



‘YBER’ PROJECT DOCUMENTATION

CP3407 – Advanced Software Engineering

Abstract

Documentation of the design process and decision making that occurred throughout the life of the ‘Yber’ project. Includes a general outline of the requirements for submission, engineering approach, and deliverable implementation.

Team 1 – Anthony Bokor, Xander Dino Caubat, Luke Hapgood, Handong Yuan

Table of Contents

Table of Contents.....	1
1. Introduction:	2
1.1. Outline:	2
1.2. Requirements:.....	2
1.2.1. User stories:	2
1.2.2. Version Control:	2
1.2.3. Architectural Design:.....	2
1.2.4. Database:	2
1.2.5. Graphical User Interface:	2
1.2.6. Testing:.....	2
1.2.7. Documentation:	3
2. Agile Software Engineering Approach:	4
2.1. Team Roles:.....	4
2.2. Building and Development Tools:	6
2.2.1. Miro.....	6
2.2.2. Figma.....	6
2.2.3. Bravo Studio	7
2.2.4. Firebase.....	7
3. Requirements Planning:.....	8
3.1. User Stories:	8
3.2. Iterations:.....	9
3.3. User and Client Feedback	12
4. Design:.....	14
4.1. Architectural Design:.....	14
4.2. Database Design.....	14
4.3. Graphical User Interface:	15
5. Implementation:	18
5.1. Back-End Functionality:.....	18
5.2. Front-End Functionality:	22
5.3. Version Control:	29
5.4. Testing.....	30
5.5. Future Improvements:	32
6. Appendix 1 – Rubric:	0

1. Introduction:

1.1. Outline:

As a team of software engineers, you have been tasked with producing a website, desktop or mobile app for "Yber-rent". This is for a future unicorn startup that connects people who need a car with people who want to rent out their cars. Something like Airbnb but instead of renting rooms, people will rent cars (bicycles, boats, pets, or anything else you may think of) from other people.

The following are acceptable project(s) for this task:

1. The project must be a software/IT development project, i.e. it requires writing source code of some sort or customisation of a software system.
2. You are strongly encouraged to use any productivity tools or software libraries, e.g. mobile app builders, or website builders.
3. The project must have a modern database, e.g. relational database such as MySQL.
4. The project must have a modern graphical user interface (GUI), i.e. command-line printout is not a GUI.

1.2. Requirements:

Generally speaking, the project should follow the agile structure as described in the textbook. Must include exemplary use of software development tools, building tools, and external libraries.

1.2.1. User stories:

Each team member is required to individually produce a set of user stories consisting of a title, description, and time estimation. As a group, the team will then assign priorities, ensuring the proposed features are correctly planned for implementation in a justified order and within a given budget. Your IT solution is planning to deliver "what is needed, on time, and on budget". It is strongly encouraged to study similar services like Air-B&B and Uber to learn and discover your user stories.

1.2.2. Version Control:

Exemplary use of GitHub/git or equivalent.

1.2.3. Architectural Design:

Architectural design. Must use online UML diagram tool, e.g.

<https://www.glify.com/uses/umlsoftware/>

Help is available in textbook Chapter 5.

1.2.4. Database:

Database designs. Must use an online tool, e.g. <https://www.genmymodel.com/database-diagramonline>

Help is available in textbook Chapter 5.

1.2.5. Graphical User Interface:

Interface design. Must use prototyping tool, e.g. <https://ninjamock.com/>

Help is available in textbook Chapter 5.

1.2.6. Testing:

Exemplary testing of all components. Test-driven development. Acceptance testing of all delivered features. Appropriate testing data sets. Delivered implementation matches the planning.

Help is available in textbook chapters 7, 8 and 9.

1.2.7. Documentation:

Add a page to your GitHub to explain and justify:

- Design of all major components.
- Your testing
- What building and development tools were utilised and how?

Finally, an additional page is required to illustrate/promote your delivered solution.

2. Agile Software Engineering Approach:

It was the teams' intention to utilise the 'Agile Software Engineering' approach to the project. This is an iterative and flexible methodology that prioritises customer collaboration, adaptive planning, and the rapid delivery of working software in short, fixed-time cycles (iterations), allowing for continuous improvement, user-centric design, and the ability to respond to changing requirements throughout the development process.

There were some small deviations from the standard agile approach, as certain aspects were found to be inappropriate for the scale of the project and team. One such example was that weekly meetings replaced daily standup meetings, as we did not have enough work occurring daily for this to be an effective use of our time.

2.1. Team Roles:

Early on in development, this required each member to take on one or more roles that align with the typical workflow seen in this style of engineering. In order to determine the roles each person would play within the group; a team charter was completed. This process involved allocating responsibilities for each role and comparing each team member's strengths and experience to see who suited each role best. The outcomes of this can be seen on the following page.

Anthony took on the role of product owner. Being our primary contact with Dmitry, he was able to represent the customer and stakeholders effectively and make decisions as to the prioritisation of certain aspects of the product backlog. Furthermore, upon completion of iterations, Anthony was responsible for communicating our progress back to Dmitry and ensuring that our development and outcomes remained in line with what was ultimately required of us.

Luke primarily took on the role of scrum master, facilitating the scrum process, taking meeting notes, and keeping track of the process and progress via thorough documentation. This meant ensuring that team members had tasks to complete, such that a reasonable workflow was maintained throughout, and confirming that tasks necessary for the completion of both individual iterations and the final submission were completed.

Xander was the technical lead of our development team. Having the most experience with the chosen software, and the greatest accessibility to each of the interconnected software. As such, he was able to provide guidance to the entire development team, helping especially to make architectural and design decisions, including database design and production.

Handong rounded out our development team as more of a free agent, filling in to help with whatever aspects of the project required prioritisation within each iteration. This helped to ensure that no one person was ever overloaded, and we could evenly and effectively distribute the workload in the event that a task proved more challenging than originally planned for.

Each team member made some level of technical contributions to our final product, and no one team member stuck strictly within each role. This is obviously an expected outcome in the context of the project, as relatively speaking we are small team producing an alpha software prototype. As such there were periods where more people were allocated to technical development for example.

To achieve our goals we need a
Product Owner 🦊

Their top 3 responsibilities are:

Prioritizing
and Managing
the Product
Backlog

Decision-
Making

Alignment
with
Stakeholders



?

Anthony

To achieve our goals we need a
Scrum Master 🐻

Their top 3 responsibilities are:

Project
Organisation

Servant
Leadership

Continuous
Improvement



?

Luke

To achieve our goals we need a
Lead Developer 🦄

Their top 3 responsibilities are:

Technical
Leadership

Mentoring
and Coaching

Team
Collaboration



?

Xander

To achieve our goals we need a
Developer 🦊

Their top 3 responsibilities are:

Coding and
Implementation

Problem Solving
and
Troubleshooting

Collaboration and
Communication



?

Handong

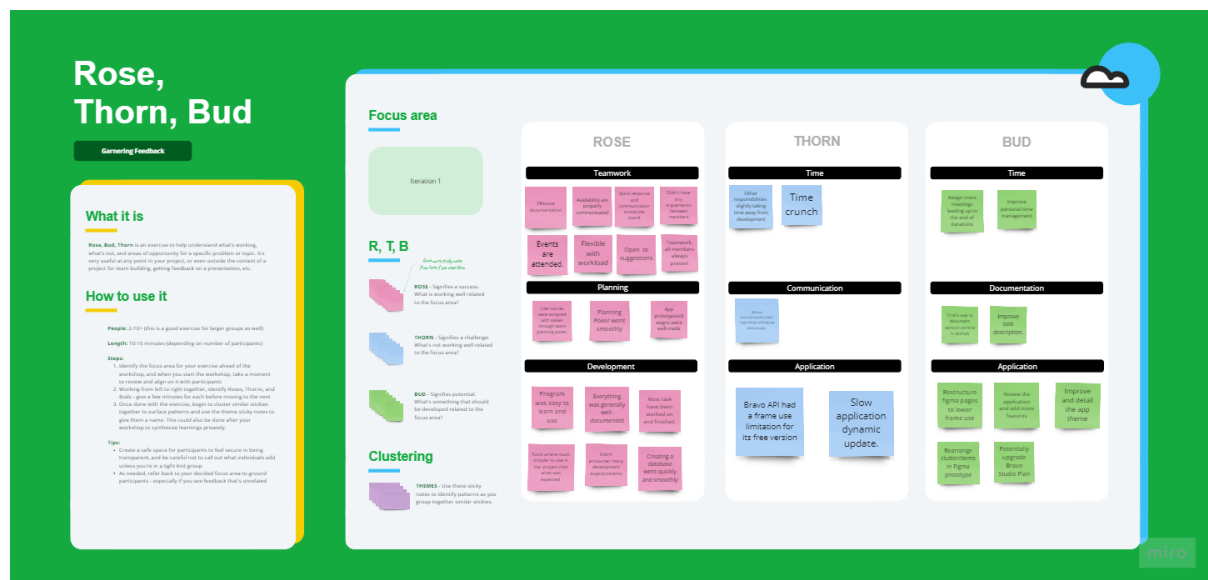
2.2. Building and Development Tools:

For the purposes of the project, a period of market research occurred with the goal of determining the ideal software to be used. Initially, this involved researching the general pros and cons of both website and app building, before branching more specifically into what was available within each. The team's prior experience was a key factor in the early decision to explore app building. Android Studio was an early consideration; however, it was quickly ruled out in favour of a combination of programs that would streamline our workflow significantly.

Development took place through the use of a combination the following software and services:

2.2.1. Miro

Miro is a versatile online collaborative whiteboard platform that supports visual thinking, brainstorming, and project planning. It is ideal for writing, creating and collaborating on user stories, as well as retrospectives and reviews, such as those at the end of each iteration as seen below. It was used for the product backlog and the team carter above among several other things.



2.2.2. Figma

Figma is a cloud-based design and prototyping tool primarily used for user interface (UI) and user experience (UX) design. It offers a collaborative platform for design teams to create, share, and iterate on design assets and prototypes. It allows multiple team members, including designers, developers, product managers, and stakeholders, to collaborate in real-time on design work. This aligns with Agile principles of constant collaboration and communication among cross-functional teams. Furthermore, it enables quick and easy modifications to design elements, accommodating agile processes where requirements may evolve throughout the project. It also accommodates user testing through its prototype feature, which assists in the validation of product outcomes, and ensures alignment with stakeholder interests. Several team members were previously familiar with Figma, and some research revealed that our version control could take place within this program.

2.2.3. Bravo Studio

Bravo Studio is a platform that specializes in turning design files, particularly those created in design tools like Figma, into functional mobile apps without the need for traditional coding. It allows designers to quickly turn their Figma design files into interactive, functional prototypes. This is beneficial in Agile development for creating early prototypes to validate design concepts and functionalities with stakeholders and end-users. Airtable

The first of our two database systems was produced via Airtable. Airtable's database capabilities provide Agile teams with a powerful tool for data organization, collaboration, and adaptability, supporting the core principles of Agile development such as transparency, flexibility, and continuous improvement. It empowers teams to manage data effectively, making it a valuable asset in Agile software development projects.

2.2.4. Firebase

Airtable on its own however was not entirely adequate as it did not provide any authentication services. As a result, an additional database was required to manage users signing in and out of the app. Firebase provides robust user authentication and access control features, enhancing security and ensuring that Agile teams can manage user data and permissions effectively.

3. Requirements Planning:

3.1. User Stories:

As a team, we developed a series of user stories centred around the prompt. The following is a raw list of those, in order of priority:

Title	Description	Length	Priority
Database	Yber-rent will have a database recording history, products, etc	14	50
Home Page	Welcome page for Yber-rent users, where most of links are present (such as filter selection and booking)	5	50
List of Vehicles	Users can display a list of nearby vehicles	5	50
Online or cash payment	Yber-rent users will be able to pay either online (credit card) or in-person with cash	5	40
Booking page	Booking site for pick-up/drop-off dates and/or general reservations	5	40
Cancel booking	Yber-rent users will be able to cancel their orders beforehand if they change their minds	1	40
Accounts	Yber-rent users will be able to login and logout of their accounts	3	30
Sign-up	A sign-up page for Yber users to register.	3	30
Log-out	A Log-out page for Yber users to go offline.	1	30
Settings Page	Yber-rent users can adjust their settings, such as light/dark mode, and font size	3	20
History	Yber-rent users will be able to view their purchase/ride history and relevant details	3	20
Support Page	Yber-rent users can send support tickets, inquiries, and issue resolutions through this page	3	10
Interactive Map	Simple map of local area	7	10
Reviews	Yber-rent users will be able to leave reviews	1	10
Filter by price	Users should be able to find vehicles within their price range	1	10
Filter by vehicle size	Users should be able to filter by vehicle size to only see vehicles which suit their needs	1	10

Filter by range	Users should be able to filter by how far away vehicles are	1	10
Filter by reviews	Yber-rent users will be able to filter by reviews	1	10

3.2. Iterations:

In accordance with standard agile software engineering procedure, the project was split into a series of iterations. This required each of the user stories to be analysed and allocated certain values. Both the priorities and lengths of each user story were assigned through rounds of 'planning poker'. The development team gathered for an estimation session, in which each team member was initially allocated a set of Planning Poker tokens ranging through 1, 3, 5, 7, and 14. Initially, a large bank of user stories was developed. Following this, the team grouped similar items, removed redundancies, and went through each item to clarify any doubts or uncertainties about the associated requirements to ensure each member was on the same page.

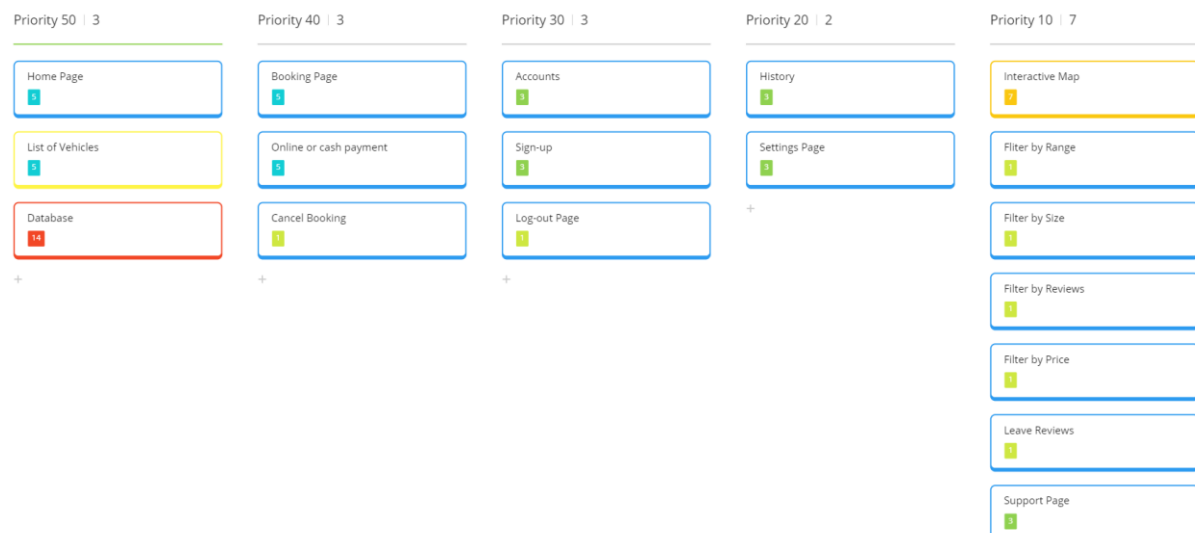
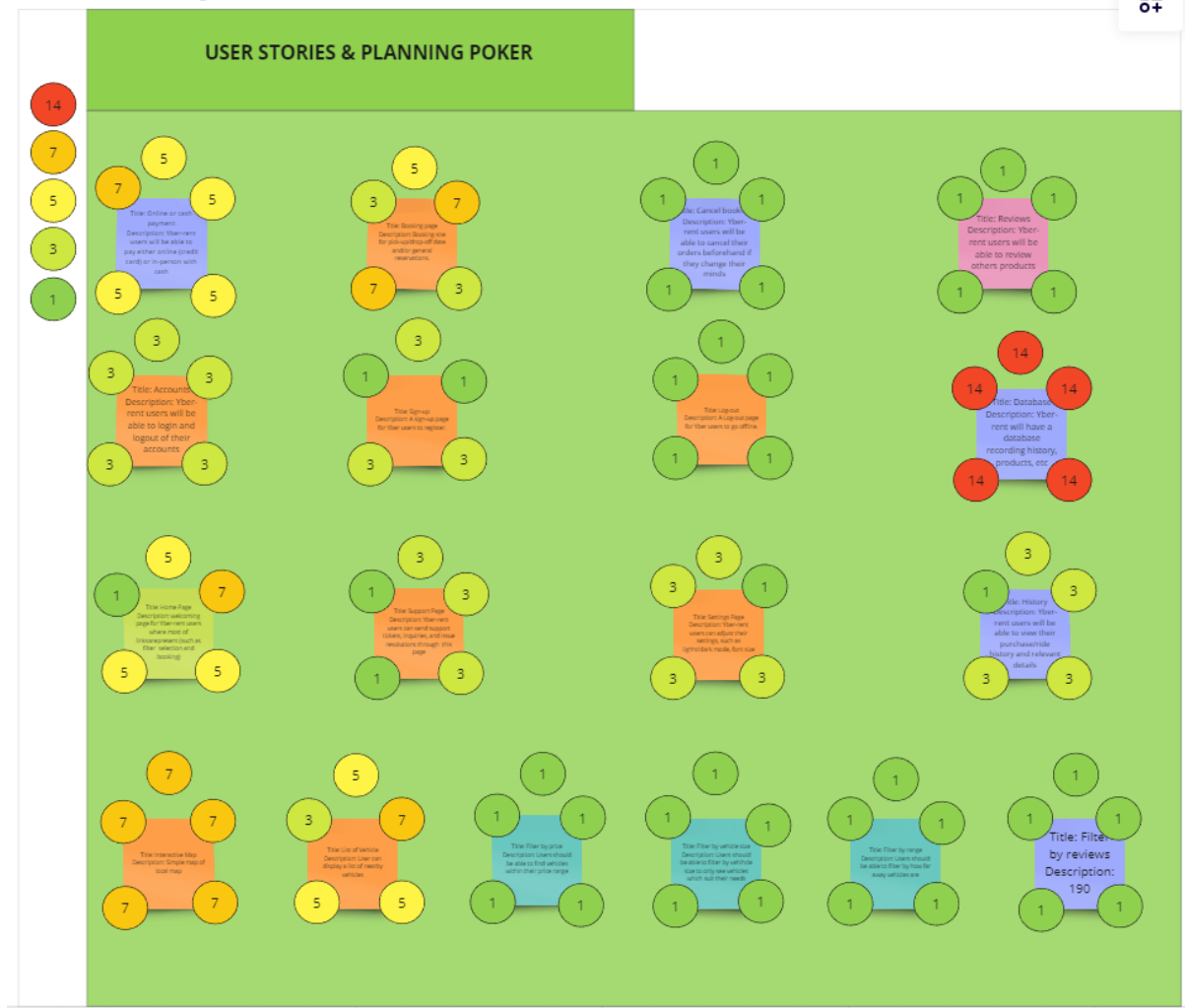
Subsequently, each member went through and allocated a token to represent their estimate of the user story length, based on their expectation of the effort required to consider it satisfactorily complete. Some examples of factors that influenced this decision include complexity, uncertainty, and the amount of work involved. Once complete, the team regrouped to discuss their opinions. If there was a unanimous estimate for any user story, it was accepted. Otherwise, a discussion occurred until a consensus was reached, and a length was officially allocated.

Prioritisation took a very similar, however less formal approach, as the limited complexity of the project meant the key items were very apparent. This process involved discussion from item to item, where features that were necessary for functionality were given the highest priority, followed by desirable additions outlined in the project brief. Any additional features that were deemed 'cosmetic' or 'unnecessary' were given the lowest priority and determined to be optional additions that would be completed if time permitted. This helped to ensure our goal of delivering what was needed on time and on budget.

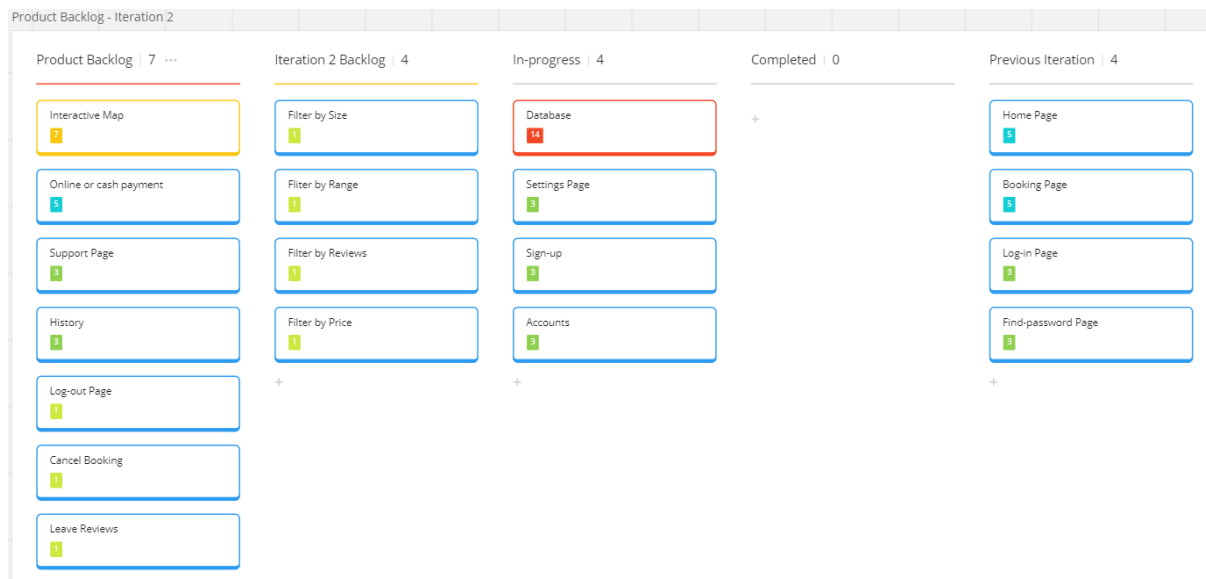
This process took place via an online interactive Miro board, which allowed users to leave comments, update their contributions to the user stories, and visualise the relationships between items quickly and effectively. Below are a series of screenshots that demonstrate the completed work, including the artifacts for User Stories, Planning Poker, and Product Backlog.

User Stories & Planning Poker

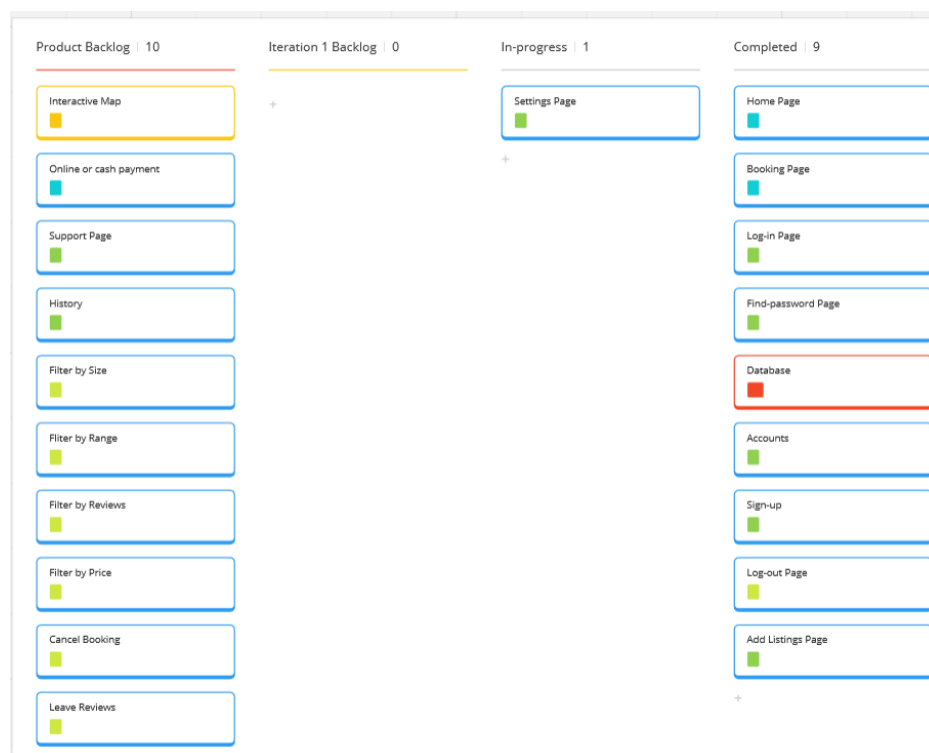
84



Finally, each fortnight a new iteration began, and a set of tasks were allocated to be completed within the designated period. For example, this is the product backlog after organising the second iteration.



It is worth noting that due to the nature of the team dynamic, and competing schedules, the iterations were not strictly followed, as it was rare that all team members had enough time to allocate to any iteration when it was originally scheduled, however regardless of this, it was always planned as such and schedules were accommodated around the work that needed to be done. One such example of our Product Owner ensuring the custom requirements were fulfilled came after iterations 1 and 2, in which inquiries as to the appropriateness of the chosen database software's style and complexity were made.



Finalised Product Backlog.

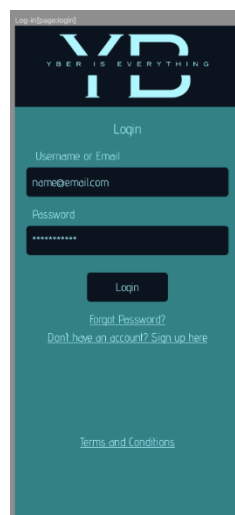
Due to the natural development of the project, the original product backlog did not remain perfectly consistent throughout the course of the project. Some items were deemed to not belong in the scope of the prototype such as the interactive map. Others such as the filters were replaced by an all-encompassing user story, which was the search bar. On top of this however, there was a series of user stories that were added that did not feature in the original list. For example, the ability for all users to add their own listings was not a feature we had considered in the beginning. Furthermore, a forgot password page was added once it was discovered that this was feasible. As expected, with the completion of each iteration, and hence the acquisition of further information about the state of the project, this list was adapted to accommodate feedback and changing requirements.

3.3. User and Client Feedback

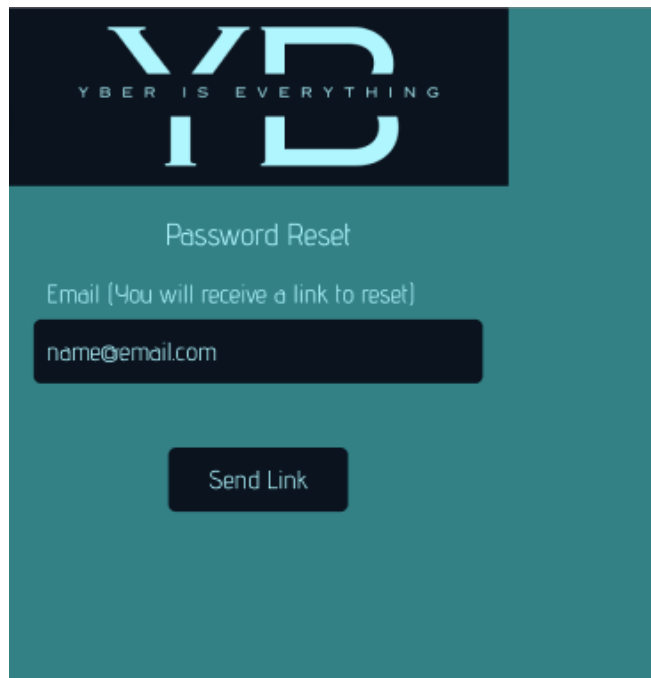
For the purposes of this report, and for the individuals who provided feedback on the application after each iteration, their identities will be kept anonymous. Furthermore, the feedback will be grouped together per iteration and formatted with bullet points. Also included are explanations and analysis on some aspects of the feedback with various screenshots to showcase what area the feedback refers to.

Iteration 1 Feedback:

1. The logo is too big and looks unprofessional.



2. It was easy to navigate it, it was very intuitive.
3. The colour scheme works very well together.
4. The pages sometimes take a while to load.
5. Some text doesn't display properly, and some sections didn't seem to be aligned properly.



Regarding Iteration Item 4. The page's occasional slow loading is entirely out of our hands. It is a byproduct of Bravo Studio and cannot be avoided. Thankfully on average the application is very quick to respond based on the users input and usually only takes a moment longer when switching pages. it is not a major issue especially as it does not inhibit the apps core functionality.

As for Item 5, some elements that did not display properly were found to be caused by incorrect element constraints. These were promptly fixed in the next iteration. A typical example of this (as seen in the image above) was that the logo was constrained vertically to the top of the screen, and horizontally to the left of the screen. With larger phone sizes this meant the logo would appear a few centimeters from the very top right corner where it was originally intended to be, leaving a gap in the background.

Aside from the few bugs the 1st iteration was a massive success, at least as far as our user testing indicated. A large portion of the app's core functionality as well as its framework was laid out and working. This can be tracked via our Product Backlog.

Iteration 2 Feedback:

It should be stated, this testing was done with different people than the first round of testers. This was done so as not to create an unconscious bias in testers who had already tested an objectively worse/smaller version of the application.

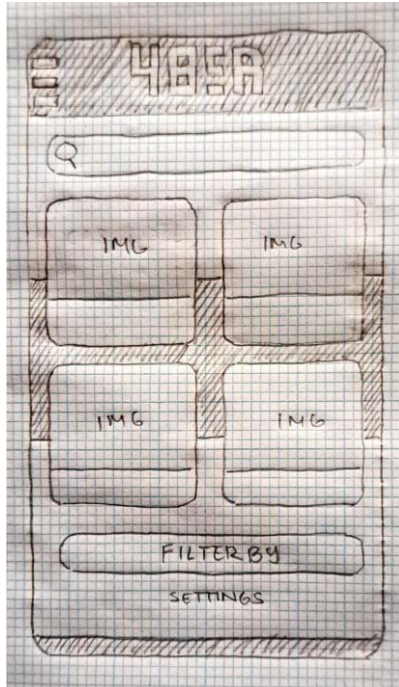
1. I can't easily go back to the home page.
2. I like how the logo is at the top right, it looks good.
3. It seemed well-put together.

Referring to Item 1, this became more prevalent of an issue as the app developed and more pages were added. Increasing past the core pages from the 1st iteration which could be easily navigated through using only the bottom menu, meant more navigation options had to be added. In the following period of development 'logout' and 'back' buttons were incorporated. For example, someone adding a new vehicle listing could now return to the previously visited page with a single action from the top left of the screen.

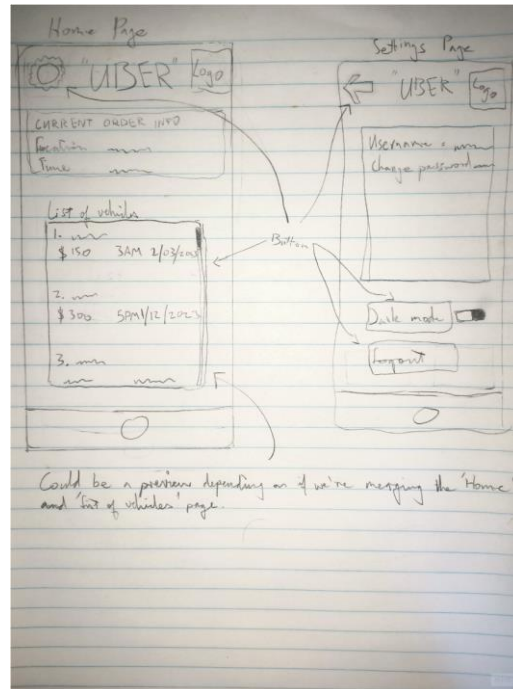
4.3. Graphical User Interface:

Initially, a series of static prototypes were developed by each team member.

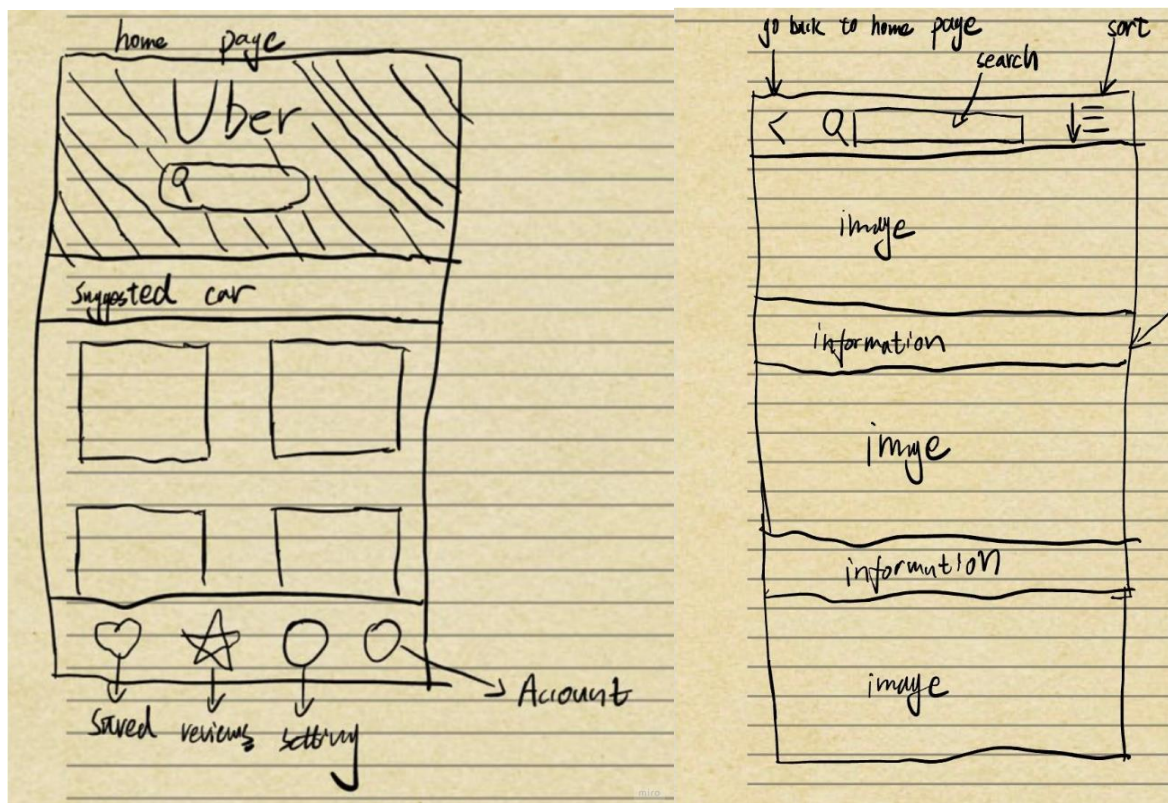
Xander



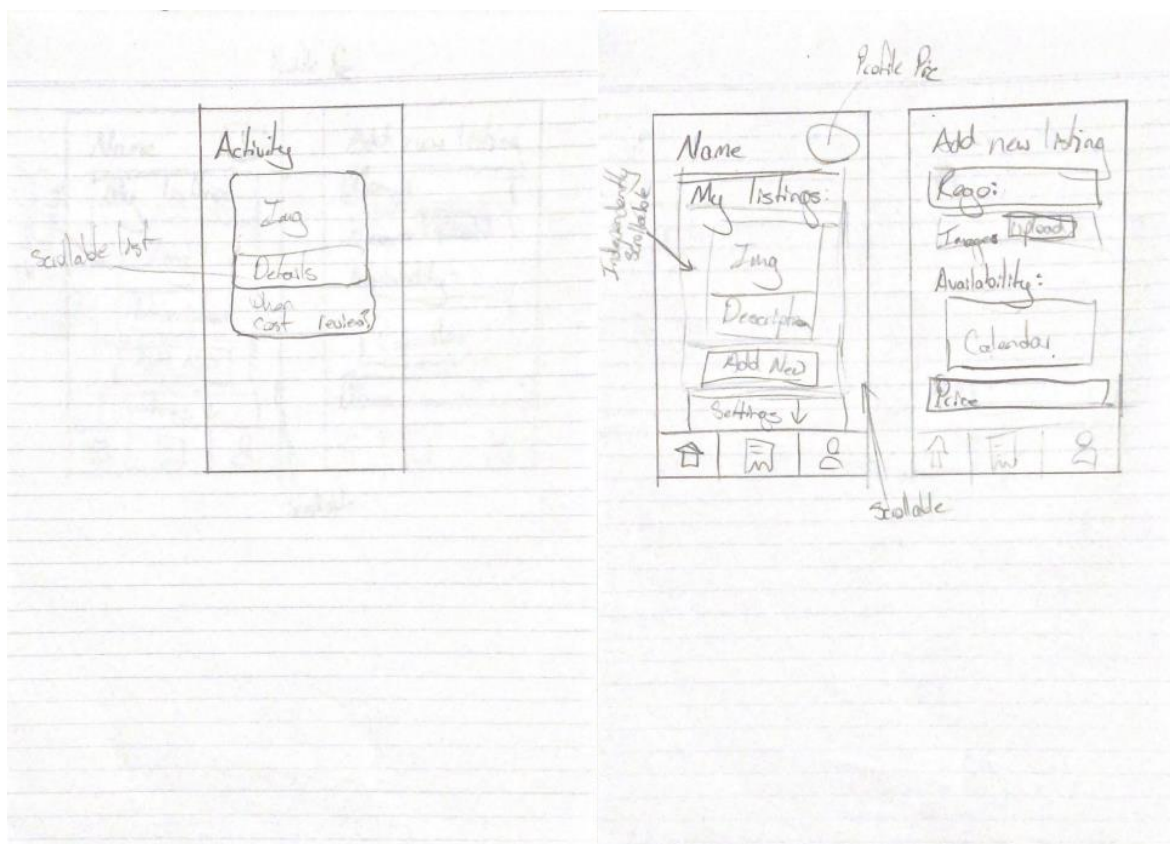
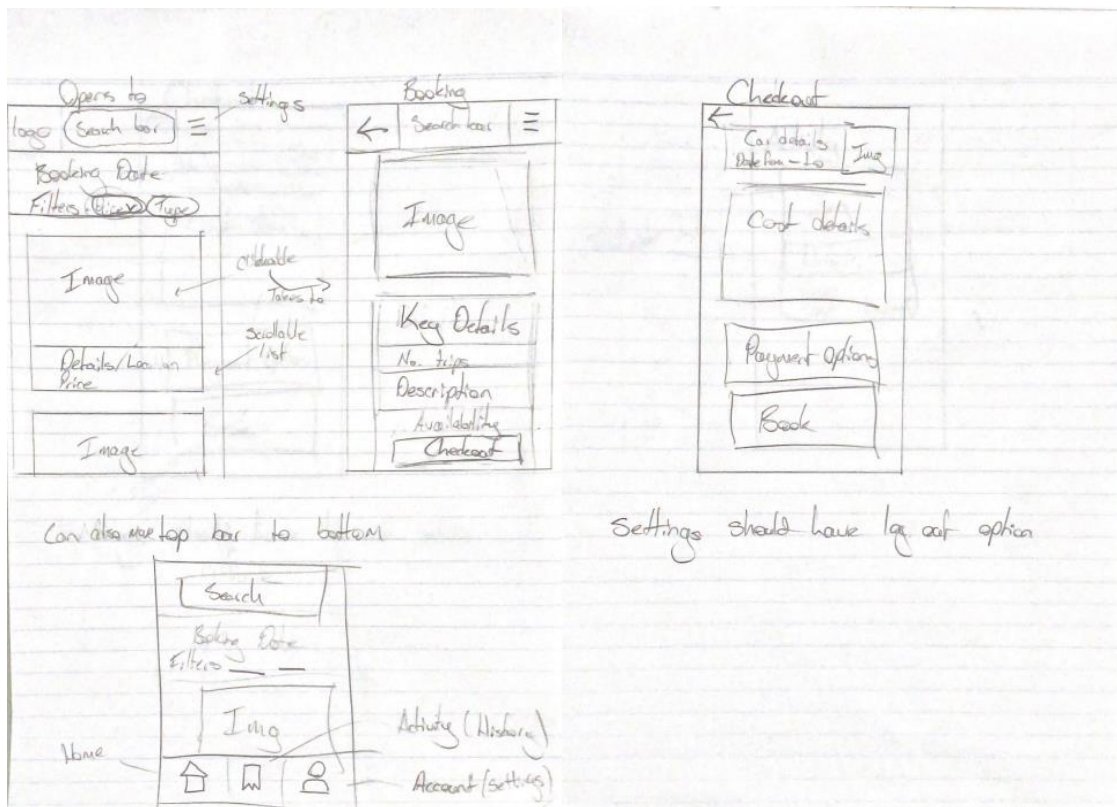
Anthony



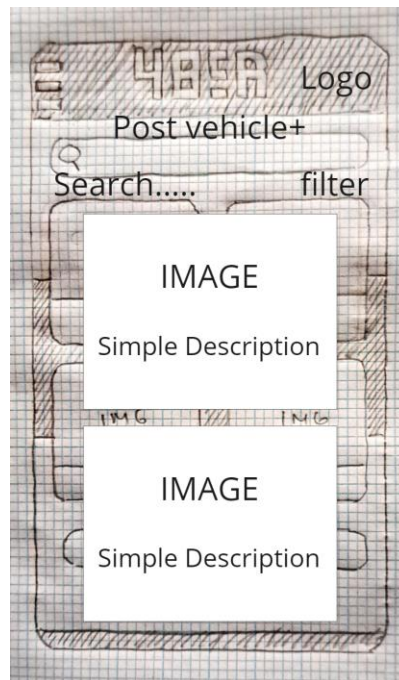
Handong



Luke



The primary purpose of these prototypes was to inform the detailed GUI design. Similar to planning poker, the team debated the style of a series of features and assets to settle on how the final app would look. These decisions were collated, and one page was designed with the accepted features.



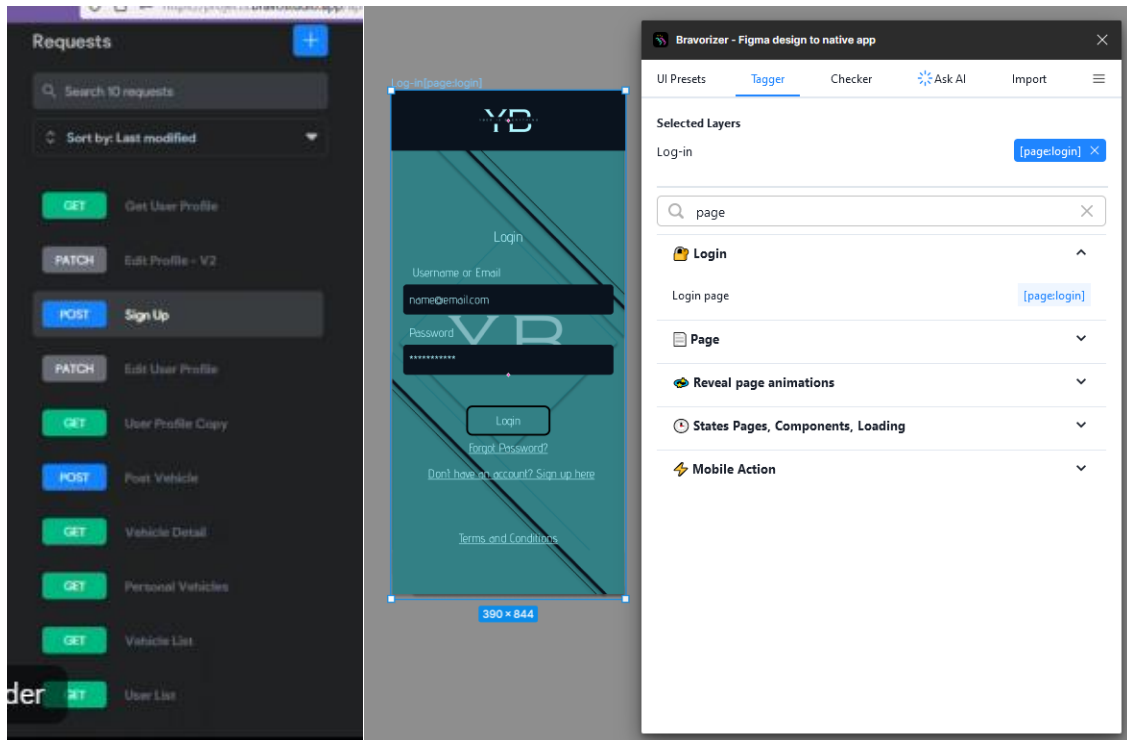
This then led to the Figma prototype which encompassed each of these design feature and decisions.



5. Implementation:

5.1. Back-End Functionality:

Back-end functionality of the application uses the in-built API functions of Bravo Studio and its associated Figma plug-in, Bravorizer. There are two main functions to ensure communications between Airtable, Bravo Studio, Figma, and Firebase. Those are the Bravorizer plug-in (tagging system) and the API Collections Tab (API request function) (refer to the image below).



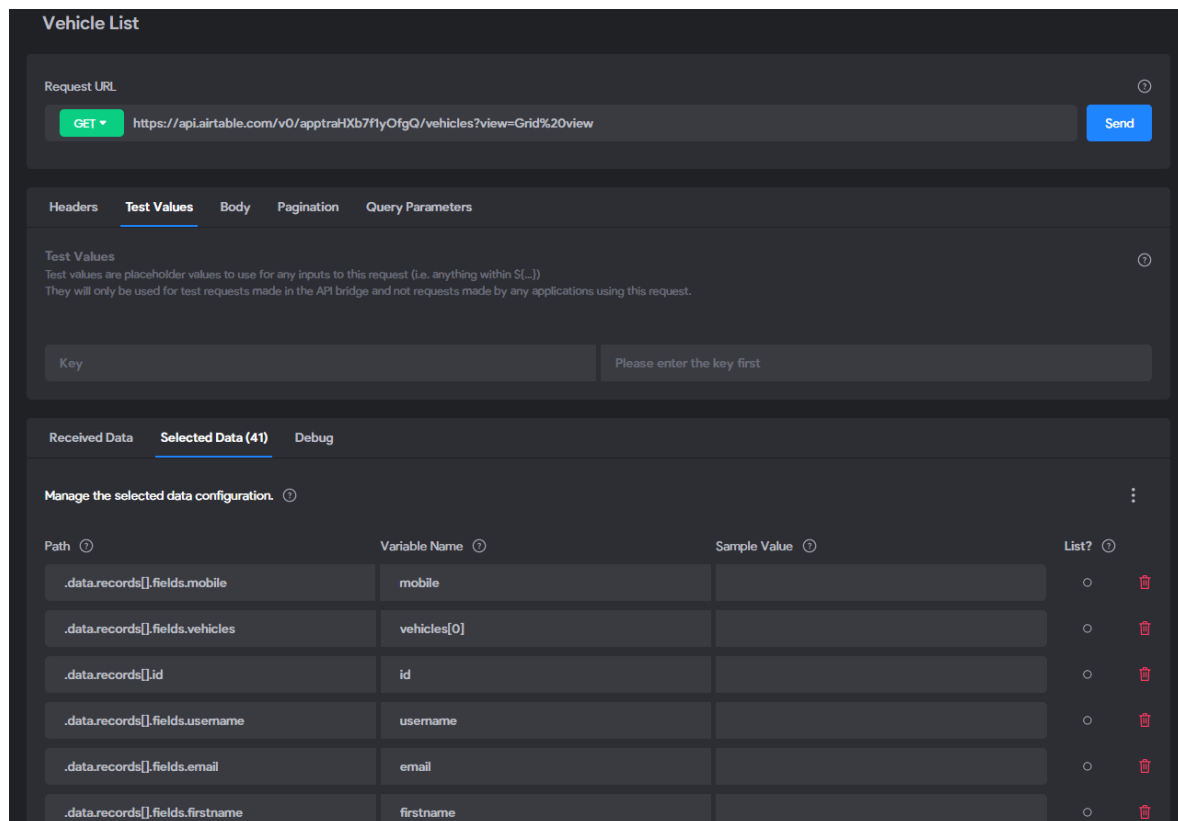
Bravo API Collections and Bravorizer plug-In

The Bravorizer plug-in functions as a tagging system that tells Bravo Studio to automatically write and build pages, features and functions based on the label given to the prototype elements in Figma. For example;

- To create a coded version of the login page or the home page, a container (div) is created when the frame layers it's related to, is labelled as "[containers]".
- Input boxes such as login details and personal information would be labelled as "[component:input-<specify data>]".
- Features such as sending data/personal information to the databases will be labelled as "[action:submit:<database>]" or simply "[action:submit]".

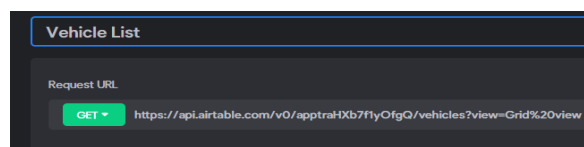
Bravorizer contains multiple tags that can be, and was, used to build most of the basic functions and features in the application. It serves as the backbone of Yber App.

API Collections Tab is where most of the back-end communications of the Yber App are created. Each request has a different purpose depending on the page it is used for. For example, the Home Page needs a list of information about vehicles from the Airtable Database, so an API GET request is created where a list of vehicles and seller's email are requested from the database.



GET request snapshot from Brav Studio

The API process works as follows (GET request for Vehicle List):

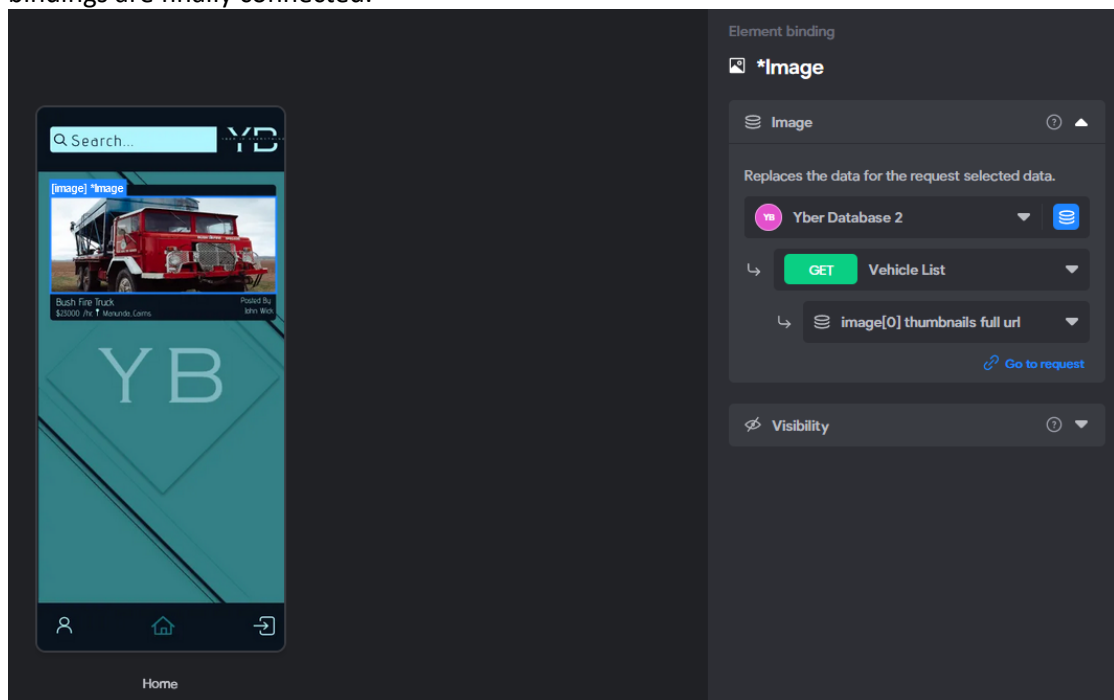


- A GET request URL is needed, which is provided by the Airtable API documentation. All of the Airtable documentation is created automatically for the database owner, thus there are only little to no modifications made to the JSON code itself.

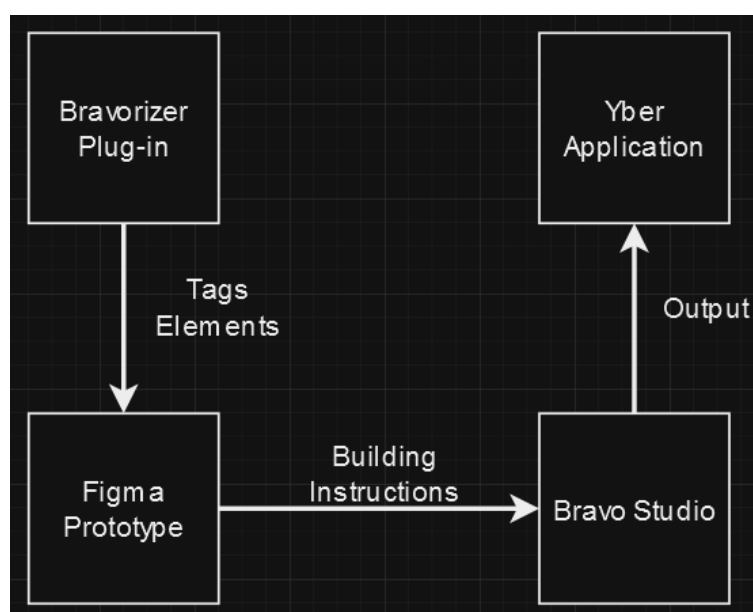
Yb	Airtable API for "Yber Database"	Open base
INTRODUCTION	List vehicles records	
METADATA	To list records in vehicles , issue a GET request to the vehicles endpoint. Note that table names and table ids can be used interchangeably. Using table ids means table name changes do not require modifications to your API request.	
RATE LIMITS	Returned records do not include any fields with "empty" values, eg. "", [] or false.	
AUTHENTICATION	You can filter, sort, and format the results with the following query parameters. Note that these parameters need to be URL encoded. You can use our API URL encoder tool to help with this. If you are using a helper library like Airtables , these parameters will be automatically encoded.	
USERS TABLE:		
FIELDS		
LIST RECORDS		
Retrieve a record		
Create records		
Update/upsert records		
Delete records		
VEHICLES TABLE		
FIELDS		
Lik records	fields Only data for fields whose names are in this list will be included in the result. If you don't need every field, you can use this parameter to reduce the amount of data transferred. <i>array of strings optional</i>	
Retrieve a record	For example, to only return data from vehicle_name and type , send these two query parameters: fields=SSR5SO-vehicle_name&fields=SR5O-type	
Create records	You can also perform the same action with field id's (they can be found in the Fields section).	
Update/upsert records	f=iLd=XSRSO=fJduKkIO6AI0BmR8&f=iLd=XSRSO=fJduTgWqAO4PZtNE	
Delete records	Note: XSRSO may be omitted when specifying multiple fields, but must always be	
ERRORS		
		curl JavaScript
		EXAMPLE REQUEST curl "https://api.airtable.com/v0/apprsBBF7jQzG/wheels?maxRecords=1&viewId=LSDtwcrw%20-%20Autosession&Bearer YOUR_ACCESS_API_TOKEN"
		EXAMPLE RESPONSE <pre>{ "records": [{ "id": "tbljjddagpbvbm", "createdTime": "2020-08-10T03:28:23.000Z", "fields": { "image": { "id": "tblsoo680OCapewr", "link": null, "width": 1000, "height": 1000, "url": "https://airtableusercontent.com/v1/22/149893080000/cwBTU-XcCQX388mgDQ/L6MNMN_uNBtl-OVreqlOTGLTOLCWldMLn7lEStNeNgldOd_nbfI93SC-/ccCTFrncxThnmMngdlAgdsyGrlTrpqpy-qgiExms33qgwCdpUr_PwMaVeWegTghysNNlStralntDTGHta/" }, "modelName": "Porsche Panier-Sanit-Jon-Dietrich.jpg", "year": 1988888, "type": "Image/jpeg", "thumbnailUrl": "https://airtableusercontent.com/v1/22/149893080000/twl?url=https://vs-airtableusercontent.com/v1/22/149893080000/Zjsabed4VybqtqrTw/qmwScL_yah_-1lt8YpldtITshpfDt23NmtlUpptglLfytq124ts-QtiSeWofdgmpflmfSGPfwrrenPhblcmBNatEd6AGlyBgOpdwpeWsd6e" } }]}</pre>

API Documentation and JSON body provided automatically.

- The Airtable then accepts the request and sends a test record to the “Received Data Tab”. This is where a sample record is displayed to ensure that communication between Airtable and API Collection is alive. If there’s an error or there is no data sent/received, the tab will display an error message.
- Different types of data are listed which correspond to different fields that exist within the database. Then, specific information is defined in the “Selected Data Tab” and selected in the “Received Data Tab”. For example, to retrieve information regarding a vehicle name, the path name “.data.records[].fields.vehicle_name” is defined in both tabs.
- After all the required data are selected, the GET request is retested again to check if all data are properly received.
- The GET request is then used in the element interactions in Bravo Studio where the API bindings are finally connected.



GET request being used in API bindings



Simple diagram of the process

In short, the Bravorizer Plugins is in charge of building the necessary codes and functions automatically, by labeling/tagging elements in Figma, which in turn send instructions to Bravo Studio to write and create the actual application.

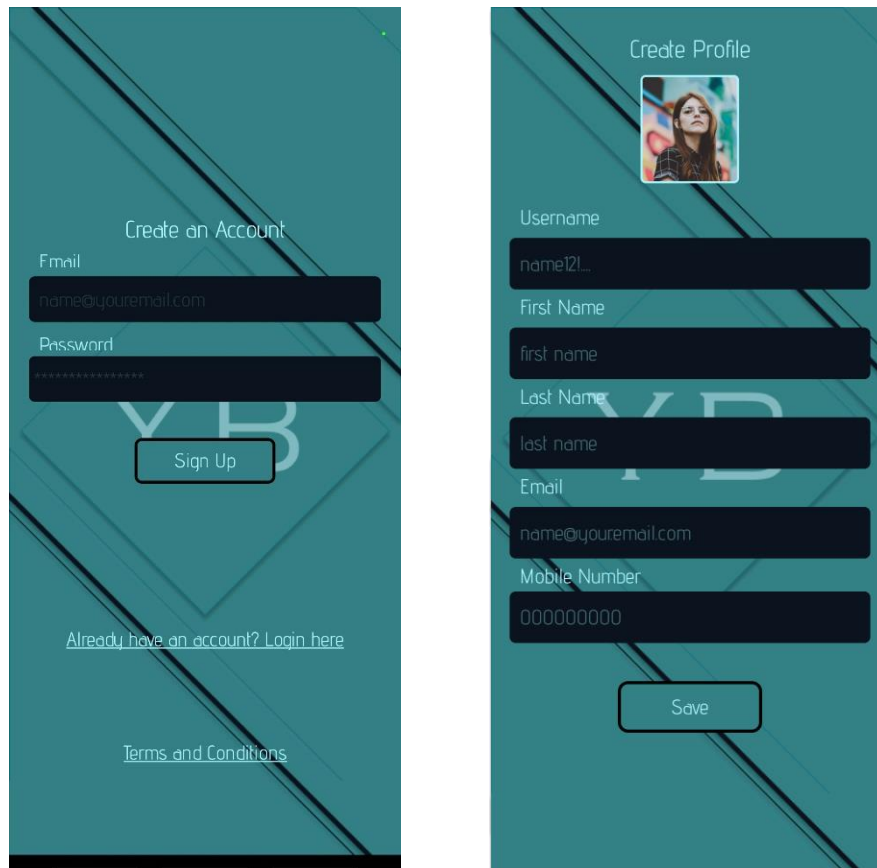
On the other hand, "API Collection Tab" has the necessary API requests to send and retrieve information from the database which is used for dynamic data e.g., listing different vehicles in the Home Page.

However, there are few key functions that are needed to be mentioned:

- First, there are pages that do not need navigational input in Figma that are required for Bravo Studio but rather, are instructed to create the code through the Bravorizer plug-in. For example, the Log-in page does not need to have navigation interaction in Figma and is handled by Bravo Studio instead.
- "GET", "POST", and "PATCH" requests are processed in similar ways, but they ultimately have different ways of calling and interacting with Airtable.
- Listing items and displaying the item information are different, thus requiring two different GET requests.
- The tags are extremely case sensitive and require careful planning. Each tag also requires different levels of hierarchy when using them in Figma. For example, the tag [Container] can only be used at the top-most level of the frame layer it belongs to.
- Airtable and Firebase do not communicate with each other. The Firebase database is only used for authentication purposes and handled directly by the tag [action:login:firebase-email-password], thus not requiring any API request. It uses the inputs from the user and checks if they have their credentials registered. If not, it will display an error.
- Since the Firebase only handles authentication, it was required to create an On-Boarding page where a user fills in their personal information a second time. The API "POST" request handles the process of sending that information to Airtable.

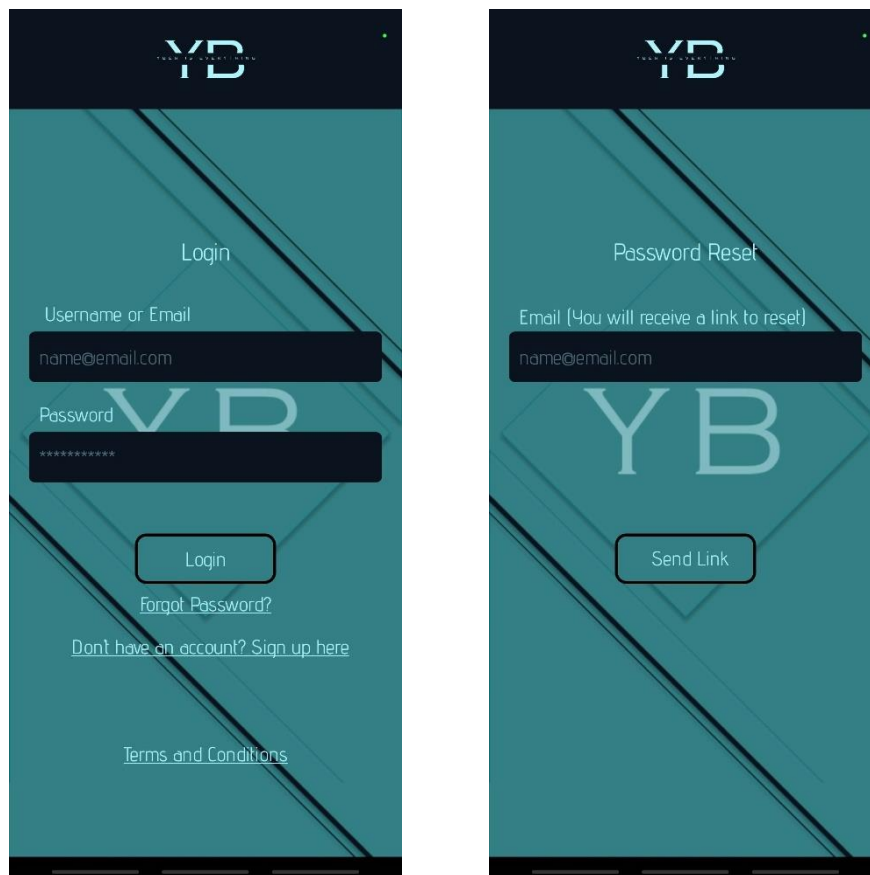
5.2. Front-End Functionality:

The final deliverable product is a simple and intuitive app, that effectively delivers what is needed, on time, and on budget. Initially opening the app greets the user with the sign-up page, where customers are prompted to enter a username and password to access the service. At this stage, the entered information is sent through to be registered with Firebase's authentication service. Presuming a unique email has been entered, the subsequent create profile page is presented to the user.

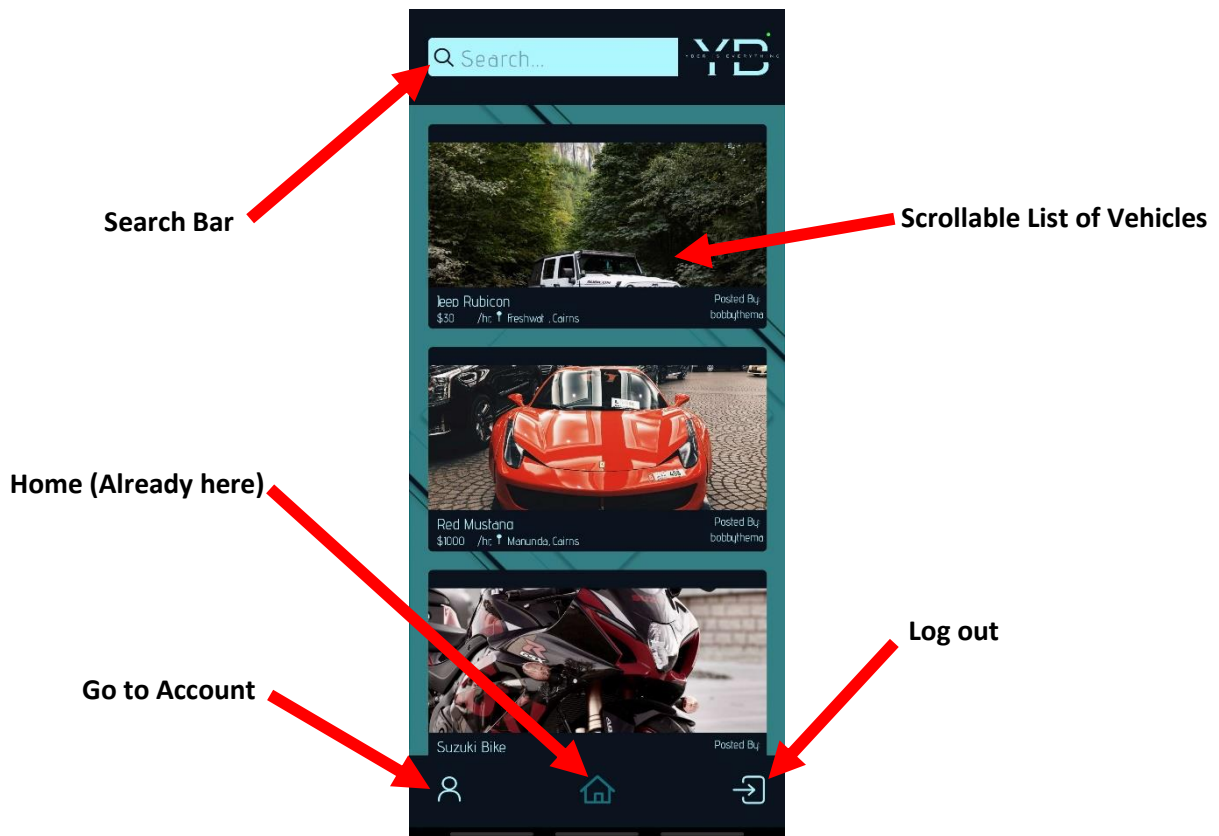


The image displays two screenshots of a mobile application interface. The left screenshot, titled 'Create an Account', features a teal background with a large, faint 'YBER' watermark. It contains input fields for 'Email' (with the placeholder 'name@youremail.com') and 'Password' (with a masked input 'xxxxxxxxxx'). A 'Sign Up' button is positioned below these fields. At the bottom, there are links for 'Already have an account? Login here' and 'Terms and Conditions'. The right screenshot, titled 'Create Profile', also has a teal background with the 'YBER' watermark. It includes a profile picture placeholder at the top. Below it are input fields for 'Username' (placeholder 'name121...'), 'First Name' (placeholder 'first name'), 'Last Name' (placeholder 'last name'), 'Email' (placeholder 'name@youremail.com'), and 'Mobile Number' (placeholder '000000000'). A 'Save' button is located at the bottom of the form.

The second screen is used to create an Airtable user and gather all the information that would be required to be able to post and purchase listings. As of the development of this project, there is no way to directly communicate between Firebase and Airtable, meaning the user must input their email a second time on the 'Create Profile' screen. This leaves potential for error that should be improved upon, when possible, as if these two prompts do not match, it could lead to significant errors regarding both peer-to-peer communication, and that between 'Yber' and its users. Once sign up is complete, any subsequent login attempts will present the user with the following screen.

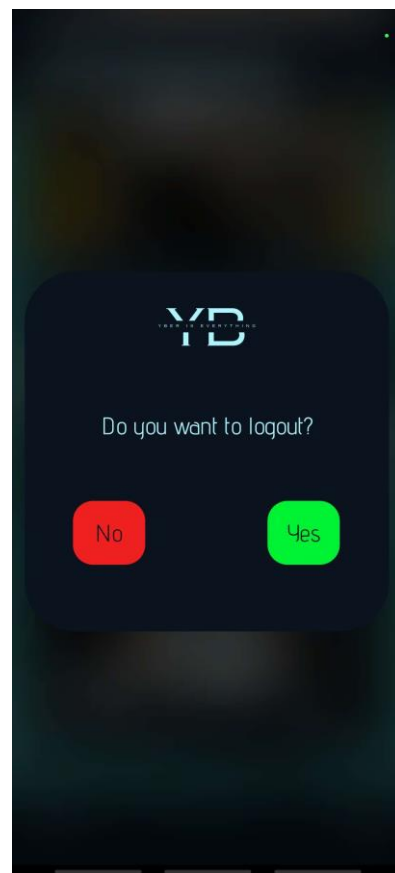
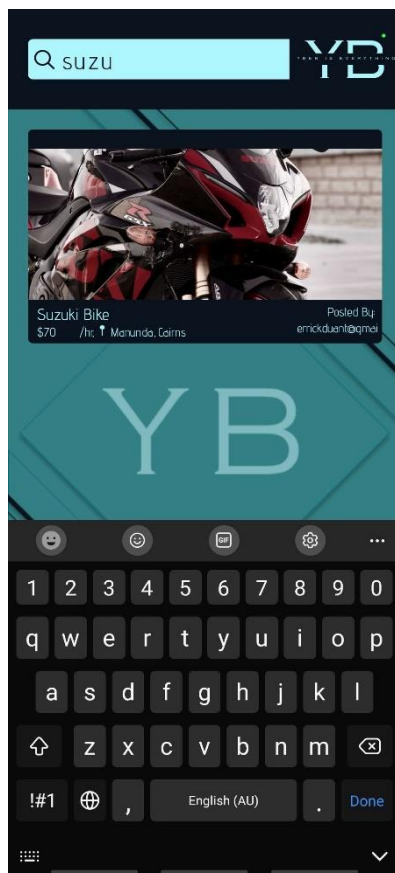


The app is intelligent enough to allow users to reset their passwords thanks to the implementation of Firebase. Users will be sent an email to reset their password if their email exists within the database and linked to the appropriate location to change their password. Once logged in, the home page begins to display any listings that are available from the Airtable.



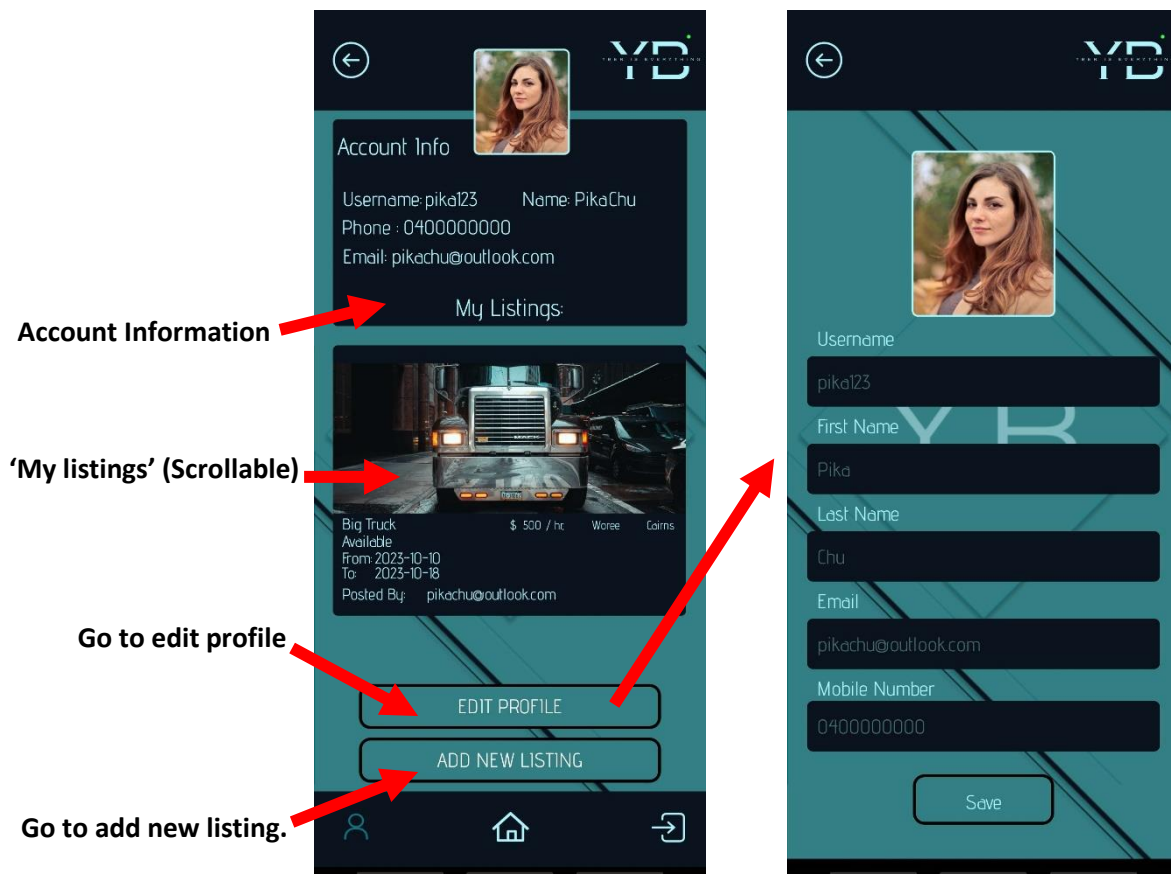
Here the user is presented with a scrollable list of vehicles they can view, complete with a summary of the name, hourly price, location, and owner. Each of these is a clickable button, which leads to a more detailed explanation of the listing. The bottom of the screen is the Navbar which serves two primary purposes. First of all, it allows the user to quickly and easily jump between their own profile, the home page, and logging out. Furthermore, however it helps new users understand where they are in the app if they were to get lost, as your current location is clearly dimmed down, relative to the other two.

The top of the screen is a search bar, which replaced the user stories for different filters, as it allows rapid dynamic filtering of the viewable listings. Future prototypes should trial the possibility of a combination of both the search bar and strict filters. Pressing the logout button brings up this prompt, which gives the user a chance to avoid having to log back in if they did not mean to press it.



The accounts page is also multifunctional. It first serves as a point of reference where users can see, confirm and theoretically edit their personal information. Unfortunately, as previously discussed, the functionality for patch requests proved stubborn, and was not able to be completed within the allocated timeframe. As a result, the account information is set once the account is created, an issue which should be addressed in later productions.

The second purpose of the accounts page is to view and add your own listings. As 'Yber' is meant to be a peer-to-peer rental service, users must be able to list their own vehicles to rent. Listings which you have uploaded before are seen here, and there is another option to add another via the bottom button.

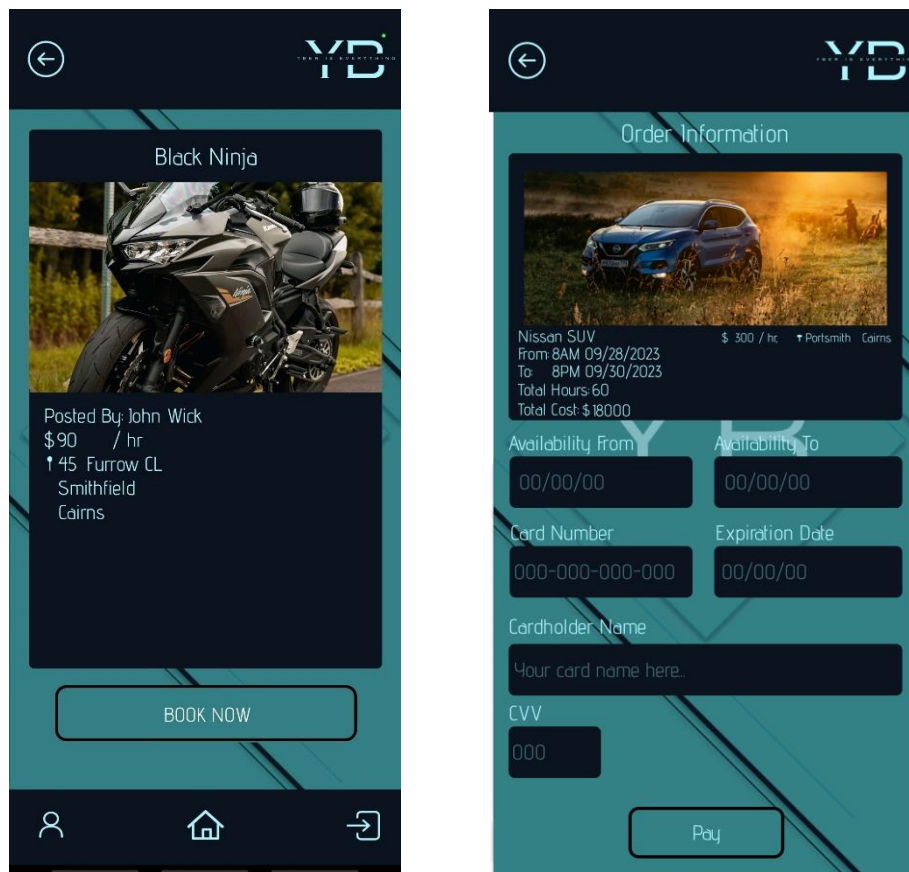


Adding a new listing requires the above information. Photos can be uploaded directly from your camera roll, and dates are selected by a pop-up calendar.



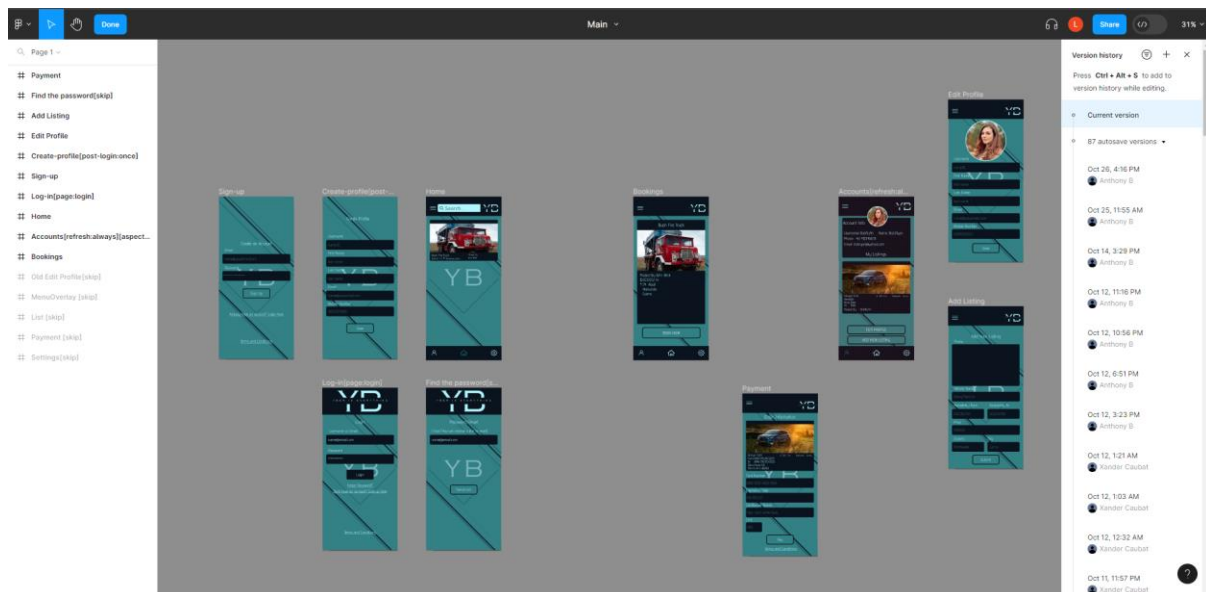
The screenshot shows a mobile application interface for adding a new listing. At the top, there is a dark blue header with a back arrow icon on the left and the 'YD' logo on the right. Below the header, the title 'Add Your Listing' is centered. The form consists of several input fields: a 'Photo' field with a placeholder image of a red truck; a 'Vehicle Name' field; a 'Brand/Version' field; two date pickers for 'Availability From' and 'Availability To', both showing '00/00/00'; a 'Price' field showing '000.00'; and two dropdown menus for 'Suburb' (showing 'Freshwater') and 'City' (showing 'Coims'). A 'Submit' button is located at the bottom of the form.

Finally, as stated earlier, clicking any of the listings on the home page will take you to its booking page. Here we can see a detailed summary of the vehicle, including the full resolution image, the owner, the location and price. The user can select when they wish to rent the vehicle to and from and proceed with their order. Clicking the book now button takes the user through to the payment screen. A summary of their order is visible at the top of the screen, detailing the date to and from, as well as the number of hours it is being booked for, the rate and the total cost. Below this is space for the customers card information and then a 'Pay' button. In this current prototype, this does not have any functionality as it exceeded the project scope, however the fundamental concept is present.

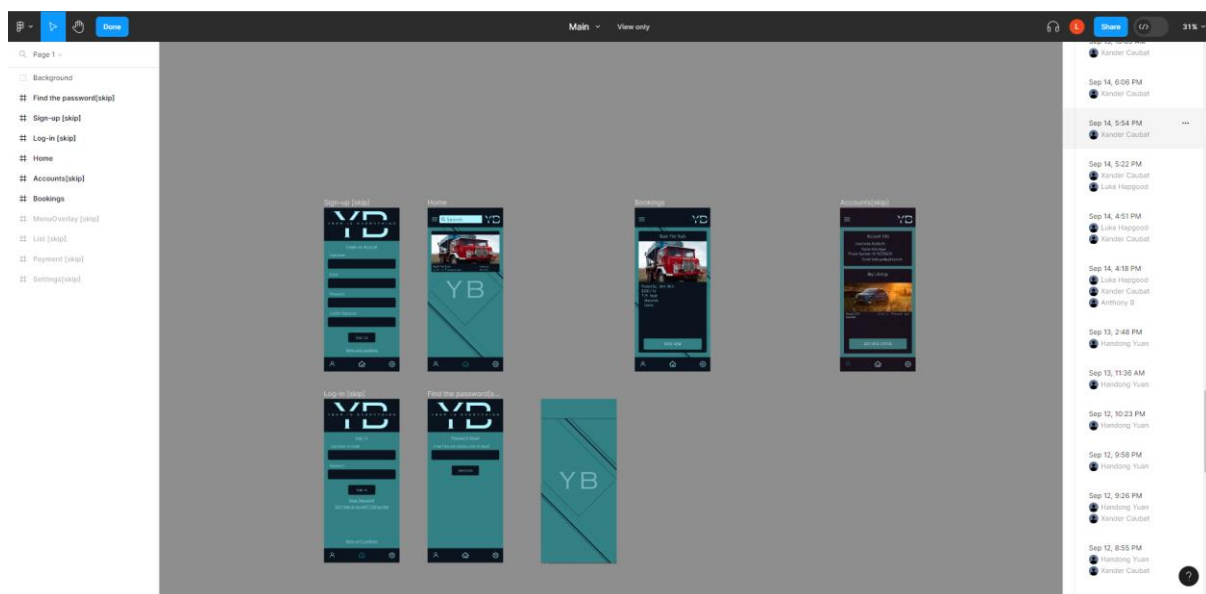


5.3. Version Control:

Figma automatically maintains a version history of design files that can be accessed by clicking on the "Version History" button in the Figma interface. This allows you to see a list of past versions, including the date and time of each save. Practically speaking, this looked as follows:



Here we can see one version, as of October 27th. The right-hand side of the screen shows the recent edits and their authors, any of which can be selected to instantly return the prototype to a previous state. For example,

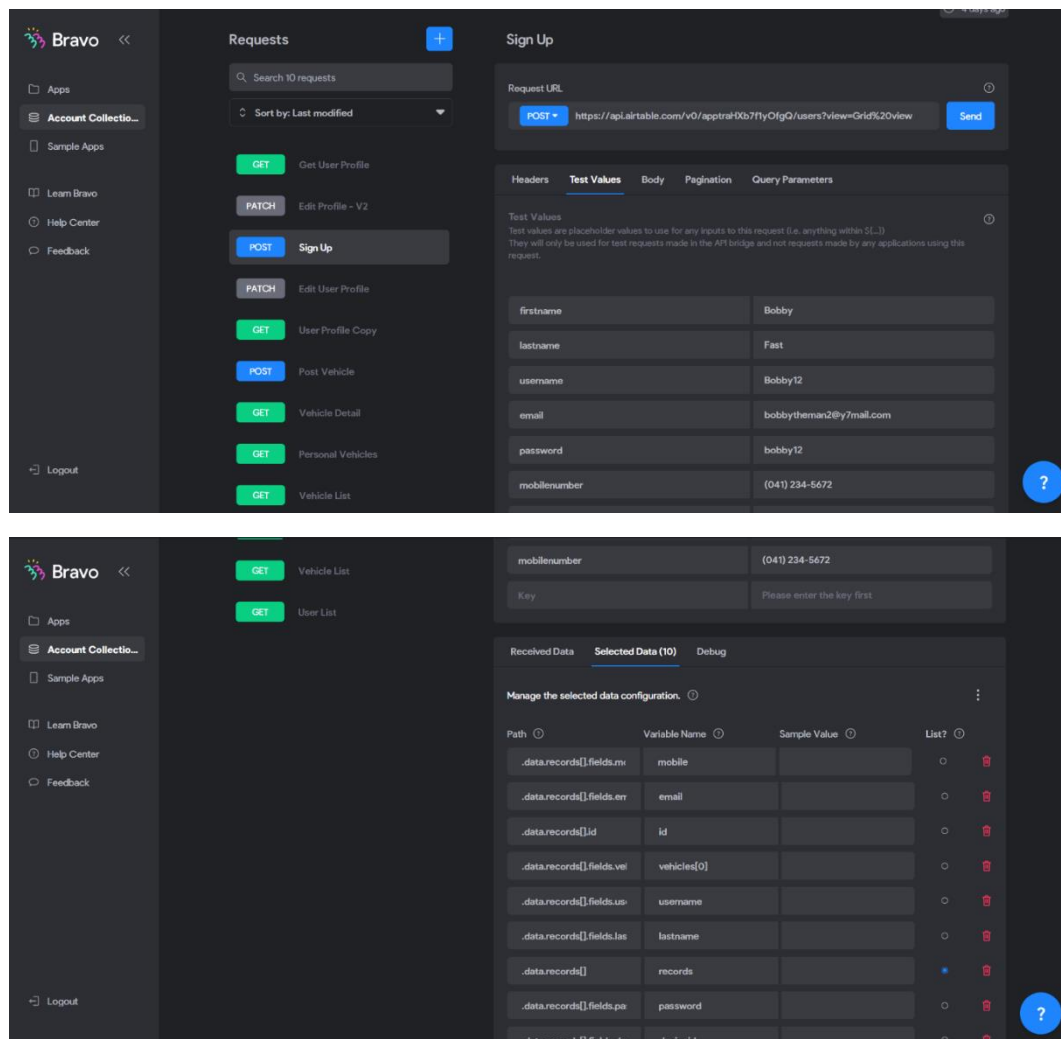


This screenshot shows an earlier state from September 14th, which was the conclusion of our teams first major iteration. Clearly many decisions had yet to be finalised, however the group had begun to collate a coherent prototype. This tool was highly effective throughout the development process as it allowed users to keep track and understand each other's changes between sessions, without requiring a formal meeting for a complete explanation. This was found to be highly beneficial due to the nature of our remote team. We also made use of google docs for documentation and its version control, and Airtable also has the capacity for static data snapshots, which would likely become useful as the project expands, although they were not utilised as of this stage.

5.4. Testing

User testing was a primary method of validation and verification employed by our development team. It played a crucial role in ensuring that the application we were building not only met the technical specifications but also addressed the real-world needs and expectations of our end-users. Throughout the Agile development process, we integrated user testing as an ongoing practice to continuously validate and verify our work. At the heart of our user testing strategy was the pursuit of genuine and unfiltered user feedback. External parties were invited to interact with the product after each iteration. Initially, we intentionally limited our guidance and input during these testing sessions to gauge raw and honest reactions. This approach allowed us to capture the immediate user experience and their initial impressions.

Following the initial raw reactions, we transitioned to guided testing sessions. The purpose here was to dig deeper into the user experience, uncover pain points, and identify areas for improvement. During these sessions, we provided users with some guidance, but we did so carefully to maintain objectivity. We aimed to distinguish between user difficulties stemming from a lack of familiarity with the tools and those resulting from the application's inherent usability or functionality issues. The feedback received during user testing was integral to our Agile development process. It created an iterative feedback loop, enabling us to make informed decisions about the project's direction. It allowed us to identify and prioritize enhancements, bug fixes, and new features, all driven by real-world user needs and expectations.



One of the pivotal aspects of our application was its ability to communicate effectively with the database. Continuous testing was imperative to ensure that any changes we made to the application did not negatively impact this critical component. Bravo Studio provided a robust set of functions that facilitated continuous and rigorous testing throughout the project's lifecycle. Our application's success hinged on maintaining data integrity and ensuring that user interactions were accurately reflected in the database. Continuous testing was necessary to identify any discrepancies, data corruption, or anomalies that could disrupt the user experience. The continuous testing capabilities in Bravo Studio were aligned with Agile principles of adaptability and responsiveness. It provided us with the flexibility to swiftly respond to changes, iterate on functionality, and maintain a high level of confidence in the application's database communication.

5.5. Future Improvements:

The limitations of the project scope leave room for improvement, moving forward into the beta and initial launch of the app. First and foremost, the functionality associated with payments, while conceptually intact, remains non-functional within the prototype. In future iterations, a plausible approach is to integrate the payment screen as a third-party application. This strategic choice is rooted in the necessity of adhering to stringent security standards and safeguarding users' sensitive card data, a challenge that the current project does not accommodate.

Another notable consideration revolves around the designation of the primary key within the Airtable. Presently, the primary key is assigned based on the vehicle's name, a decision that may not be optimal for several reasons. The use of registration numbers (Rego numbers) as an alternative primary key is also infeasible due to potential conflicts arising from variations across different states. Consequently, to ensure the uniqueness of each vehicle listing, a vehicle ID should be generated. Listings then require verification to prevent users from listing the same vehicle multiple times.

Furthermore, the homepage could undergo several improvements. First of all, users should initially be presented with vehicles situated within their geographical proximity. This feature becomes especially pertinent when dealing with databases housing a considerable volume of data, potentially numbering in the tens or hundreds of thousands. In such cases, presenting the entirety of the dataset to the user would prove impractical and overwhelming. Additionally, one user story mentioned the possibility of searching for vehicles with an interactable map. This feature could significantly improve the user experience, however, was abandoned due to technical complexity and time constraints. A simple polished deliverable was considered preferable to an incomplete complicated one.

The Edit Profile page and its API PATCH request bugs are probably the hardest bugs that the team has encountered. The API Patch request logs state that all send and retrieve processes are normal and there are no errors. However, when interacting with the page itself, it can only display personal information but could not modify nor send the changes to the database. Unfortunately, the team was unable to determine what the cause of this error was in the allocated time. Future prototypes should work to correct this error.

6. Appendix 1 – Rubric:

CRITERIA	HD (9-10)	DISTINCTION (7-8)	CREDIT (5-6)	PASS (3-4)	FAIL (0-2)
REQUIREMENTS	All requirements (user stories) are correct and correctly estimated with justified priorities. An exemplary set of features are correctly planned for implementation in justified order and within given budget. Your IT solution is planning to deliver “what is needed, on time and on budget”	Few minor errors	Many minor errors	Few major errors	Many major errors
DESIGN	Architectural, database, and user interface designs are exemplary and justified. Help: textbook chapter 5. Add a page to your GitHub to explain the design of all major components				
IMPLEMENTATION / CODE	Your IT solution delivered “what is needed, on time and on budget” for each iteration. Demonstration of client feedback on your deployed solution after each iteration. Exemplary UI, database and deployment choices. Add a page to your GitHub to illustrate/promote your delivered solution.				
TEST	Exemplary testing of all components. Test-driven development. Acceptance testing of all delivered features. Appropriate testing data sets. Delivered implementation matches the planning. Help: Chapters 7, 8 and 9. Add a page to your GitHub to explain your testing				
VERSION CONTROL	Exemplary use of GitHub/git or equivalent.				
BUILDING AND DEVELOPMENT TOOLS	Exemplary use of software development tools, building tools and external libraries. Add a page to your GitHub to explain what and how you used it.				
AGILE SOFTWARE ENGINEERING	Exemplary application of agile iterative development as per textbook.				
PROJECT TECHNICAL WRITING	Exemplary technical writing in your project GitHub pages.				