UNSW SYDNEY

COMP3900 Computer Science Final Report

P7. Car Rental Management System

Parkify

Aditya Banik

Table of Contents

1. Overview

The car space renting management system is a comprehensive solution designed to streamline the process of renting parking spaces. It incorporates a modern tech stack consisting of React for the frontend, Python for the backend, and MongoDB for the database. Additionally, the system integrates two key third-party APIs: Stripe for

payment processing and OpenStreetMap for mapping services. At its core, the system comprises several interconnected components that w5.3ork harmoniously to provide a seamless user experience.

1.1. Architecture Overview

The system architecture follows a client-server model, where users interact with a web-based interface or mobile application, while the server handles data storage, processing, and business logic.

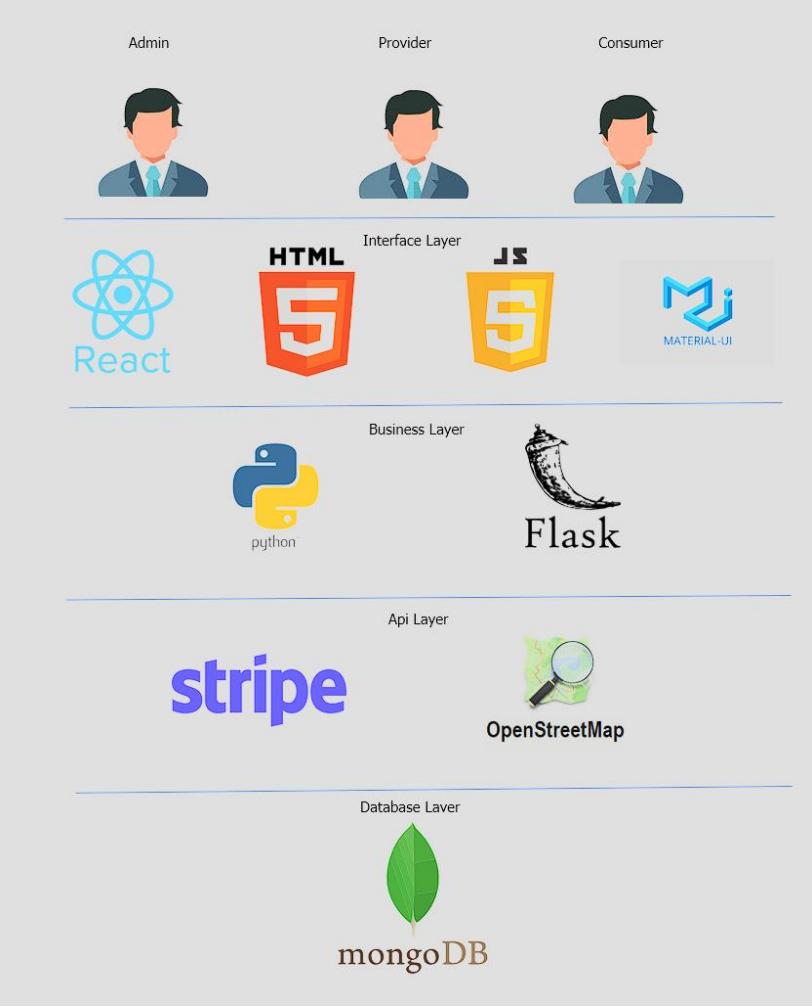


Figure 1: Software Architecture Diagram

1.1.1. Frontend (React):

The frontend of the system is developed using React, a popular JavaScript library for building user interfaces. React's component-based architecture allows for the creation of modular and reusable UI components, enabling a responsive and interactive user experience. With React, users can seamlessly navigate through the system, search for parking spaces, and make bookings with ease.

1.1.2. Backend (Python):

Flask serves as the backend framework, providing a lightweight yet powerful foundation for implementing server-side logic and handling user requests. With Flask, RESTful APIs are developed to manage authentication, interact with the database, and execute business logic efficiently. Its simplicity and flexibility make it well-suited for building scalable web applications, ensuring smooth communication between the frontend and the database.

1.1.3. Third-Party APIs:

- **Stripe API (Payment Processing)**: Stripe provides a secure platform for handling online payments. Integration with the Stripe API enables users to make payments for parking space bookings using various payment methods, including credit cards, debit cards, and digital wallets. Stripe's advanced features, such as fraud prevention and international currency handling, ensure secure and seamless transactions.

- **OpenStreetMap API (Mapping Services)**: OpenStreetMap offers detailed and up-to-date mapping data through its API. Integration with the OpenStreetMap API enables users to visualize parking space locations, nearby amenities, and directions.

- **Database (MongoDB)**: MongoDB is chosen as the database for storing and managing data related to user profiles, parking space listings, bookings, and transactions. As a NoSQL database, MongoDB offers flexibility in data modelling and schema design, making it suitable for applications with evolving requirements. Its scalability and performance ensure optimal data storage and retrieval, contributing to the overall efficiency of the system.

1.2. Functionality Overview

The car space renting management system is engineered with a paramount focus on ensuring a secure and personalized experience for all users. Its foundation rests on robust user authentication and management functionalities that establish a secure environment from the moment users create their accounts. Employing stringent authentication measures, the system safeguards user data, instilling confidence in

users about the privacy and confidentiality of their information. This approach not only fosters trust but also lays the groundwork for seamless interactions within the platform.

For users, the system offers a myriad of features designed to enhance convenience and ease of use. Beyond basic profile management, users can securely update personal details and payment information, ensuring hassle-free transactions. Meanwhile, providers are empowered to showcase their parking spaces comprehensively, leveraging detailed descriptions and high-quality images to attract potential renters. This comprehensive approach streamlines the browsing and booking process, enabling users to find suitable parking spots quickly and efficiently. Moreover, the system's integration of secure payment processing mechanisms, such as the Stripe API, adds an extra layer of protection, bolstering confidence in financial transactions.

The booking process itself is a testament to the system's user-centric design. With an intuitive interface and a 10-minute window for users to reach their chosen parking spot, the system prioritizes accessibility and convenience. A persistent timer serves as a helpful tool, allowing users to monitor the remaining time regardless of their location within the platform. Once a booking is initiated, a real-time timer tracks the duration, providing users with instant feedback and facilitating effective management of their parking sessions.

In the event of disputes or discrepancies, the system offers a recourse mechanism for users, further underlining its commitment to fairness and accountability. Users can raise disputes within the platform, triggering a structured resolution process facilitated by the system. This feature not only addresses user concerns promptly but also reinforces trust and confidence in the platform's integrity. Overall, the car space renting management system stands as a testament to the fusion of robust security measures, intuitive functionalities, and user-centric design, ultimately delivering a secure, seamless, and satisfying experience for all stakeholders involved.

2. Web-App Functionalities

2.1. User and Admin authentication

The user authentication is done by keeping security as our top priority. Considering this, when a new user is joining the platform, the user is prompted to verify that they are a real user and not a bot by verifying their email. A randomly generated number will be emailed to them, when received, they are required to enter it on the website to be verified. The system checks if this value is same as the number generated and grants access to the user if it is correct. The user cannot use other functionalities of the webpage if they have not verified their email. Moreover, the password that a user enters is SHA256 hashed. This is integral for data security, even in the case if the database is compromised, the password of the person will not be compromised at all. Whenever a user logs in, the system compares the hash value of the password entered with the hash value of the password that is stored in the database. As a result, even the developers do not know the passwords of their users and we prevent passwords being leaked in case of any attacks. Whenever an admin is onboarded, they are onboarded directly from the backend to prevent unauthorized onboarding of the admin, this prevents common users from having admin capabilities. This functionality addresses the project requirement of "System must be able to authenticate admin, provider, and consumer before updating any sensitive information."

2.2. Booking Functionalities

Being able to book a car space is one of the most vital functionalities of this software. Consumers are able to book for a **live** car space. A car space is said to be live if the provider puts it their listing as available between a certain time frame that they decide and they can deactivate a listing if they wish to do so by pressing a button, more details on this is given in section 1.3 of the report. If a consumer is interested in a listing provided that the listing is live and is not currently occupied by another consumer, they can book a listing. We recognize that when a person books a listing, it is not necessary that they are at that listing. In order to tackle this situation, when a person books a listing, that listing gets reserved for that consumer for 10 minutes. This grace period allows the consumers to travel to the place of listing and not be charged extra for that space since they have not used it yet. Once they have arrived at the listing, they can let the system know by clicking the "I'm here" button after which, the booking has been confirmed and the consumers get charged for time that they are using the parking space for. Once the consumer clicks the "Finish" button, the system ends that booking and calculates the cost for using that parking space using the formula $amount\ to\ charge = cost\ per\ hour * time\ (hours)$. It is

important to note that the user will be charged for the next hour. For example, if they have booked for 10 minutes, they will be charged for the whole hour or if the booking time is 1 hour and 2 minutes, they will be charged for 2 hours of parking. This was a design choice made by the team keeping the integrity of the software in mind. We recognized that there could be a case where the consumer could park for a small amount of time (~5 minutes) and the provider charges hourly, if in this case we had a grace time where the consumer is not charged for, they could get away with parking for free, which is not fair to the provider. Hence this design choice was implemented in order to ensure fairness amongst our different user bases. This functionality meets the requirements: “System must be able to calculate the total cost incurred for parking based on the time of booking”, “Consumer must be able to book an available car space and specify the duration of the booking”, “System must be able to accept booking of car spaces based on availability”, “System must be able to auto-cancel reservation if the consumer fails to succeed in among the window period”, and “Consumer must be able to cancel the booked car space.”. The entire booking system is also persistent, so no matter what phase you are in (prebook, booking, end booking), if you logout or go to another page and login or go back to the home page, the phase you were in will be shown with the correct information (e.g. time). This was and is important to do, as users cannot be guaranteed to always keep the website open and it must be expected that they may close the site and come back to it later when they for example arrive at the parking spot. This is a bonus feature that is not mentioned in the requirements, but is very important.

2.3. Search and filtering

The searching functionality is implemented keeping partial search queries for users in mind. For example, if someone enters a partial search query for a listing named “88 Christie Street” as just “88”, then the searching implementation will give a result as “88 Christie Street” and other similar addresses. This is done using regex where the user’s search query is simply matched against all the active listing’s addresses and the matching listings with a partial or complete match are shown. This functionality meets the requirement “Consumer must be able to find a car space from the list of car spaces registered in the system.”

2.4. Recommendation System

One of the most impressive features of our web app is the ML based recommendation system. The system custom-makes a proprietary dataframe for the entire user database and the listing database on the web-app. Once this data frame is created, the system makes a similarity matrix which learns the human-listing interaction of the user in real-time and assigns a score to relevant listings. The

parameters that it gives a score is based on the frequency of that listing being liked and the frequency of other users booking that listing. Once the system creates this similarity matrix, it uses the K-nearest algorithm with a size of 3 nearest neighbors to give recommendations on parking space to users.

This technique is a hybrid of Similarity Learning and K-nearest algorithm. We have found this to provide the most accurate predictions of parking spaces to users through extensive research and trial-and-error. The scoring metric has been kept very simple and does not use a weighted metric to score, rather, one score is given to a listing if it has been booked by a user previously and/or if it has been liked by the user. For example, if a person books a listing and likes it, the score for that cell (user email, listing_id) will be 1+1 = 2. This is done in order to maintain simplicity while running the algorithm. A picture of the similarity matrix is attached in the appendix for reference. This functionality meets the requirement of "System must provide consumers with recommendations for car spaces they have not booked yet but may wish to."

2.5. History

For the functionality of history, we allow both the user and provider to view their own respective history pages'. From the user's perspective, they can view all their bookings that they have made. From the provider's perspective, they can select any of their listings and then view all the bookings made under those listings. Both the user and provider can make disputes against each other (e.g. the user had a tree fall on their car, the provider was not happy that the user overstayed) and then these disputes are sent over to the admin for review. The disputes are submitted with the message of the disputer and any images that they would like to attach as evidence. Additionally, the users can click on a booking in their history page which will send them to that booking's listing's page where they can then like the listing (as they have booked that listing at least once). Thus, this functionality fills the requirement "Consumers can browse through their bookings and express their content with a car space they have completed one or more bookings for by liking it on the platform". The disputes functionality is a bonus.

2.6. Admin Functionalities

The admin has two main abilities. The first one is to be able to manage the users and their respective listings, where the admin can: edit/delete any user's account in the system and be able to edit/delete/live/unlive any user's listings as well. Essentially, the admin through this ability has full control over the system and is able to make any changes as they like. The second main ability of the admin allows them to view the aforementioned disputes that the user or provider sends. The admin is able to

view these disputes and can see full details such as the booking, listing, the complaining user's email and the complainant's email as well. From here, we give the admin manual control in terms of deciding what actions to take (email the users, make a police report, etc) as of course, the nature of the disputes can be quite dynamic. Then the admin is able to resolve and mark the disputes as resolved. These abilities cover the requirement "Admin must be able to view/update/delete information of all registered car spaces.". The disputes functionality again is a bonus here too.

2.7. Listing Functionalities

2.7.1. List Management

The provider users can manage their listing and perform related operation. And they have to provide their information to the payment gateway before starting to navigate the provider listings management page. The create listings functionality allows providers to provide their information of their car space and make a listing based on the information that they provide. The form provides an address search bar, a map, a quantity field, rate field, details text input field and restriction text filed, a thumbnail image file input and an additional images file input. The address search bar can allow users to search for their address and if a valid address exists, address options will show on the selection bar, allowing users to click and choose their intended input. "No location" is shown if the search location is not found by the search engine. Under the address search bar, we also get a map to show the selected result more clearly to provide a better user experience. We also provide the functionality that the map will pan to and zoom in to the address that the user selected. Combining the address search bar and map, our website is able to provide a good UI/UX on the address related operation. This functionality could be considered novel since an address input field with searching functionality and a corresponding real-world map is not a strict requirement. While filling out the details of their listing, providers are able to provide the quantity (number of cars the space can hold), price per hour, restrictions and any extra details such as amenities for their space. In addition to this, the provider can also upload a thumbnail and any additional images for their space too. The upload of image is optional but if user didn't upload a thumbnail. The system will attach a default thumbnail with their listing. The provider can also preview/remove/upload the uploaded images. We treated this as a bonus since the operation of uploaded images is not a strict requirement. After users fill in all the information and click the create button. Users will receive an alert as confirmation of the create listing operation. Thus, the above functionality fulfills the project requirement: "Providers must be able to register a

new car space, the cost per hour/day, availability, bank account details, and other details."

2.7.2. Edit Listings

After the listing is created, providers can manage their listing on the provider listing management page. They will see a list of listing cards which they can view their listing details and perform operation on. Users can edit the listing by clicking "edit" button on the listing. And the user can modify any input field except the address to update the listing and click edit to confirm. And they can also click remove to delete the current listing. This functionality is implemented similarly to the create listing described in the above section and fulfills the requirement "Provider must be able to update data of her/his existing car spaces."

2.7.3. Activate/Deactivate Listings

Providers can click on the "live status" button on the listing card to activate and deactivate the listing. Deactivated listings will show red color on the button and activated shows green. After the live status button is clicked a pop up is opened. For a deactivated listing, the default input range will be the today to a week later. User can click to select or type in the date input and click activate to activate a listing, which will be confirmed by an alert on the listing. If a listing has been made live, a pop up shows the current published date range and a deactivate button. Users can click on the button to deactivate the listing. This functionality fulfills the project requirement: "Providers must be able to register a new car space, the cost per hour/day, availability, bank account details, and other details."

2.7.4. Like/Unlike Listings

This functionality allows the user to like a booking if they have had a good experience with the listing that they have booked. The functionality of liking a booking is only available to users once the booking has ended. This design choice was made in order to remove inaccuracy on the user's part while liking a listing. It does not make sense that users would be able to like a certain listing without actually booking it. This helps in reducing bias, enabling the Machine Learning algorithm to produce accurate results for recommendations and not just put the most popular listings up only because the number of likes eclipses the number of bookings. This functionality meets the requirement of

"Consumers can browse through their bookings and express their content with a car space they have completed one or more bookings for by liking it on the platform."

2.8. Map Implementation

As we described in create listing, we used maps to provide better UI/UX experience to user. The map is included in individual listing detail pages for the user to navigate. In the home page of our website, we used a map to show listing locations to user, each black marker represents an individual listing. Each marker is clickable to show a listing card in order to see the details of listing, and the card is also clickable and redirect user to the listing detail page. Our application will ask users' permission to get their current location and centre the map on the user's current location. This helps users to navigate other listings around their actual physical location. This functionality is another novel feature as it further enhances the user experience of our application.

2.9. Payment system

Our website implemented a payment system to allow providers to receive 85% of the charge, customers pay for the parking slot and our platform receives 15% of service charge. The payment system is divided into the provider side and customer side. Although the data on stripe is separated for the provider/customer, our users won't be aware of it since we abstract the implementation details and provide a unique interface to them. As the requirement of stripe for a connected account, users must provide some necessary personal detail such legal names, bank accounts, phone numbers before being able to receive money from the platform. They only have to provide that information if they are interested in renting out their car slot.

2.9.1. Set up Provider Information

User can update their rent out information by clicking the menu "set up rent out information" Otherwise, if they haven't set up their information properly, once they click on parking slot management page. The website will redirect the user to the same page to the payment gateway link to set up the necessary information. (see appendix [1]) Furthermore, to use the service of the payment gateway, we have to set up our frontend on https connection, so all the data sent is encrypted. After they finish the form, the website will redirect them back to our website. After a few minutes for stripe to process, users are free to offer their listing and receive money from our system. The admin can view the connected

account detail on the stripe website (including money transfer record, personal information, bank account, see appendix [2])

2.9.2. Customer Payment Method management

Before making any booking, users must provide their payment details to us first. This is done to ensure that when users book a parking space, that space is paid for and not left used for free. Users can add/manage their payment method by clicking the “Add payment method” on the menu nav bar. Our business logic is the user provides the payment method first and we issue the bill and charge them directly on their default payment method after they end a booking. Therefore, User can enter their intended payment methods including direct debit /credit cards on our system. This is also a bonus feature since the saving of payment method and supporting multiple payment methods is not on the requirement. After a payment method is added, the admin can also inspect the change on stripe. (see appendix [3])

2.9.3. Customers can Remove/Set Default Payment Method

In the payment method management page, a list of payments added can be inspected. If the user clicks the “set default” and “remove” button on the list, a card can be set as default (from which all of the payments can be made) or the card can be removed, respectively. This allows customers to manage their own payment method and choose between saved method to pay for their services. This is another bonus feature since the payment method management is not a requirement.

2.9.4. Promo Code

Customers have the option to apply promo codes during checkout to receive discounts on their bookings in the end booking page. Valid promo codes are verified against the list of predefined codes stored in the system. If a valid promo code is approved, the discount is calculated and reflected in the final booking price before the payment processing. Again, this is also a bonus feature.

2.9.5. Bank Transfer

When the booking is ended, the money transfer is handled by backend. While transferring, 85% of the money goes to the provider using the connected account

id stored in our database and 15% of the money is stayed in our (the admin) stripe account as a service charge. After the payment is done, admin can also inspect the record on stripe. The customer related record is in the customer account (see appendix [4]) and the provider accounts and their money transfer records are on the connect field of the side bar (see appendix [5]).

All the above functionality satisfies the requirements: “System must be able to issue bills to consumers”, “Consumer must be able to pay bills online”, “System must be able to make payments to provider’s bank account (minus 15% service fee)”, “System must be able to authenticate admin, provider, and consumer before updating any sensitive information.”

2.10. Profile Page

On the profile page, users can manage their profiles efficiently through a set of functionalities. Firstly, they can upload a new profile image by clicking on the "Change Image" button. Once an image is selected, users can save the changes by clicking the "Save Changes" button, which initiates the update process through the backend route specified below. Additionally, users have the option to delete their profiles entirely, ensuring a seamless experience. A critical aspect of this process was ensuring thorough deletion of user details to prevent any potential conflicts, such as booking a deleted listing by another user. This is a novel functionality

2.11. Notifications

When a customer books a spot, notifications are promptly dispatched to the provider. These notifications are conveniently displayed within a modal dropdown accessed from the icon located in the top right corner of the interface. To ensure smooth notification management, relevant data is stored in the user database and subsequently retrieved when needed. Upon retrieval, we employ the split method to extract pertinent information, such as the address, facilitating effective communication and action by the provider. This is a novel functionality,

3. Third Party Functionalities

3.1. Frontend

3.1.1. React

React was chosen as the framework for its robust and flexible architecture, which allows for the creation of dynamic and interactive user interfaces. Its component-based structure promotes modularity and reusability, making it easier to maintain complex applications. Additionally, React's virtual DOM (Document Object Model) efficiently updates and renders UI components, resulting in better performance and a smoother user experience. Its strong community support, extensive documentation, and ecosystem of third-party libraries and tools also played a significant role in its selection, providing developers with the resources they need to tackle various challenges and accelerate development. Overall, React's combination of powerful features, performance optimizations, and thriving community make it an ideal choice for building modern web applications.

React hold the MIT license that they are permissible for modification, commercial usage, distribution (Clark, 2022). But they are not responsible for the stableness of the library. In terms of license, React library will not limit our ability to develop our applications or develop our business. However, there is potential risk that we have to bear the financial lost due to the vulnerability of the library.

3.1.2. Material UI

Material UI was selected primarily for its seamless integration with React. Material UI offers a visually appealing and intuitive user interface out of the box. Its extensive library of components and styling options enables developers to quickly build consistent and polished UIs without the need for extensive customization. The tight integration with React ensures that Material UI components work smoothly within React applications, providing a cohesive development experience. Overall, Material UI's combination of design principles, ease of integration, and comprehensive component library makes it a popular choice for building modern web applications with React.

Material UI also holds a MIT license (Material-Ui/LICENSE at next · Mui/Material-Ui, n.d.). The limitations of it are similar to React library since they are using the same license.

3.1.3. Emotion/React

This is one of the dependences of MUI so we use this with the same reason of MUI.

Again, it holds a MIT license (Nguyen, 2022)

3.1.4. Leaflet

Leaflet is an open source javaSript based map construction library. It supports most of the applications including mobile phones or desktops. We used it to present the location visible on our application and enhance the navigation experience of our user. It is vital in our application, since it is not feasible to construct our own map component by the given development time constraint.

Leaflet hold a BSD 2-Clause "Simplified" License (Sen, 2024).It is nearly the same as a MIT license in terms of limitations and usage described above.

And the difference of them is BSD require the license to be included in both source form and binary form, but MIT license didn't specify the inclusion format. It is noticeable that the license inclusion format and this difference is relatively irrelevant to our project. Therefore, no further clarification will be given on it.

3.1.5. react-countdown-circle-timer

react-countdown-circle-timer is an open-source JavaScript based timer component. It provides a high quality and customizable react component (Dimitrov, 2021). By using this library, we can shorten the development time and focus on other functionality of the application.

Again, it holds a MIT license.

3.1.6. react-splide

react-splide is an open-source JavaScript based library that provide a react image carousel component. It provides a stable, lightweight and visually pleasing react component. We used it to present multiple images on our application frontend. And by using it we can shorten the development time and provide high quality UI/UX to user.

Again, it holds a MIT license (Fujita, 2020)

3.1.7. react-stripe-js

react-stripe-js is an open-source JavaScript based library that provide the component to interact with the stripe payment gateway backend. It is necessary in our application, as we can't perform any real-world bank transfer or bill issuing without using it.

Again, it holds a MIT license (Stripe, 2019)

3.1.8. react-router-dom

React-router-dom is a routering library for react web application. This is necessary for a react 23b project otherwise we can't provide or get use of any web router functionality.

Again, it holds a MIT license (Jackson, 2023)

3.1.9. leaflet-gesture-handling

Leaflet-gesture-handling is a mapping library that is used to prevent map focus/zoom on scroll. This is important for our project as on the main page the map takes up a very large area of the view port and during usage it can be very annoying to properly scroll down the page without focusing/zooming on the map. This library makes it so that you must ctrl+scroll in order to zoom, allowing the user to have a much better experience when scrolling on the site.

Again, it holds a MIT license (Marquis, 2021).

3.2. Backend

3.2.1. Python

Python is an open-source and has a vast ecosystem of libraries and frameworks for various tasks, including web development, data analysis, machine learning, etc. This allows us to use existing libraries and modules to build robust application involving the use of machine learning. Python's syntax is simple and easy to understand and allows us to focus on solving problems rather than dealing with language complexities.

The license is governed by the Python Software Foundation License, which is a permissive license like the MIT License. This license allows us to use and service python for our project.

3.2.2. SSL

SSL provides secure sockets layer support for network connection. In our codebase, we used SSL to create sockets for SMTP implementation.

It is distributed under the OpenSSL license which is a permissive license.

3.2.3. Smtplib

The smtplib module in Python is part of the standard library. It provides an SMTP (Simple Mail Transfer Protocol) client session object. It is used to establish a secure connection over SSL/TLS when sending emails.

It is distributed under the same licensing terms as Python itself.

3.2.4. Scikit-learn

Scikit-learn is a machine learning library for Python. It provides simple and efficient tools for data mining and data analysis. In the code, scikit-learn is used specifically for implementing a recommendation system. The 'NearestNeighbors' algorithm from scikit-learn is used to find nearest neighbors based on user interactions.

It is an open-source machine learning library for Python. It is a 3-Clause BSD License, which is a permissive license that allows for free usage, modification, and distribution of the library (see appendix [6]).

3.2.5. Flask and Flask – CORS

Flask is a micro web framework for Python. It provides tools and libraries for web development and defining routes.
It is a permissive license allowing us to use the framework for both open source and commercial projects.

This extension handles Cross-Origin Resource Sharing (CORS) and allows you to make requests to the application from a different domain. It is licensed under the BSD license which allows us to use it for open-source projects.

3.2.6. MongoDB with PyMongo

MongoDB is a NoSQL database and PyMongo is a Python driver for working with MongoDB databases. This flexibility allows us to store data in a format that can evolve over time. It provides the ability to build robust applications for a larger database.

PyMongo is licensed under the Apache License. We can use the MongoDB services to showcase the project, but we are not allowed to distribute MongoDB as a service.

3.2.7. Certifi

This is a Python package that ensures secure communication with MongoDB by providing the necessary SSL certificates. Using Certifi for our project, ensures that our Python application has access to the updated bundle of CA certificates, which is crucial for establishing secure connections over HTTPS or other SSL/TLS protocols.

It is an open-source licensed package.

3.2.8. Geopy

Used for calculating distances between geographical coordinates given the longitudinal and latitudinal values. The ‘geopy.distance.geodesic’ function is part of the ‘geopy’ library, which is a Python client for geocoding web services.
The project requires calculating distances between geographical coordinates, such as finding the distance between a user's location and available parking listings. See appendix [7]

Geopy is distributed under the open-source license. Therefore, the licensing terms of 'geopy' do not possess any significant impact on the project's objective.

3.2.9. Stripe-python

Stripe-python is a python-based library used for accessing the API service of payment gateway Stripe. It is also vital in our application. It enables our application to interact with the server of the payment gateway and access their service so that our platform can issue bills to service consumer and pay to the provider.

Again, it holds a MIT license (Bellone, 2018)

3.2.10. Passlib

Passlib is a password hashing library for Python, providing secure password hashing. It is used to secure user passwords. In the source code, password is encrypted before storing it in the user database. This enhances security by protecting user passwords in case of a data breach, as the hashed passwords are difficult to reverse engineer into the original form (See appendix [8]).

It is an open-source library and can be freely used in both commercial and non-commercial projects.

3.2.11. Json

The json module provides functions for encoding and decoding JSON data. It's commonly used for serializing and deserializing data when working with web APIs or exchanging data between applications.

Json is a module in Python's standard library, so it follows the licensing rules discussed with python.

3.2.12. Math

Math is a module in Python's standard library that provides mathematical functions and constants. It's used for mathematical operations in the codebase.

It follows the licensing rules discussed with python.

3.2.13. Copyreg

Copyreg is an important module in Python's standard library that provides functions for registering pickle support functions.

Copyreg inherits the same licensing terms as Python itself. As part of the Python's standard library, it can be freely used and distributed for both open-source and commercial projects.

3.2.14. Uuid

The uuid is a Python module used for generating universally unique identifiers (UUIDs). This is very essential to create and manage unique ids in the codebase.

The uuid module is distributed under the same licensing terms as Python itself.

3.2.15. Threading

Threading is a module in Python's standard library. Threading is employed to execute long-running persistence operations, such as database queries or updates, in separate threads at the same time.

This license is a permissive open-source license that allows for free usage, modification, and distribution of Python and its standard library modules, including threading.

3.2.16. Requests

'requests' is a popular HTTP library for Python. It is used for making HTTP requests to web servers such as GET, POST, PUT, DELETE, etc., to interact with web API.

The 'requests' module is distributed under the same licensing terms as Python itself.

3.2.17. Datetime

The 'datetime' is a module in Python's standard library for working with dates and times. It is used for handling date and time-related operations in the code base. The date is a class in the datetime module that represents a date (year, month, day).

The Datetime module is distributed under the same licensing terms as Python itself.

3.2.18. Bson.json_util

The 'bson.json_util' is a module provided by the bson library, which is used for working with BSON (Binary JSON) data in MongoDB. json_util provides utilities for encoding and decoding BSON data to and from JSON format.

The BSON library, including the bson.json_util module, is distributed under the terms of the Apache License 2.0. We can distribute the BSON library, including bson.json_util, in both its original form and modified form. We must include a copy of the Apache License 2.0 along with the modified source code.

3.2.19. HTTP exceptions

The werkzeug.exceptions module is part of the Werkzeug library, which is a comprehensive WSGI (Web Server Gateway Interface) utility library for Python. It provides utilities for building web applications such as request handling and exception handling.

Werkzeug is distributed under the BSD 3-Clause License, which is a permissive open-source license. Therefore, there are no significant licensing restrictions that would impact the results of the project.

4. Implementation Challenges

4.1. Listing Implementation

While implementing the functionality for Listings, it was a bit of a challenge to determine if a listing created by a provider would be active instantly or not due to the possibility of providers not always wanting to make their listing active. We did some research into what current systems do and saw that most of them just publish that listing. In order to cater to the user's convenience, we decided on implementing this functionality so that providers could make a listing active or unactive; but we were unsure about how it would be designed. In the end, we decided that once a listing is created, the provider can set a window between which they want to keep it active. Doing so, the listing can automatically deactivate once the specified end date has passed. This is helpful as the provider will not have to worry about remembering the end date in order to make the listing unactive. Once the end date has passed, if the provider wishes to make the listing active again, they can set a new window in which they want to set it active. This design choice makes sure that convenience for the provider is given very high priority. It is designed and thought out so carefully that it considers almost every real-world scenario that one may encounter while making a listing.

While implementing the create listing, the first technical difficulty we faced is to handle the user input address and try to get back and provide real world address option to user. We had initially planned to use google map api but we don't have the budget to pay for their services. So we used a free map api 'open street map' in substitute.
Once we get a user input address, we perform a delay search the open street map API https://nominatim.openstreetmap.org/search?, to search a range of possible address option for user to choose, And the search results are limited to addresses in Australia as we provided the country code as "AU" in the query parameter.

However, if the user types too fast or type in an abnormal frequency. The map API may limit our access to ensure the fair use of their service.
To address this, we used a set time out function inside the useEffect hook to delay the address search by 380ms once the user makes a character change. And use the clean-up function to stop the previous unexecuted search being executed (eg. If user makes 10 characters changes in 380 ms, only the last character change will trigger a search request to the map API after 380ms and others outdated request is cleaned up by the clean-up function in the useEffect hook). We feed those addresses as the option of the autocomplete of MUI and let user selects among them. A real time search result is presented to users in a dynamically visible way.
And under the address search bar, we also get a map to show the selected result more clearly to provide a better user experience.
In order to achieve this, we used the react-leaflet library to render a real world map in the frontend. The difficulty of this is to present the user address input clearly on the map. To address this, we first done some research on the react-leaflet library to understand how to center the map to a certain location and make a customized marker on the map.
we referenced a few online resource (eg. Youtube tutorials) (Coding, 2022)and leaflet API documents to make our own customized use of the map. After we get the lat and lon geo information from the open street map api, we pass the location to the map container as the center of the map container. And ensure that we also render a Marker on that location once we get the geo information.

4.2. Machine Learning Based Recommendation Implementation

While implementing the recommendations part, we were contemplating whether or not we would use machine learning but one team member insisted on using machine learning based recommendations so we went ahead with using machine learning to recommend car spaces. None of the people in the team has ever made ML based recommendations so extensive research was required before we even implemented it. Research showed that there are mainly two popular methods of recommending items using AI: Contend-based recommendation and Collaborative-based recommendation. Content-based recommendation basically classifies users and items by their known metadata and recommendations are made by learning the user's past activity. Collaborative-based recommendation basically learns the interaction of every single user with every single item in the web-app and leverages feedback or activity history of all users in order to give recommendations. We decided to use a hybrid approach in which the algorithm uses collaborative based recommendations to make a similarity matrix and then gives a score for each

user-listing cell based on the number of likes and the frequency of the booking that listing has been in. This design choice was made so that the algorithm recommends listings that are tried-and-tested to be good and not give recommendations for bookings that are not great. The reason for using K-nearest neighbouring algorithm to train the dataset because we wanted to not only give the highest recommended listing by others but also other listings similar to this listing.

4.3. Payment system

While implementing the payment system, the challenges we faced is that we need to integrate our application with a third-party payment gateway. And we did more study on Stripe payment gateway document, and we followed the guideline and instructions of Stripe to build and test our website, and solved half of our problems.

However, since the expected payment logic of stripe is customer pay for a product through a checkout page which is different from our website. So, we must track and store the user default payment method id by ourselves in our own backend. And built the way user can view and manage their entered payment method entirely on both frontend and backend.

4.4. Showing searched listings

The way the home page shows listings is to use a standard useeffect() to fetch all the listings and then display them. The component of the search functionality is located on a separate file than the home page file (it is located on the nav bar file as we believe that this looks better on the UI). So, when the user searches for something, the backend returns the matching listings in that nav bar file. Obviously we need it in the home page file, so we leverage a react state variable “listings” in a higher tier file that includes both the nav bar and the home page file in order to share the new listings to the home page file. There, we have another useeffect() which has that “props.listings” state variable in its dependency array. Once triggered, the useeffect() runs and the shown listings are set to that search’s returned “props.listings”.

However, there is an issue here. The way react works is to run all the useeffects() first and then render the page. This is an issue here as we don’t want to run the second useeffect (as it is only meant to run when the search functionality is used).

To combat this, we create a state variable in the home page file called “initialListingsLoaded” and initialise it as false. In the first useeffect (the main initial fetch of all listings), we set it to be equal “true”. Then in the second

useeffect (the one that only is meant to activate when a search occurs), we have an if statement check before it runs (if (initialListingsLoaded === true)).

You may spot an issue here. Won't initialListingsLoaded be set to true always in the first useeffect, and thus therefore the second useeffect will always run no matter what? (when instead we only want it to run if it's dependency array (props.listings) changes and not on initial page load). The way react works for the initial page load cycle is that first, the useeffects are ran and then the component is rendered. However, any state variable changes (such as seInitialListingsLoaded(true)) are not reflected until after the useeffects and render are fully completed. This means that on initial render, the second useeffect will never run as react only updates the initialListingsLoaded state variable to true after everything is loaded. So therefore only after the initial page load can this useeffect be run (i.e. when the dependency array: props.listings changes).

Thus, we leverage this knowledge about the nuances of React in order to achieve this design.

5. Installation/User document/manual

5.1. Setup/Installation

We recommend installing and testing our application on wsl or linux.

1. Unzip the downloaded repo.
2. Open the file and create two terminals. One will be used to run the frontend server; one will be used to run the backend server.
3. For the frontend, in one terminal, cd to the frontend directory with the command "cd frontend/"
    a. Node is required. If you don't have node installed on your system, check this website and follow the instructions for your OS: https://nodejs.org/en/download
    b. Use the command "npm install" to install required dependencies after installing node.
4. For the backend, in the other terminal, cd to the backend directory with the command "cd server/"

    a. Use the command “pip3 install -r requirements.txt” to install required libraries.
5. If you are on a linux os (or wsl), on the frontend terminal to start the frontend server use either the command “npm run start-linux” or “export HTTPS="true" && npm start”. If you are directly on windows (no wsl), use the command “($env:HTTPS = "true") -and (npm start)” instead
6. Now on the backend terminal to start the backend server use the command “python3 app.py”.
    a. If you are having certificate issues here when you are trying to test the system and you are on a mac OS, then use finder to find the “Install Certificates.command”. Right click “Get Info”, then click open with “iTerm.app”. After this, double click ‘Install Certificates.command”.
7. The site is located at https://localhost:3000/

We recommend you use chrome for running the site. When you open the web for the first time, it will give you a warning since it is running on local host. You are getting a warning since the certificate does not match the ‘localhost’ domain and ‘localhost’ domain can’t be verified.

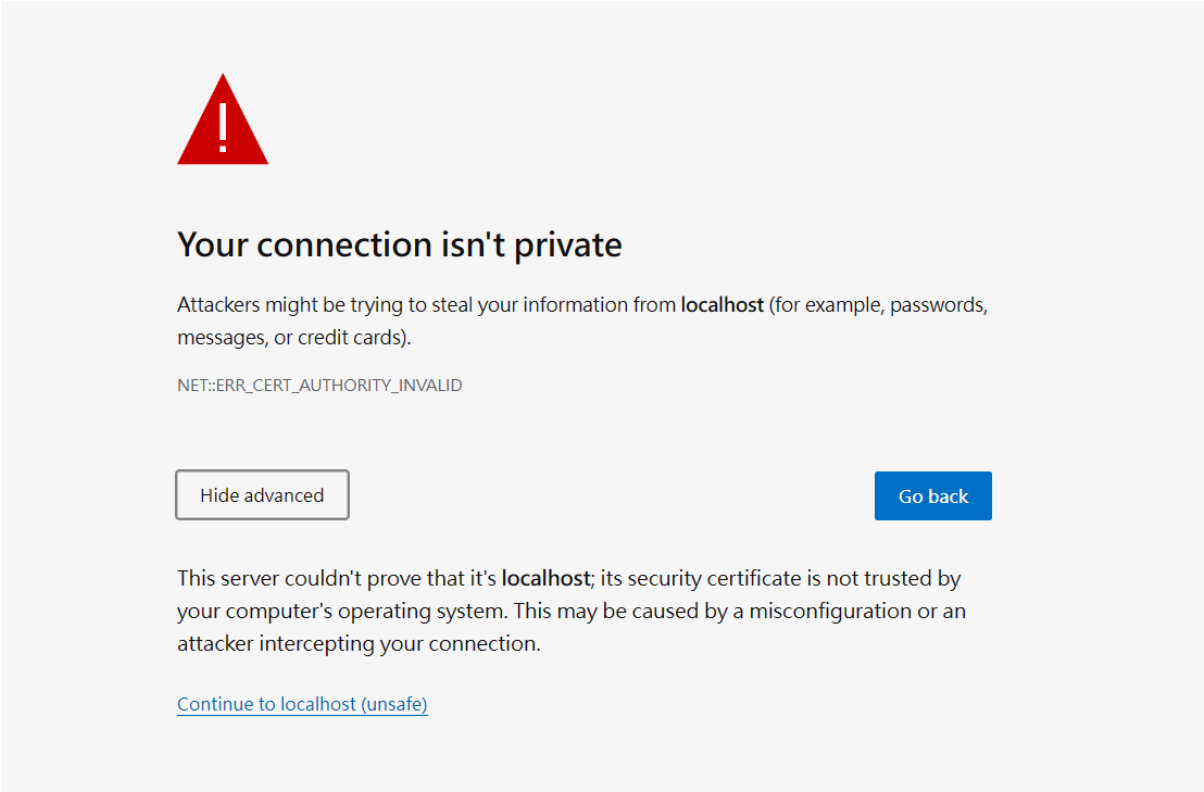


Figure 2: Initial Loading Page When Loading For the First Time

Click “advance” and click “continue to localhost(unsafe)”.

After the above installation step, you can navigate our website now.

5.2. Backend and API service account

You can navigate to our API services and MongoDB database with the following websites:

- *https://dashboard.stripe.com/dashboard*
- *https://www.mongodb.com/*

5.2.2. Stripe

Account name: parkify3900@outlook.com

Password: COMP3900stripe

You need to use the Google MFA to login with these details. Please make sure the dashboard is on test mode when you are navigating the dashboard.

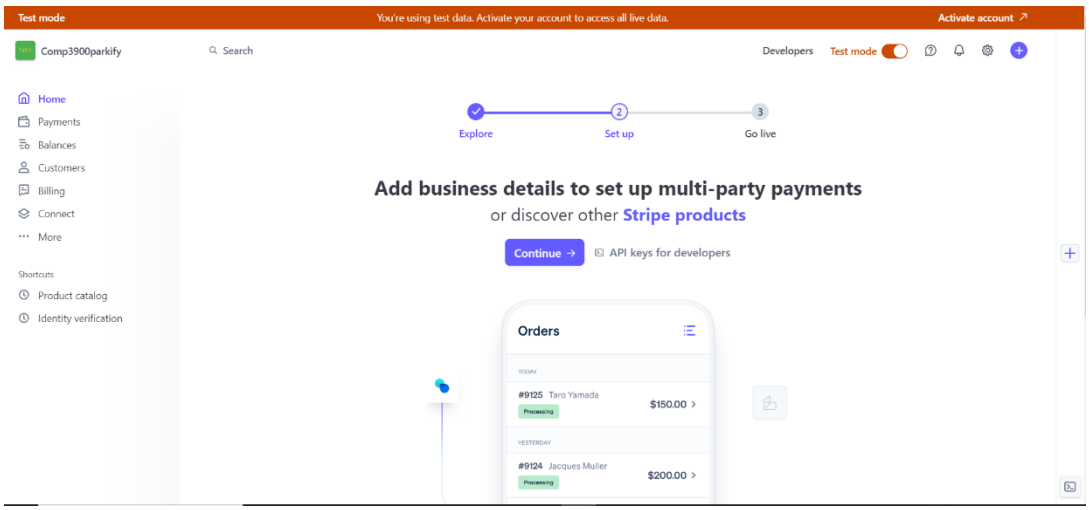


Figure 3: Home Page for an Account On Stripe

The account is not fully activated so you can only use the testing card and testing data provided by stripe to test our website. And we can't fully register our website since it requires us to register our business and provide real business details.

The payment system is divided into the provider side and customer side.

The customer and their payment related items (eg. Credit card added) will be stored in the customer account on the stripe. You can navigate to the customer related details on the sidebar “customer” field.

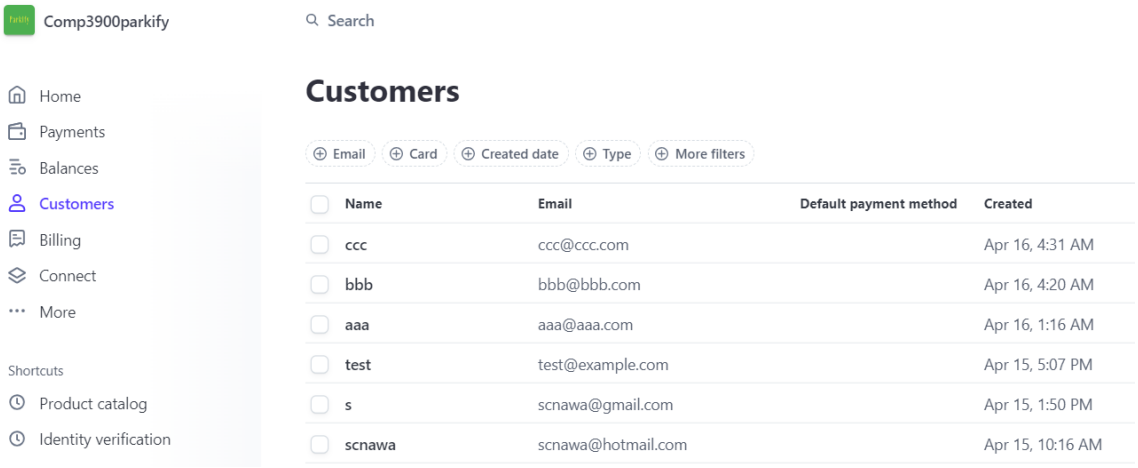


Figure 4: Customer Details Page on Stripe

And there are connected accounts for provider. Only a connected account has the ability to receive money from our platform through stripe.

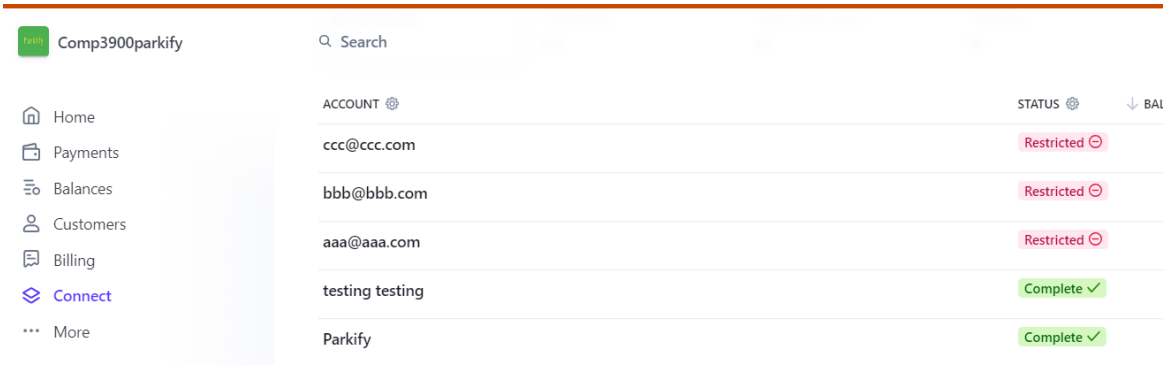


Figure 5: Connected Accounts Page on Stripe

The status ”Restricted” means that users haven’t provided the details to become a provider so they can’t create listing or receive payment from stripe. More details of this will be given in the sections below (how to use the system).

5.2.2. MongoDB

Account name: parkify.auth@gmail.com

Password: COMP3900Project

You can login in and observe our backend data on it by going on the collection titled “Parkify”. Click on database, collections, then click on one of the dbs. In order to log into the MongoDB account, you will need to sign into Gmail with the

above credentials and on the login page, you will need to click on the “Gmail” button to sign in.

5.3. How to use the system

5.3.1. User/Admin Authentication

First of all, we provide you with a user/provider testing account:

Email: tutor@example.com

Password: tutor

You can use it to login in and skip the signup section if you don't want to use your own email to register or don't have the time to go through the whole registration process on Stripe in our payment section. But if you have time and are interested in that, we strongly encourage you to start from scratch.

Initially, when we load the website (https://localhost:3000/), we are greeted with the following home page:

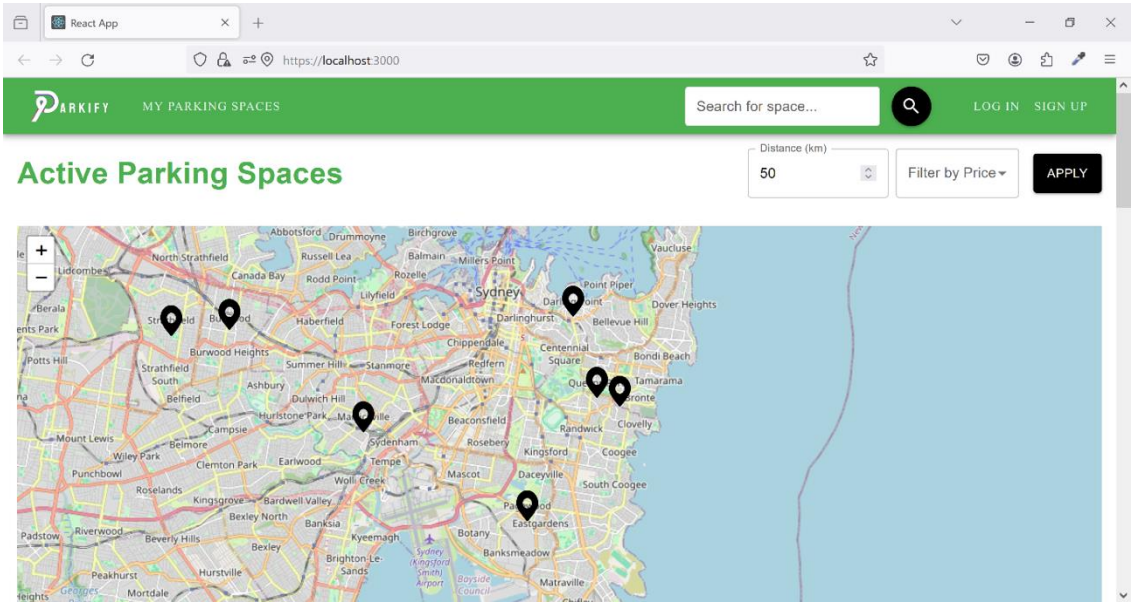


Figure 6: Homepage of Parkify

First, we have to sign up and make an account. Click "Sign up" and you will be redirected to the sign-up page where you will fill out your details as below:

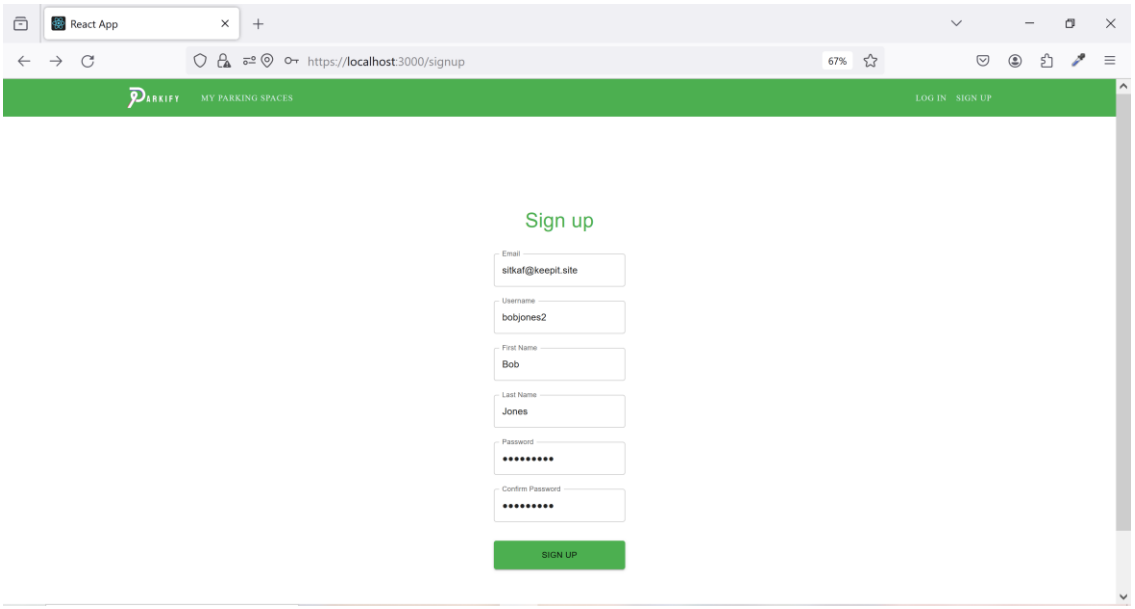


Figure 7: Signup page of Parkify

Now click “sign-up” and you will be redirected to the verification page, where you will then fill out the verification code that is sent to your email. The email will look like this:

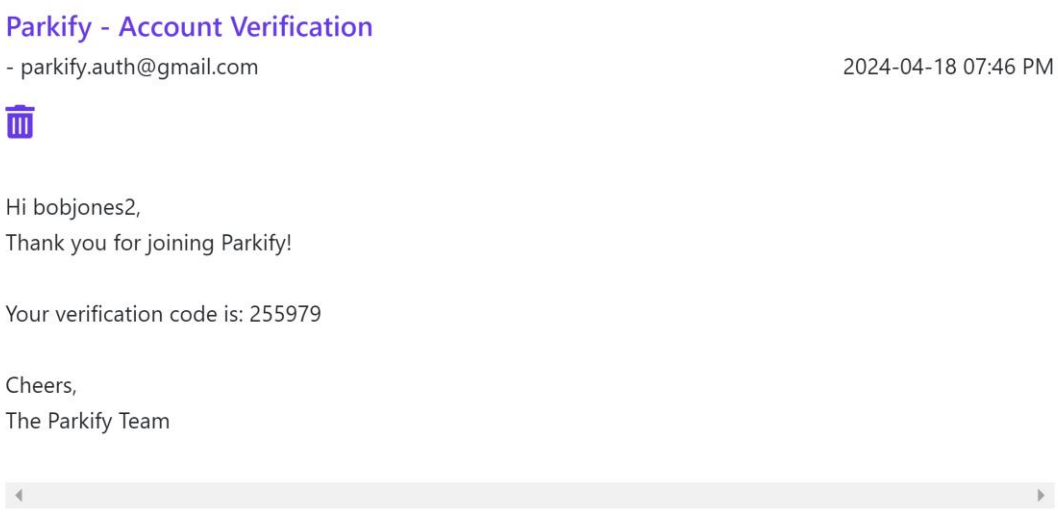


Figure 8: Automated Verification Email Sent by Parkify Account

Fill out the code and click verify:

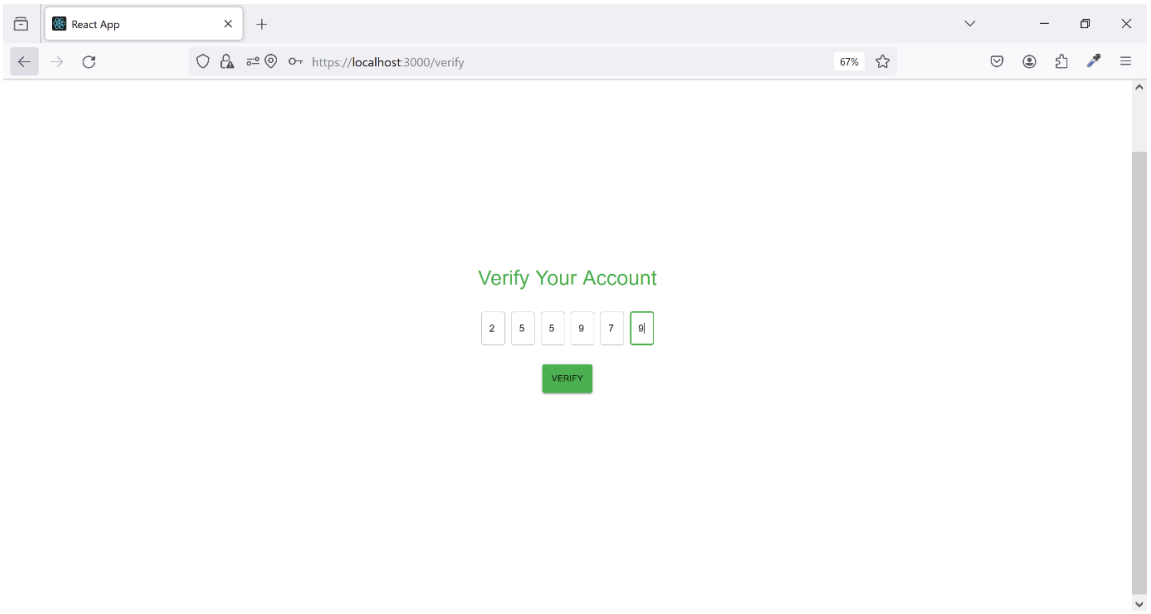


Figure 9: Verification Page

You will then be redirected back to the home page. Now you have a user/provider account. To demonstrate the logout functionality, click the account button on the top right and then click “logout”:

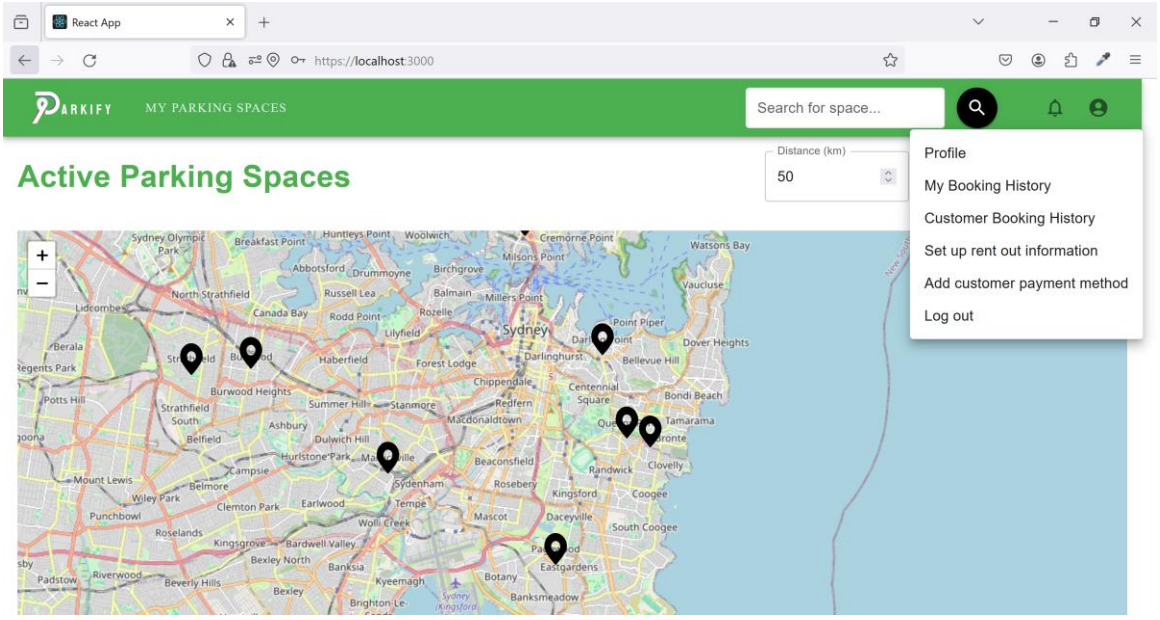


Figure 10: Navbar with Profile Icon is Clicked

The navbar (the green top bar) will then turn into the following with the main homepage being displayed.

One thing to note for mobile responsiveness is that as the viewport decreases, the items on the navbar (such as “MY PARKING SPACES”) will disappear from the nav bar and instead be moved onto the account menu bar that you can see in figure 10.

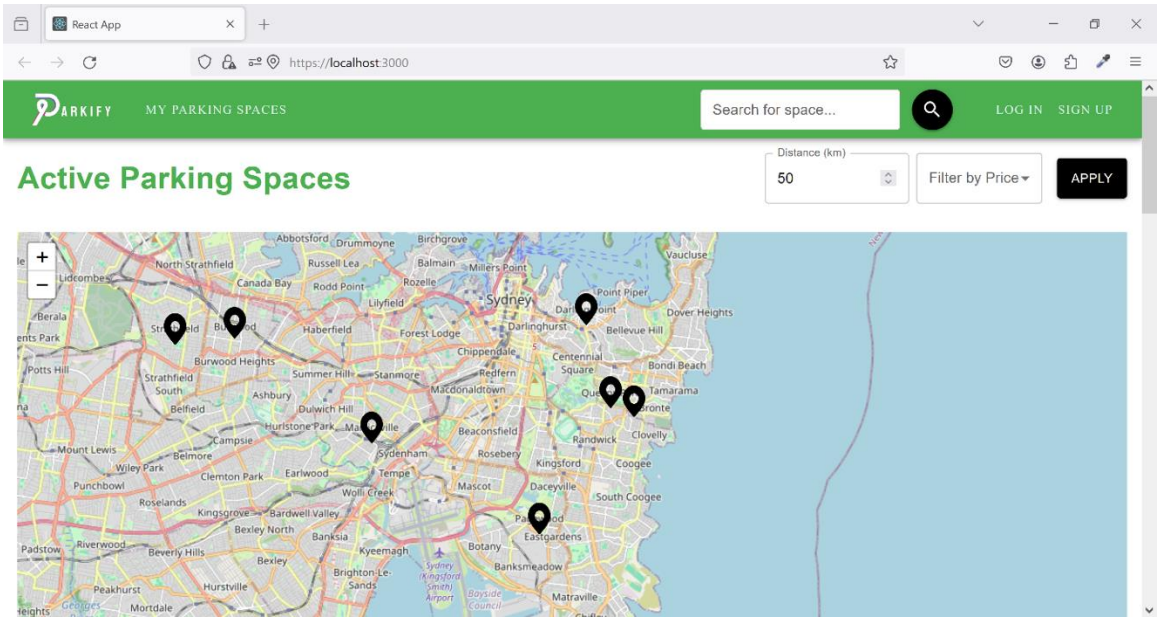


Figure 11: Navbar when Dropdown Menu is not clicked

Now before we log back in, the functionality of making an admin account will be shown.

For time purposes, There is already an Admin account setup:
email: admin@admin.com
password: admin

Admin accounts cannot be setup in the front end and must be entered through a service such as postman. Enter the link and the request as shown, with your

admin details:

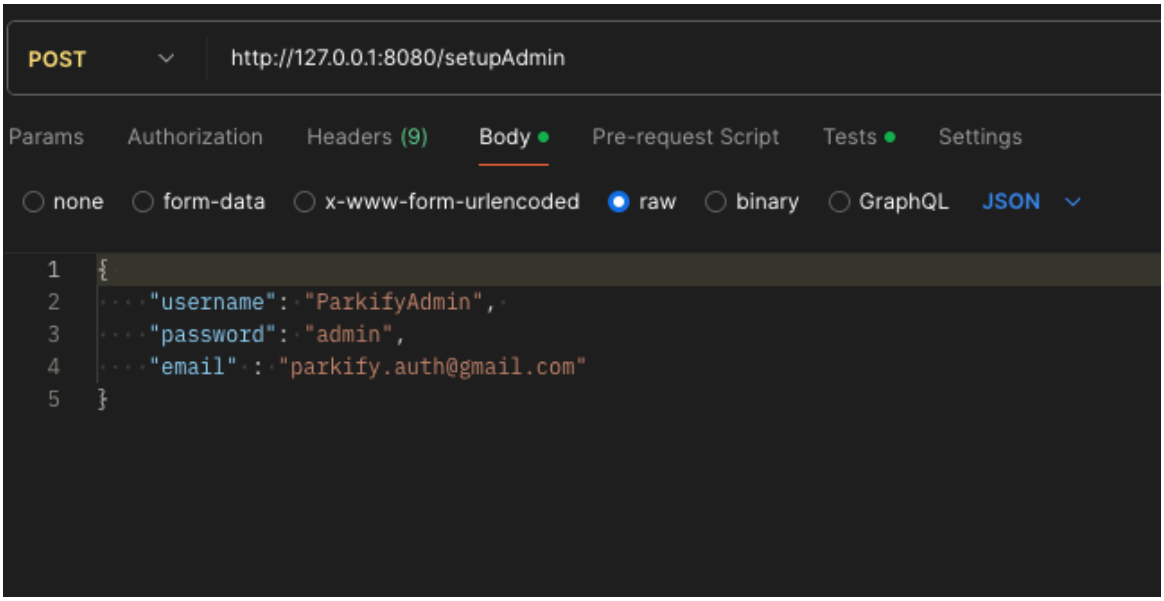


Figure 12: Postman Method to Add Admin

Then, click the “login” button in the nav bar and you will be redirected to the login page. Here you can login as the admin or as the user/provider account. We will login as the user/provider account we made, as for now we will demonstrate the user/provider functionalities. Click the “login button” as below and you will be sent to the home page.

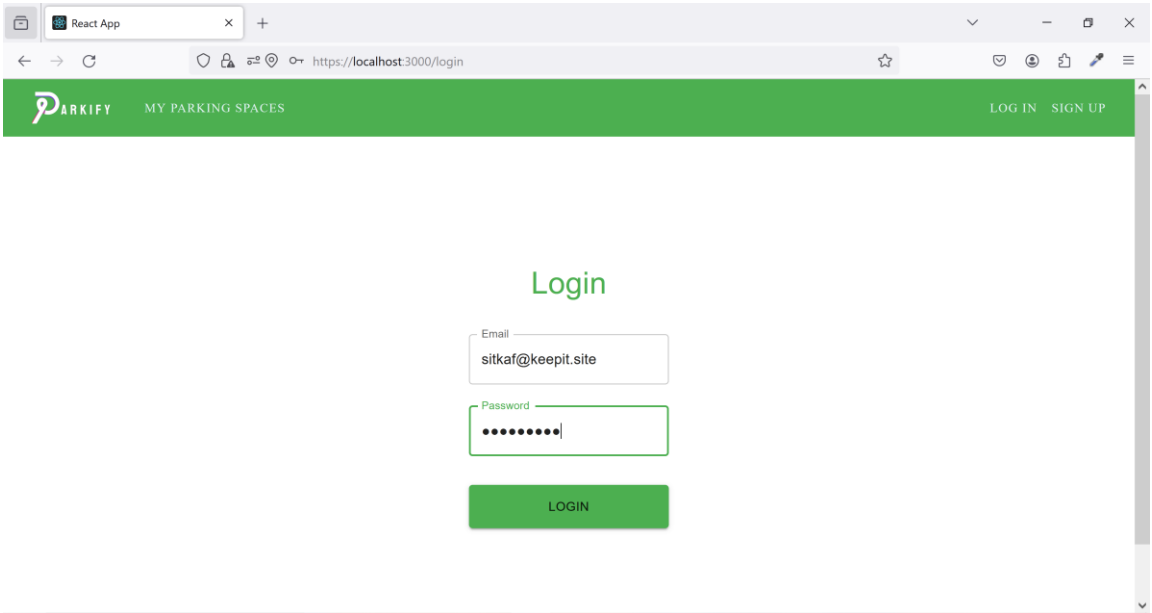


Figure 13: Login Page of Parkify

Then:

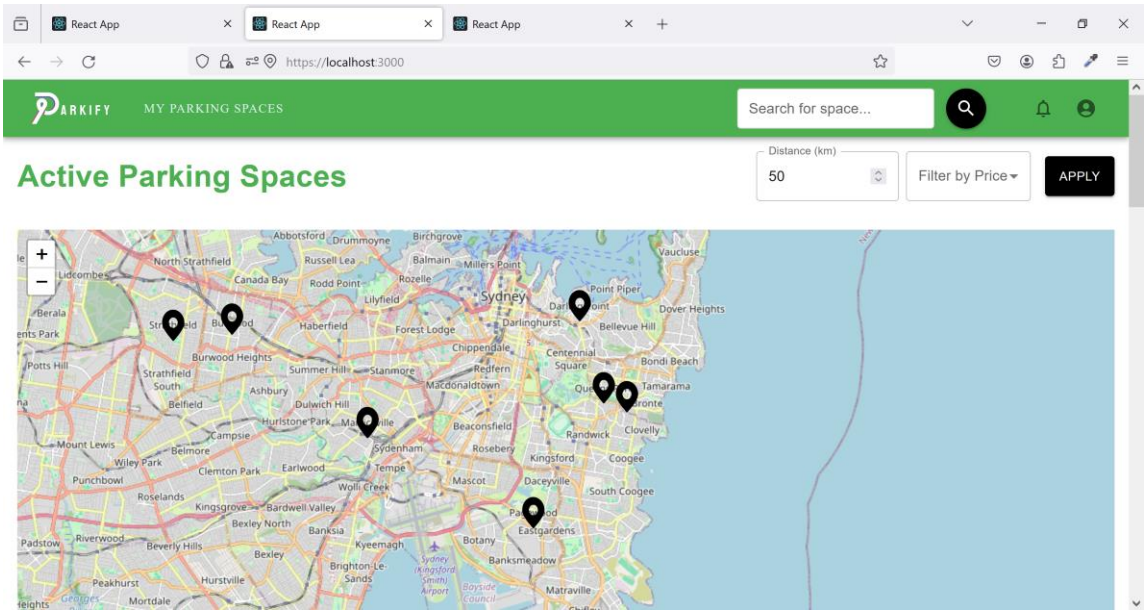


Figure 14: HomePage of Parkify

5.3.2. Payment system

Users have to provide necessary details to stripe before they can rent out their space. If users click on “MY PARKING SPACE” on navbar before providing those details, an alert will show up and prevent them visiting the space management page.

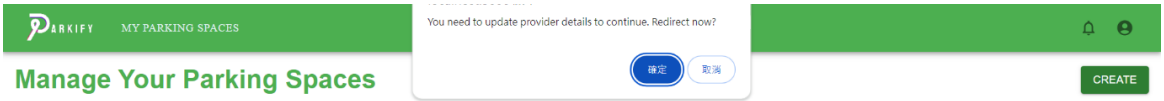


Figure 15: Prompt to Add Card

The user can click yes and be redirected to the stripe hosted page to fill in their provider details.

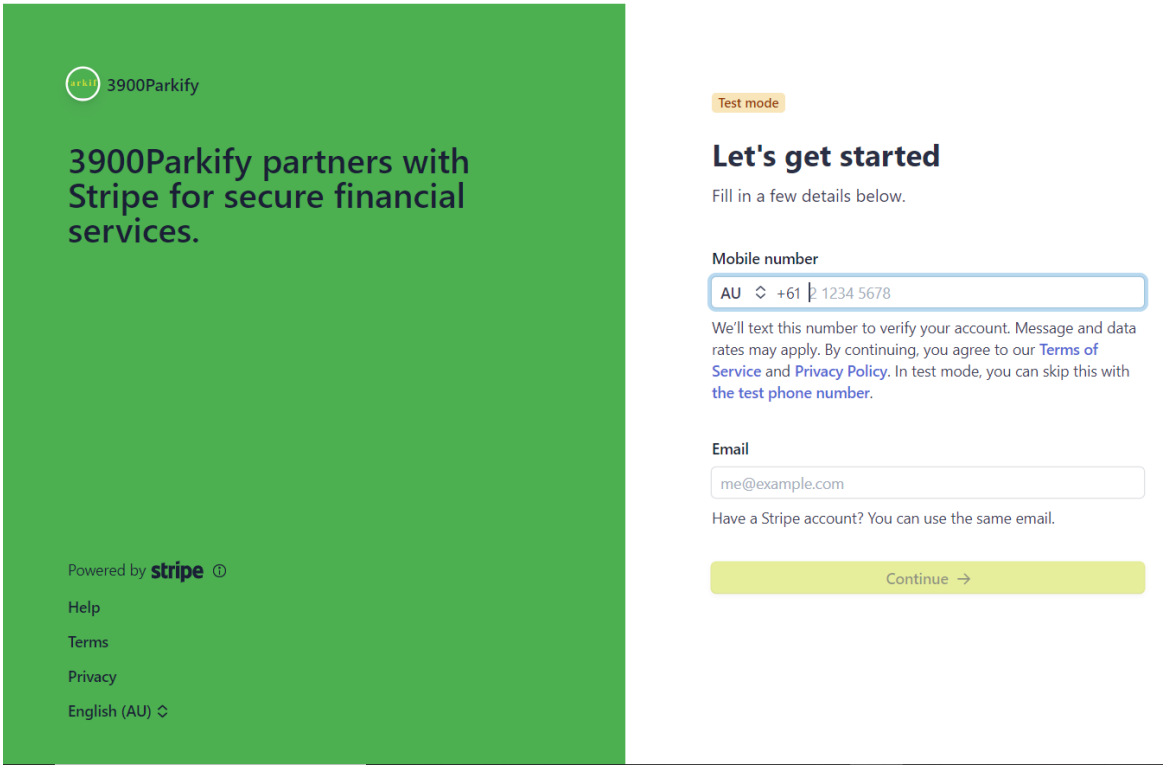


Figure 16: Stripe Onboarding

You can click the blue text "the test phone number" to fill in the test phone number and fill in "me@example.com" for the email. If you are really interested in the verification process, you can fill in your real phone number and stripe should send you a message with a verification code for you to fill in.

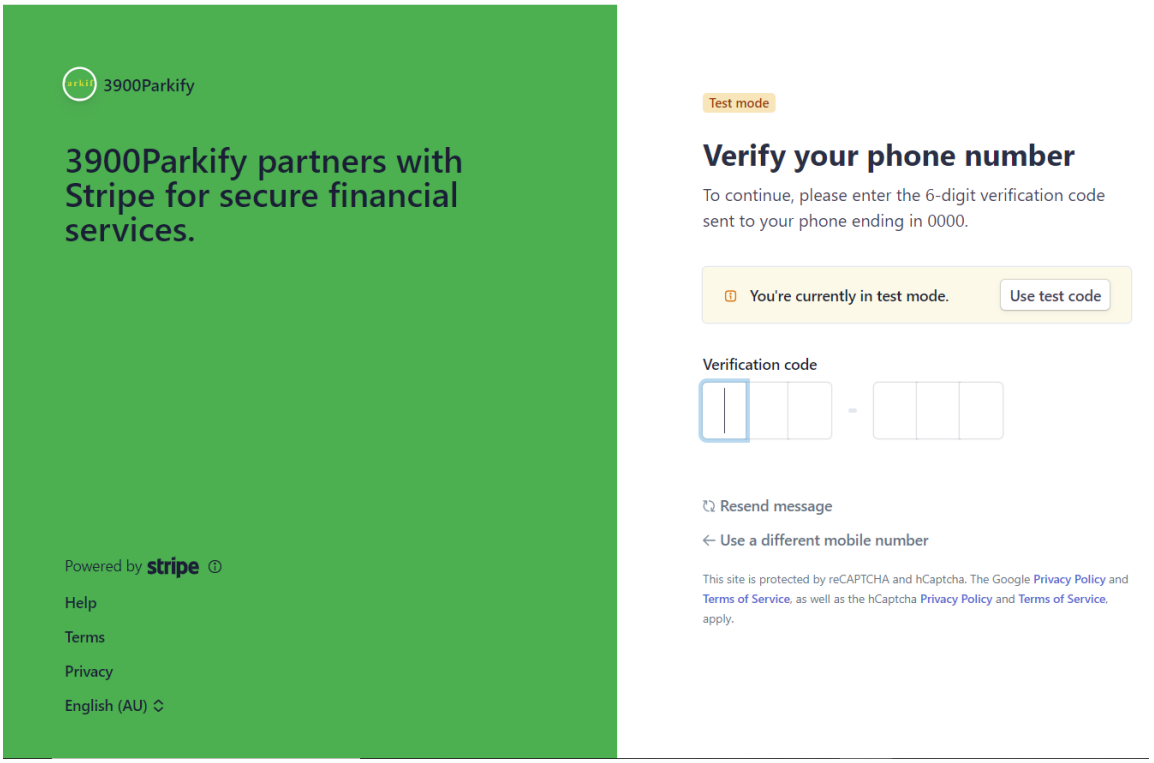


Figure 17: Verification Prompt For Stripe

Then you can click “Use test code”.

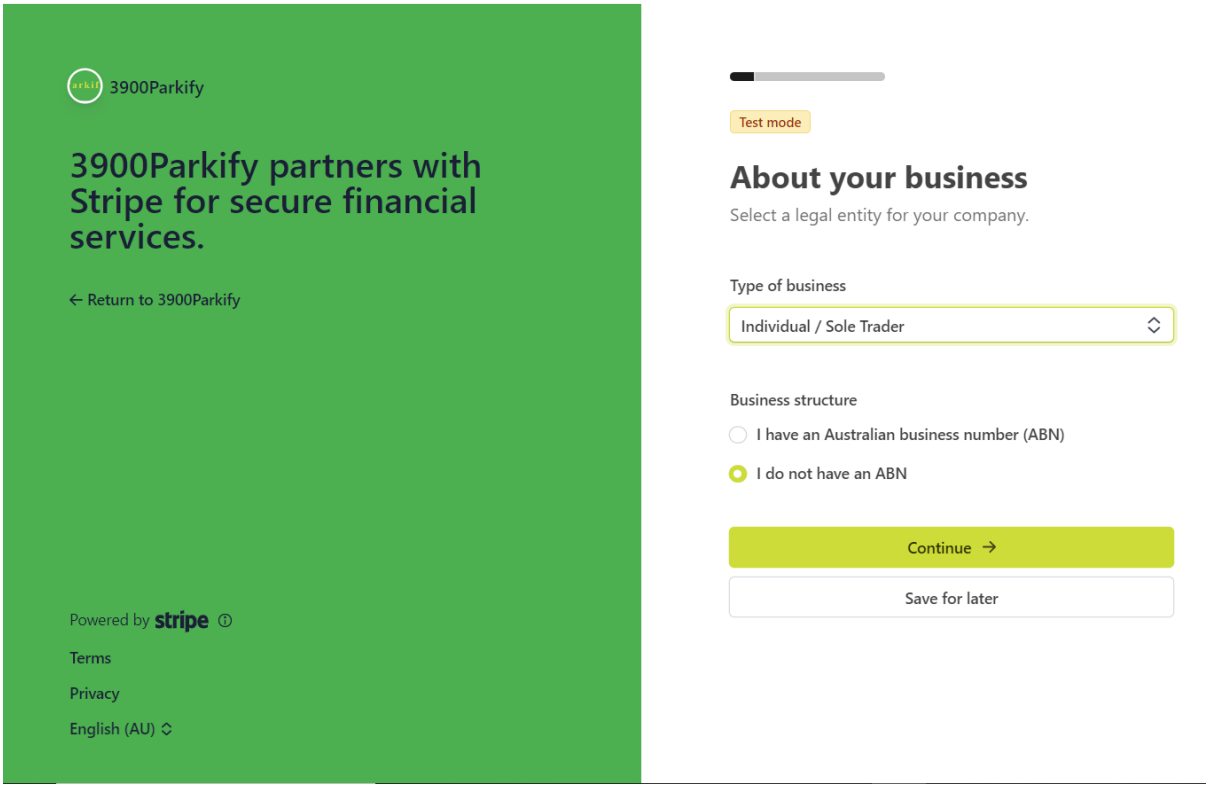


Figure 18: Business Information Stripe

Then select “individual/Sole Trader” for the type of business. Select “I do not have an ABN” for Business structure. Then click Continue.

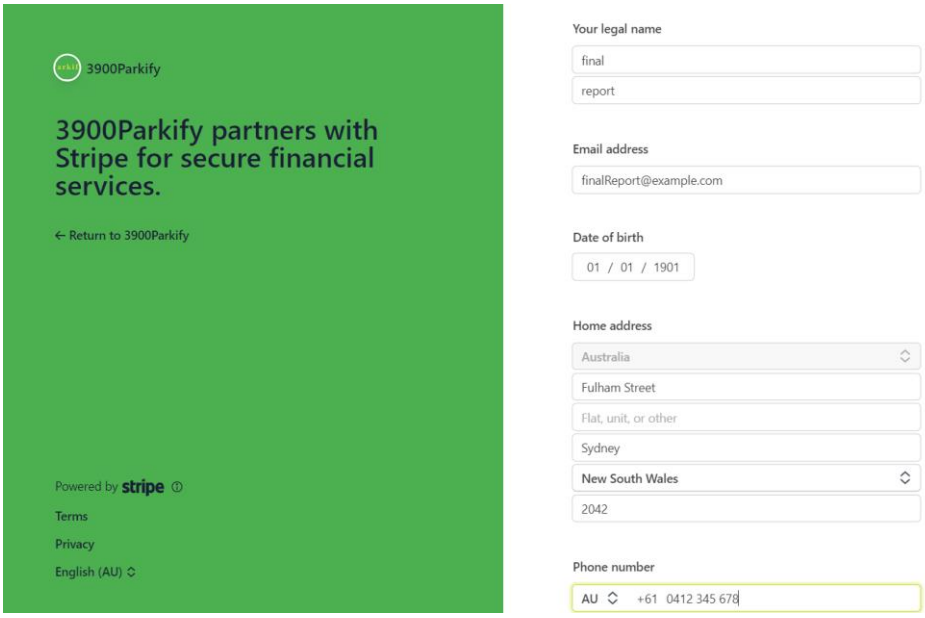


Figure 19: Personal Details Prompt on Stripe

- For “Legal name”, fill in your name or a random text (eg. Final report).
- For “Email address”, fill in a random email address in a normal email format (eg.finalReport@example.com)
- For “Date of birth”, fill in 01/01/1901.
- For “Home address”, type/select a random address
- For “Phone number”, type in 0412345678.

Then click continue.

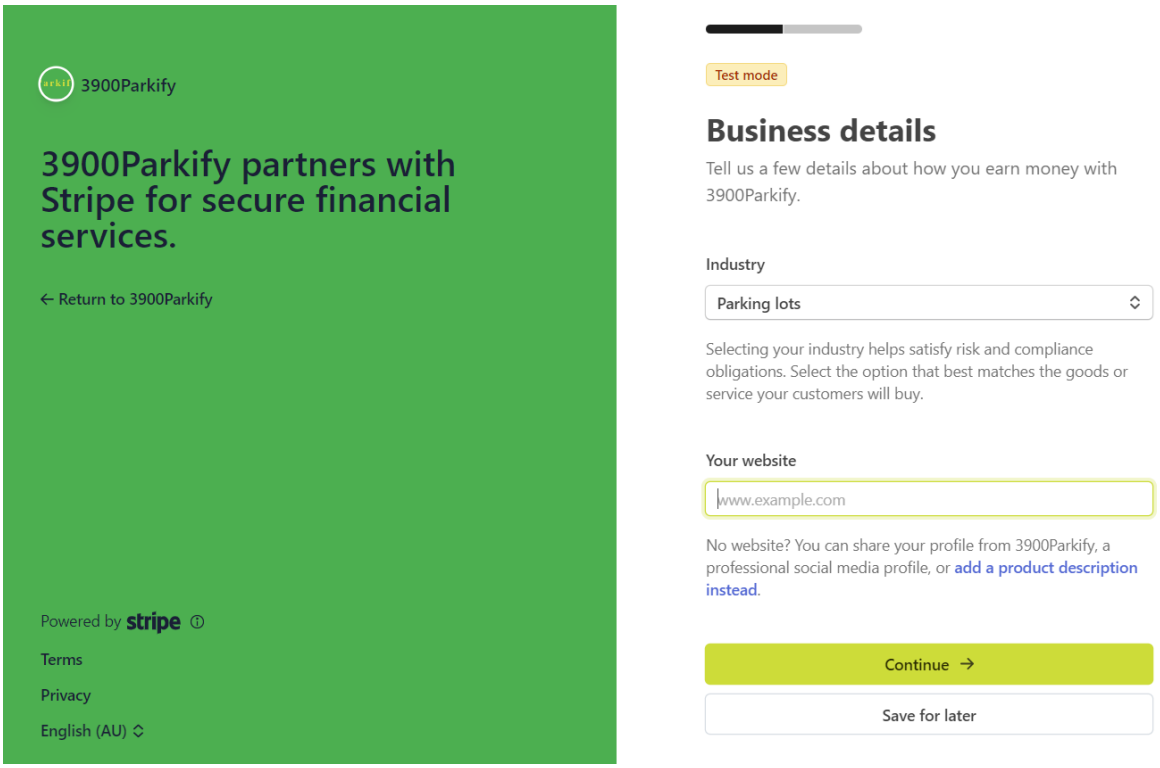


Figure 20: Stripe Enter Business Details

- For “industry”, select transportation and select Parking lots.
- For “website”, select add a product description and fill in parking lots.

Then click continue.

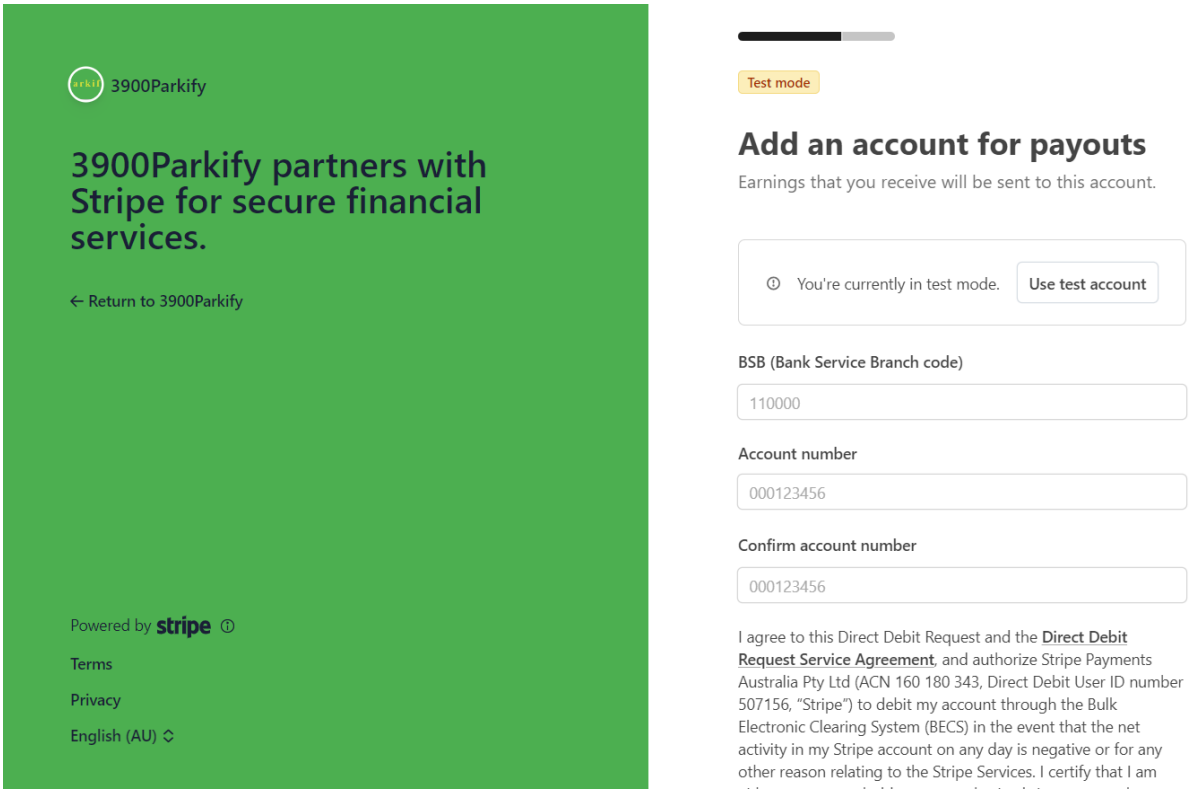


Figure 21: Payout Account Onboarding

Then click ”Use test account” to fill in test account details.

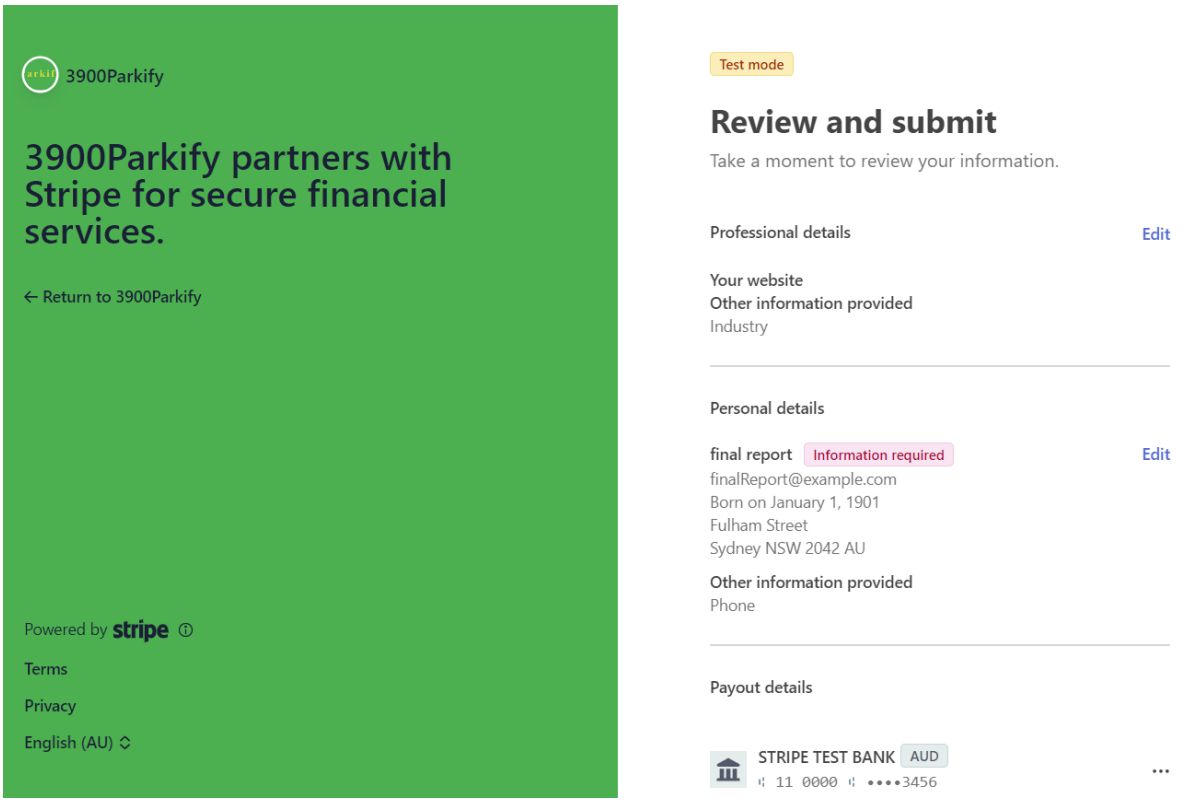


Figure 22: Review Page

Then click Edit button on the Personal details

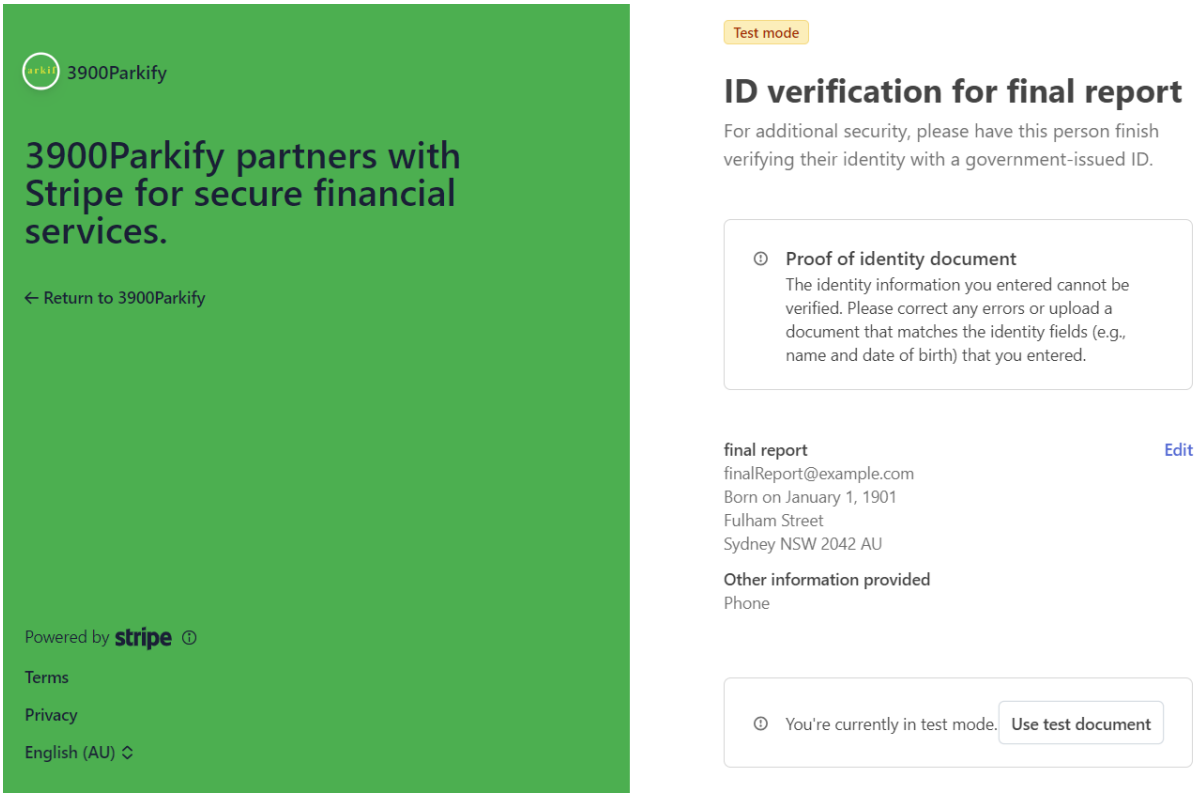


Figure 23: ID Verification Page

Then click ”Use test document”

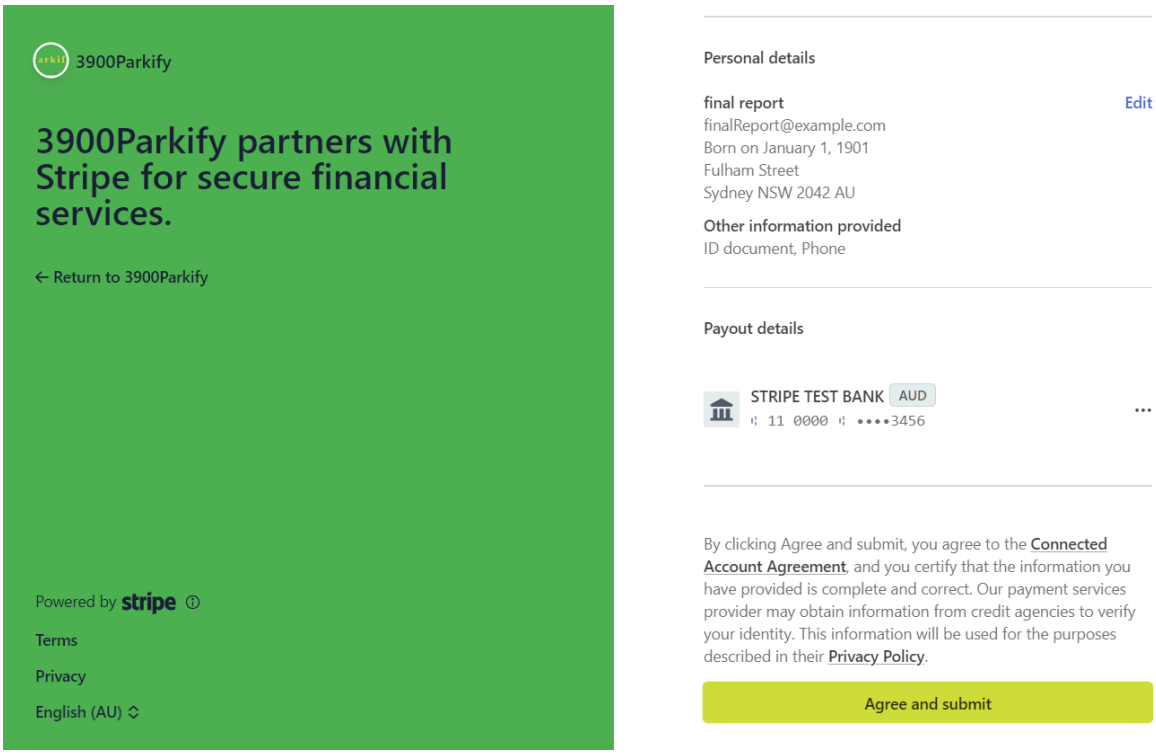


Figure 24: Review Page

Then click “Agree and submit” to submit the form and you will be redirected back to our website.

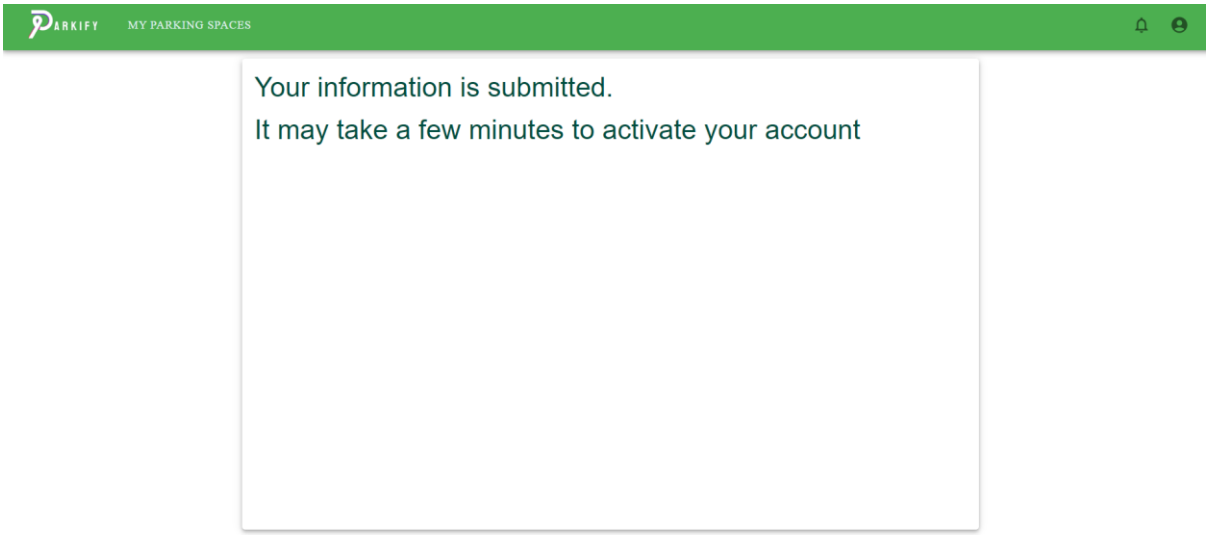


Figure 25: Page after submitted provide details

Then you can click “MY PARKING SPACES” to create a listing without getting a redirect. Sometimes it takes a few minutes for Stripe to update. However, if this situation persists then you may have to revisit the form again to check if there is any missing out field that you forgot to fill in. Then you also should be able to see the stripe connected account status turn to “connected” on stripe website.

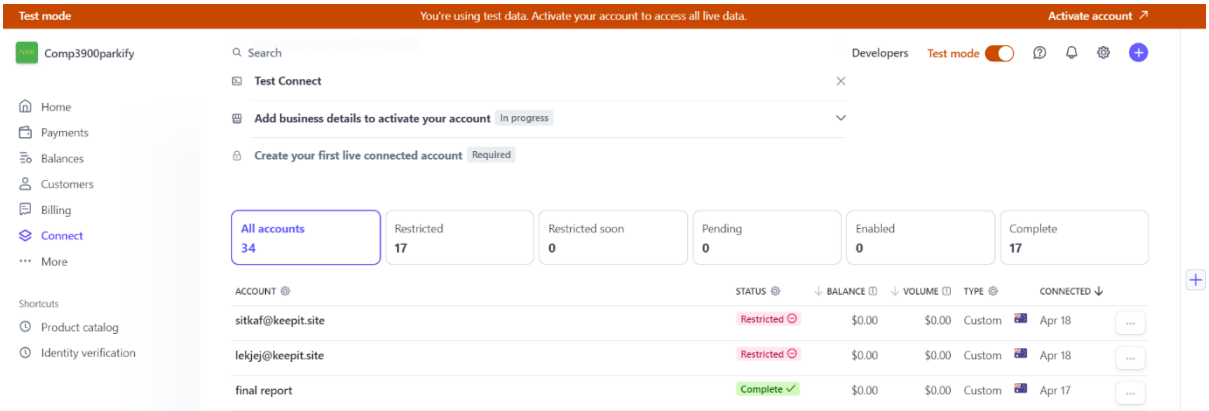


Figure 26: Stripe Accounts Page

You can view the details you just filled in by clicking the account. The user on our Parkify website can also click “Set up rent out information” on the nav bar menu to update the provider details (or as well to set the stripe up for the first time as well).

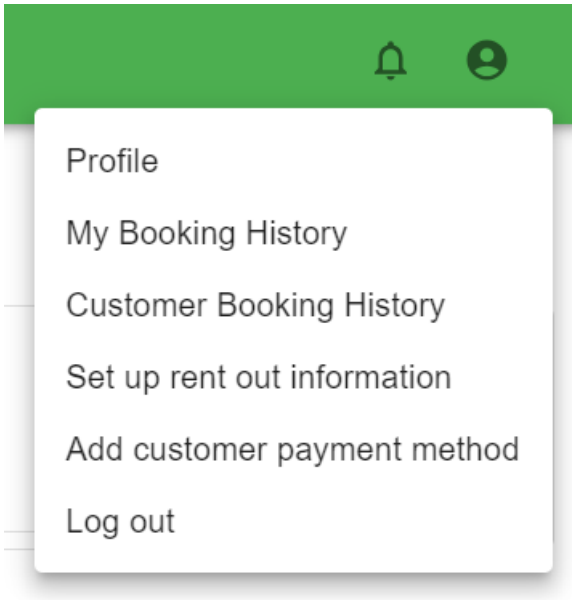


Figure 27: Dropdown Menu

Before the Customer able book a booking, they have to provide their payment method first. They can manage their payment method by clicking “Add customer payment method” on the nav bar menu.

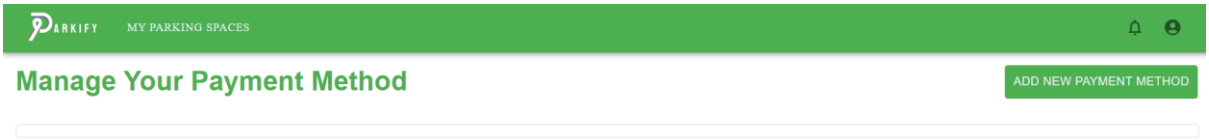


Figure 28: Manage Payment Page

Then customer can click “ADD NEW PAYMENT METHOD“ to add their payment method.

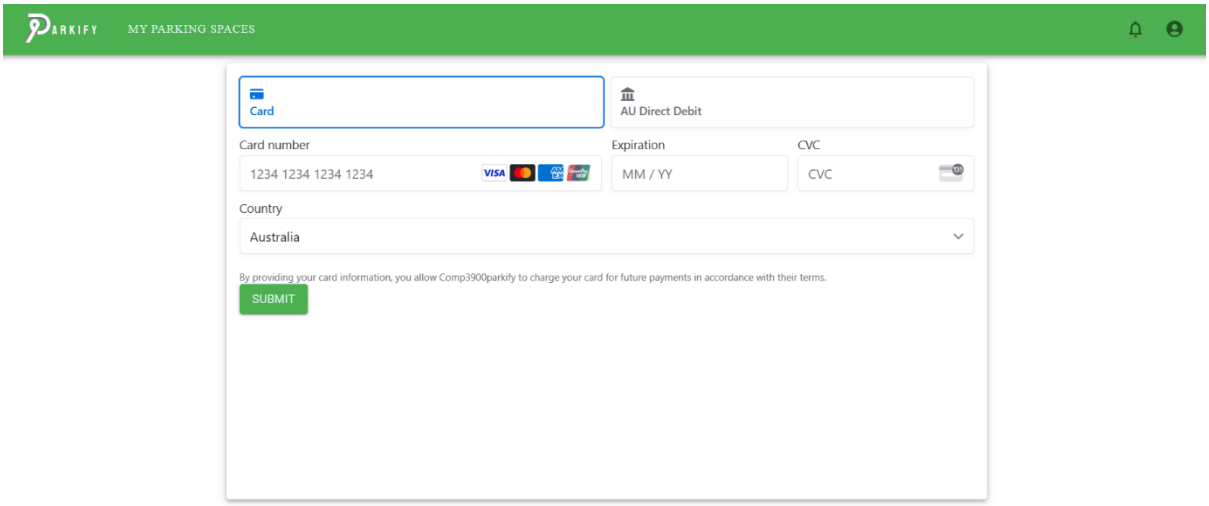


Figure 29: Page to Enter Card Details

Then user can fill in their payment methods.
For testing purposes, stripe provides some testing card for us to test this functionality.

- For Visa credit card number: 4242424242424242
- For Master debit card number: 5200828282828210
- Expiration: any future dates (eg. 12/34)
- CVC: any number (eg.333)

Then click submit and it will redirect to a page with a success message.

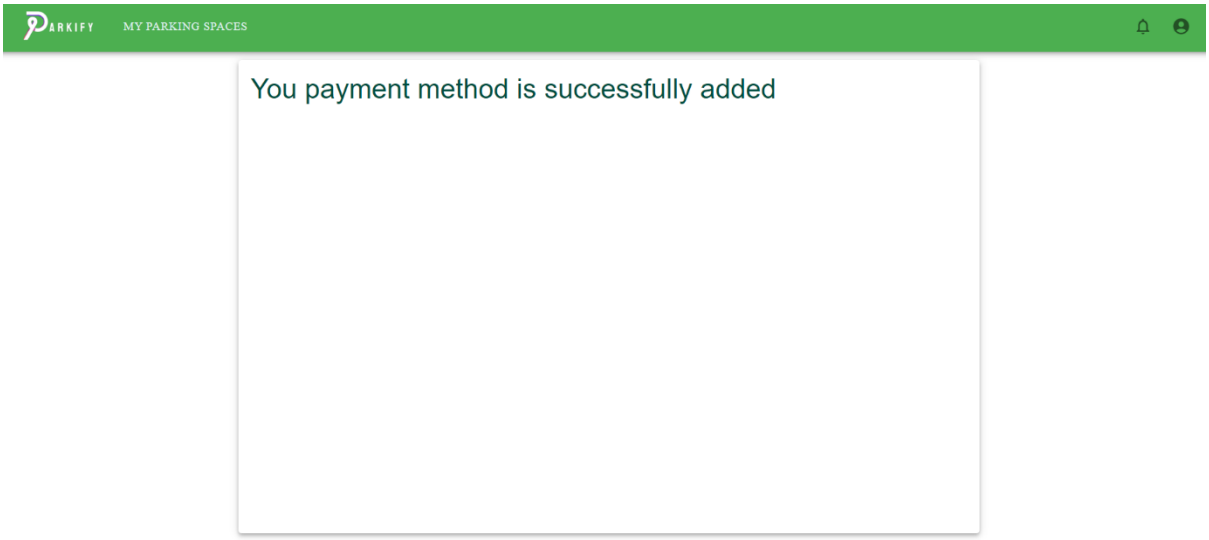


Figure 30: Success Page

Then if you revisit the payment method page again, you should be able to visit and manage the payment method have entered.

Figure 31: Manage Payment Page

And you can click ”set default” or ”remove” button to set a default payment method and remove a payment method. And you can also view the added payment on stripe customers tap.

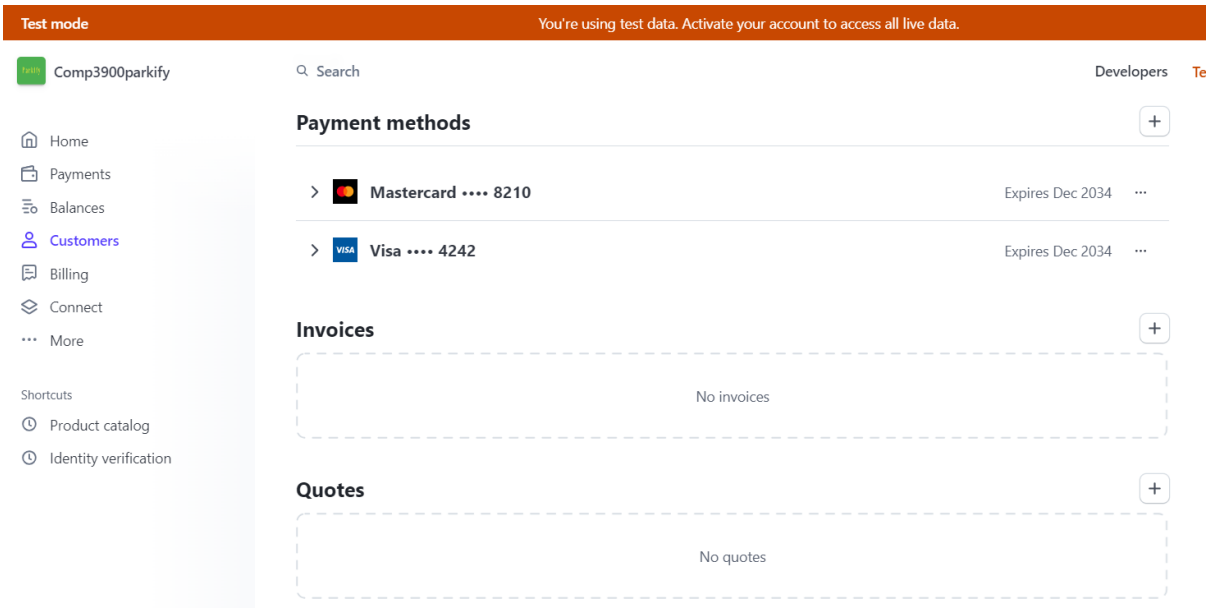


Figure 32: Stripe Payment Methods Page

5.3.3. Listing functionalities (activate/deactivate/create/delete/update)

After the provider sets up their details, they can start creating and managing their listings by clicking “MY PARKING SPACES”.

Figure 33: Manage Parking Spaces Page when Access for The First Time

To create a listing, click the “Create” button.

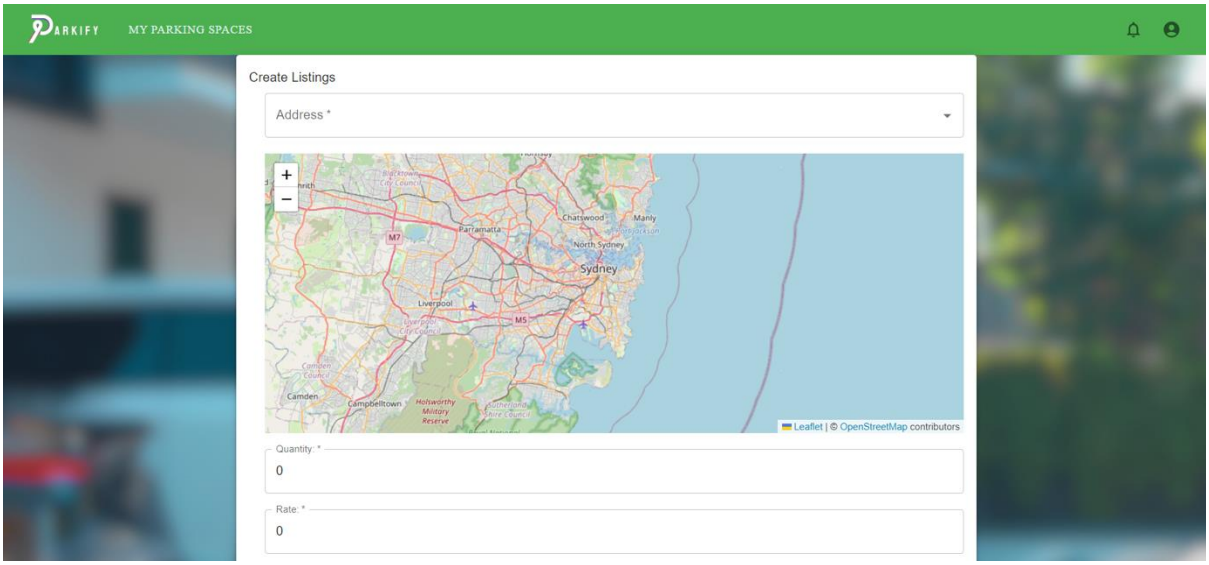


Figure 34: Create Listing Page

You can fill in information of the listing here. Select an address and fill out the quantity and rate.

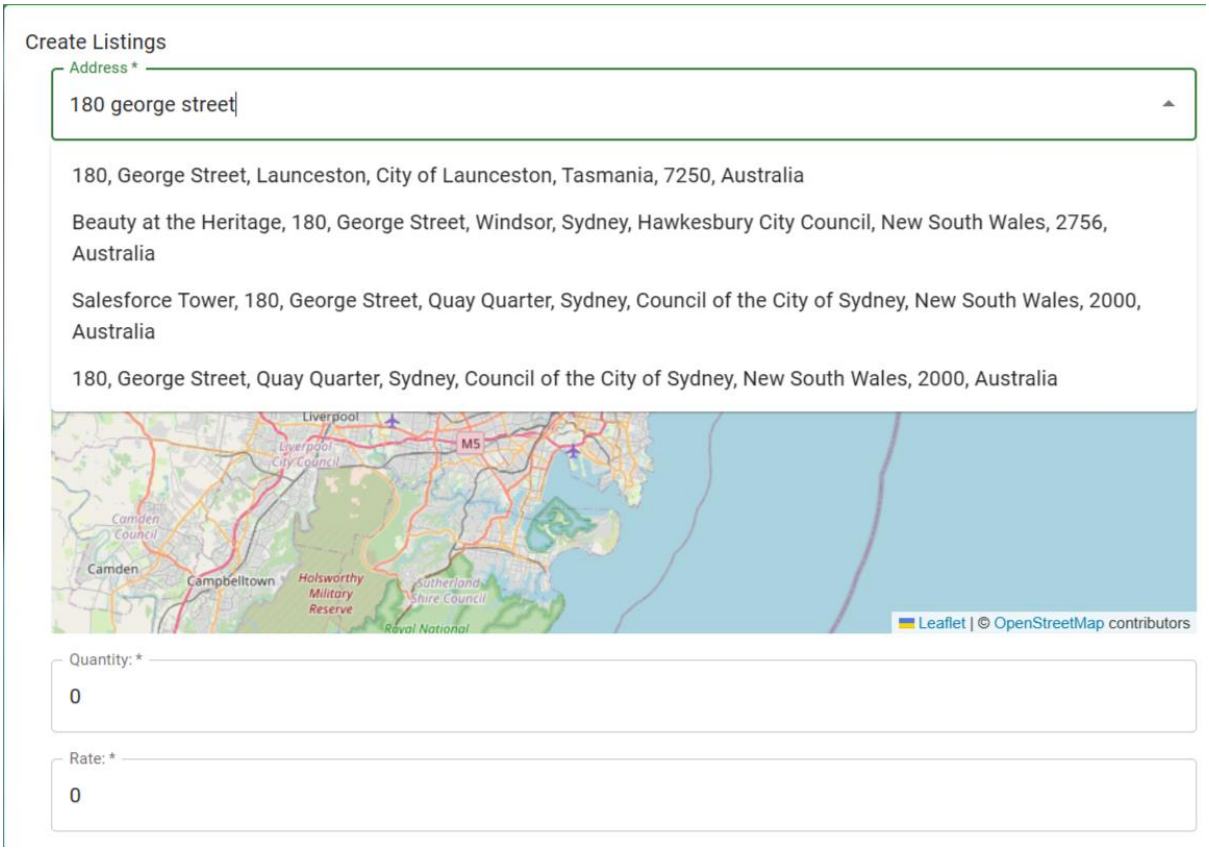


Figure 35: Address Input Results

And users can upload a single image for thumbnail and multiple additional images. Both uploads are optional, and a default image is given if no thumbnail is provided by the user.

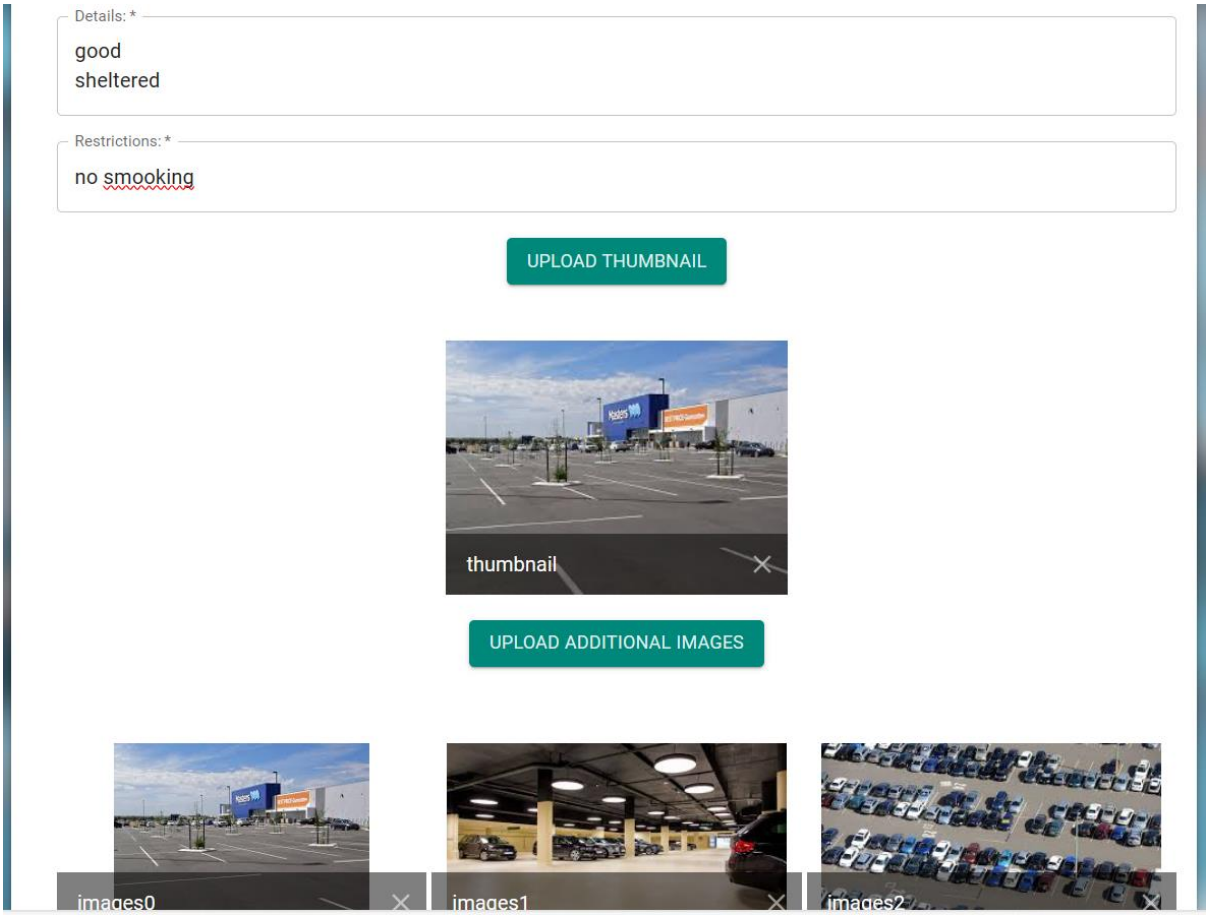


Figure 36: Restrictions and Upload Thumbnails Page

And user then can click ”create” at the button of the form to create a listing.

Then now user can view the listing they just created

Figure 37: Manage Your Parking Spaces After Listing Is Added

Click the “live status” button on the listing to be able to activate it.

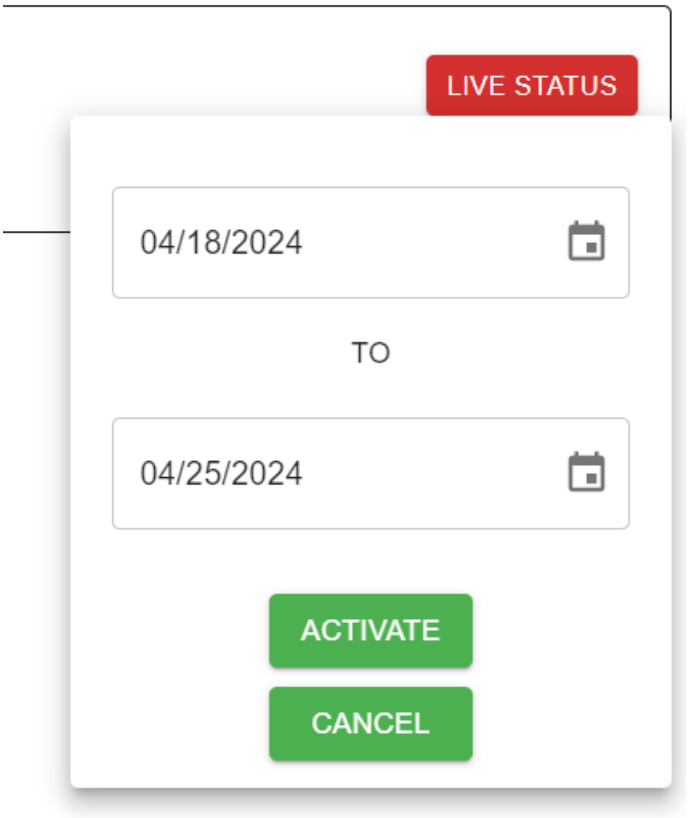


Figure 38: Making the Listing Live

The user can click the calendar to change the default start date and end date. Click ‘ACTIVATE’ to activate the listing. And the ‘LIVE STATUS’ button will turn green after the listing is activated. Once activated, the listing will appear in the main home page. It can be reached by clicking the “Parkify” logo on the nav (green) bar on the top left of the site. On the home page, the user can now see their live listing both on the map, and on the bottom of the page itself as a card:

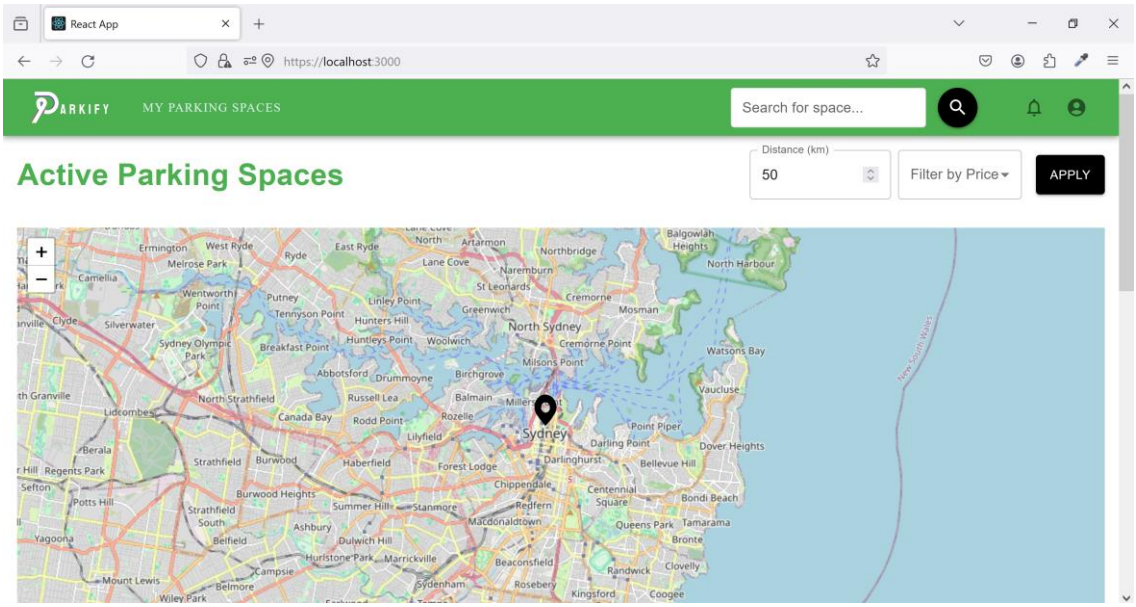


Figure 39: Listing that was Just Added Showing Up on Map

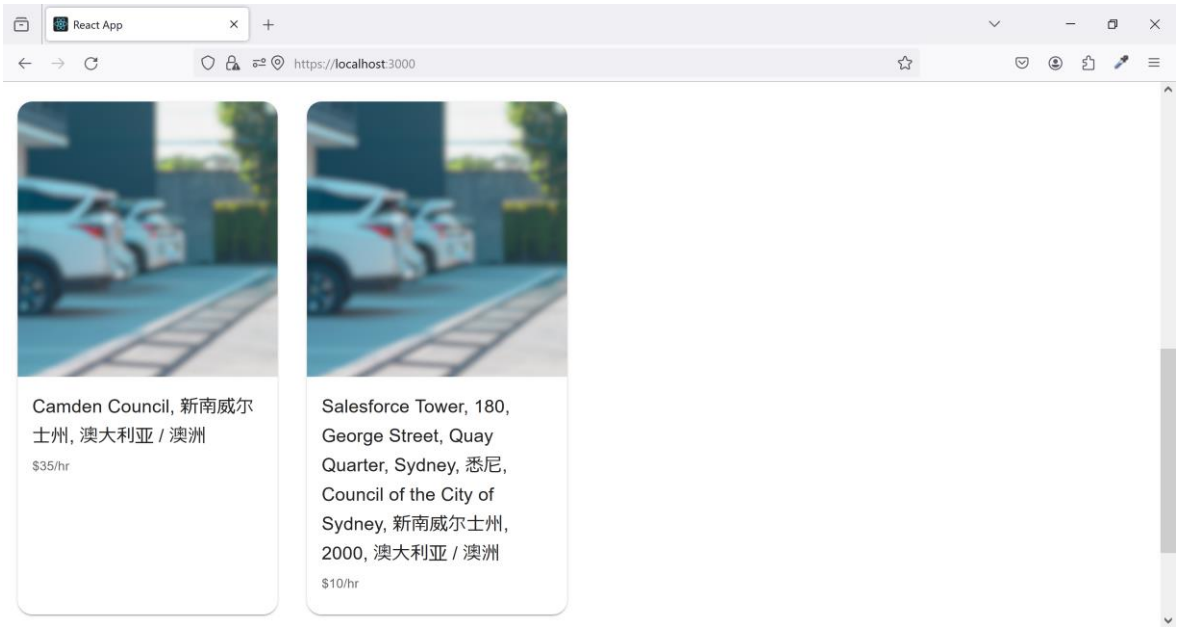


Figure 40: Listing that was Just Added Showing Up on Dashboard

Back at the “My Parking Spaces” page, the user can click the ‘LIVE STATUS’ button again to view the listing dates or deactivate it.

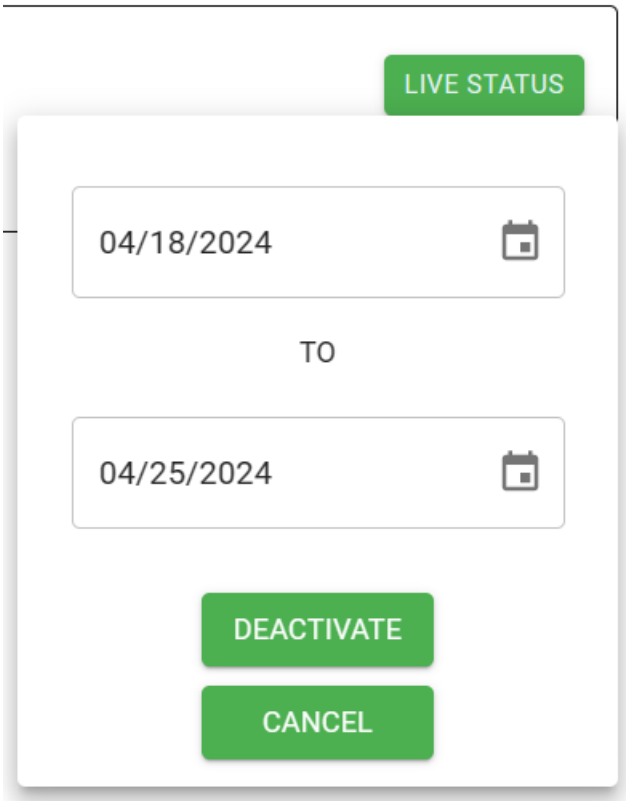


Figure 41: Deactivate Listing Prompt

And after the 'DEACTIVATE' is clicked, the listing will be deactivated again. The user can also edit their listing by clicking on the 'EDIT' button on the listing

Figure 42: Manage Parking Spaces Page after Listing is Deactivated

The user can modify anything except the address of the listing.

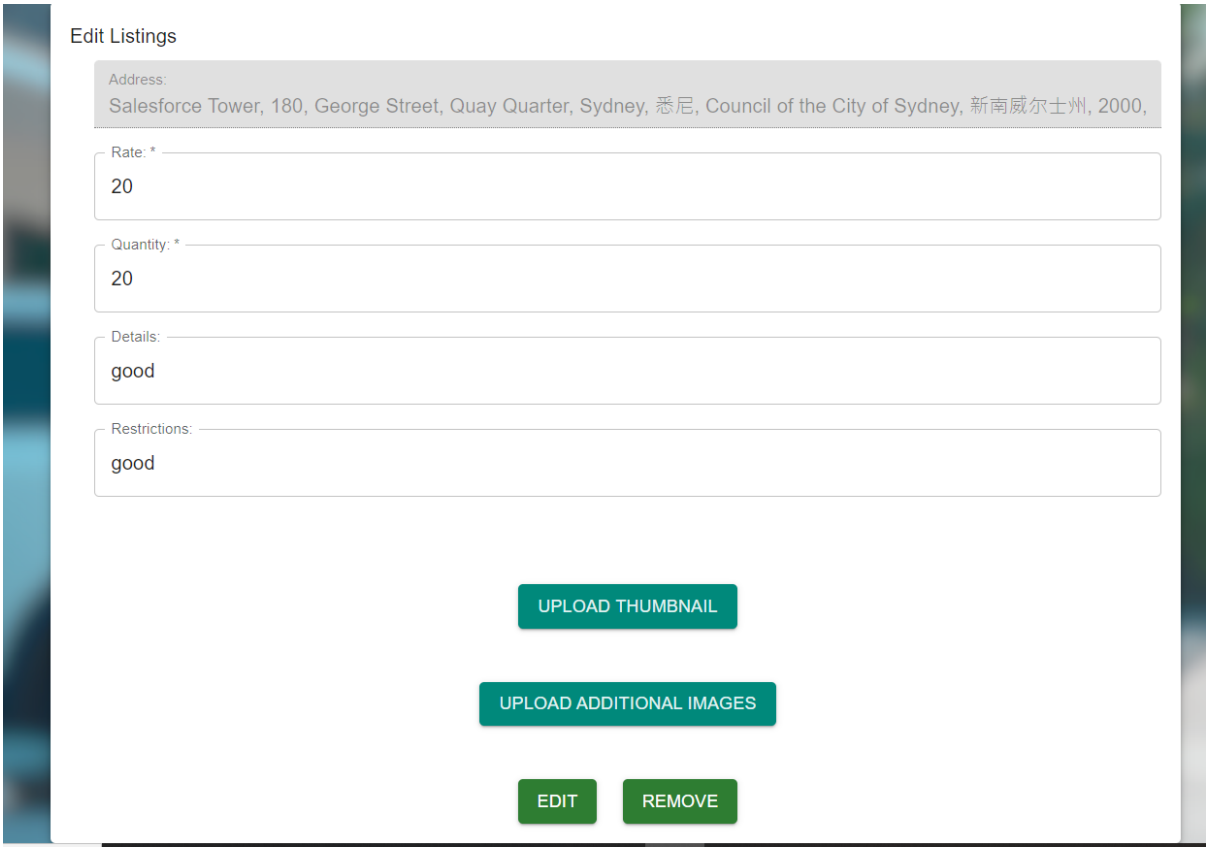


Figure 43: Edit Listings Page

And User then can click 'EDIT' to apply the changes or click 'REMOVE' to delete the whole listing.

5.3.4. Viewing the listings by searching and filtering

In the home page of the website, the web will ask user for permission to get their current location.

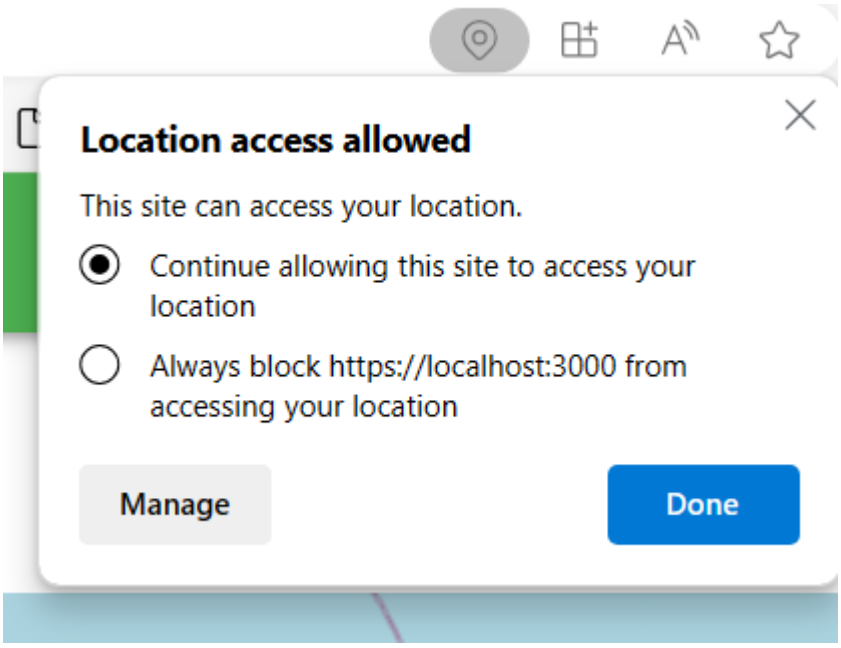


Figure 44: Prompt to Allow Access to Location

And then the page will contain a map and center on your current location. And if you block the access of location, the map will center on UNSW.

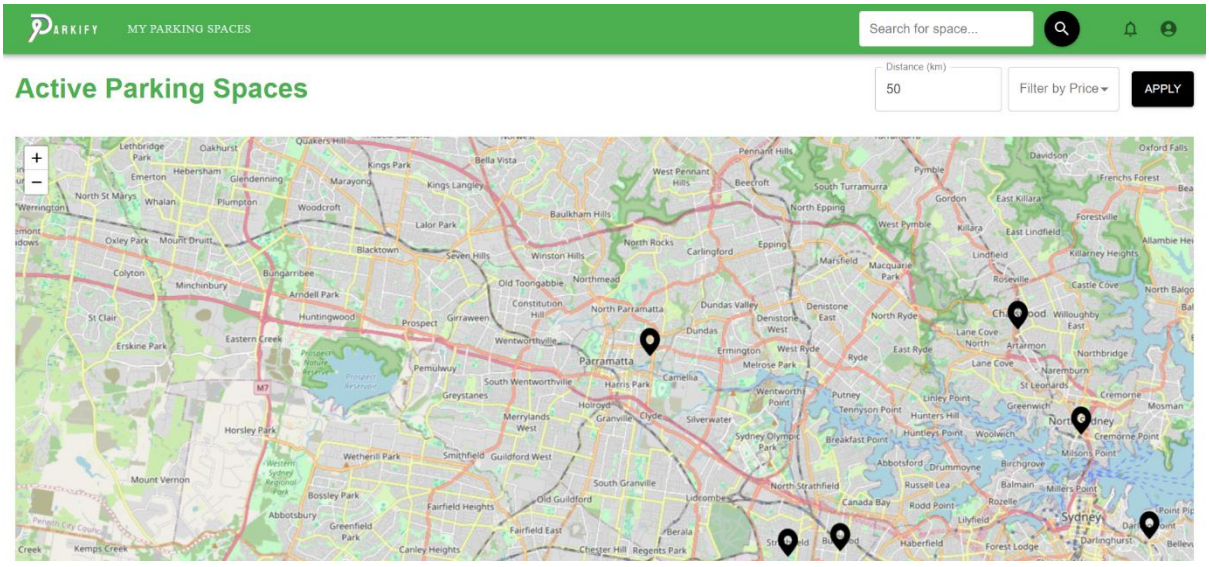


Figure 45: Map Centered Around User's Location

And there are corresponding listing cards shown on the bottom of the page with a show recommendations button.

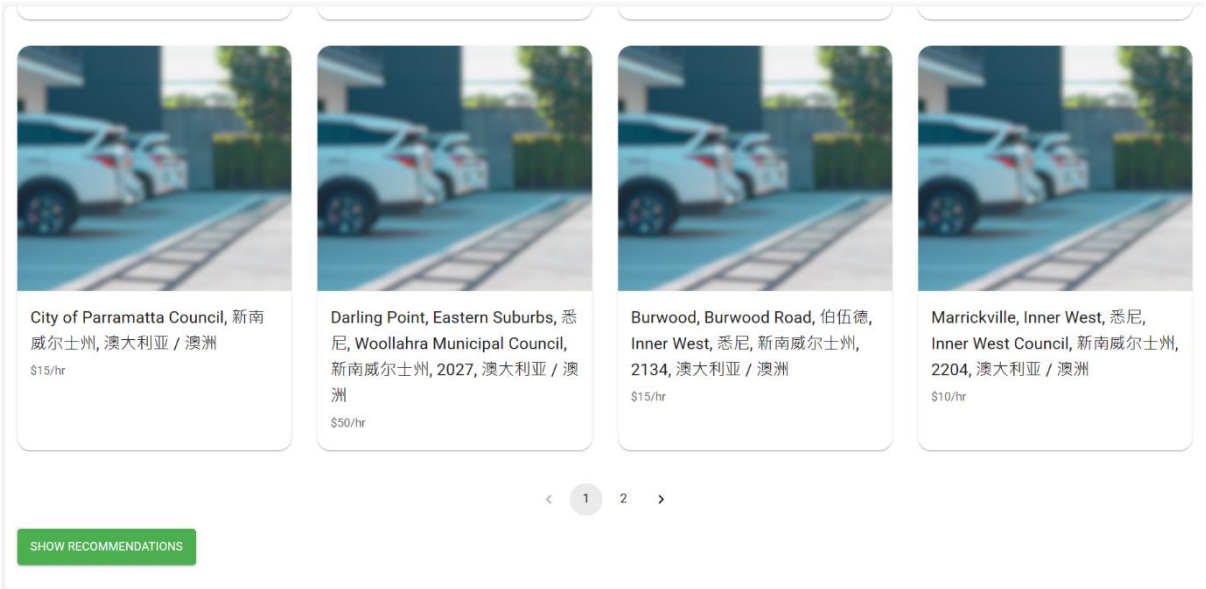


Figure 46: Pagination of Listings

The listings are paginated, and user can click to navigate different pages.

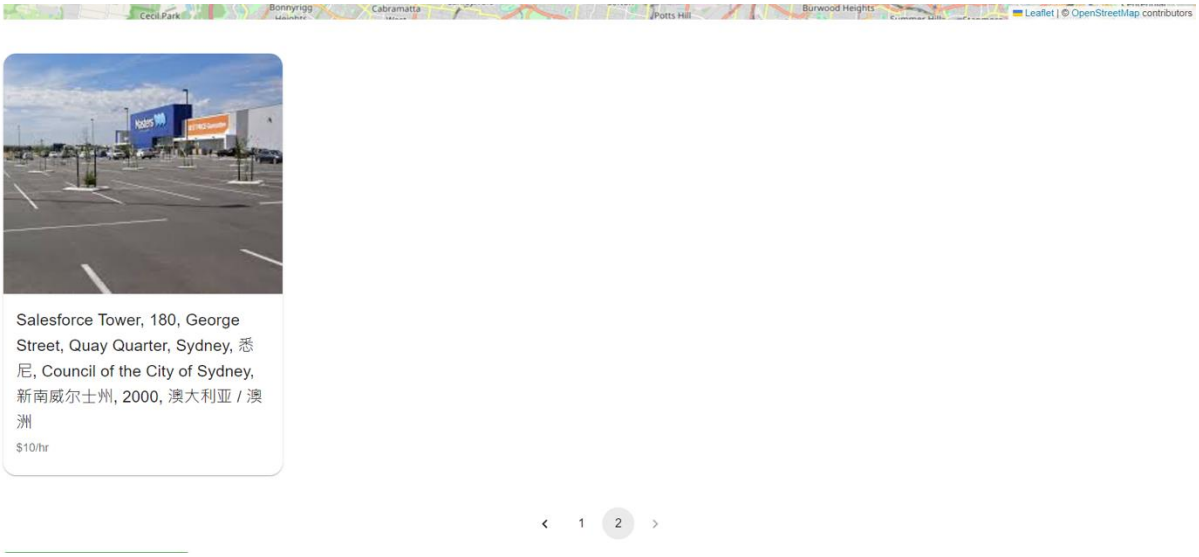


Figure 47: Second Page of Pagination

The page will show a search bar, distance filter and price sort filed on the top of it.

The distance filter will filter the listing around you within a certain number of km. And it has a default number of 50km. So, if there is no activated listing around you within 50km. You may not see any black marker on it. And you may want to higher the distance value and click apply or create more listing around you.

The price sort allows the listings to be sorted from lowest to highest or vice versa.

After users modify the filter and click apply. The change will apply to the map and the below listing cards.

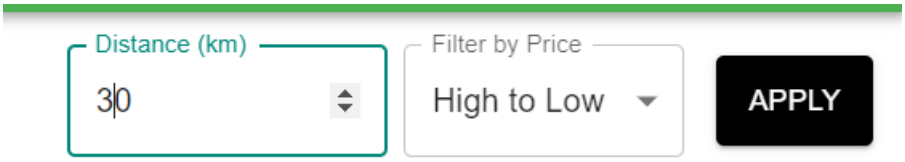


Figure 48: Application of Filters Based On Distance

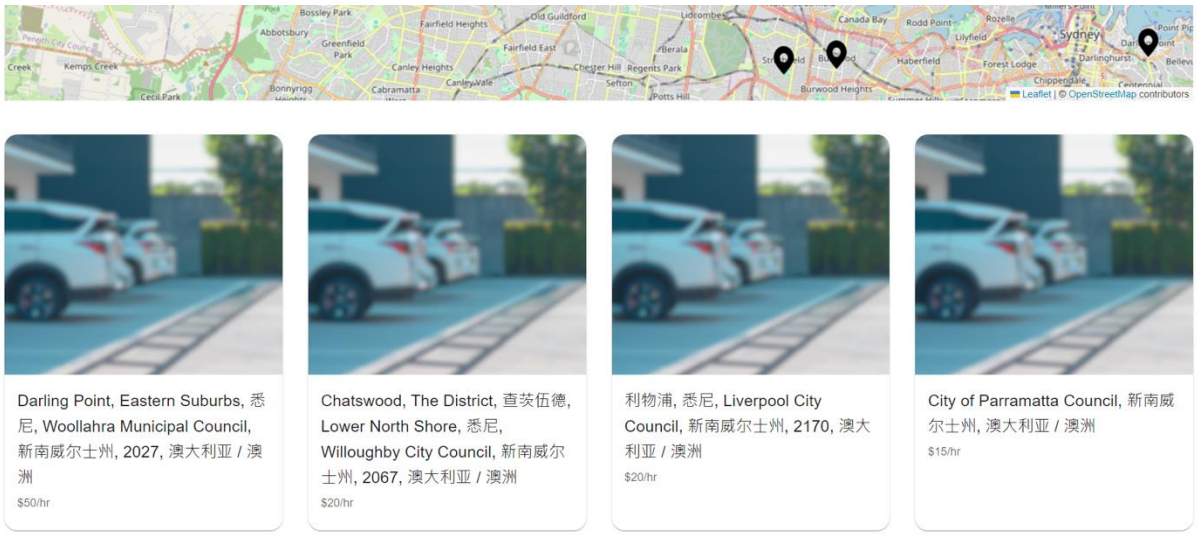


Figure 49: Result On The Dashboard

And the search bar allows you to search for an address and the result will also be reflected on both the map and listing cards.

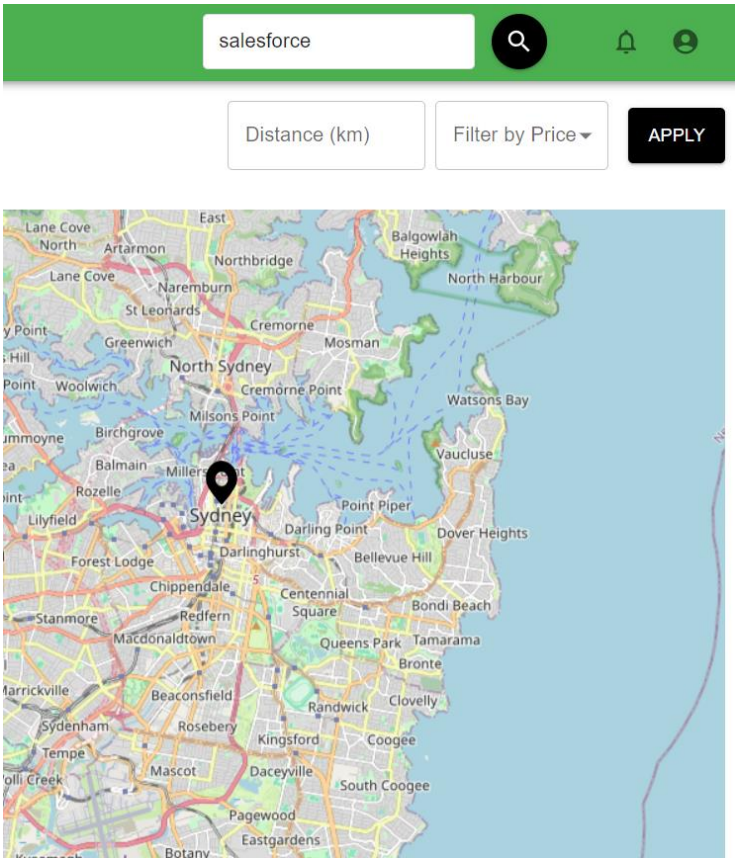


Figure 50: Search Result for Query on Map

5.3.5. Booking Functionality

First, the user has to select a listing. There are two ways to do this, by either using the map or by checking the listing cards beneath the map.

1. For the method of using the map, the user can look through the map to find a listing (Note: a listing may not appear on the map if it is on a different page. Only listings on the current page are shown in the map). Once found, they can click the listing (shown as a black icon) and the listing card will be shown

as such:

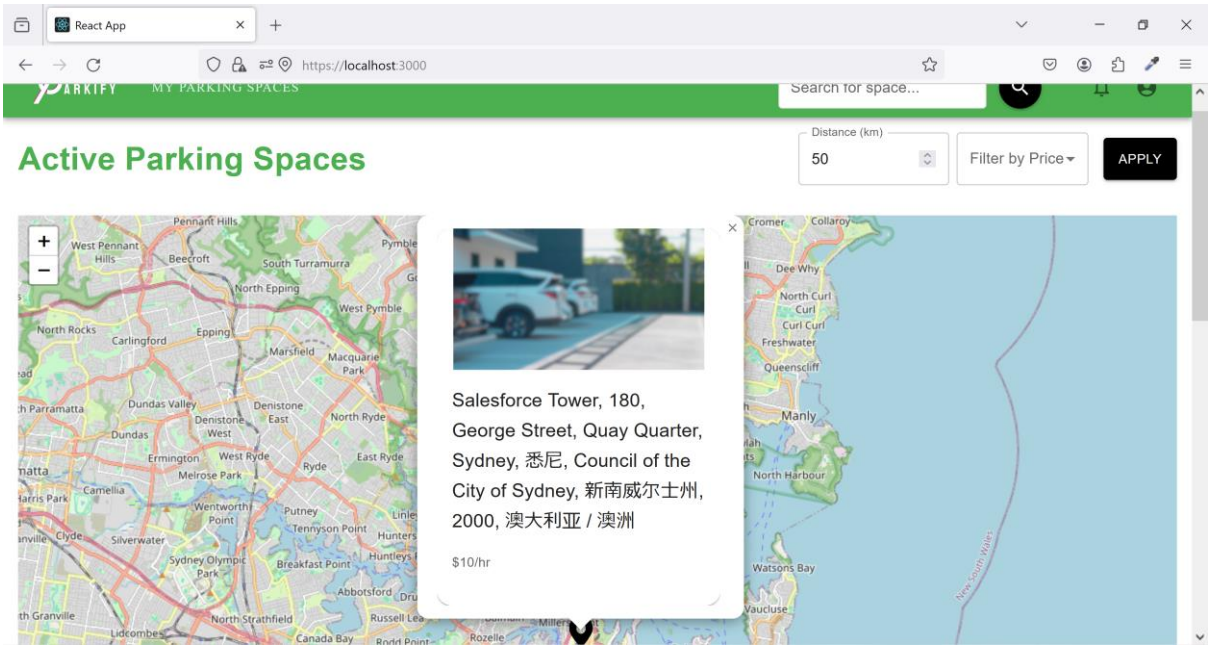


Figure 51: Listing Card Showed On Clicking Pin

The user can then click this listing to view its details specifically.

2. For the method of finding a listing via the cards beneath the map, the user can similarly find and click the listing card to view its details:

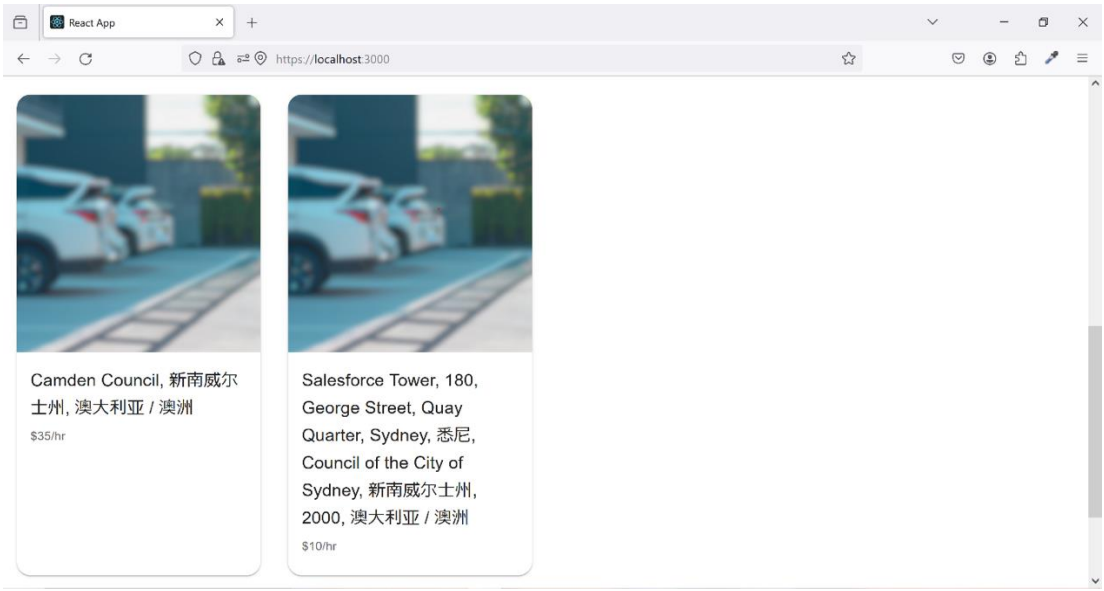


Figure 52: Dashboard Showing Listings

Once the listing is clicked, a new tab will be opened with the listing details:

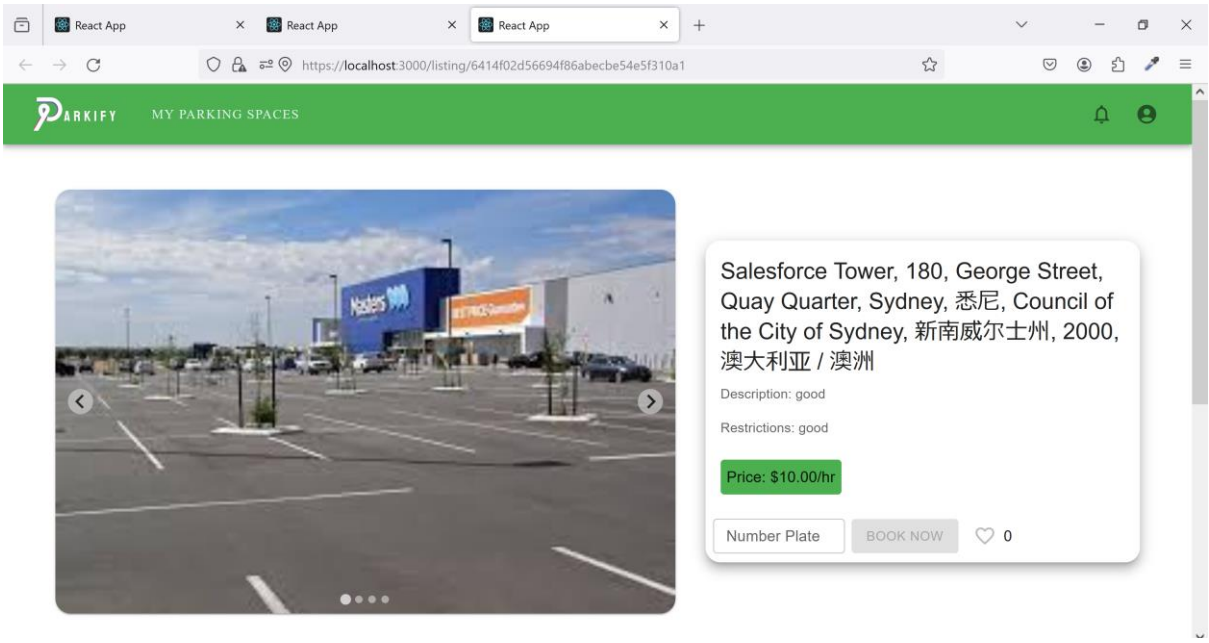


Figure 53: Details Of The Listing

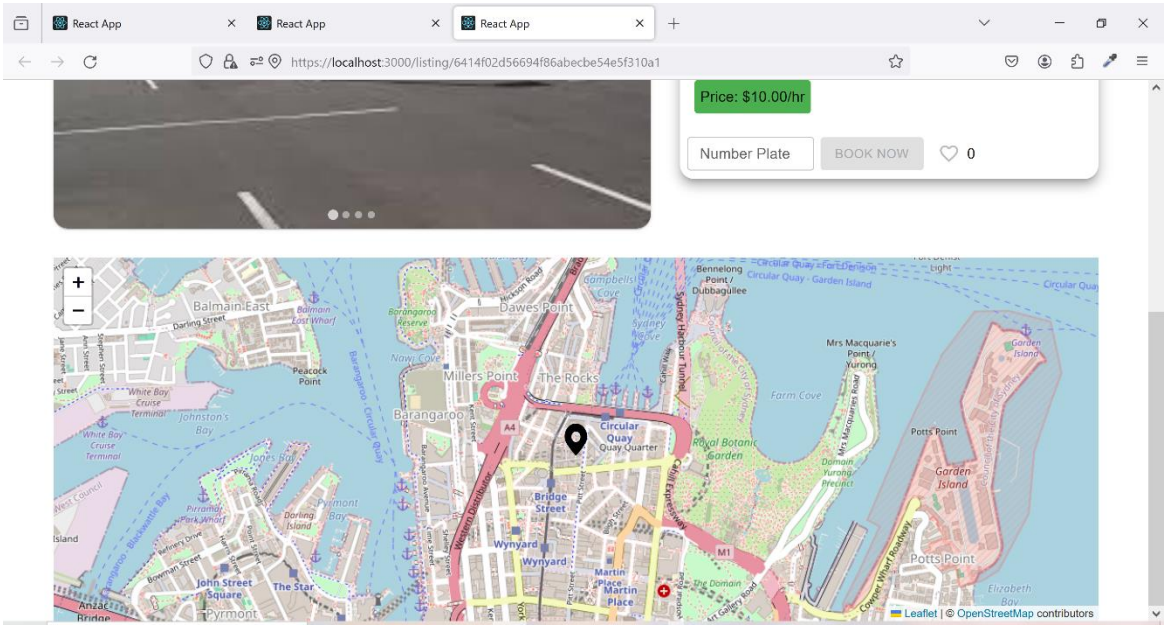


Figure 54: Number plate, listing images, other details and a book button

Here the user can see the listing images and details. To create a booking for this listing, the user needs to enter a 6-digit plate number of the vehicle they intend to park there in the “Number Plate” input box. Once this is done, the “Book Now” button can be clicked to enter the next phase of the booking process, which is the prebooking stage.

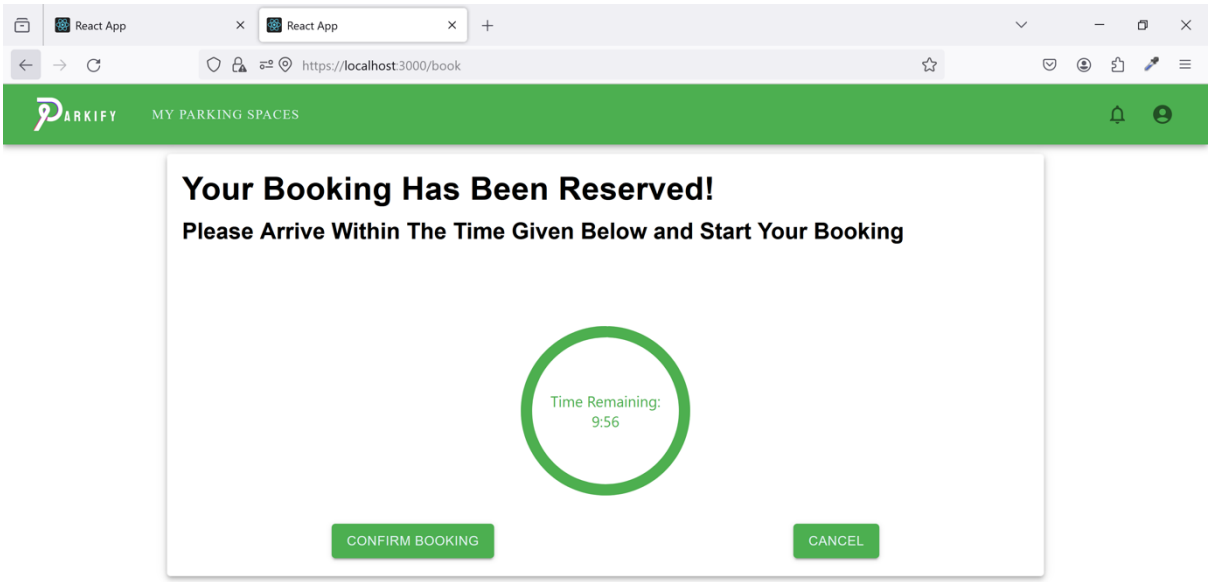


Figure 55: Pre Booking Timer Page

Here the user is given a 10-minute window to arrive at the parking space and start their actual booking. The listing for the duration of the booking from now is held and thus removed from the active listings home page. If the time lapses or the “cancel” button is pressed, the booking process is cancelled, the listing returns to being active in the homepage, and the user is redirected to the home page as well. Otherwise, if the user clicks “confirm booking” they are redirected to the booking timer page, where their booking is started:

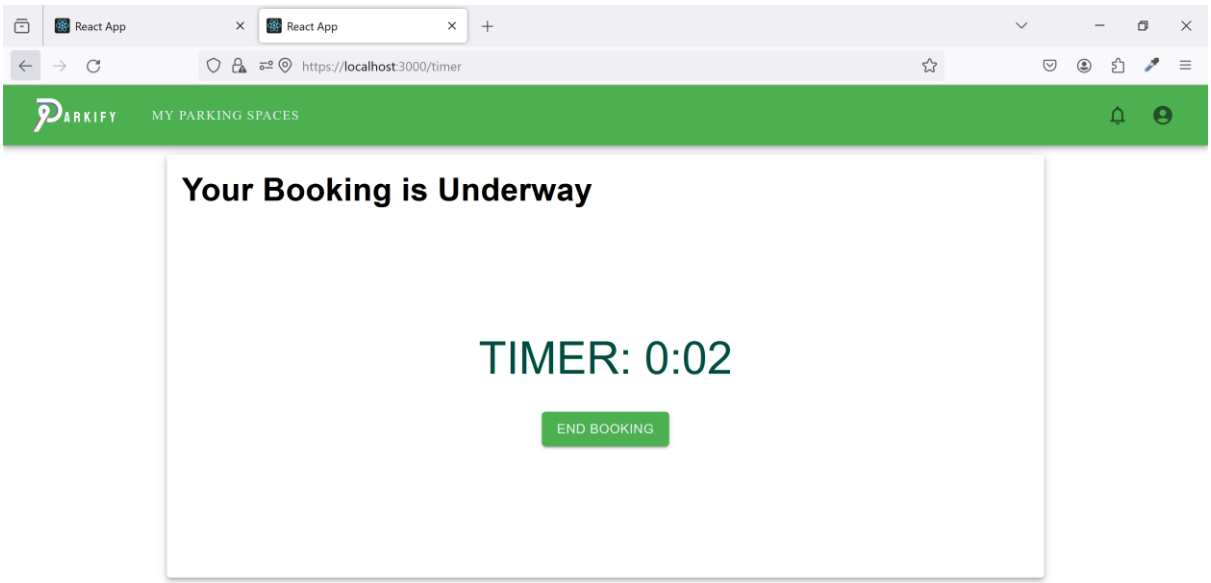


Figure 56: Booking Timer Page

When the user is finished with their booking/parking and decides to leave, they click “end booking”:

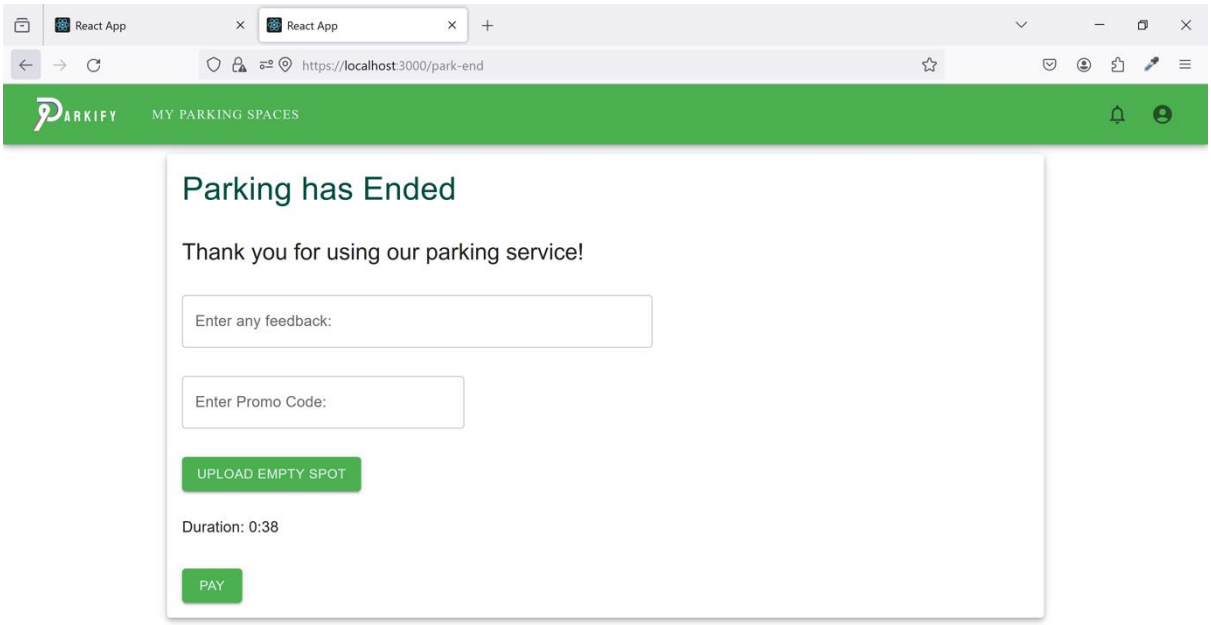


Figure 57: End Booking Page

On the end booking page, the user can enter any feedback they have and also any promo codes that might reduce the price. The list of promo codes can be found in ‘/server/promoCodes.txt’. Then before being able to pay, the user must upload an image of the empty parking spot to confirm that they have left:

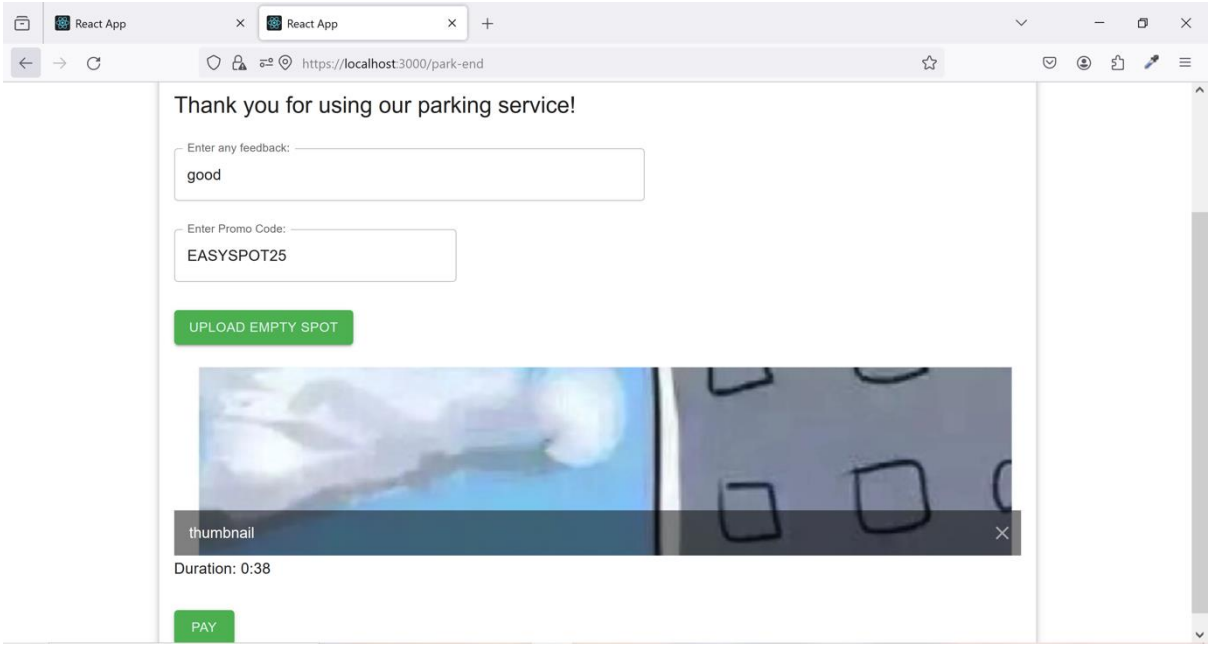


Figure 58: Prompt to Add Pictures of Empty Space

When they click pay, the payment is automatically made against their previously entered card:

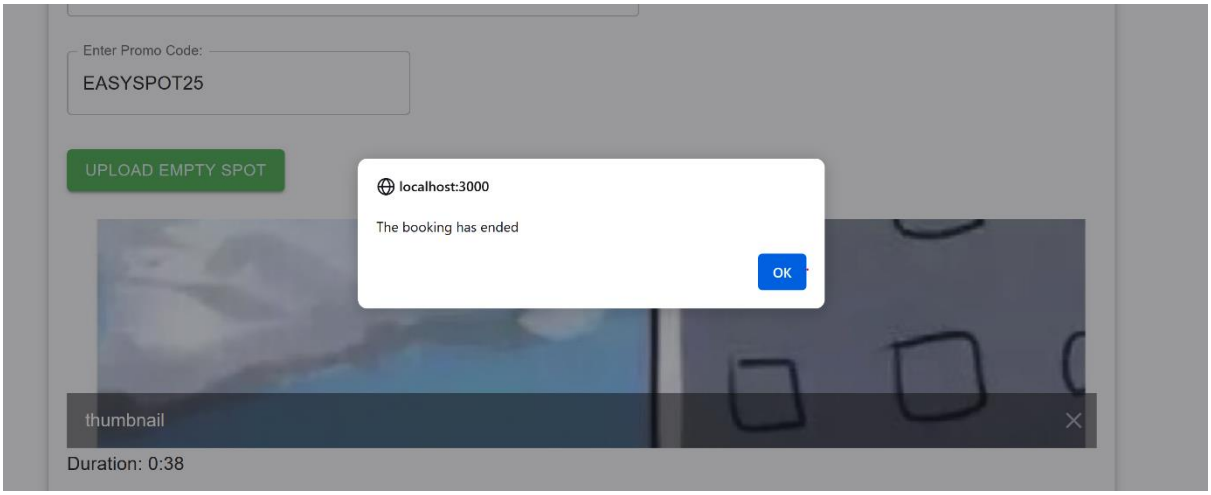


Figure 59: End Booking Alert

They are then redirected to the home page and the listing is re-activated (as the user is no longer using the listing):

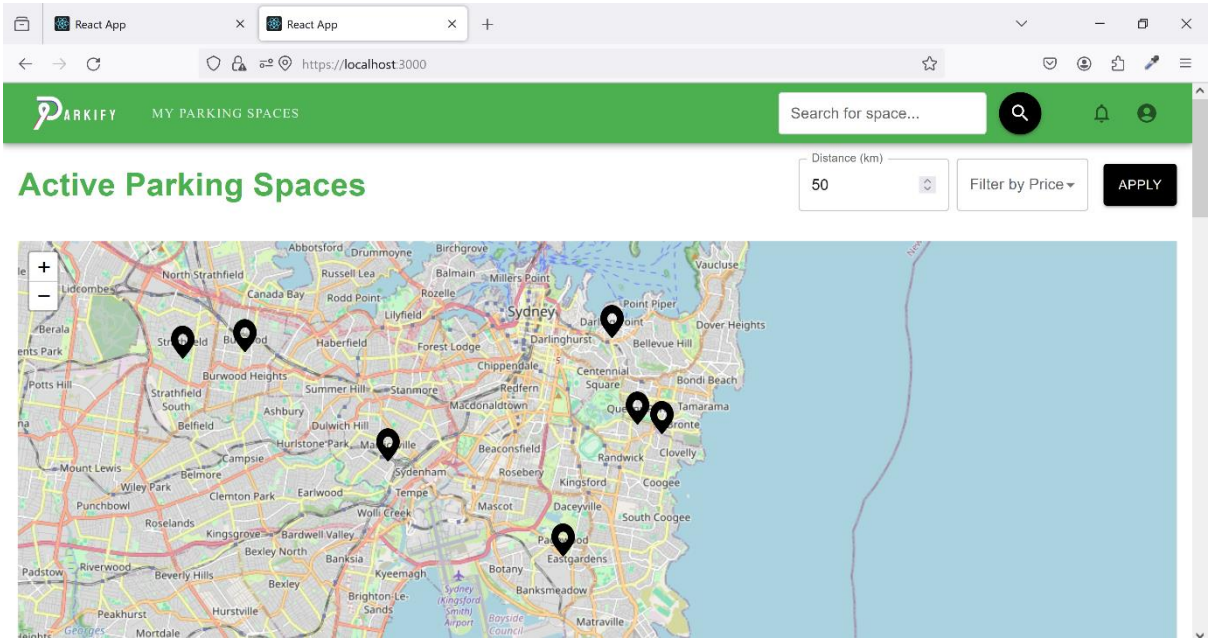


Figure 60: Parkify Home Page

The booking is then saved in the history of the user and the listing provider's customer history page for that listing (more details in 5.3.7).

Throughout this whole process, persistence is maintained. This means that at any given time in the booking process (prebooking 10-minute timer, actual booking timer, end payment page) the user can logout, change sites, or do anything else. When they come back and click the "Parkify" logo on the top left (or just login), depending on the booking phase that they were in, they will be shown that page (prebooking, booking, end payment).

5.3.6. User profiles

Users can edit their profile by clicking the account circle in the top right corner then clicking profile.

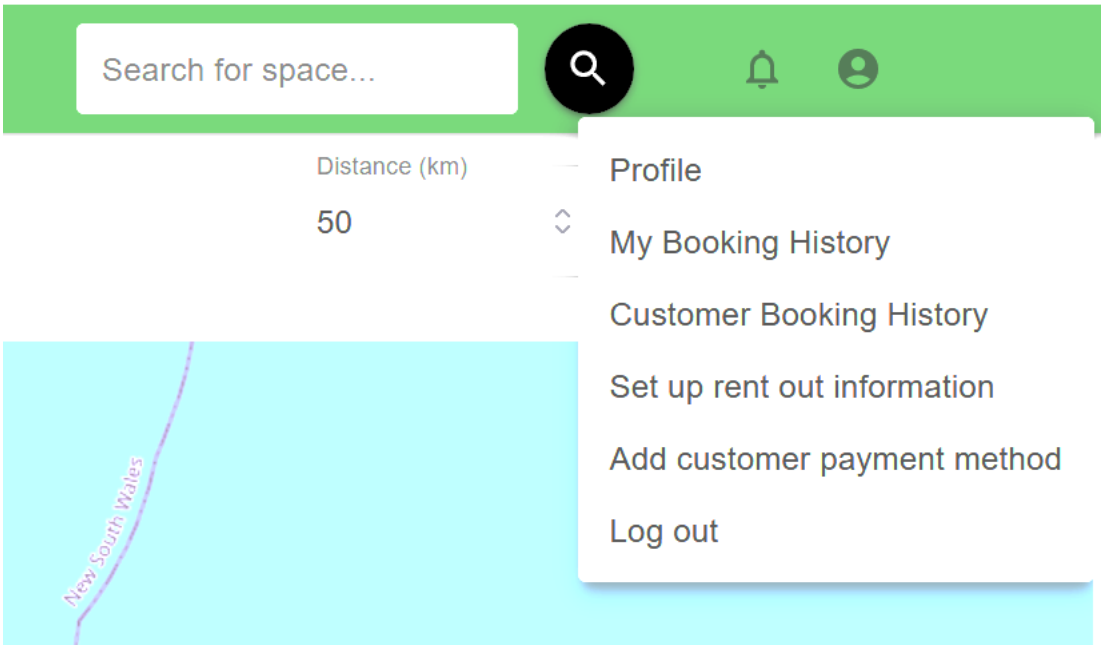


Figure 61: Dropdown page After Clicking Icon

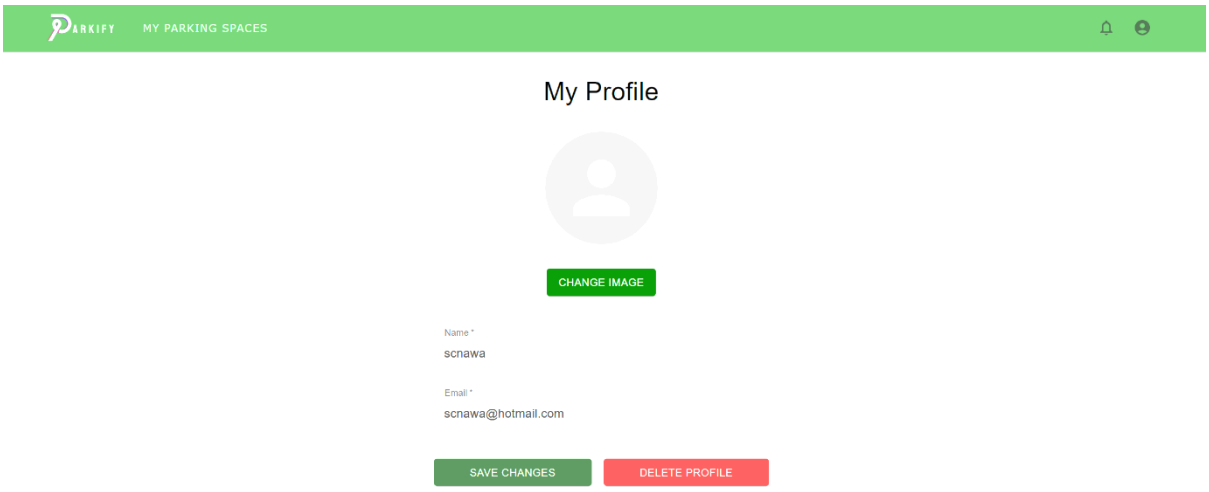


Figure 62: Profiles Page

User can then change their name or email address or upload a new profile image by clicking the save changes button.

They can also delete their profile by clicking on the delete button.

5.3.7. Histories and disputes

Users are able to view their booking history.

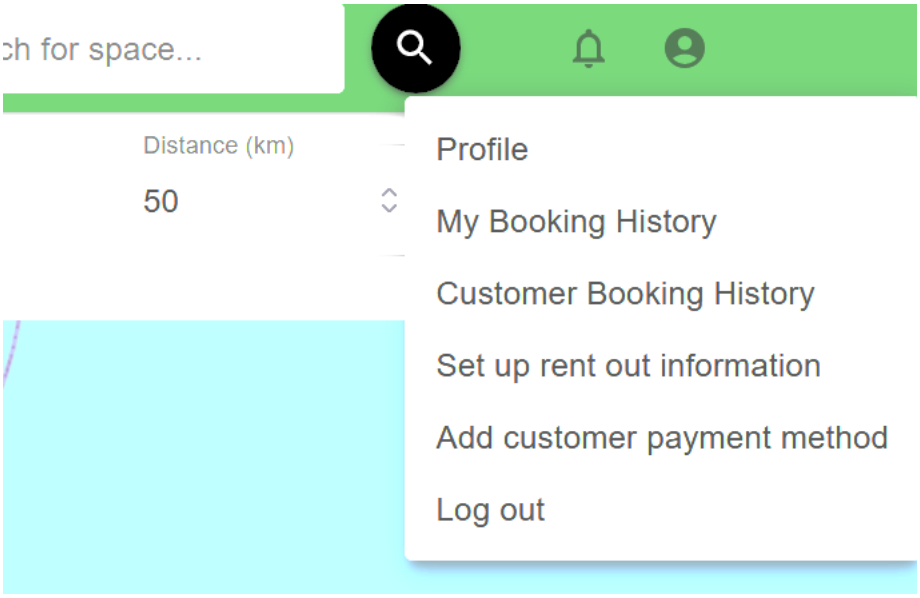


Figure 63: Dropdown Menu After Clicking Icon

To check their bookings, they can click “My Booking History”.

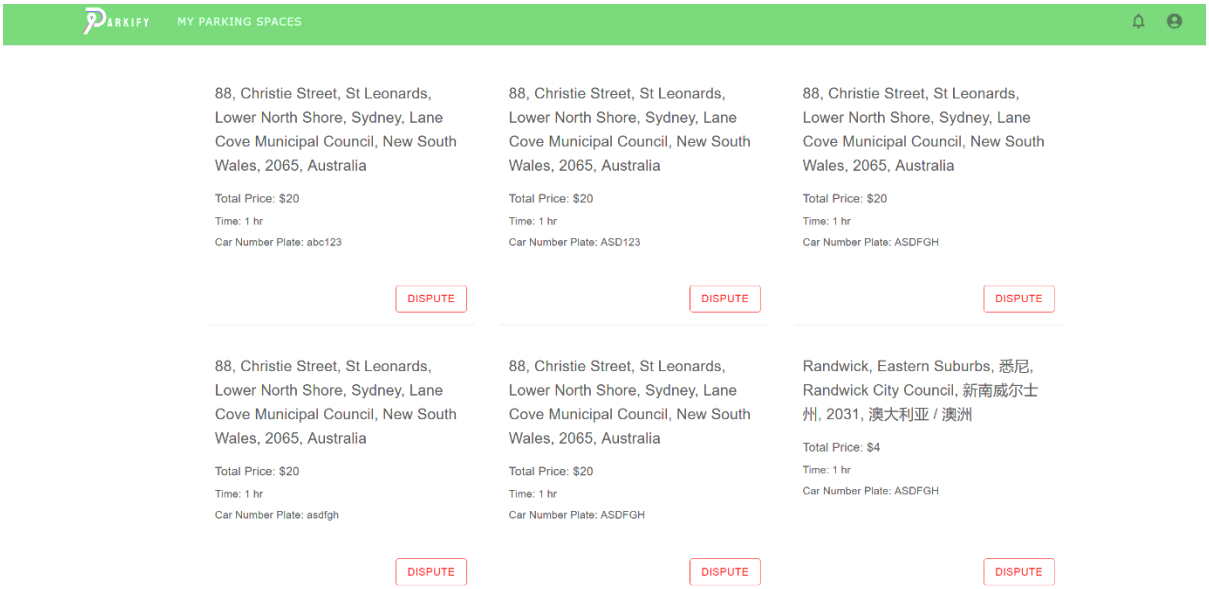


Figure 64: Booking History Page

They can click any booking card (not on the dispute button but anywhere on the card) to navigate to the listings specific details page (the same page that is shown when listing card is clicked on the home page). A dispute can be raised with any of their bookings by clicking the red “dispute” button.

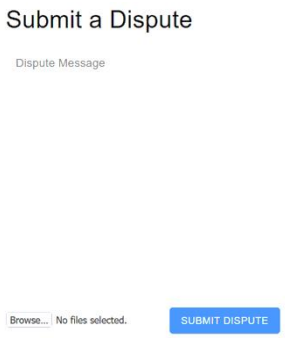


Figure 65: Disputes Page

They will then be directed to this page in which they can raise a dispute by explaining the issue, uploading image evidence, and clicking “submit dispute”.

5.3.8. Recommendation system

Users can scroll to the bottom of the dashboard to see recommendations provided by the system (which are unique to them).

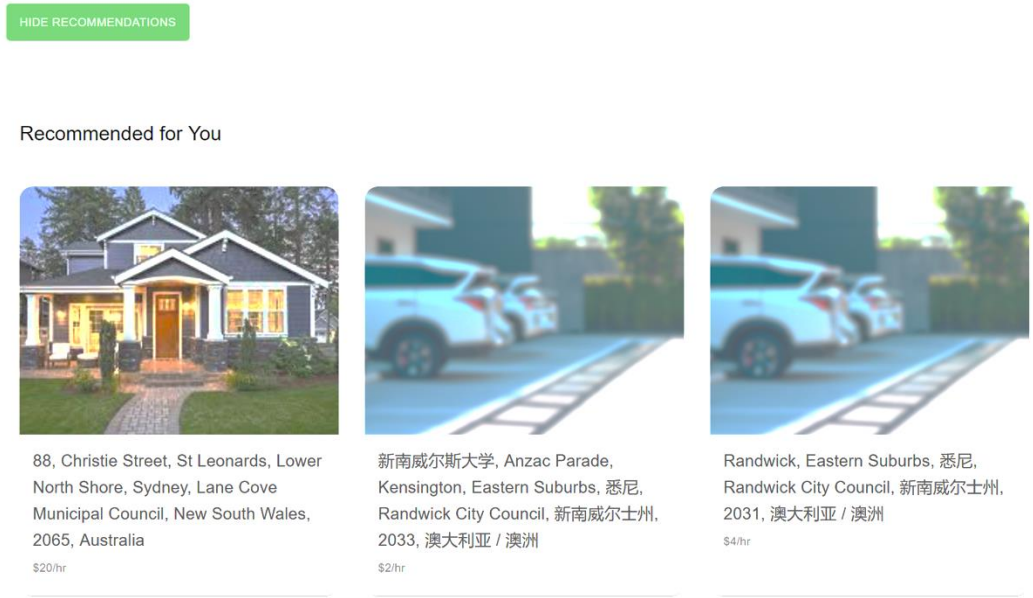


Figure 66: Recommendations Page

5.3.9. Like/Unlike listings

User can click the heart button to like/unlike a listing they have booked before.

Figure 67: Like Booking Functionality

5.3.10. Admin functionalities (Disputes, Manage Users)

For admin functionalities, the user needs to login as admin (the admin setup process is shown in 5.3.1)

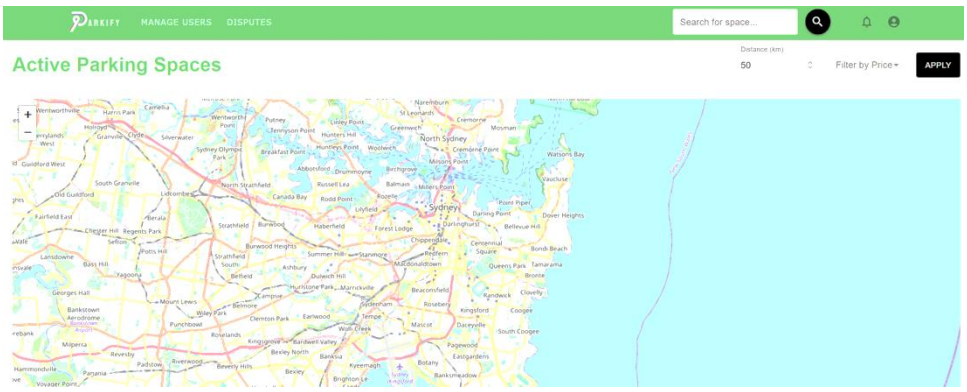


Figure 68: Parkify Admin Home Page

The admin can manage users and mange disputes as seen in the green nav bar.

1. Manage users: First the admin selects the user to manage by clicking the dropdown menu on the top left where they can search for a user or find them by scrolling down the dropdown. The admin has the ability to edit the account details or delete the account. They can also manage that user's listings/parking spaces which are shown below

the profile, where the same capabilities that a user has, the admin also has (edit/delete listing).

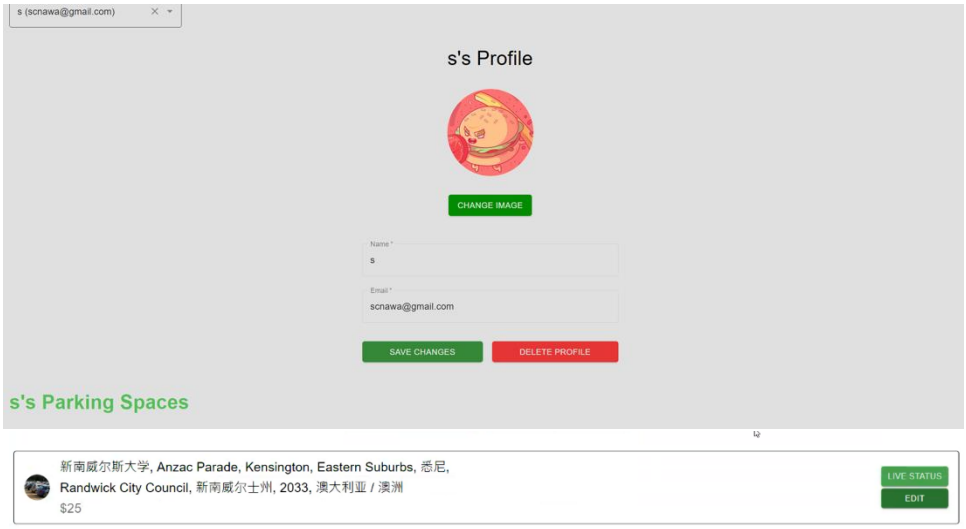


Figure 69: Admin reviewing a User's Details

2. The admin can also manage disputes raised by providers or consumers by clicking disputes. They can click “view details” to view the full details (images, etc) of that dispute. They can filter by all, resolved, or unresolved disputes. The admin should manually fix any disputes by contacting all involved parties and providing solutions (e.g. worst-case scenario contacting the authorities). Once the issue is resolved, the admin can click “mark as resolved” to resolve the issue.

Figure 70: Admin Checking All the Disputes

5.3.11. Notifications

Users can click the notification icon (left of the account icon) to see their notifications.

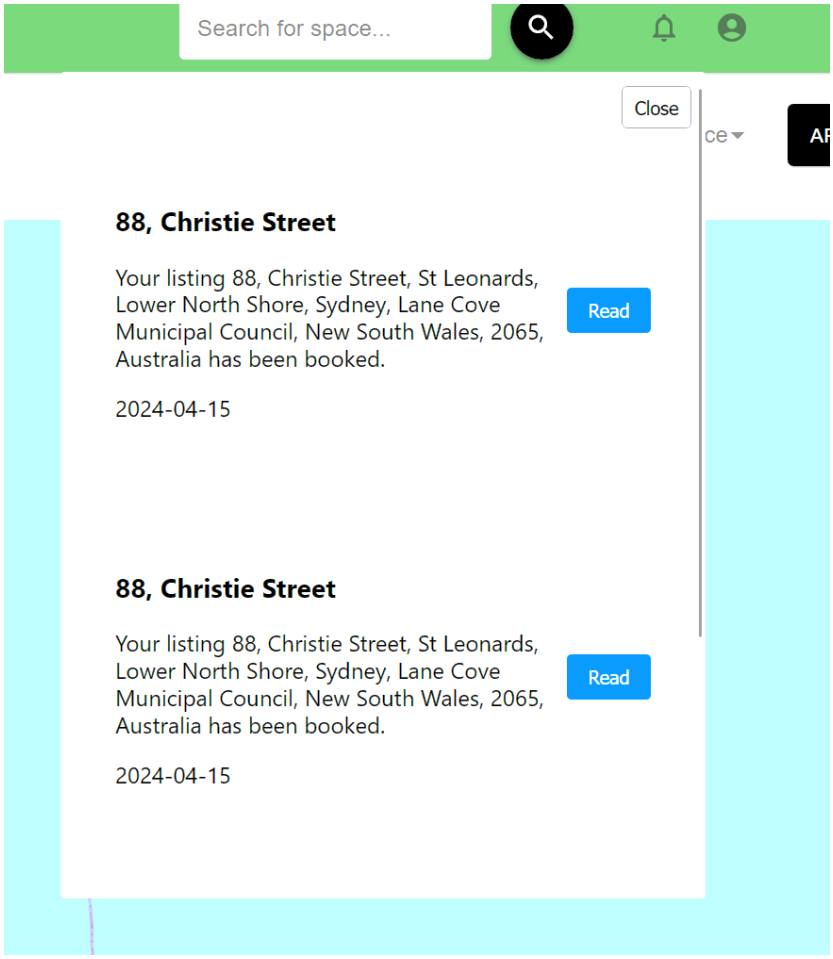
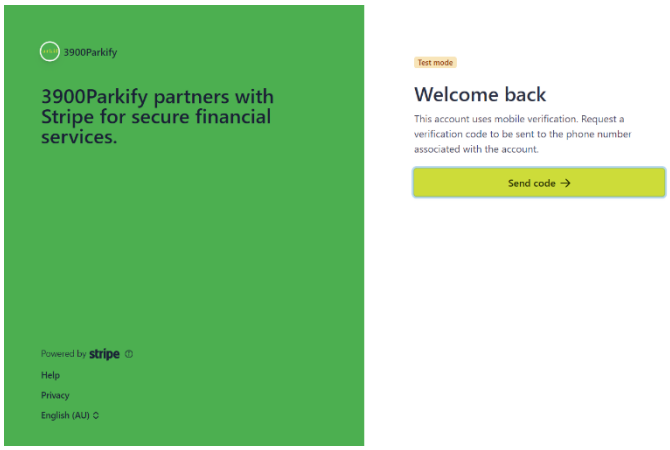


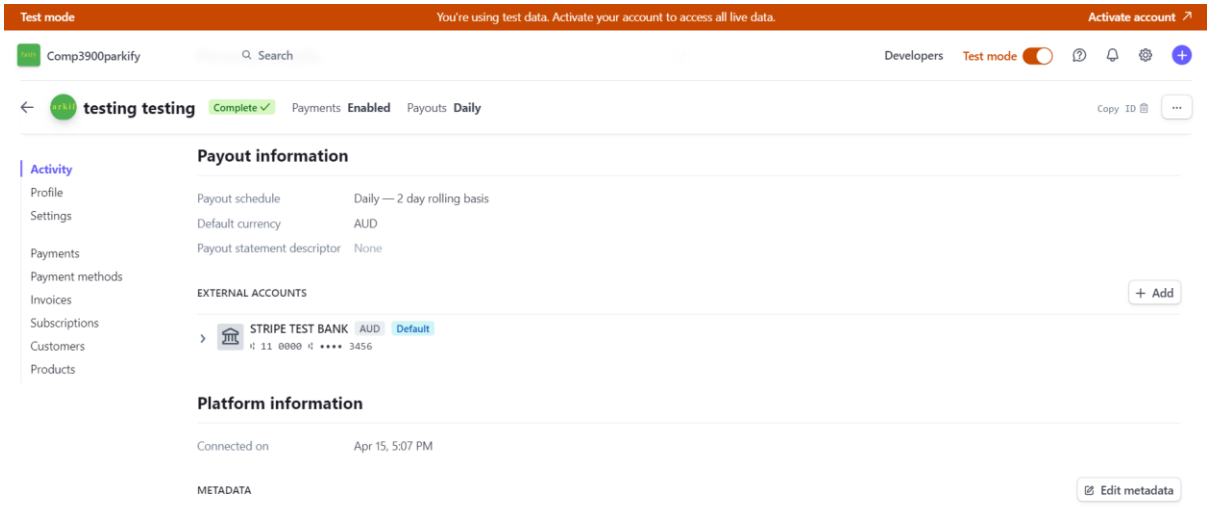
Figure 71: Notifications Dropdown

6. Appendix

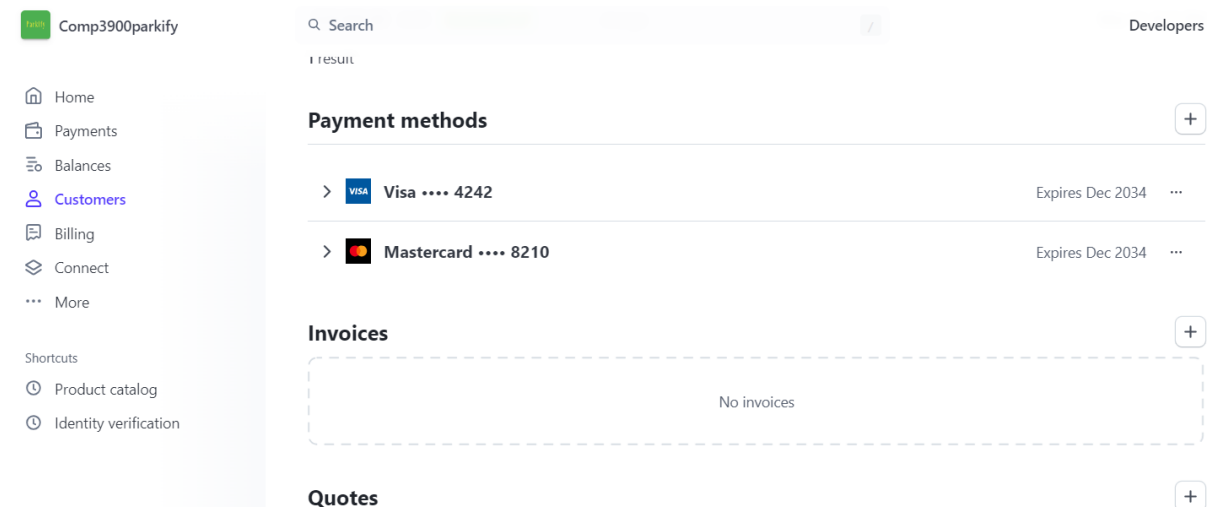
[1]



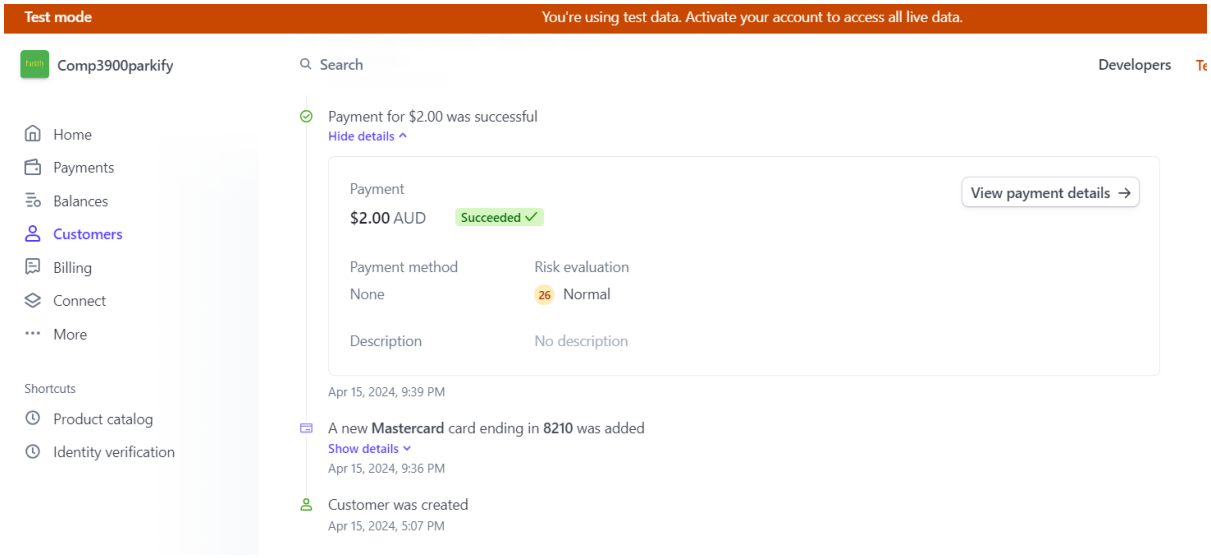
[2]



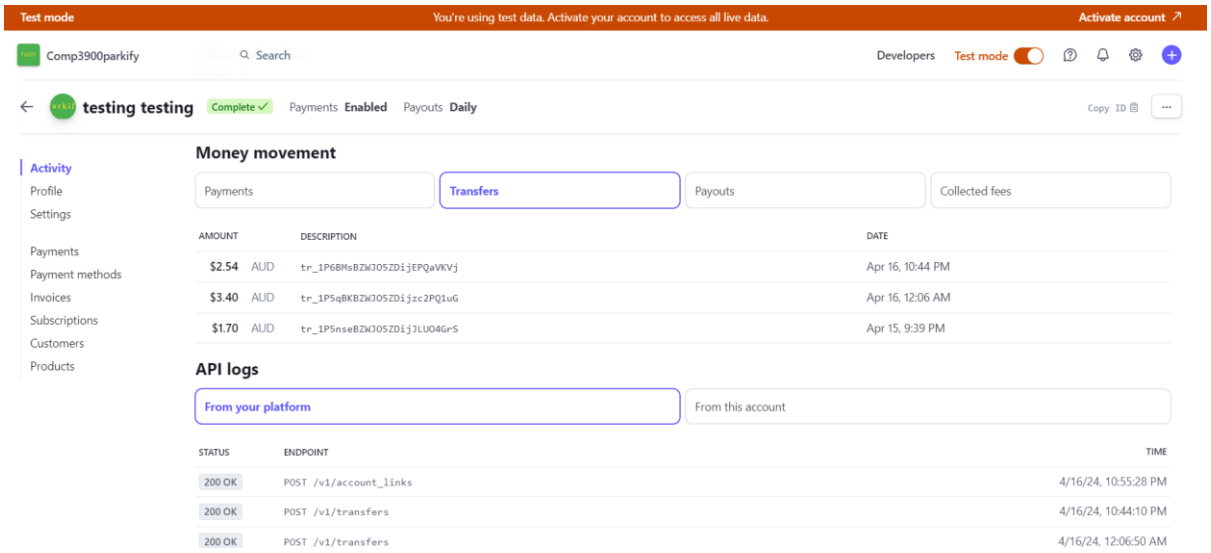
[3]



[4]


Test mode
You're using test data. Activate your account to access all live data.
Comp3900parkify
Search
Developers
Home
Payments
Balances
Customers
Billing
Connect
More
Shortcuts
Product catalog
Identity verification
Payment for $2.00 was successful
Hide details
Payment
$2.00 AUD
Succeeded
View payment details
Payment method
None
Risk evaluation
26 Normal
Description
No description
Apr 15, 2024, 9:39 PM
A new Mastercard card ending in 8210 was added
Show details
Apr 15, 2024, 9:36 PM
Customer was created
Apr 15, 2024, 5:07 PM

[5]


Test mode
You're using test data. Activate your account to access all live data.
Activate account
Comp3900parkify
Search
Developers
Test mode
testing testing
Complete
Payments Enabled
Payouts Daily
Copy ID
Activity
Profile
Settings
Payments
Payment methods
Invoices
Subscriptions
Customers
Products
Money movement
Payments
Transfers
Payouts
Collected fees
AMOUNT
DESCRIPTION
DATE
$2.54 AUD
tr_1P6BMsBZWJO5ZDijEPQaVKVj
Apr 16, 10:44 PM
$3.40 AUD
tr_1P5qBKBZWJO5ZDijzc2PQ1uG
Apr 16, 12:06 AM
$1.70 AUD
tr_1P5nseBZWJO5ZDijJLUO4GrS
Apr 15, 9:39 PM
API logs
From your platform
From this account
STATUS
ENDPOINT
TIME
200 OK
POST /v1/account_links
4/16/24, 10:55:28 PM
200 OK
POST /v1/transfers
4/16/24, 10:44:10 PM
200 OK
POST /v1/transfers
4/16/24, 12:06:50 AM

[6]

```
def make_reco(self, headers):
    user = db.userbase_data.find_one({"session_id": headers['token']})
    if user:
        matrix_df = helper.make_df(db)
        # fit the model with the user interactions
        model = NearestNeighbors(n_neighbors=3, metric='cosine')
        model.fit(matrix_df)
        user_interaction = matrix_df.loc[headers['email']]
        distances, indices = model.kneighbors([user_interaction])
        # Retrieve recommended car spaces
        recommendations = []
        target_user_idx = matrix_df.index.get_loc(headers['email'])
        for idx in indices.flatten():
            if idx != target_user_idx:  # Exclude target user
                recommendations.extend(matrix_df.iloc[idx][matrix_df.iloc[idx] > 0].index)

        # Remove duplicates and sort recommendations by interaction strength
        recommendations = list(set(recommendations))
        recommendations.sort(reverse=True)
        listings_db = db.listing_data.find({})
        for listing in listings_db:
            if listing['listing_id'] in recommendations:
                recommendations.append(listing)
                recommendations.remove(listing['listing_id'])
        return json_util.dumps(recommendations)
```

[7]

```
def calcLatLong(address):
    API_KEY = "44278b4afd944530a529b53bc76f7110"
    url = f"https://api.geoapify.com/v1/geocode/search?text={address}&limit=1&apiKey={API_KEY}"

    response = requests.get(url)

    if response.status_code == 200:
        data = response.json()
        # Check if the 'features' list is not empty before accessing its items
        if "features" in data and data["features"]:
            result = data["features"][0]
            latitude = result["geometry"]["coordinates"][1]
            longitude = result["geometry"]["coordinates"][0]
            return latitude, longitude
        else:
            # Handle the case where 'features' is empty
            print(f"No geolocation data found for address: {address}")
    else:
        print(f"Request failed with status code {response.status_code}")

    # Return a default value if the API call fails, the data is not as expected, or no features are found
    return (0, 0)


def calculateDistance(sourceLat, destLat, sourceLon, destLon):
    return geodesic((sourceLat,sourceLon),(destLat, destLon)).kilometers
```

[8]

```
def signup(self, userData):
    latitude, longitude = geocoder.ip('me').latlng
    user = {
        "username": userData['username'],
        "password": userData['password'],
        "listings": userData["listings"], # one list of dictionaries of different listings
        "creditCards" : userData['creditCards'],
        "email" : userData["email"],
        "session_id": [],
        "recentBookings": [],
        "isVerified" : False,
        "latitude": latitude,
        "longitude": longitude,
        "payment_id": "",
        "payOut_id": "",
        "default_payment_id":"",
        "is_stripe_connected": False,
        "pre_booking_time": "",
        "profile_picture": "",
        "liked_listings": []

    }
    user['password'] = pbkdf2_sha256.encrypt(
        user['password'])
```

7. References

Anon., n.d. *Nomatim*. [Online]
Available at: https://nominatim.openstreetmap.org/search?

Bellone, O., 2018. *Github*. [Online]
Available at: https://github.com/stripe/stripe-python/blob/master/LICENSE
[Accessed 2024].

Clark, A., 2022. [Online]
Available at: https://github.com/facebook/react/blob/main/LICENSE
[Accessed 19 04 2024].

Coding, A. o., 2022. *Youtube*. [Online]
Available at: https://www.youtube.com/watch?v=rmlhGPy8rSY
[Accessed 19 04 2024].

Dimitrov, V., 2021. *Github*. [Online]
Available at: https://github.com/vydimitrov/react-countdown-circle-timer/blob/master/LICENSE
[Accessed 19 04 2024].

Fujita, N., 2020. *Github*. [Online]
Available at: https://github.com/Splidejs/react-splide/blob/master/LICENSE
[Accessed 19 04 2024].

Jackson, M., 2023. *Github*. [Online]
Available at: https://github.com/remix-run/react-router/blob/main/LICENSE.md
[Accessed 19 04 2024].

Nguyen, H., 2022. *Github*. [Online]
Available at: https://github.com/mui/material-ui/blob/next/LICENSE
[Accessed 19 04 2024].

Reva, M., 2018. *StackOverflow*. [Online]
Available at: https://stackoverflow.com/questions/46935678/how-to-avoid-site-not-secure-on-an-https-configured-express-server
[Accessed 18 04 2024].

Sen, A., 2024. *Github*. [Online]
Available at: https://github.com/Leaflet/Leaflet/blob/main/LICENSE
[Accessed 19 04 2024].

Stripe, D., 2019. *Github*. [Online]
Available at: https://github.com/stripe/react-stripe-js/blob/master/LICENSE
[Accessed 19 04 2024].

Lévy L Marquis (2021). Github. [online]
Available at: https://github.com/elmarquis/Leaflet.GestureHandling?tab=readme-ov-file
[Accessed 18 Apr. 2024].