eventually decided to go for a more flat design which seemed to be more the modern approach to app design. To avoid user input errors and accidental actions we also made sure to use Alert popups to confirm any major changes as well as inform users when changes were made.

If we were to release Trippr to the public domain we would need to spend some time going through the app and testing on a number of different size devices to ensure that the app renders correctly across all devices and fix up naming and small typos. Importantly, the app and the features that we have implemented on the current version generally work as intended.

Colour palette

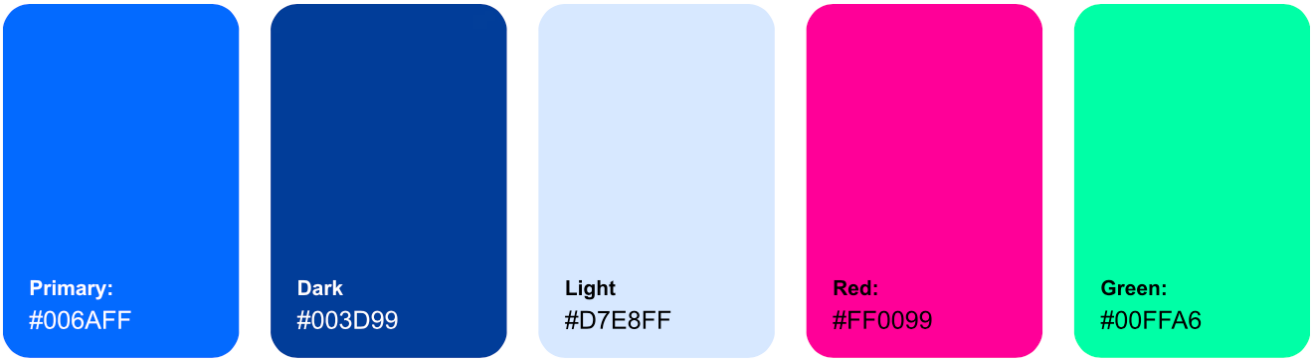


Fig 6.11 - Colour Palette

From meetings with our client, we learnt that he wanted a minimalist colour scheme. After experimenting with various colours, including purples, greys and greens, we eventually settled on #006AFF. The fresh and vibrant blue of #006AFF assures the user of a user experience as fluid as water and as dependable as the sky.

The primary app colour, #006AFF, also provides excellent contrast against the texts on our app so that it can remain readable no matter its size. The Dark and Light variants of #006AFF allowed us to have variety throughout the app while maintaining a central colour theme. Red and Green accent colours also tone well with Primary,

Dark and Light but allow us to use colour to communicate an element’s status to the user, such as on or off, accept or decline, as seen in Fig 1.62.

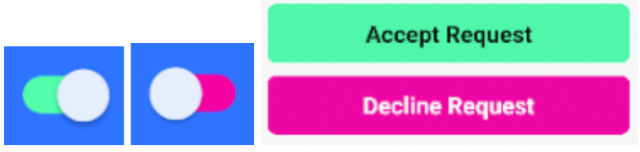


Fig 1.62 - Colour Palette

Gestalt Principles

Throughout the design of the app Gestalt Principles heavily influenced our design decisions as a team.

As our client enjoyed the minimalistic design approach that many successful apps have adopted it was essential that we understood gestalts principles inside and out to ensure we could separate elements without cluttering the page. As a carpooling app a challenge that we would often have to face is trying to show the user enough information for the user to feel the transparency/safety that our app advocates without cluttering parts of the UI. Below we will be going over just a couple examples of how we leveraged Gestalt's Principles when designing our UI.

The Ride Searching pages from figure 1.4 shows how we have considered these principles in every section as within just these couple of pages all 6 principles were used to influence our decisions.

The first principle of proximity is used when displaying all the ride information which is grouped together via proximity. Using the principle of similarity we have made a choice to follow a consistent colour scheme and font style and shaping. For example, we ensure that every card/button had a rounding of ten to ensure design consistency, especially when buttons were laid overtop of cards like how the enquire button is laid overtop of the RideDetails card.

The third and fourth principle is continuity and closure which go hand in hand in this example.

It was used here on the ride details screen as the card overlay was used to encapsulate the written details and separate the foreground from the background which contained the map. Which brings me on to the next principle, figure-ground. The principle focuses on the distinction between the main subject (figure) and the background. Here the written details card is separated from the map which is the background via the card as explained before.

The intractable nature of the background and having the card overlap the map also allows the map to be a background-object where people would intuitively want to engage with. Especially with the routing information having lines that lead off of the screen which invite users to drag along and follow the route.

To separate the figure/ground on the card we used coloured buttons. The final principle of symmetry and order. People tend to perceive symmetrical and well-organised arrangements as more appealing and easier to understand. Applying this principle in our design we ensured the buttons and font sizing was the same while trying to keep to the same ratio for buttons of different sizes, such as the bookmark button.

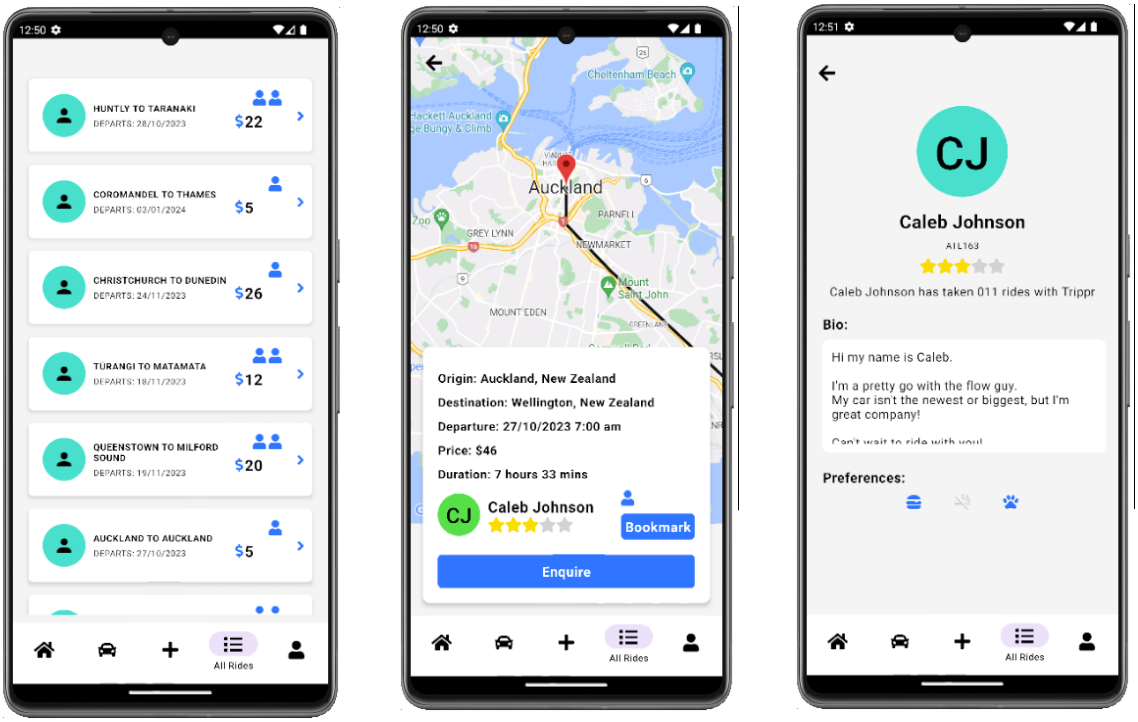
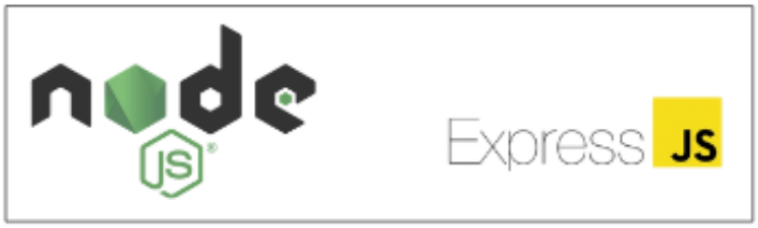


Fig 1.4 - Ride Searching

Design of the Backend

Components

Frontend	
Backend	
Essential Tools	

Process

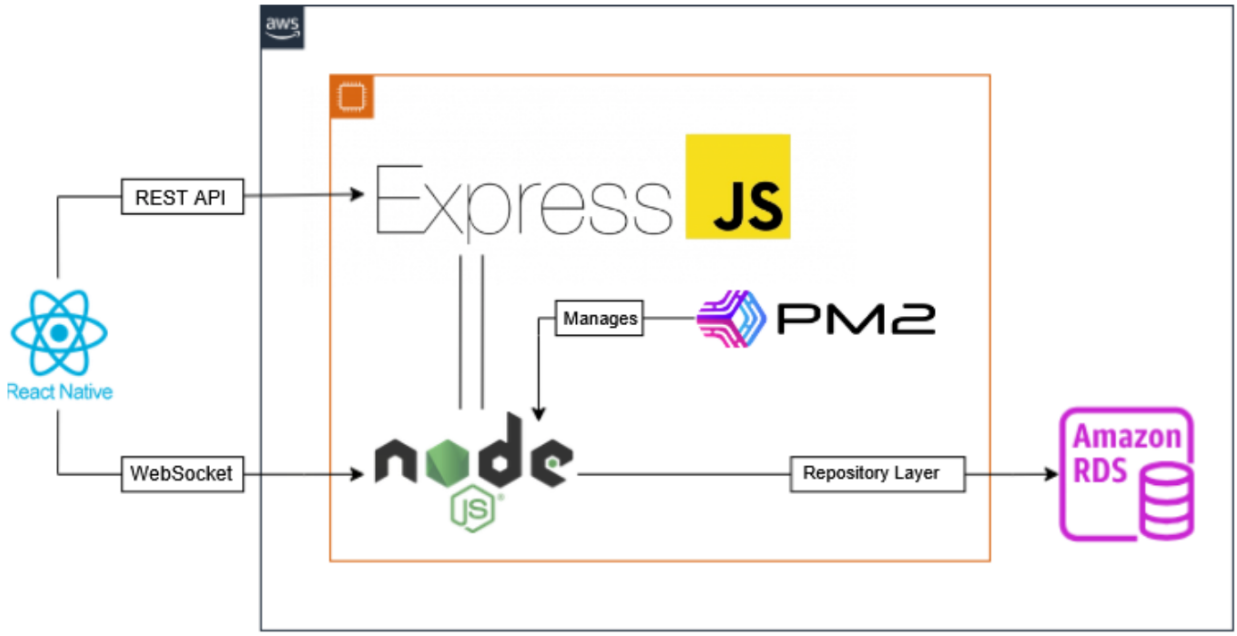
When designing our backend, we began with a user-centric approach. Our initial choice was Node.js, a highly scalable and industry-standard platform for numerous applications.

To enhance our Node.js implementation, we integrated Express in order to leverage its robust routing capabilities to efficiently structure our app endpoints.

For our chosen database, we opted for Postgres its open source nature resulting in consistent updates, and its SQL compliance.

With the core of our backend in place, we extended our commitment to Postman for testing and api documentation. Additionally, we employed PM2 to effectively monitor, log, and maintain the continuous operation of our Node app within the EC2 instance.

With scalability in our minds from the beginning stages, we have carefully designed each part of our backend application to be as scalable and readable as possible.



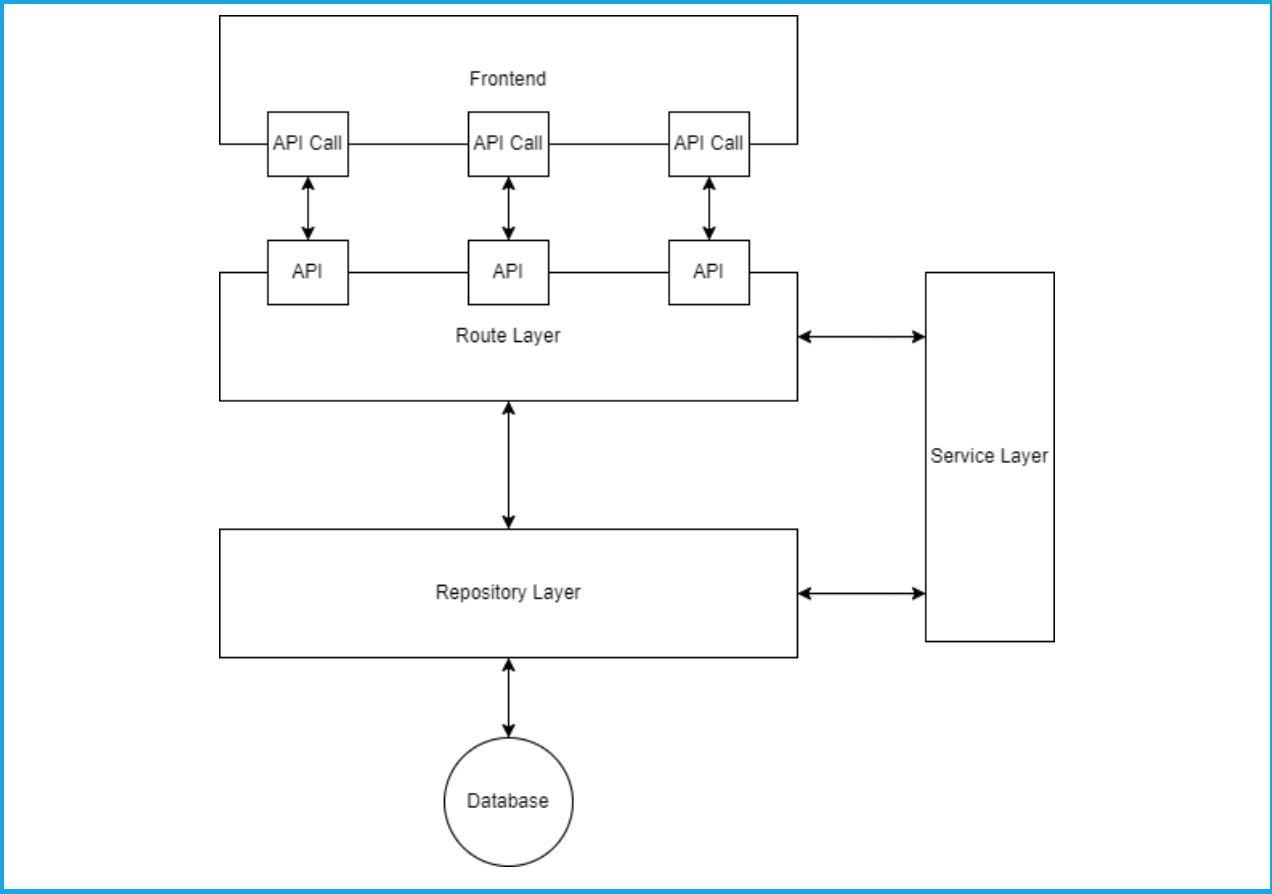
User-centric approach

consistent updates

Highly scalable

Node structure

To delve into the fundamental design choices of our backend, we've adopted a structured approach by breaking down our Node application into three distinct layers:



Routing Layer

This layer serves as the repository for all endpoints, and it also accommodates the WebSocket handler.

Services Layer

Here, the core business logic is computed, and this layer supports the Routing Layer by providing essential helper functions.

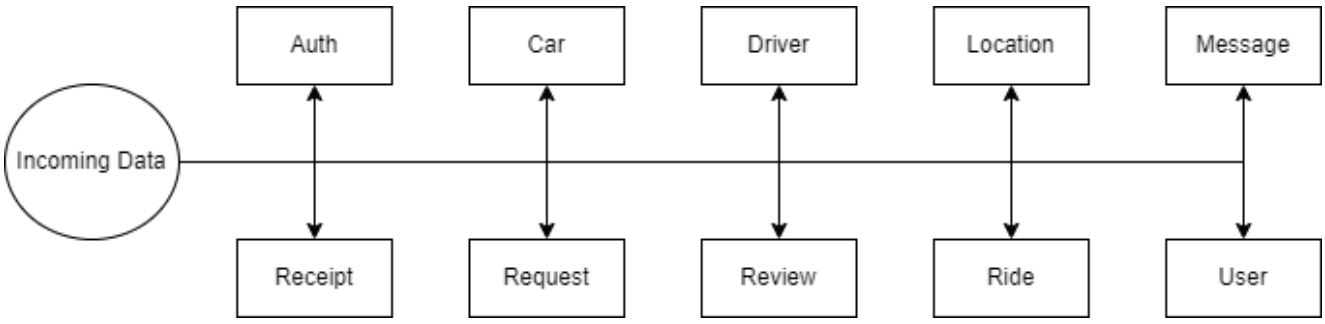
Repository Layer

This segment is responsible for housing all the SQL logic required for execution.

This layered architecture enhances the organisation and maintainability of our Node application, making both the expansion of the application, and the onboarding of future developers a seamless transition.

API Grouping

When dissecting our API design, our approach involved categorising endpoints to ensure that as the application scales, each endpoint's purpose remains easy to locate and comprehend.



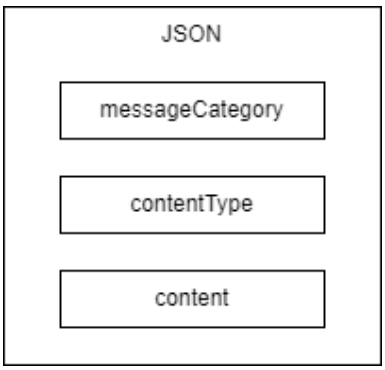
With this structure, the frontend benefits from greater readability due to the api usage information being embedded into the location. Backend developers also benefit due to the codebase components being logically categorised.

Web Socket Messaging

For seamless communication

Websocket works differently to API calls which can be differentiated by their location. In websocket any info is received without any information about what it contains. To solve this problem, the team decided on a formatting standard that we would implement in our communications.

After discussing and adapting our formatting so that any level of expansion of the application would be seamless, we settled on the following:



messageCategory

Grouping of the call much like the API groups above

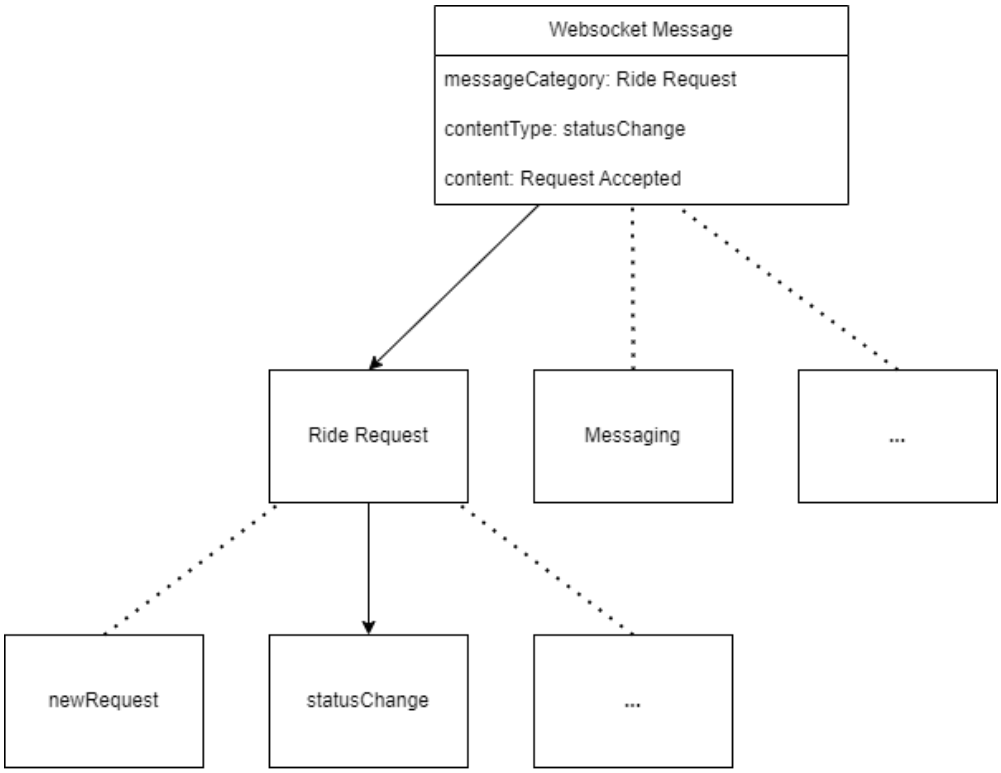
contentType

This field is used to notify the other application what operation it wants performed

content

This field holds all of the information needed to perform the operation

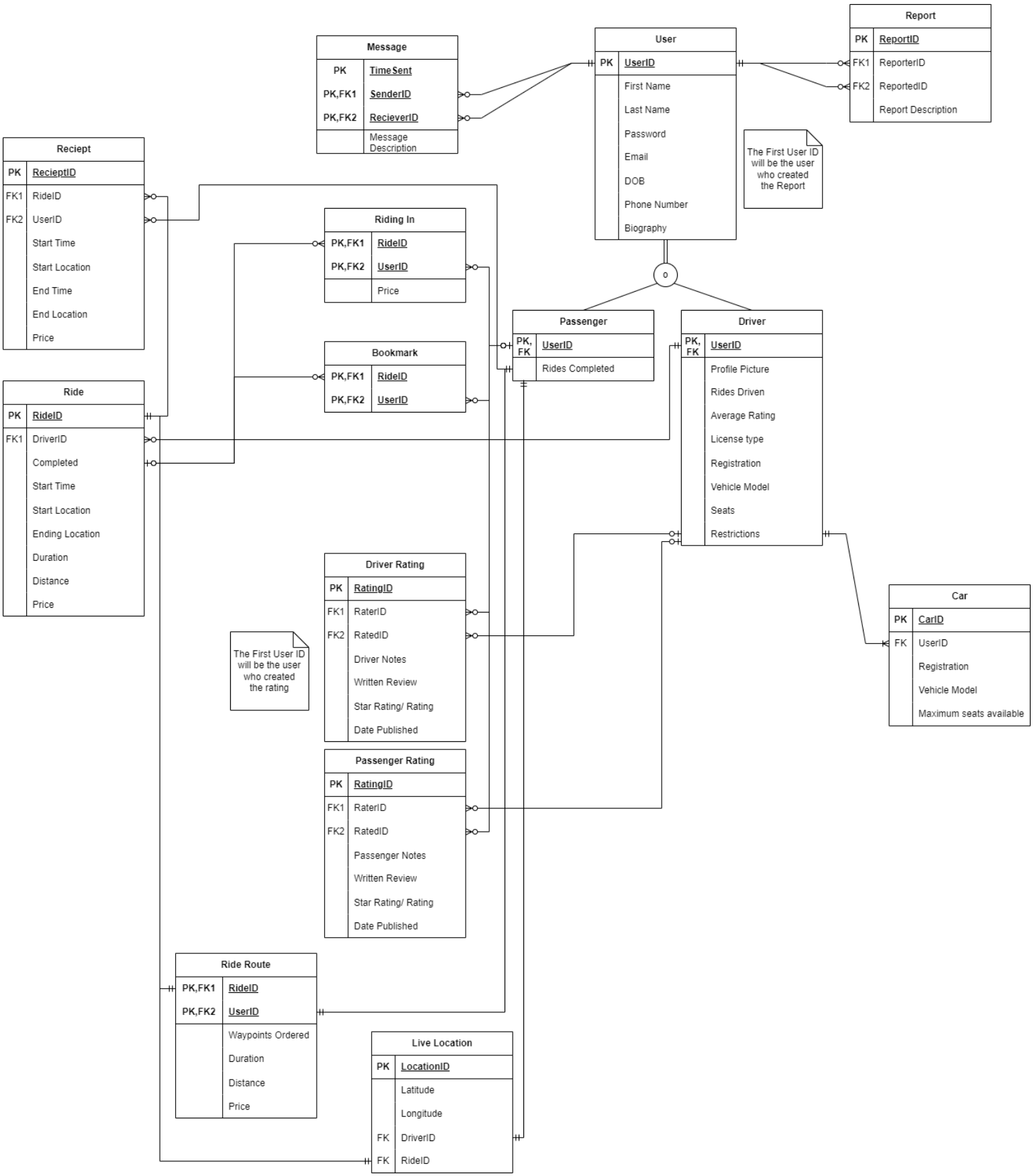
The efficiency of this structure truly becomes evident when incorporating new operations. It requires only the definition of a new category or content type, and adding the implementation of the new operation. The 'messageCategory' feature also plays a pivotal role in streamlining the WebSocket response times, especially in a scaled application, where the app is tasked with searching through a considerably reduced number of options, resulting in improved efficiency.



We meticulously designed our database using a Crow's Foot Entity-Relationship Diagram as the foundation.

This diagram served as our blueprint, allowing us to seamlessly integrate new project constraints and requirements as they emerged, ensuring the database's adaptability and robustness.

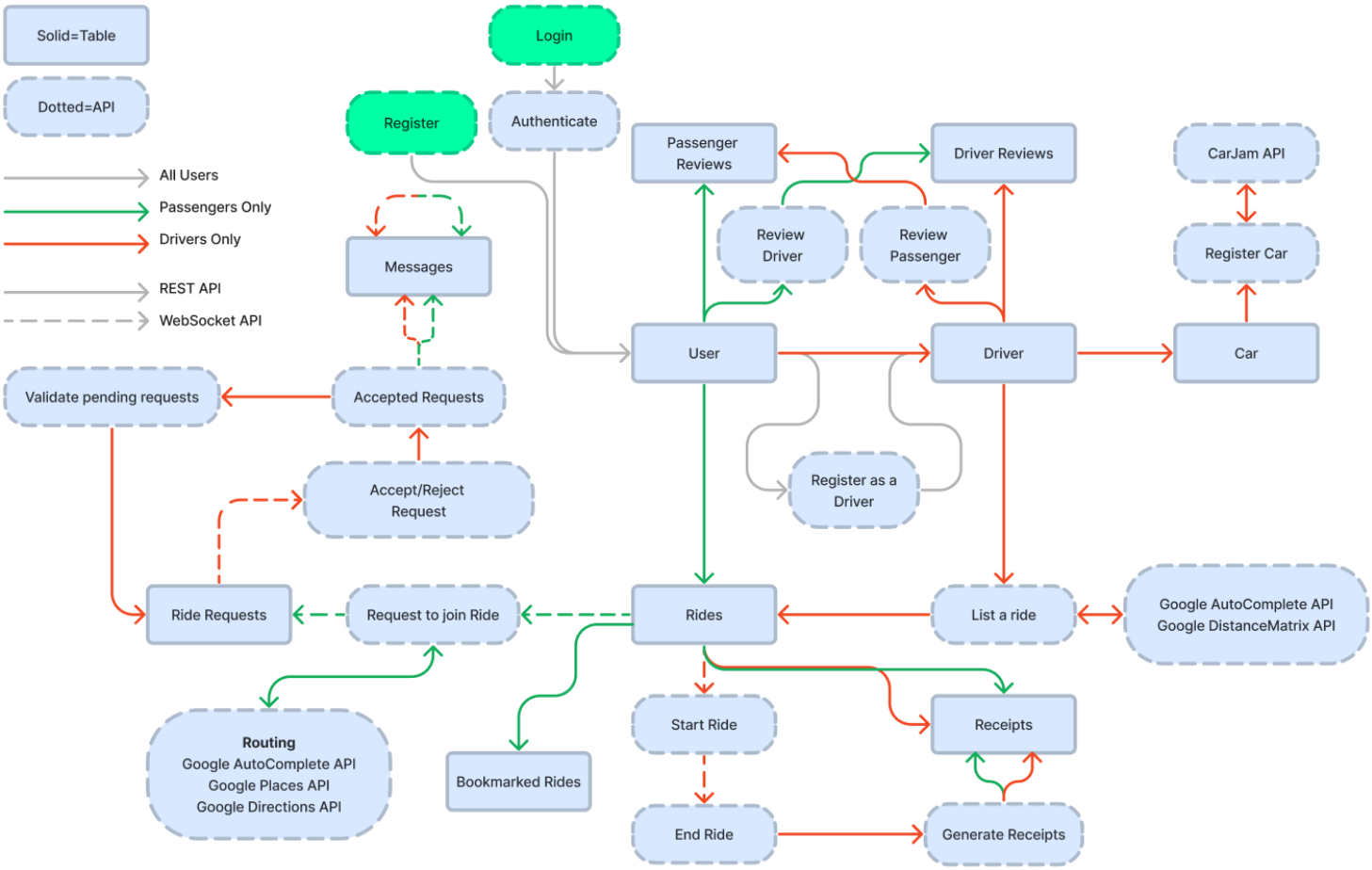
We found this to be an important step in our success as this visual representation allowed us to see potential problems before they emerged such as many to many relationships.


Message
PK TimeSent
PK,FK1 SenderID
PK,FK2 RecieverID
Message Description
User
PK UserID
First Name
Last Name
Password
Email
DOB
Phone Number
Biography
Report
PK ReportID
FK1 ReporterID
FK2 ReportedID
Report Description
The First User ID will be the user who created the Report
Reciept
PK RecieptID
FK1 RideID
FK2 UserID
Start Time
Start Location
End Time
End Location
Price
Riding In
PK,FK1 RideID
PK,FK2 UserID
Price
0
Passenger
PK, FK UserID
Rides Completed
Driver
PK, FK UserID
Profile Picture
Rides Driven
Average Rating
License type
Registration
Vehicle Model
Seats
Restrictions
Bookmark
PK,FK1 RideID
PK,FK2 UserID
Ride
PK RideID
FK1 DriverID
Completed
Start Time
Start Location
Ending Location
Duration
Distance
Price
Driver Rating
PK RatingID
FK1 RaterID
FK2 RatedID
Driver Notes
Written Review
Star Rating/ Rating
Date Published
The First User ID will be the user who created the rating
Car
PK CarID
FK UserID
Registration
Vehicle Model
Maximum seats available
Passenger Rating
PK RatingID
FK1 RaterID
FK2 RatedID
Passenger Notes
Written Review
Star Rating/ Rating
Date Published
Ride Route
PK,FK1 RideID
PK,FK2 UserID
Waypoints Ordered
Duration
Distance
Price
Live Location
PK LocationID
Latitude
Longitude
FK DriverID
FK RideID

Implementation

Overview of the backend:

Note that many APIs that update or return information are not included in the diagram but exist in the app.



Implementation (cont.)

Backend

Users

When a user starts Trippr for the first time, they will be prompted with a login screen with the option to register. The stored password is hashed, and we use Passport.js to authenticate our users.
Once the user is logged in, they can view the available rides or register as a driver.

Driver

Drivers need to register a car before they can list a ride. When listing a ride, the driver will enter their trip origin and destination, the number of seats in their car, and the time they begin the trip.

Google AutoComplete will always try to complete the location, limiting them within New Zealand and ensuring we can't get overseas places or problems calculating trip information. The DistanceMatrix API calculates the distance of the journey and the duration.

Passenger

When a user requests to join a ride, they enter the number of seats they wish to book and the departure and arrival locations, again using the AutoComplete API. The Google Places API is used to get more information about the location, such as coordinates and the formatted address.

This is then passed to Google Directions API as a waypoint and any other passenger waypoints. After verifying that the newly completed route is possible, the price is calculated, considering their section of the trip, the average fuel price, and the average fuel economy of a modern car.

Driver

The driver for a ride can accept or deny passengers for their ride, accepting a driver enables messaging between the driver and the passenger, and existing pending ride requests for that journey are recalculated to ensure the new route is valid. When the driver for a ride starts the ride, the driver will no longer be able to accept pending ride requests.

When the ride is finished, receipts for the driver and passengers are automatically generated, and both parties are encouraged to leave reviews about each other.

One of our unique features is integration with the ***CarJam API***; when a driver passes through their vehicle registration, we call the ***CarJam API***. Then it returns information on the associated car, such as its ***model, make, colour, and year of manufacture.***
It costs to use real plates. However, we are using their test environment, which we can use for free, giving us this feature as more of a proof of concept.

REST API

The REST API we developed was made using Express; the client sends the server a POST, PUT, GET, or DELETE request.

Once the server has processed the request, it returns a JSON message before terminating the connection and sending the appropriate HTTP response code. Different Requests have different uses:

POST: Used when adding something to the database, such as creating a user or listing a ride

PUT: Used for updating values, such as changing the status of a ride.

GET: Used for returning information to the client. Get Requests don't accept a payload from the client

DELETE: Used to delete data from the database.

The REST API will always return a JSON object with the same format to the client for every connection.
"message": Content,
"messageType": Status

If the request is successful, the message content will be what the client requested, and the status will show "Accepted". If the request is unsuccessful, such as missing JSON parameters, the content will contain an error message "Malformed Request", while the status will give us an error code "JSONError". These error codes made debugging straightforward, as the front and back end had different developers working on the project. It also lets us inform the user what went wrong if it was a user error.

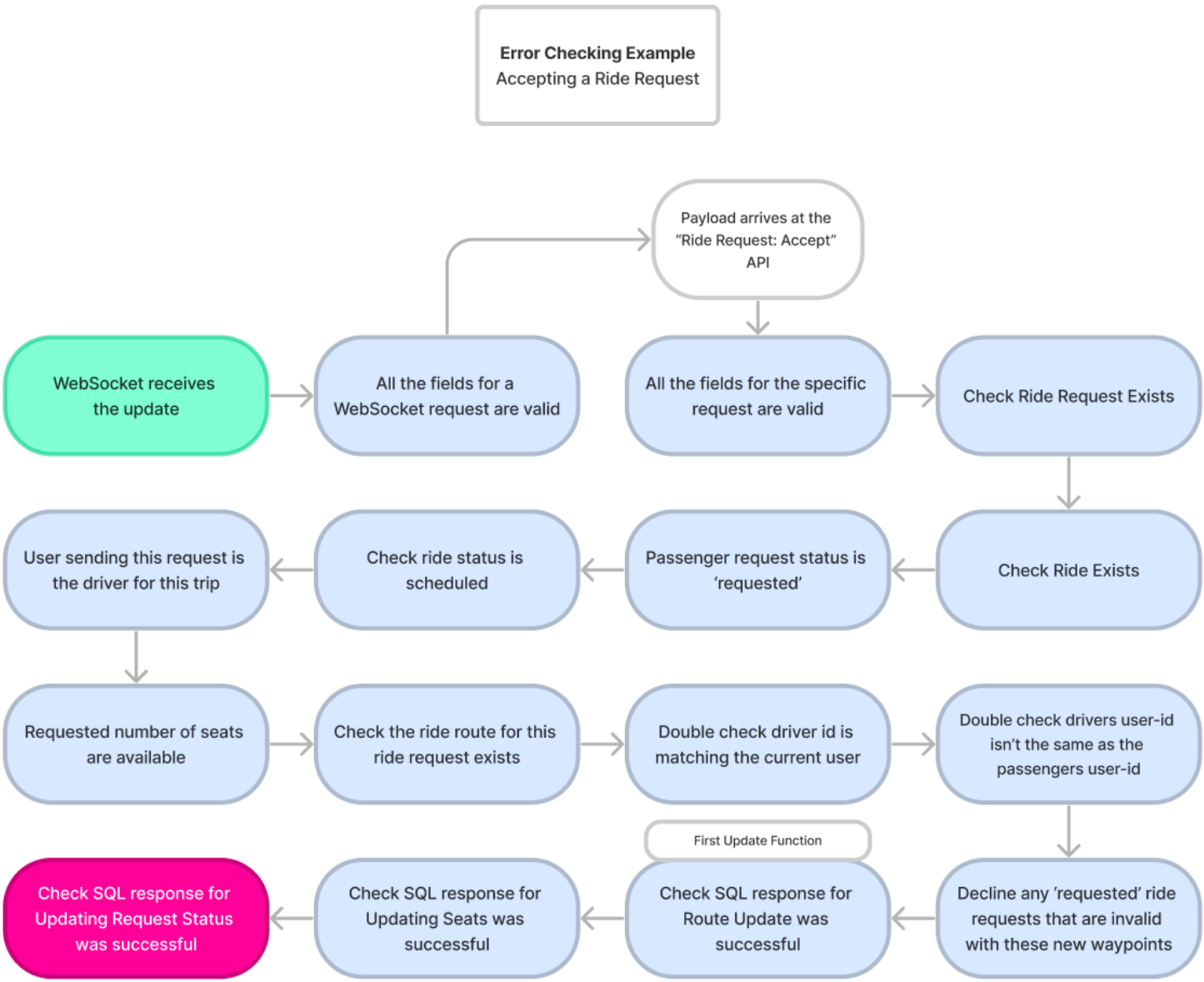
Error checking was an extensive and constant focus for the backend, especially with WebSocket, as these interactions involved multiple users. The flow for a call is generally as follows: firstly, we check that the user is authenticated/signed in. Then we inspect the JSON received, checking that the fields are valid and values are usable (Numbers get correctly parsed to float/integers, emails are in the correct format etc.).

A second round of authentication is generally done now to ensure it's the appropriate user for the API call, such as a user only changing their biography. The service layer generally checks that the request is possible, such as ensuring the user is a driver before trying to give them their driver details or allowing a user to only update their information. Some calls don't need much error checking, such as returning ride information, as we only had to ensure the ride exists, which was information accessible by everyone. Other APIs were incredibly intensive and had many checks, the most significant of which was accepting a passenger into a ride.

This example is just one API call; **we have over 50**.

Implementation (cont.)

Error checking example



The Challenge

When initially designing Trippr, we wanted to incorporate dynamic routing with multiple passengers on a trip;

however, as this wouldn't get developed until two-thirds of the project was completed, we didn't dig deeply into the complexities of it. One of the unforeseen problems we encountered was the travelling salesman problem.
The travelling salesman problem is finding the shortest path between multiple cities; it isn't a new problem, but computationally expensive. Fortunately, there is a formula for this, which we must modify, as we don't return to the starting city, and a passenger's origin must come before their destination. This changes the total amount of combinations to be (n-1).

This gives us six combinations with two passengers and 39,916,800 combinations with six passengers, which is still too large. The initial realistic solution was using the Google Places API to get coordinates, and use this to calculate the straight line distance between two places, which was free. The nearest neighbours algorithm could have been used with this new method to reduce the runtime with a runtime O(n*2), which would have been reasonable.

The Solution

Fortunately, the Directions API given by the Google Maps platform can solve this, as it has the option to optimise waypoints and returns a route of the shortest journey, charging USD one cent per call.

This saved the backend a considerable amount of work, with one flaw: the route could put the destination for the passenger before their starting location.

For example, suppose a driver is driving from Auckland to Wellington, and a passenger doesn't know their geography well. They apply to join the trip with their origin and destination as Levin and Taupo (the opposite direction the driver is travelling). In that case, the calculated route is Auckland – Taupo – Levin – Wellington, something the driver could quickly look over as the route looks correct on the map.

Fortunately, the API returns the order in which the waypoints are returned as indices; we just need to check each passenger's origin before their destination, and we can validate the route.

Implementation

Frontend design

Post-it note User Flow

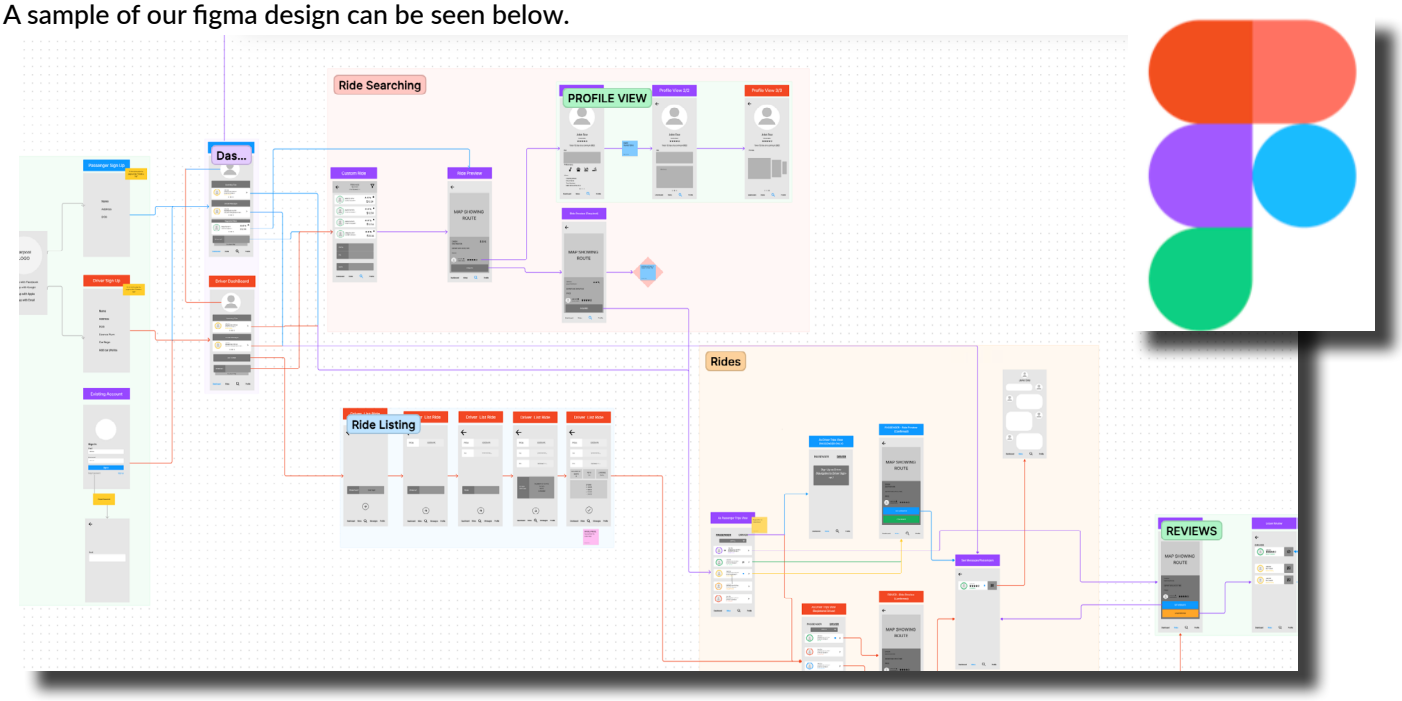
Design for our app was initially done using post-it notes on a white board. We considered what our users needed at the core of our app. These were the ability to create an account, list a ride and join a ride. Fig 5.1 shows what this initial design looks like.

Fig 5.1 Initial Post-it User

Figma Prototyping

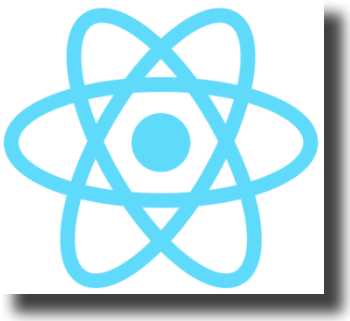
The next step of our design process was to use figma to create a more detailed overview of our apps user flow, the dependencies and different states. Using Figma, we were able to also create low-fi designs for our screens.

This figma document served as the main resource to inform our frontend app development. The front-end team would continually refer back to this document to inform design decisions and guide development.
A sample of our figma design can be seen below.



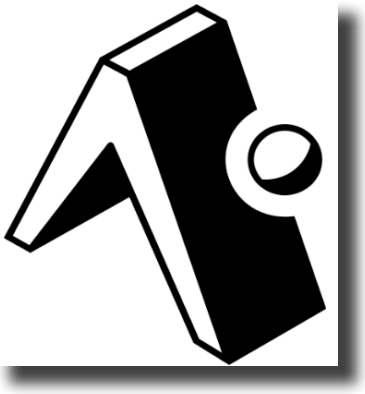
React Native and Expo

We decided to use React Native along with Expo to develop our app.



From the beginning of our project, we knew that we wanted to use React-Native to develop our app due to its cross-platform usage. Using React-Native meant that we only needed to write code once to be able to create apps that worked on both iOS and Android devices.

Another reason for using React-Native was because of its wide use throughout the industry, so taking the opportunity to have hands-on experience with it would be extremely valuable for our future employment. Because of its widespread usage, there was also a significant development community, which meant if we encountered issues in our development, we would have a good chance of finding helpful documentation online to navigate the problems.



As novice users of React Native, Expo provided a simplified and streamlined development process by removing much of the complexity of setting up a react native project. Expo allowed us to quickly begin the development of the logic and the UI of Trippr.

Expo also offers a convenient command-line interface (CLI), including a Quick-Start that allows us to check any development we did quickly. Additionally, Expo comes with a wide variety of components ready to use right from the beginning of the development.

Overall, Expo helped us to get started quickly with the development of our app using react native and be able to progress with the software as soon as possible.

Other Helpful Tools

A few additional dependencies that we utilised within the development of the app to help add the functionality we desired. These are listed below:

Async Storage

This was used to store the user token and registration complete status so that a user would only be shown screen asking if they wanted to signup as a driver once after their registration and be able to retain their login status even after closing the app.

React-Navigation

React Navigation allowed us to navigate the user around the app through button presses. The Material bottom tabs dependency meant that we could have a bottom tab navigator so that users could navigate quickly between the key modules of our app.

Date Time Picker

The date time picker was used in any situation where we required the user to input times or dates. This allowed us to control the data that we were receiving from the user to avoid errors, unusual date inputs and restrictions or limitations on them. Not only this but from the users perspective it is much more user friendly and familiar to use these widgets than text input.

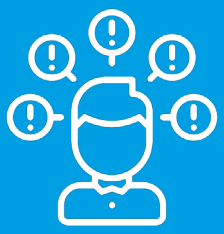
Formik and Yup

We used Formik and Yup for our data validation. Again this allowed us to verify that users were inputting the correct formatting within input fields and importantly completing all fields. By doing this, we were also able to disable submission buttons until the correct information was added.

Google Places Autocomplete

Like our other helpful tools, Google Places Autocomplete helped us to control the users input so we could avoid issues when passing the data through to the backend. Again, this was also helpful for the ease of use for our users and is almost expected by any user familiar with inputting addresses

Problem



Throughout the app's development, we had no prior knowledge of developing an app of this scale. Due to this, we needed to gain experience properly managing dependencies, whether that be installing or updating them. Furthermore, due to some of the components we used, we had to roll back dependencies for them to work. A technical challenge that we had yet to work through before.

This problem was exemplified by the limited knowledge that the team had with expo and react-native, so when trying to install dependencies, rather than sticking with one package manager, we tried to use every method until one worked. This turned out to hinder us later in the project when we could not add any more packages or build our app. We found the proper way to remove all modules and locks to rebuild our app from scratch using yarn, a package management tool to manage our packages effectively.

This was the critical issue that restrained our development. With the proper guidance, we may have been able to avoid this issue and exceeded our original MVP (Minimum Viable Product). This took away from weeks of production as without the new modules none of the app could be built and tested. Furthermore, it was only in the latter half of those weeks did we learn that it was a dependency management issue and not due to a deprecated module. Because of this we had assigned members to learn how to implement new modules into the app which took away our production time significantly.

Results & Evaluation

Good requirements are objective and testable.

Project Goals

Over the course of the project, we have managed to create an app that is successfully able to allow users to list, book and enjoy rides no matter their location in New Zealand.

At the start of the project, our client had given us the following goals:

Create an app for people in New Zealand that enables them to carpool on short or long trips.

Idea

Brainstorm

Realisation

Secondary Goals:

- Model the European app ***"BlaBlaCar"***
- Keep the app simple
- Use a non-intrusive revenue model
- Allow users to sign up as a passenger or driver
- Make signup as frictionless as possible
- Map view to see the location of users

- Integrate Bitcoin Cash
- Have a large selection of searching criteria for users finding rides
- Ban profiles when they hit a threshold (report or star rating)

The primary goal has been successfully met through collaborative efforts within our team, resulting in an app that empowers users to list, plan, book, and enjoy carpooling journeys throughout New Zealand. We have achieved this by maintaining a highly available backend and a responsive frontend, ensuring all necessary functionalities and information are readily accessible to users.

In addition to fulfilling our primary goal, we have made substantial progress in addressing the secondary goals, contributing to the overall depth and effectiveness of Trippr.

Sign up

To ensure simplicity and a frictionless signup process, our frontend team has fostered a clean and user-friendly feel through providing information only when it is needed

Revenue

For our revenue model, we have adopted a small % transaction fee on payments concept when designing our automatic pricing calculator. This low percentage enables users to maintain affordable pricing whilst also offering relief to our drivers.

Google Map & CarJam

Our integration with Google Maps and CarJam has enhanced the user experience by providing location information during journey planning and the ride itself. This significantly reduces the effort required by users to plan their travelling adventures.

Let the work speak for itself.

While we have made substantial progress, there are a few additional features we plan to refine and integrate into our operational app. **These include implementing Bitcoin Cash support, introducing a profile banning feature based on user reports and star ratings, and expanding the available search criteria to provide users with greater control over their experience.**

Results & Evaluation (cont.)

Testing

Design:

In comparison to backend, frontend testing requires more unique testing methods in order to try and cover any bugs that are present within the app. The first would be navigational errors which require stress testing by navigating to every page within scope and trying to create error logs generated by expo. An example of this would be when trying to navigate between screens where user information is transferred from one screen to another we had to ensure the right information is being passed through no matter what.

Within the application, our team delegated tasks to complete which we described as modules. For example, one person was delegated the ride listing component. When the ride listing component was complete the rest of the front-end team would go through and try to break the frontend by testing methods that they had found problems with when developing their own modules. Due to the limited time constraints our strategy was just to record any type of exploit/loophole that we could find that we would retest and ensure the same bugs/mistakes would not be repeated when developing further.

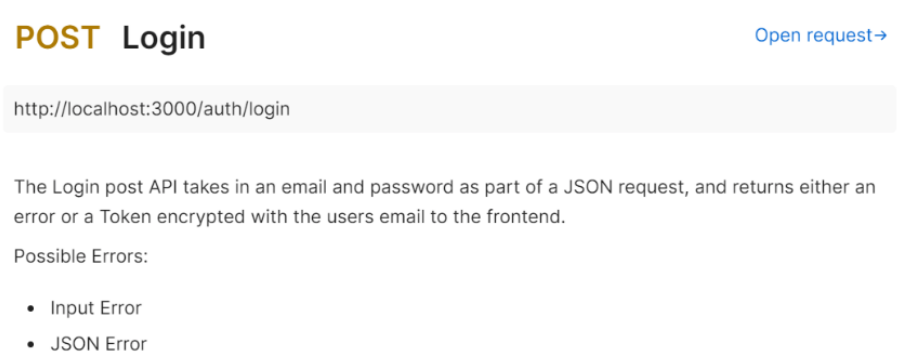
This worked well as most of us still had a lot to learn in react native so our debugging process exponentially improved as the more we coded, the more we learnt and logged. We also ensured that any UI that we create works well with any device by having a broad range of testing devices.

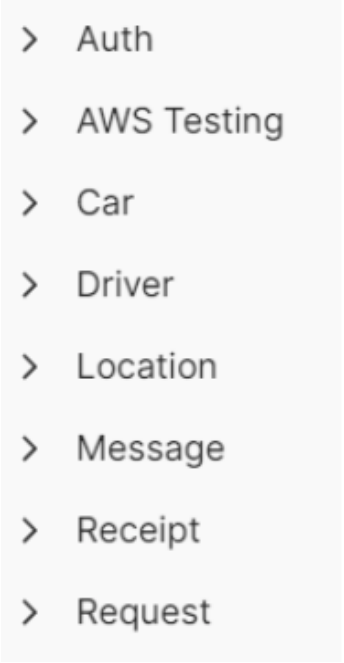
We used the most popular phone screen resolutions as a guide in order to pick phones that would fall into the top 50% of screen resolutions.

Backend functionality:

When testing the functionality of our backend, our team has incorporated Postman.

Through separating test cases into their categories, we have been able to identify the possible errors that frontend might encounter.





Each test has a description of what the endpoint or connection should be both receiving and sending.

We then display an example body if applicable:

Body raw (json)

json

```
{
    "email": "d@Y.Z",
    "password": "password"
}
```

Every test case comes with a try out section, where any developer is able to test different cases. This allows the frontend team to have a hands-on and visual way of understanding each connection and api, and is used by the backend team in their rigorous testing.

JavaScript - Fetch

```
var raw = "{\r\n    \"email\": \"X@Y.Z\",\r\n    \"password\": \"password\"\r\n}";

var requestOptions = {
  method: 'POST',
  body: raw,
  redirect: 'follow'
};

fetch("http://localhost:3000/auth/login", requestOptions)
  .then(response => response.text())
  .then(result => console.log(result))
  .catch(error => console.log('error', error));
```

App integration:

In terms of frontend testing, we also initially tested through Postman to determine how the information was being sent over to us and then created JSON objects to replicate the formatting we would receive. Then, after we parsed that information, we tested it and ensured it worked with a local JSON file before trying to call the APIs from a local database that each of us set up.

When information was being sent the other way around, we replicated the JSON and logged the JSON into the console before sending the information. Again, through Postman, we ensured that all forms of our object that we built in the frontend would be accepted. However, thanks to backend work, our APIs had exceptional error handling and would send the frontend team specific error messages that we would work on debugging.

AWS integration:

id	name	mode	□	status	cpu	memory
0	app	fork	4	online	0%	80.6mb

When it came to integrating the frontend and backend components, our initial setup involved hosting the server and database locally. However, as the semester progressed, we decided to migrate the database and Node server to AWS (Amazon Web Services). This strategic move enabled our frontend team to conduct comprehensive testing, not only on their emulators, but on their mobile devices as well.

In order to maintain a smooth transition and detect any potential issues that might arise when merging, our backend team implemented PM2 into their AWS EC2 server for error logging and to ensure permanent server uptime. This proactive approach helped in swiftly identifying and addressing any bugs and crashes that surfaced during the integration process.

Critical Evaluation:

During the course of the project, our team conducted comprehensive testing on individual components and their integration. Each team member rigorously tested their peers' work to ensure that every artefact benefited from multiple perspectives and the unique skills of our team, contributing to the continuous improvement of Trippr.

This rigorous testing resulted in us identifying the strengths and weaknesses off Trippr in its current state as such:

Strengths:

- Built with scalability and the future in mind

- The app provides a smooth and easy user experience

- Users are able to get instant feedback on their status with the app through websocket

- A sense of community is made through the messaging and biographies that each user is able to use at their own convenience.

- The app places high value on the security of each user both in the app and on their journey through reviews, encryption and transparency

Weaknesses:

- Due to monetary constraints, testing on an iOS phone rather than an emulator was not an option available.

- The app does not currently have a form of user feedback towards their experience with Trippr

- Restricted internet connection may cause problems with the users interaction with the app

Future Work

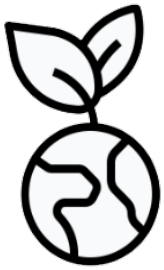
Areas for improvement and expansion in the future

Group Interactivity

By increasing user-to-user engagement through features that enhance group interactivity, we could deepen the relationships between ride participants before a ride even begins.

There are two specific features here that we, the Trippr team, believe could facilitate this sector. The first is Spotify integration. Integrating music streaming services such as Spotify into our app to allow users to create and share playlists during their carpooling journeys would enhance the overall user experience. This could be through listening to music you enjoy over a longer trip, exploring new music genres through new people, or the social interactions that occur while contributing and forming the new playlist.

Another feature would be implementing a group chat feature within the app. Allowing users to communicate and coordinate more effectively during rides. Group chat can help users plan their trips, share information, increase pre-ride interactivity and, most importantly, stay connected.



Environmental Stats

By leveraging our implementation of the Carjam API, we could develop a feature that calculates and displays the environmental impact of each user's carpooling activities.

The metrics we gathered could be used in many ways, such as gamifying our app to include these statistics when viewing others' profiles, which would also give another metric for people to evaluate a driver's status.

Payment options

Another feature we look forward to implementing would be a more streamlined payment method such as stripe and/or cryptocurrency.

Our in-house price calculation logic already streamlines the pricing factor of carpooling, so streamlining the payment process would make it that much easier for users to set up and undergo rides.

Robustness

The final factor that I would want to work on is robustness. As we found out, even with the amount of debugging that we did, the app can still be fragile at times.

One point of focus would be the Google Cloud services, as they seem to require a very stable connection in order to function correctly. A way we could get around this would be implementing some sort of offline functionality for the core sections of our app and storing rides in temporary storage that users may need access to even in unstable network connections, such as the map view and location data.

Redundancy and failover mechanisms as a backup to the Google Services could also be used to increase the reliability of the app.

Conclusion

Looking back

Throughout this project, we successfully devised a solution to alleviate traffic congestion, reduce greenhouse gas emissions, establish a cost-effective transportation system, and laid the groundwork for fostering a strong and unified community.

Diverse User Preferences:

User preferences vary significantly, particularly in an app catering to a broad audience.

Technical Expertise:

Developing an app of this calibre demands comprehensive knowledge across multiple domains, including GPS integration, real-time mapping, payment processing, and scalability planning.

Feedback Integration:

Enabling user feedback through peer review is essential for fostering trust and improving the app.

Data Privacy and Security:

Early insights emphasised the critical importance of data privacy and security for our app.

Operational Efficiency:

Streamlining the operational aspects of the app, including ride searching, route optimization, and messaging channels, is essential to provide a seamless carpooling experience.

Our major achievements encompassed the realisation of excellence through the **EEE principle (Economic, Efficient, Eco-friendly).** This accomplishment significantly bolstered the app's cohesion, thereby amplifying the benefits provided to each user.

Additionally, the integration of advanced live data APIs such as ***Carjam*** and real-time location data from the ***Google API*** proved to be a substantial game changer for our project, greatly elevating the overall user experience.

The project's most notable outcome was the successful creation of a user-friendly, enjoyable, and reliable carpooling app that remains readily accessible and has the power to become an integral part of daily routines.

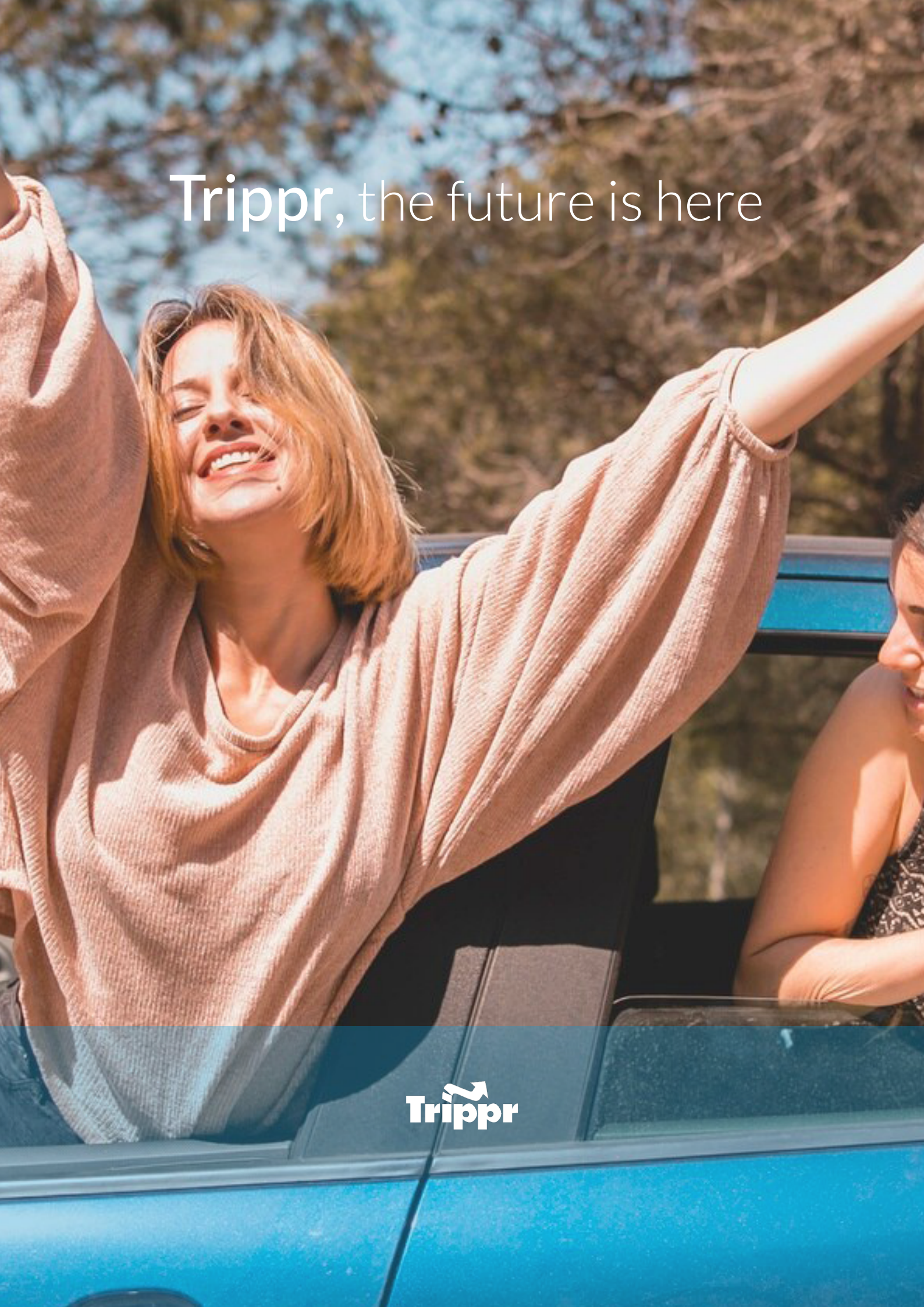
In the development of Trippr, we have successfully crafted an innovative, user-centric, and environmentally-conscious transportation solution to meet the diverse needs of individuals all around New Zealand.

Appendices

Authorship table

1. Title Page: **Jae**
2. Executive Summary: **Josh**
3. Table of Contents: **Jae**
4. Introduction: **Josh**
5. Background: **Josh**
6. Specification & Design: **Reuben, Zane**
7. Implementaion: **Michael, Reuben**
8. Results & Evaluation: **Zane, Ethan**
9. Future work: **Ethan**
10. Conclusion: **Zane**
12. Appendices: **Jae**

Editor-at-large & Presentation: **Jae**


Trippr, the future is here
Trippr