Module Code	CT100-3-3-AMCA
Intake Code	APD3F2402IT(MBT)
Lecturer Name	Mr. Amad Arshad
Hand-out Date	26 February 2024
Hand-in Date	9 June 2024

Student TP Number	TP059963
Student Name	Yip Zi Xian

Table of Contents

1. Introduction

This documentation provides a comprehensive overview of the development and functionality of a shipping service provider application named HaulEase. This project showcases the practical application of mobile backend systems using Amazon Web Services (AWS) and modern software development with Kotlin Jetpack Compose. The main objective of this document is to provide the assumptions, design choices, testing procedures, and future enhancements considered during the development of HaulEase.

This application aims to streamline the shipment process for users, offering functionalities such as shipment creation, cargo management, and location tracking. However, several assumptions and limitations have been noted and are discussed within this document. Use case diagram are designed as visual representations of the functionalities from the perspective of two different roles, users and admin whereas entity relationship diagram provides the data models and relationships used to structure the application's database on AWS Relational Database Service (RDS). User manual and automated test report are also included as guidance and testing phase of the application, respectively. Lastly, a short research discussion on mobile backend systems is conducted to explore the history, architectural evolution and current trends.

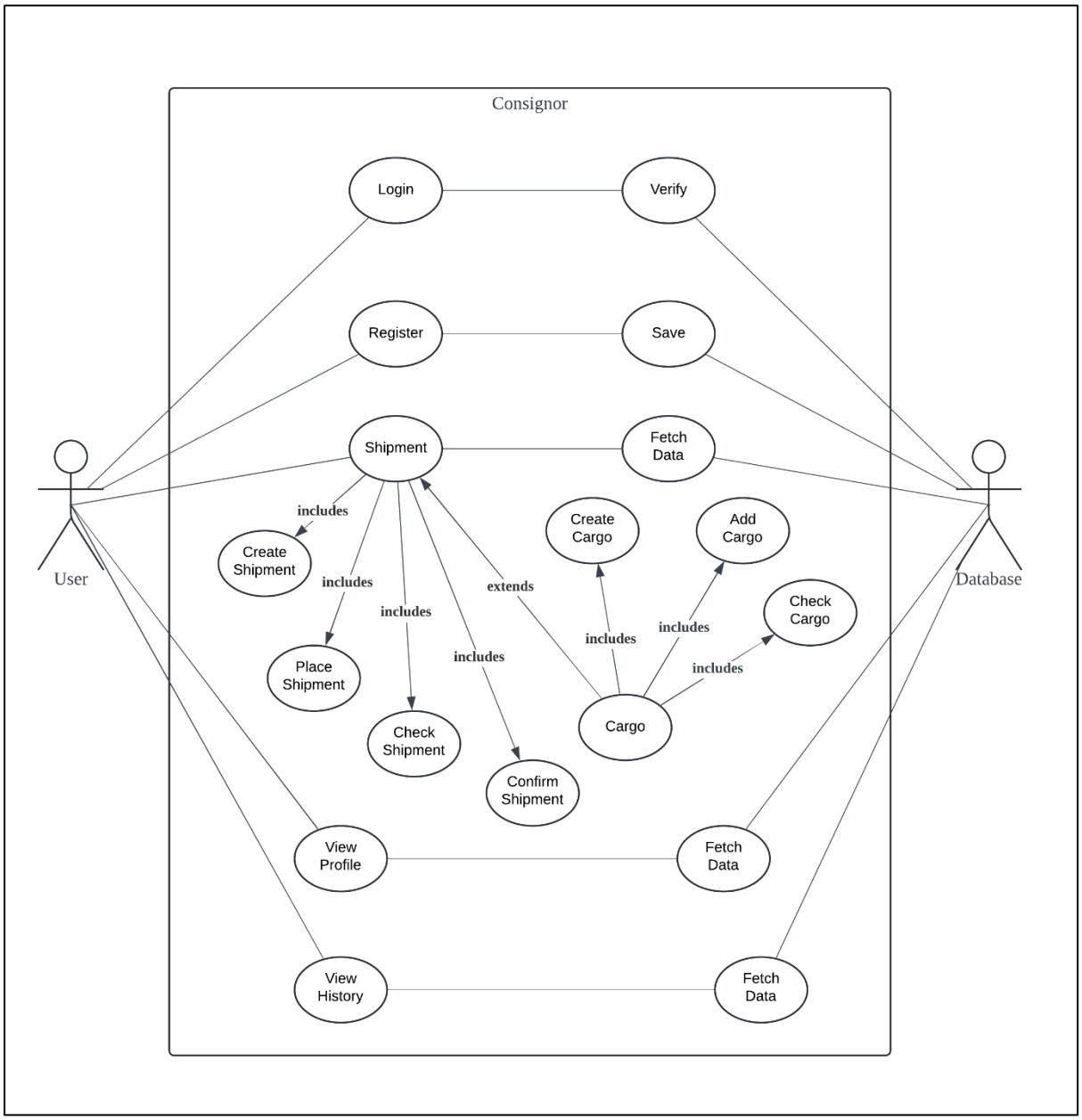
2. Assumptions

There are several assumptions made during the development of HaulEase mobile application because of time constraints and limited resources:

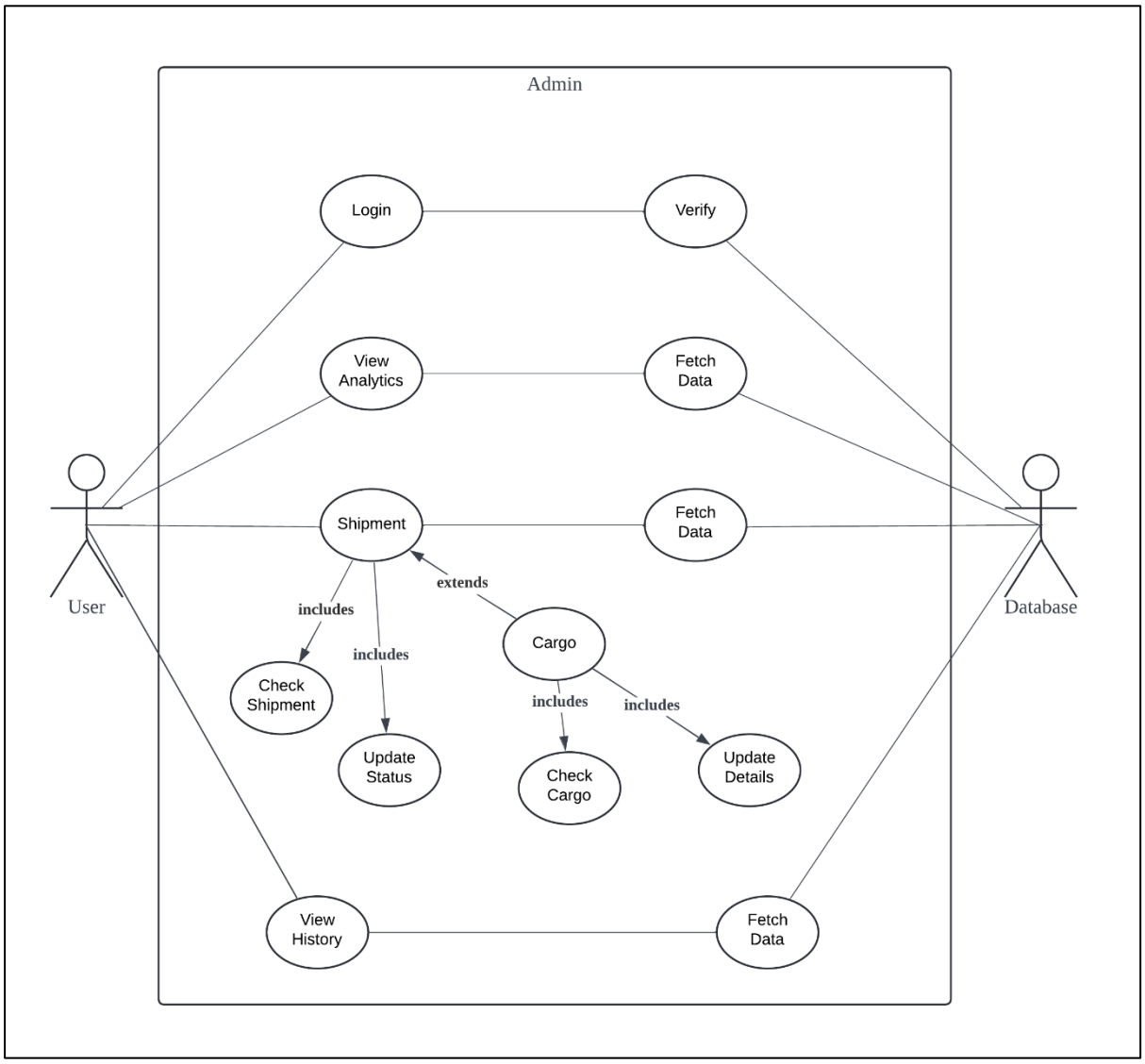
- Users are assumed to use an existing email address as it is required to reset their password.
- Users are assumed to not be able to edit their profile to ensure their respective shipment details remain the same throughout the shipping and delivery process.
- Users are assumed to refresh their page if they did not see any changes after performing an action like creating shipment or removing cargo.
- The tracking of shipment location is only a simulation that the function is doable and runnable.

3. Use Case Diagram

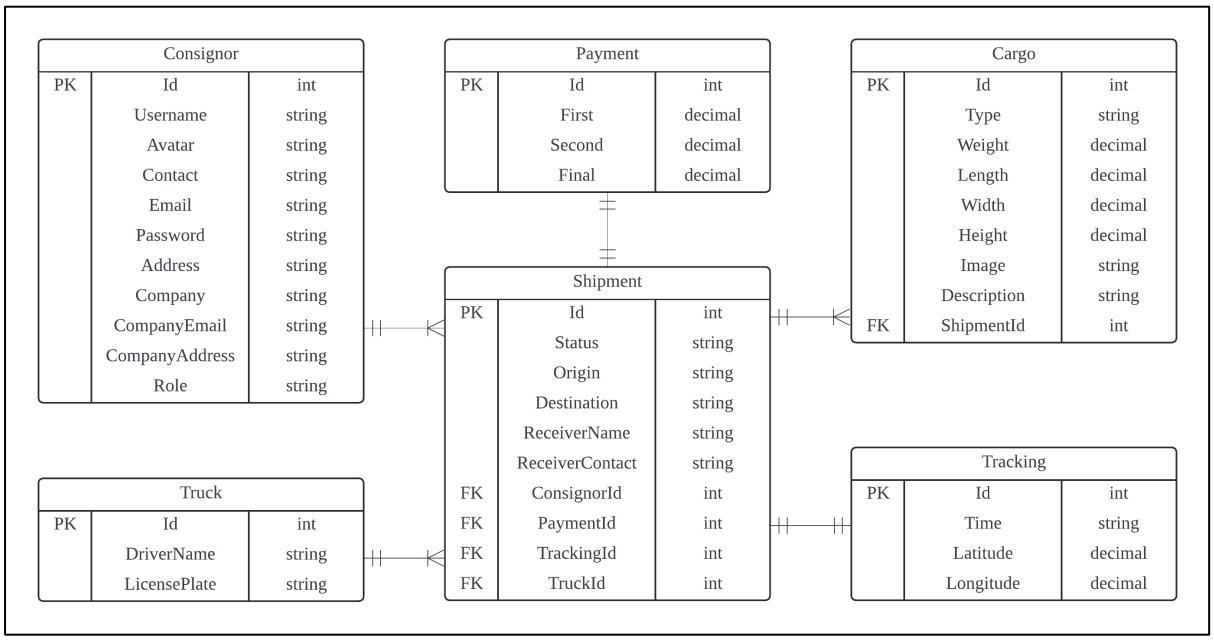
3.1. Consignor Use Case Diagram



3.2. Admin Use Case Diagram



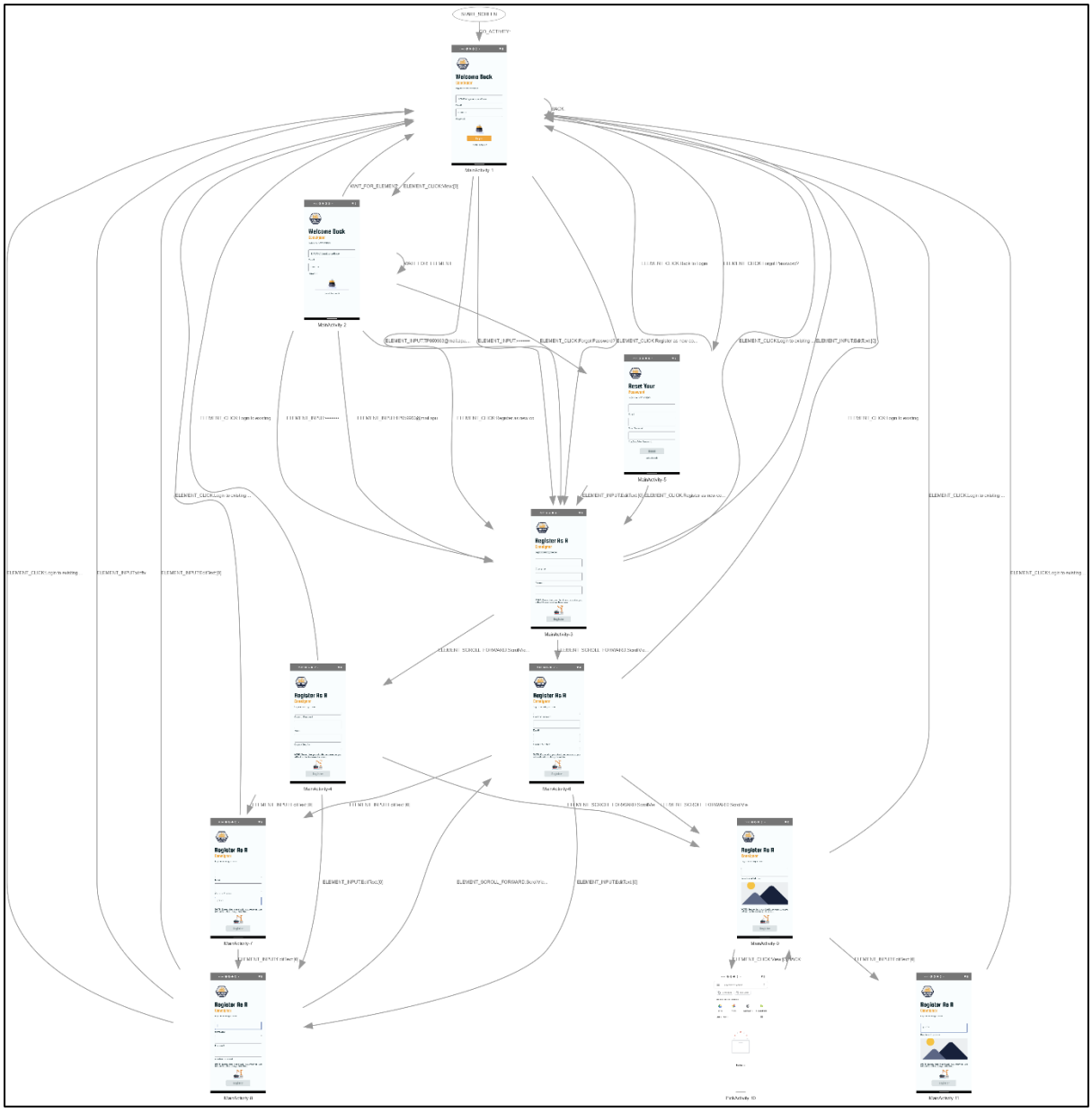
4. Entity Relationship Diagram



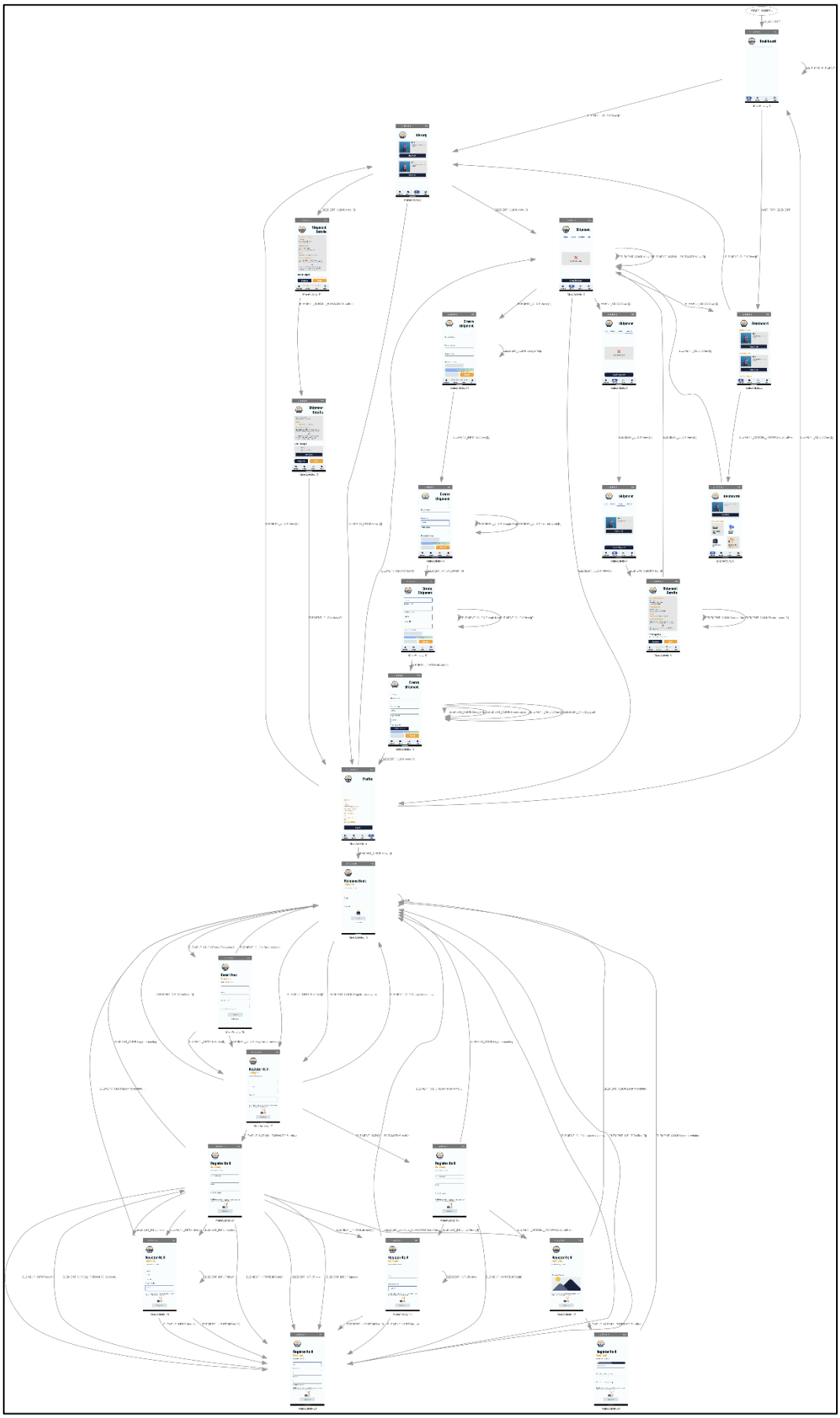
5. Automated Test Report

Three separate Robo testing are conducted using Firebase Test Lab. Noted that the session details are provided as default session details during the testing process, thus no login or registration are required. Below is evidence of the crawl graph generated on Pixel 5 device with API Level 30:

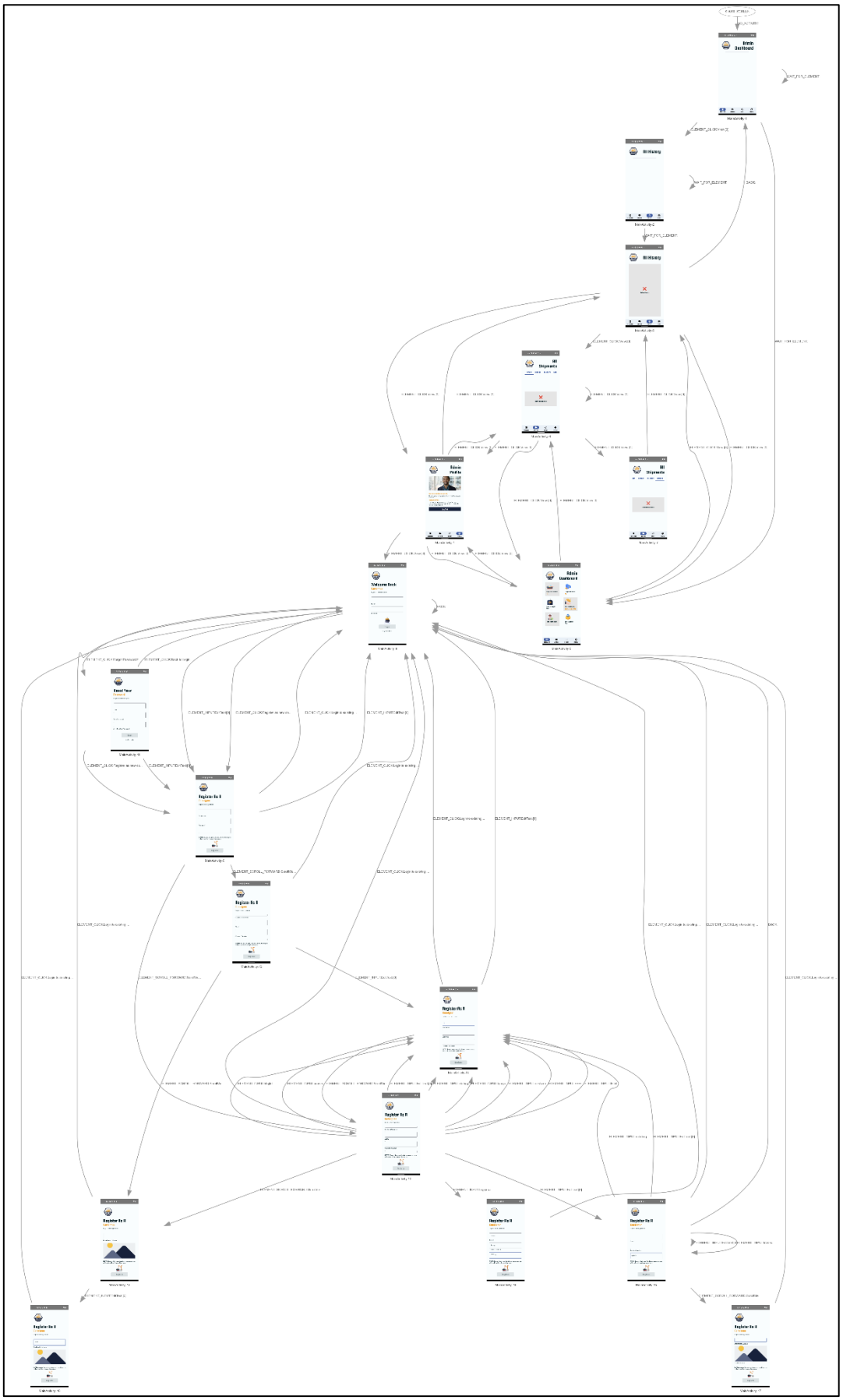
5.1. Main Screen



5.2. User Screen



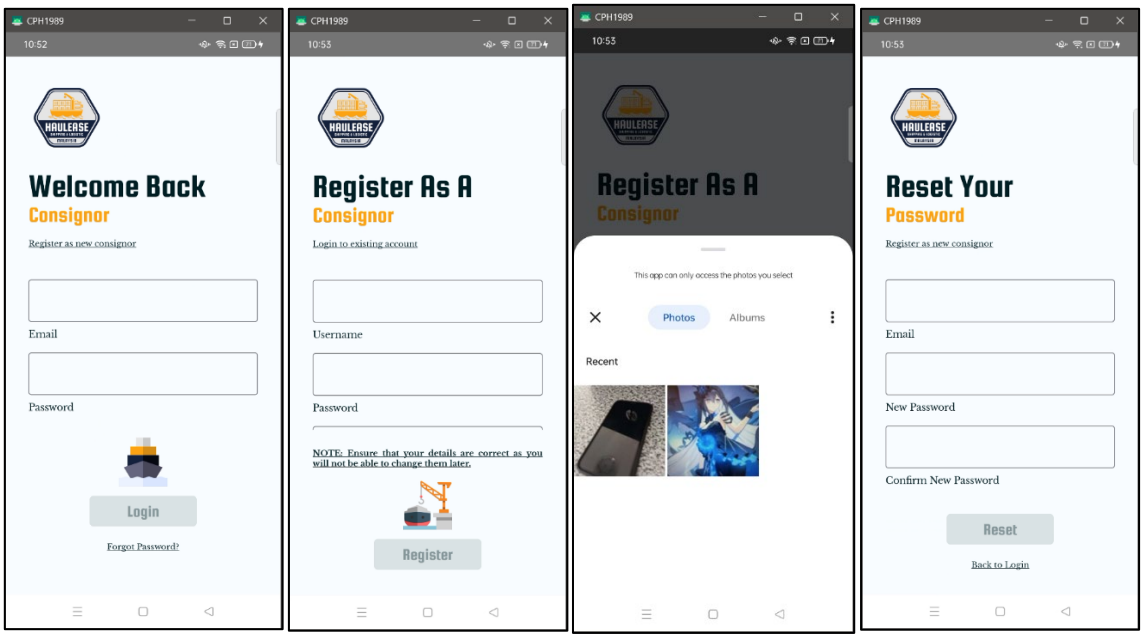
5.3. Admin Screen



6. User Manual

6.1. Consignor

Login, Register and Reset Password Screen

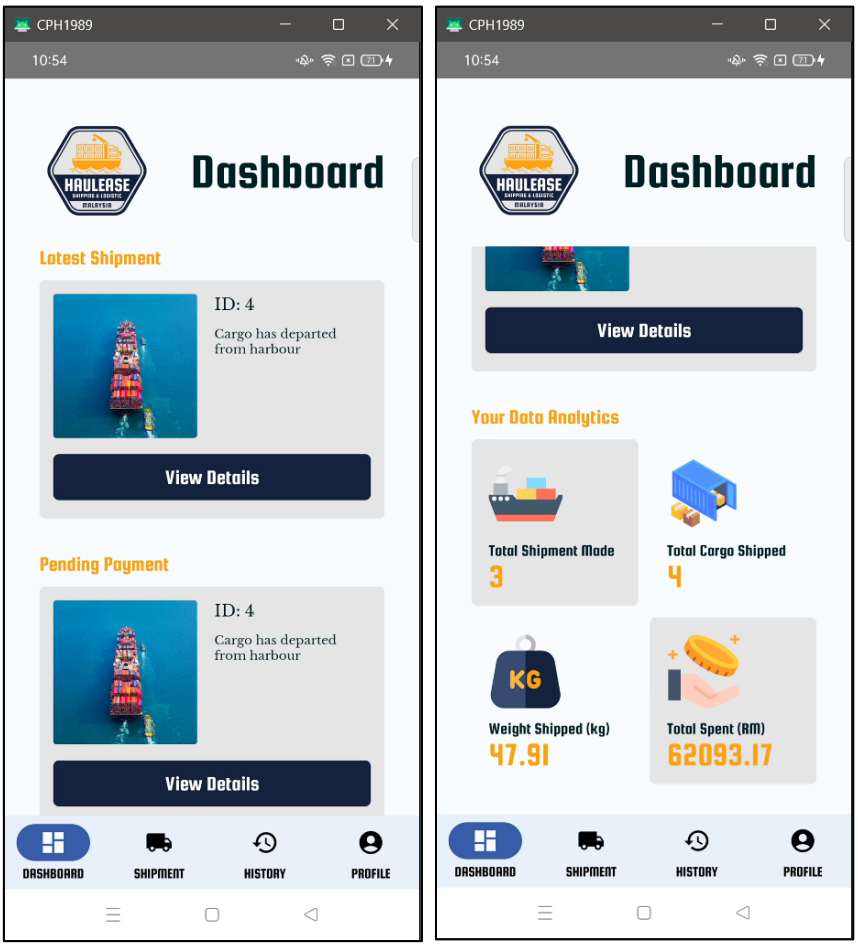


Login screen is the initial page for all unregistered users or logged out users. They are required to enter their email and password to be authenticated by Firebase. If their credentials do not match, a popup message will be shown.

Users can navigate to register screen by clicking “Register as new consignor”. They are required to enter all their details, including username, password, email address, profile picture (browse from their photos or albums), residential address, and optional inputs such as company name, company email, and company address. If there is an existing email registered, a popup message will be shown.

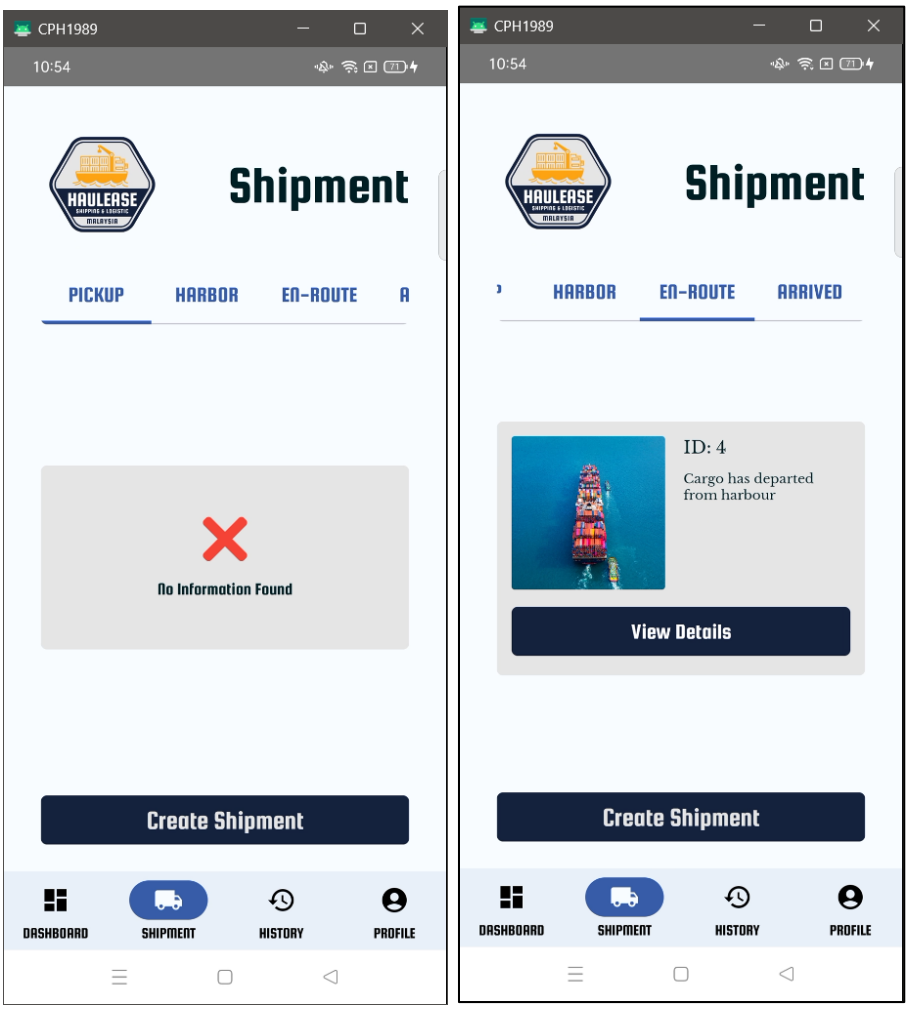
Users can also navigate to reset password screen by clicking “Forgot Password?”. They are required to enter their email address and new password before requesting Firebase to send a reset password verification link via their email address, assuming their email address is valid and usable.

Dashboard Screen



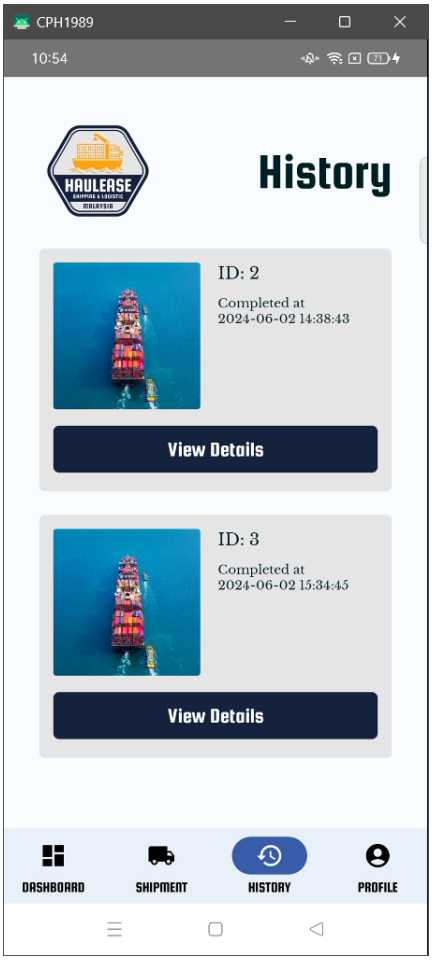
After a successful authentication, registered users will land on their dashboard. They can view their latest shipment and pending payment. Latest shipment is based on the last shipment they made while pending payment is based on the earliest shipment which they haven't paid. Besides that, there are four small boxes which displayed their account analytics like total shipment made, total cargo shipped, weight shipped (kg) and total spent (RM).

Shipment Screen



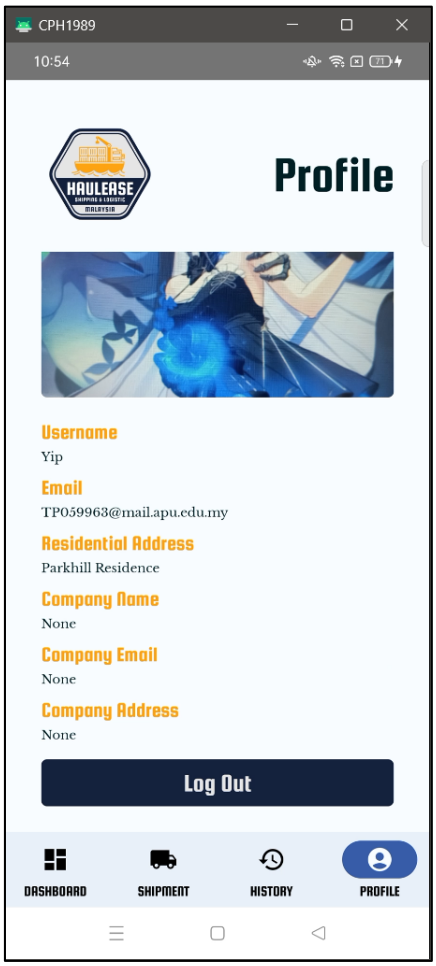
There are four categories, pickup, harbor, en-route, and arrived, to differentiate the status of each shipment placed. It is a horizontal pager where it allows users to swipe horizontally to view different categories within one screen. If there is no relevant shipment found under each category, an information message will be shown. If there is relevant shipment found, users can view their shipment status and navigate to the shipment details. There is a button at the bottom of the screen for users to create a new shipment.

History Screen



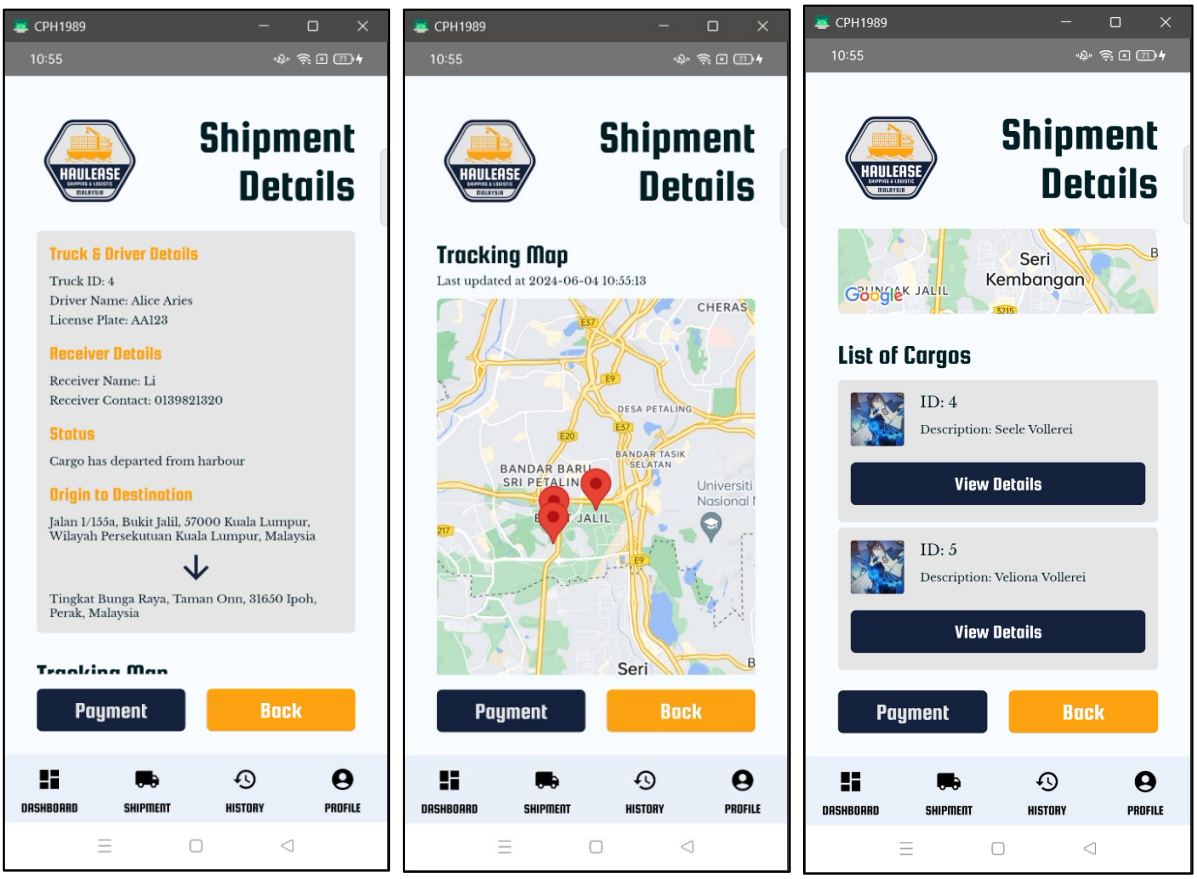
Users can find their past shipment which has been delivered along with the completion date. They can navigate to the shipment detail page to view relevant information and list of cargo attached to the shipment. Similarly, if there is no information found under completed shipment, an information message will be shown.

Profile Screen



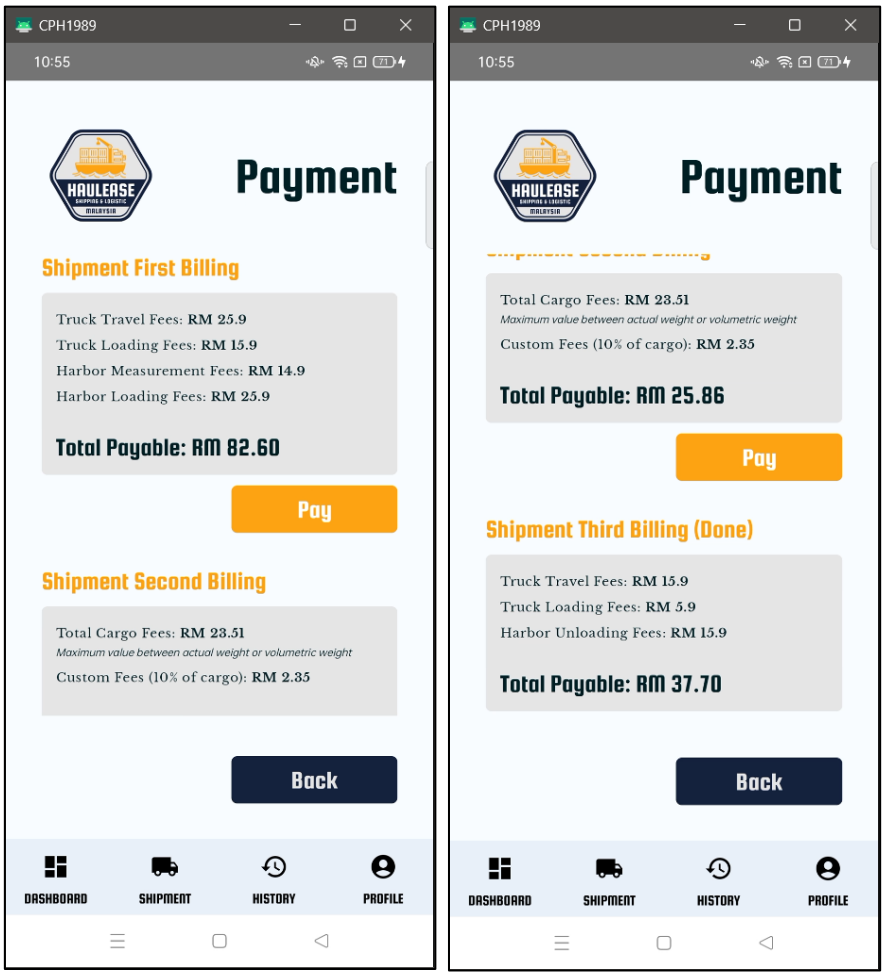
Users can view their profile details like username, email, residential address, company name, company email and company address. They can log out from their account by clicking on the “Log Out” button at the bottom of the screen. This will destroy and clear their sessions, thus they are required to login again.

Shipment Detail Screen



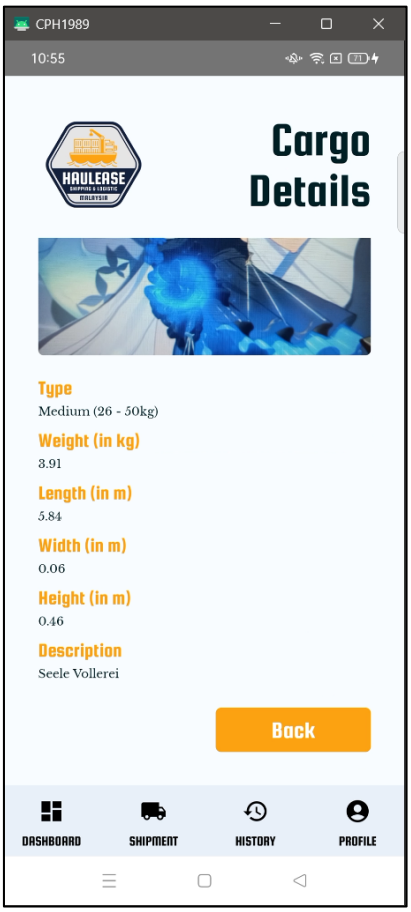
Users can view all shipment details like truck and driver details, receiver details, status, origin and destination address. Moreover, they can also track the location of the shipment, with three additional markers which will be the origin address, destination address, and current location of the user's device. The tracking map will have a message that displays the last update time. Users can also see their cargos under the shipment and navigate to view the cargo details. A button at the bottom of the screen is displayed for users to see their payment details.

Payment Screen



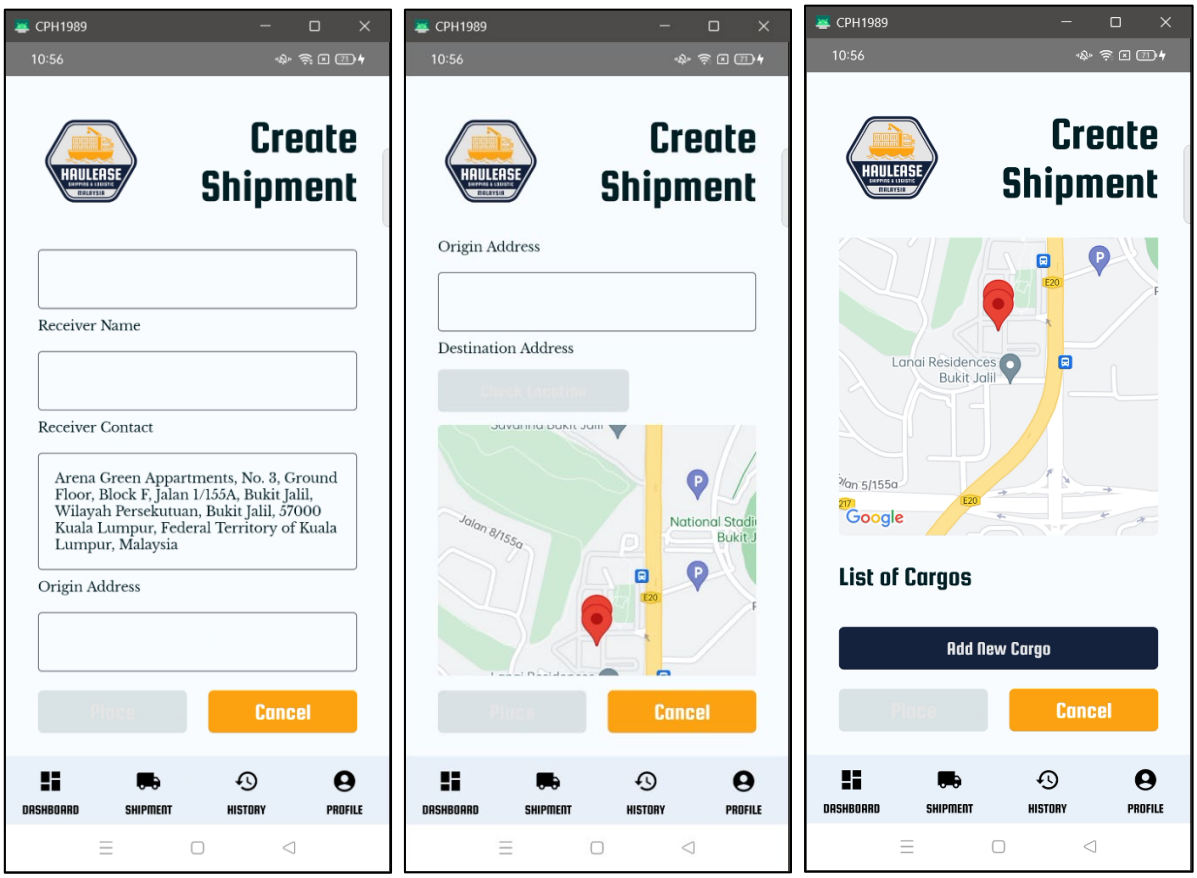
Users can see their payment details in three different billing phases. The first shipment billing is required when the truck driver come for the pickup, the second is required when the cargos have finished inspection and measurement, while the third is required when the shipment arrived at the intended destination. During the second payment, the cargo price will vary depending on the dimensions of the cargo itself and will use the maximum value between its volumetric weight or actual weight as the final price.

Cargo Detail Screen



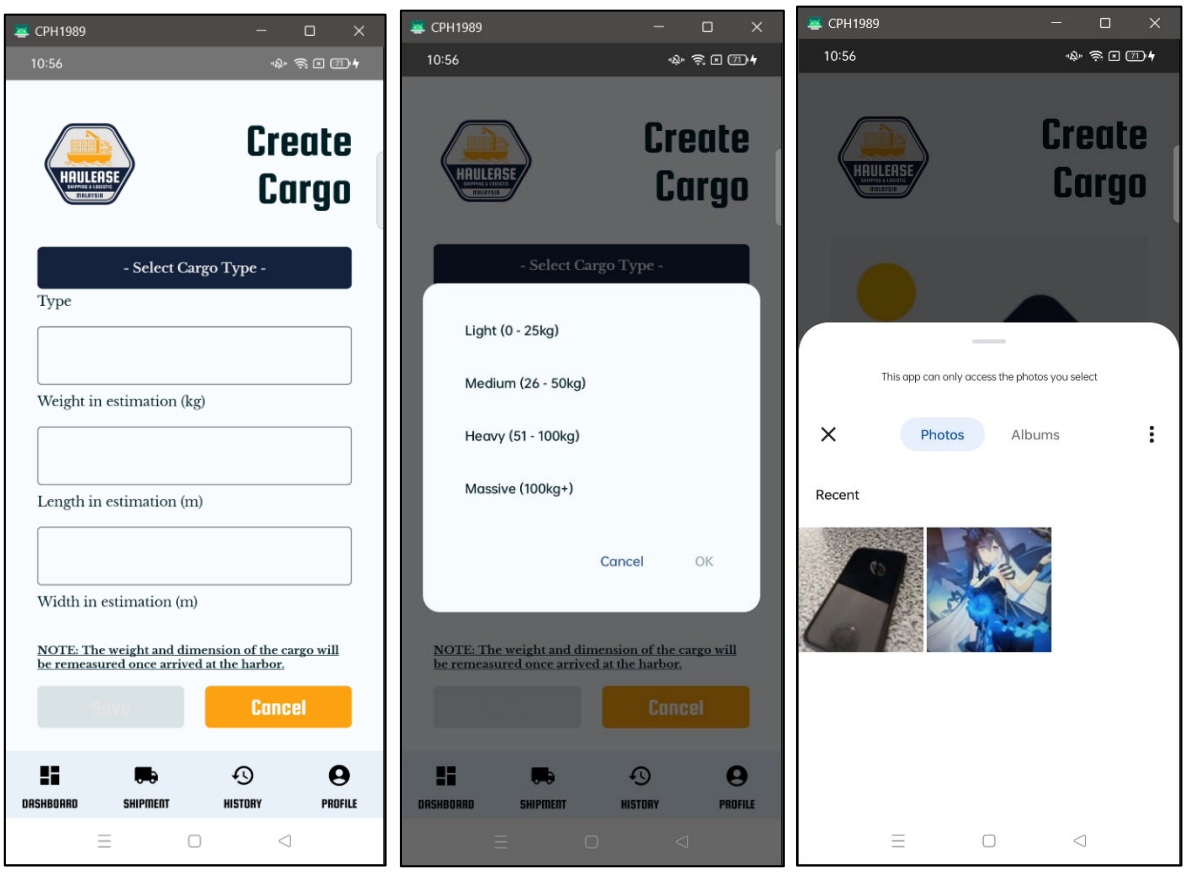
Users can view the cargo details like type of cargo, weight, length, width, height, description, and the cargo image.

Create Shipment Screen



When creating a shipment, users are required to include the receiver's name and contact, origin and destination address. Prior placing a shipment, they need to check both locations to determine whether the addresses exist or accurate. There will be three markers displayed on the map for users' references, the origin, the destination, the users' device current location, respectively. After that, they must navigate to add at least one cargo to the shipment.

Create Cargo Screen



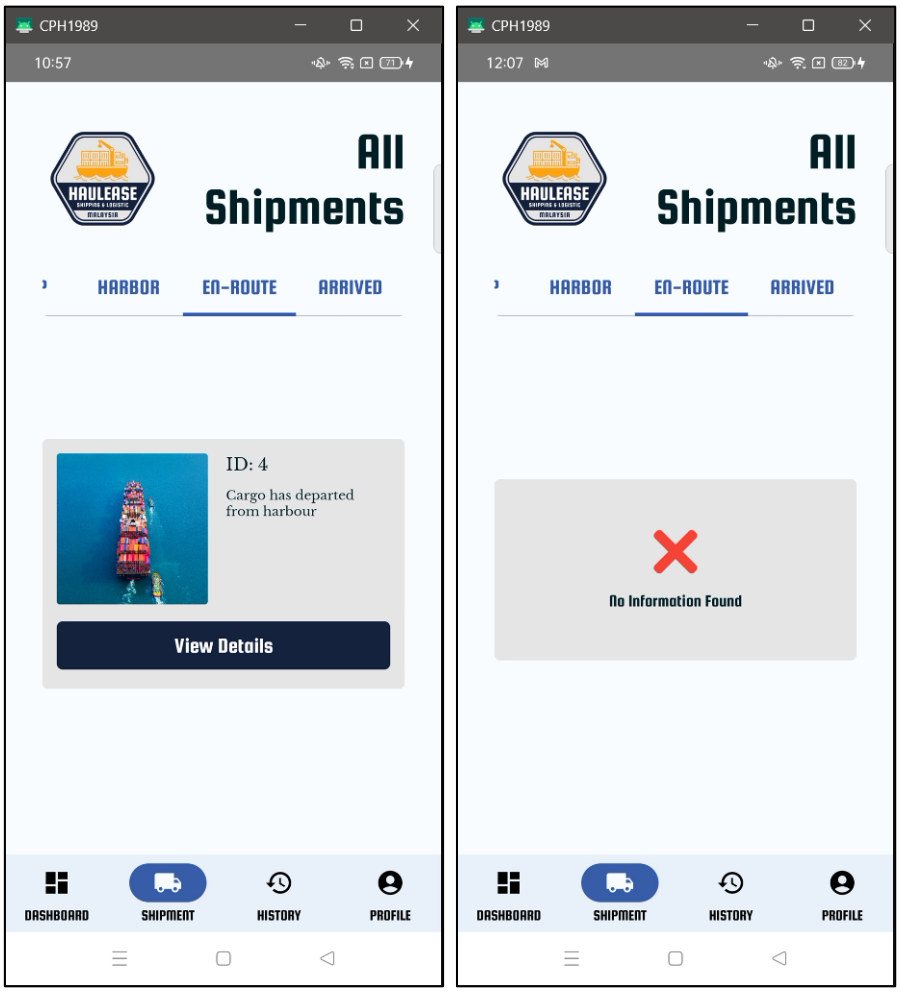
When creating a cargo, users are required to determine the type of cargo, weight, length, width, length, all in estimation as the workers at the harbor will inspect and measure the cargo to determine the actual dimension of the cargo. Users must provide a description for the cargo along with a picture.

6.2. Admin

Dashboard Screen

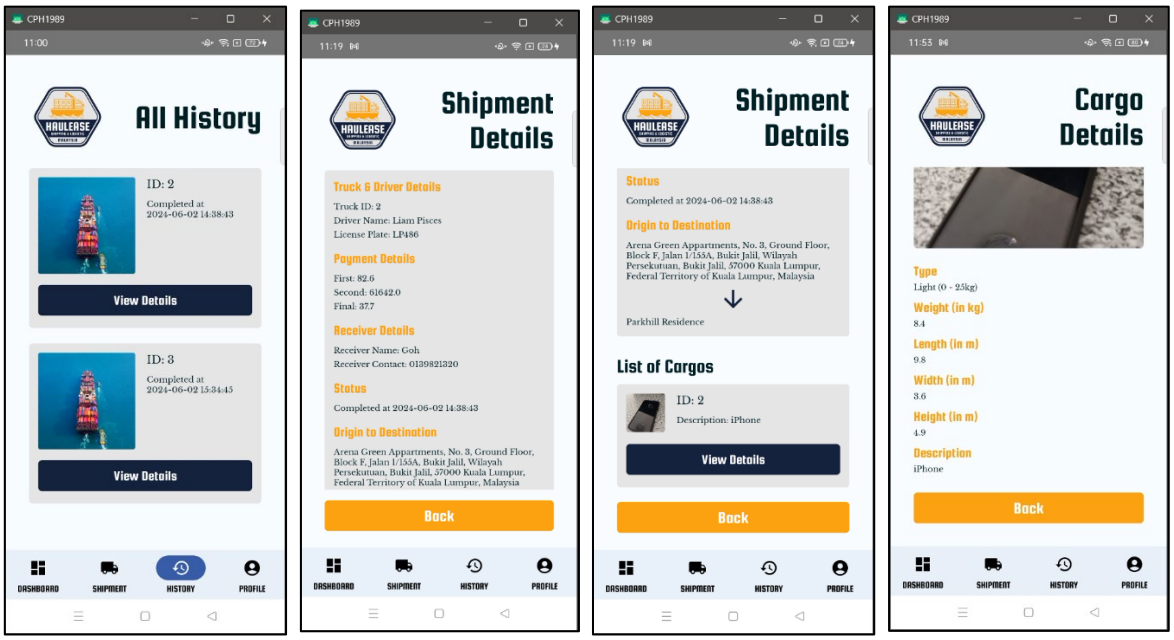
Similar to the data analytics of consignor dashboard screen, admin can view the overall service analytics like shipment received, cargo transported, weight shipped (kg), total income (RM), total active users, and total shipment done.

All Shipments Screen



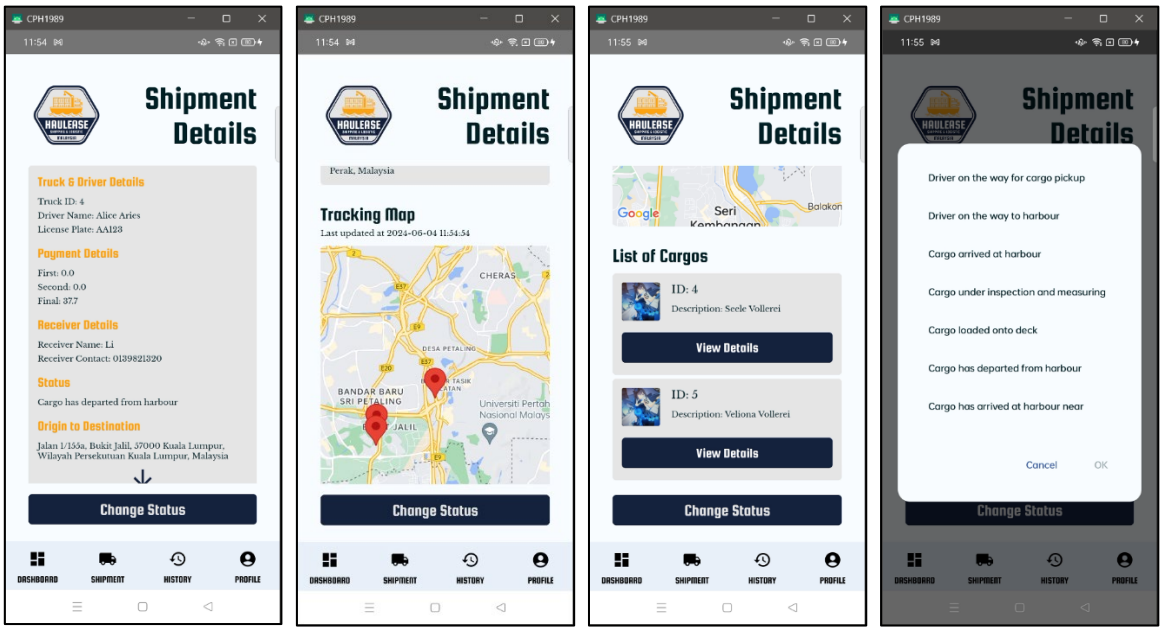
Admin can view all status categories of shipments placed similar to what users can see to their own shipments. If there is no relevant shipment found under each category, an information message will be shown. If there is relevant shipment found, admin can view the shipment status and navigate to the shipment details for management.

All History Screen



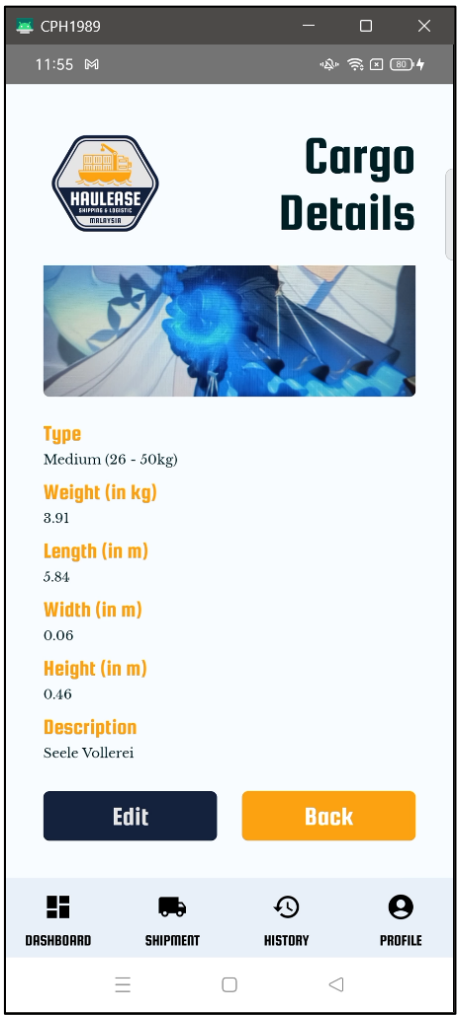
Admin can view all past shipments made by consignors along with the completion date and navigate to check each shipment details like truck and driver, payment, receiver, status, origin and destination address. They can also navigate to see each cargo details available in the shipment.

Shipment Details Screen



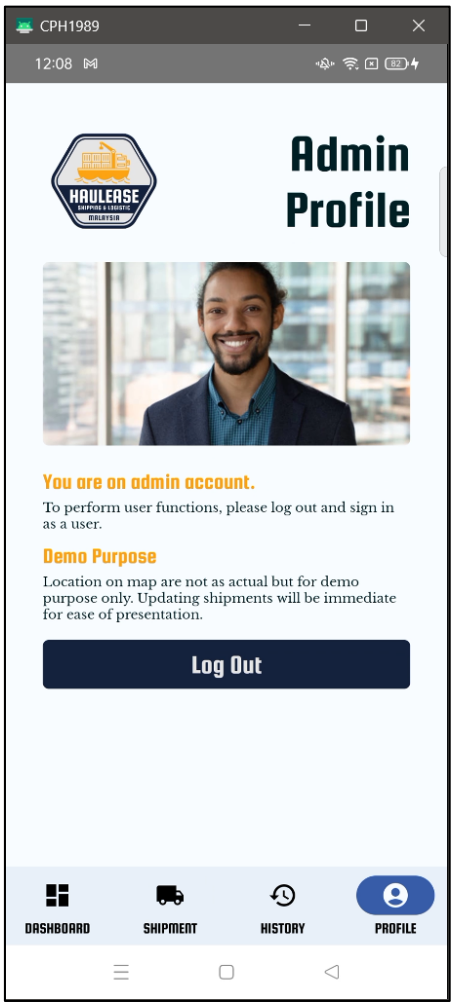
Similar to viewing shipment details through history screen, admin can see all relevant information about the shipment along with a tracking map and list of cargos as the shipment is not completed. Admin can change the status of each shipment to inform consignor and allowing them to pay their shipment fees.

Cargo Detail Screen



During the inspection and measurement of cargos, admin can edit the dimensions of the cargo to ensure that it is correctly written and calculated for payment. They will navigate to the page similar to consignor creating a new cargo, but the values are pre-tabulated into the input fields, allowing admin to change only the wrong values without altering any correct ones.

Profile Screen



There is nothing much in the profile screen as it is an admin account and only for demo purposes. Admin can only log out from their account by clicking on the “Log Out” button.

7. Critical Appraisal

7.1. Limitations

- Users are not allowed to edit their profile details.
- Users are not allowed to cancel their shipment after placing an order, they are required to contact the admin for support.
- Users can not edit their cargo detail after adding a cargo when creating a shipment.
- Users can not filter through their ongoing shipments and past history.
- The registration and login process might be vulnerable to cybersecurity attacks like swarm bots, denial of service (DoS), brute-force attack since security are not included during the development process.

7.2. Future Enhancements

- Loading screen can be improved to be fitting to the user interface.
- A collapsible menu can be implemented for easier nested navigations.
- Users should be allowed to edit their profile details.
- Users should be allowed to cancel their shipment if mistakes are made.
- Users should be allowed to edit their cargo instead of removing and recreating a new cargo.
- The tracking of shipment location should be replaced with the actual location of the truck driver or the cargo ship.
- Filter function can be included in ongoing shipment and past history for users to find their orders easier.
- Security features should be implemented to avoid cybersecurity attacks.
- Actual payment gateway should be implemented to process and verify users' payment.
- Admin should have more controls over every active users, shipping, and cargo.

8. Mobile Backend Systems

8.1. History on Mobile Backend Systems

According to Flexiple (n.d.), the history of mobile backend systems is closely connected to the evolution of mobile phones. This is because early mobile applications, often only integrated with basic functionalities like games such as Snakes, Bounce etc., or calculators, and contact number are stored in SIM cards, thus had less requirements for backend needs.

However, with the invention and development of smartphones like iPhones, Samsung, Huawei and more sophisticated applications, the need for flexible and powerful backend systems increased. The initial approach to a backend system is very primitive as it invovles building a custom backend infrastructure from scratch which can cater 100% of how the developers want but costs them for their time and resources in return. As mentioned by Dubey (2021), API-based backends such as Node.js with Express, Python with Django and Mobile Backend as a Service (MBaaS) platform are introduced as a more streamlined solutions for mobile backend systems.

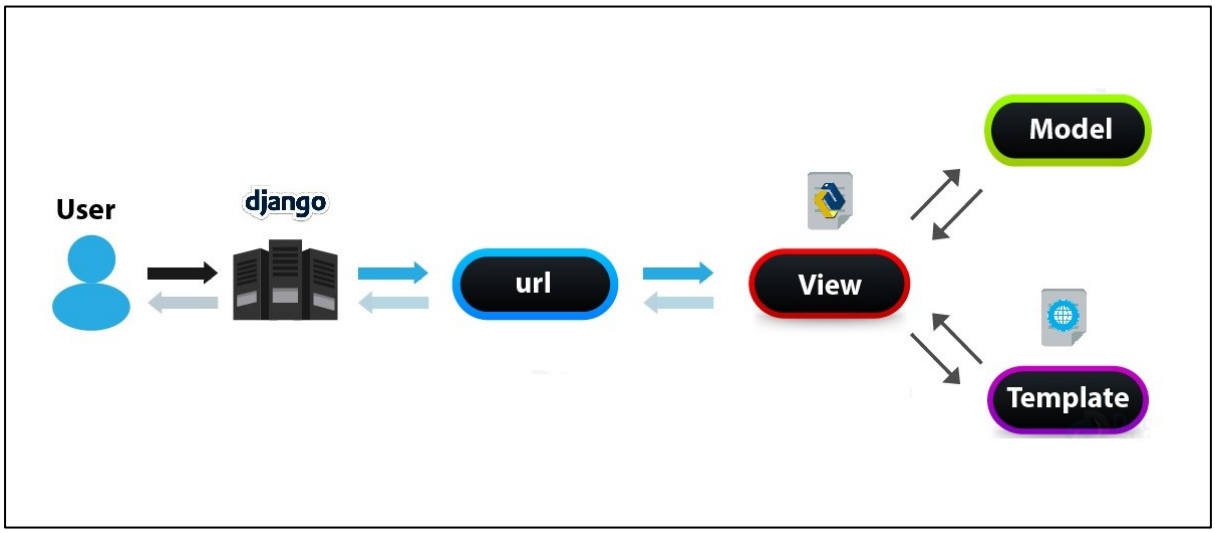


Figure 1: Example of API-based Backend using Python and Django (Adityaraj1711, 2022)

8.2. Architectural Evolution of Backend Systems

Mobile backend architectures have been through a notable transformation, adapting to the growing demands of mobile applications. There are three most popular backend system architectures, namely monolithic architecture, microservices architecture, and serverless architecture.

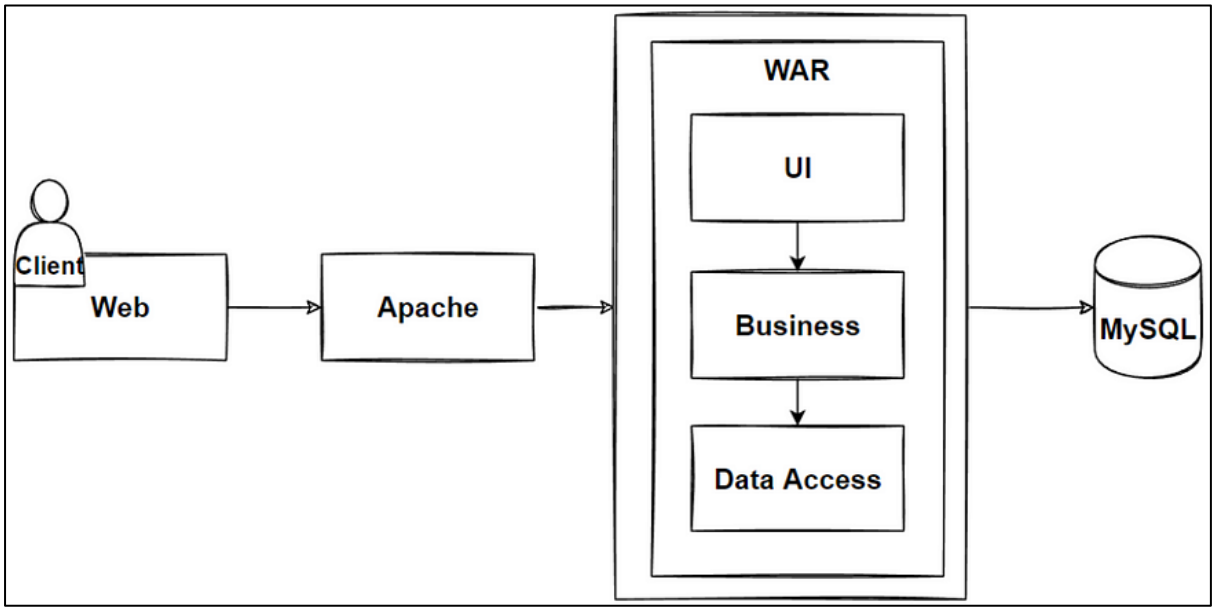


Figure 2: Illustration of Monolithic Architecture (Ozkaya (a), 2023)

Based on figure above, monolithic architecture is a traditional approach which is mostly used during the early days of mobile application development as it only involves a single server that handles all aspects of the backend (Awati & Wigmore, 2022). It is defined as a software design pattern where the whole application is built upon a single, tightly coupled unit or standalone codebase with its functionalities and data combined together. A monolithic architecture can be easy to implement and manage in the beginning, yet it will eventually encounter limitations as the application complexity increases. It will be headache to scale the entire system to adapt user growth or new features and time-consuming when performing troubleshoots and maintenance.

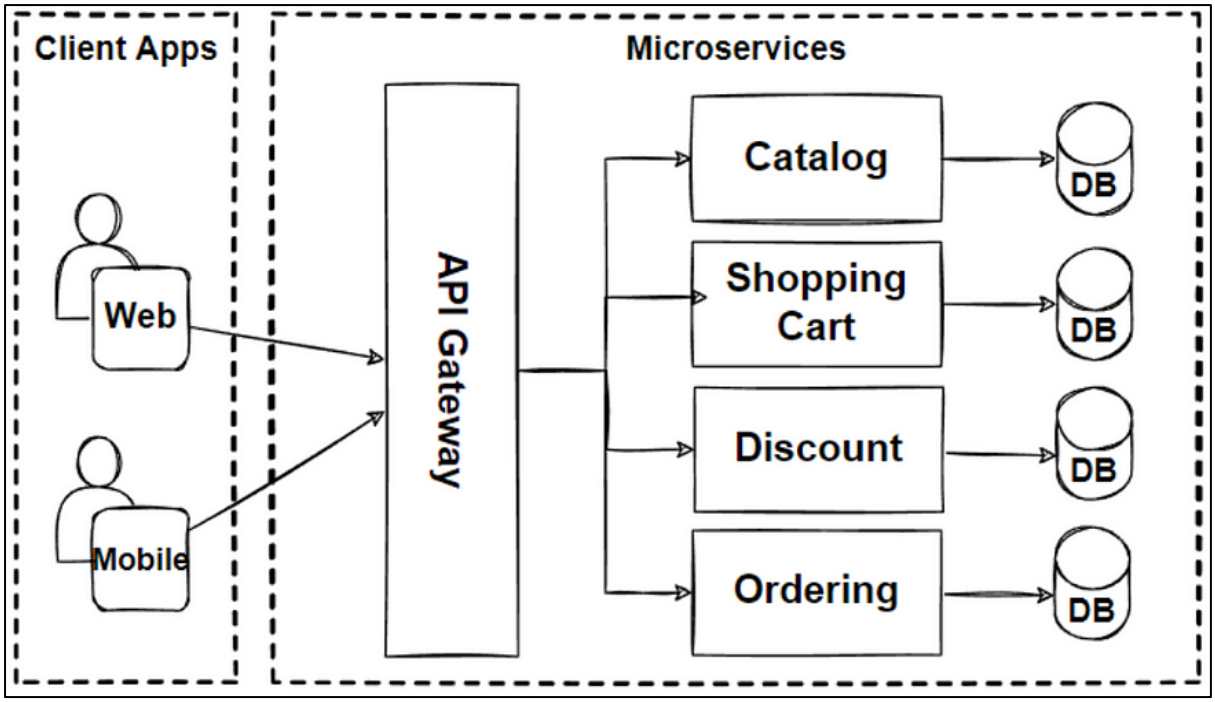


Figure 3: Illustration of Microservices Architecture (Ozkaya (b), 2023)

Based on Google Cloud (n.d.), microservices architecture is a software development style where an application is built as a single and separated unit or service. According to the figure above, each services have its own specific business function and still can communicate with other services via application programming interface (API). Microservices architecture solved the problem which monolithic architecture faced which the increased application complexity since it is more scalable and flexible. This approach offers simplicity to the development and deployment of an application as developers can divide and code each microservice independently, thus increasing productivity. Secondly, as mentioned before, it is scalable to cater for larger demands without affecting other sections of the backend. Lastly, it is fault tolerance since the application is constructed with different serivces, thus if one service fails, the others will continue to execute.

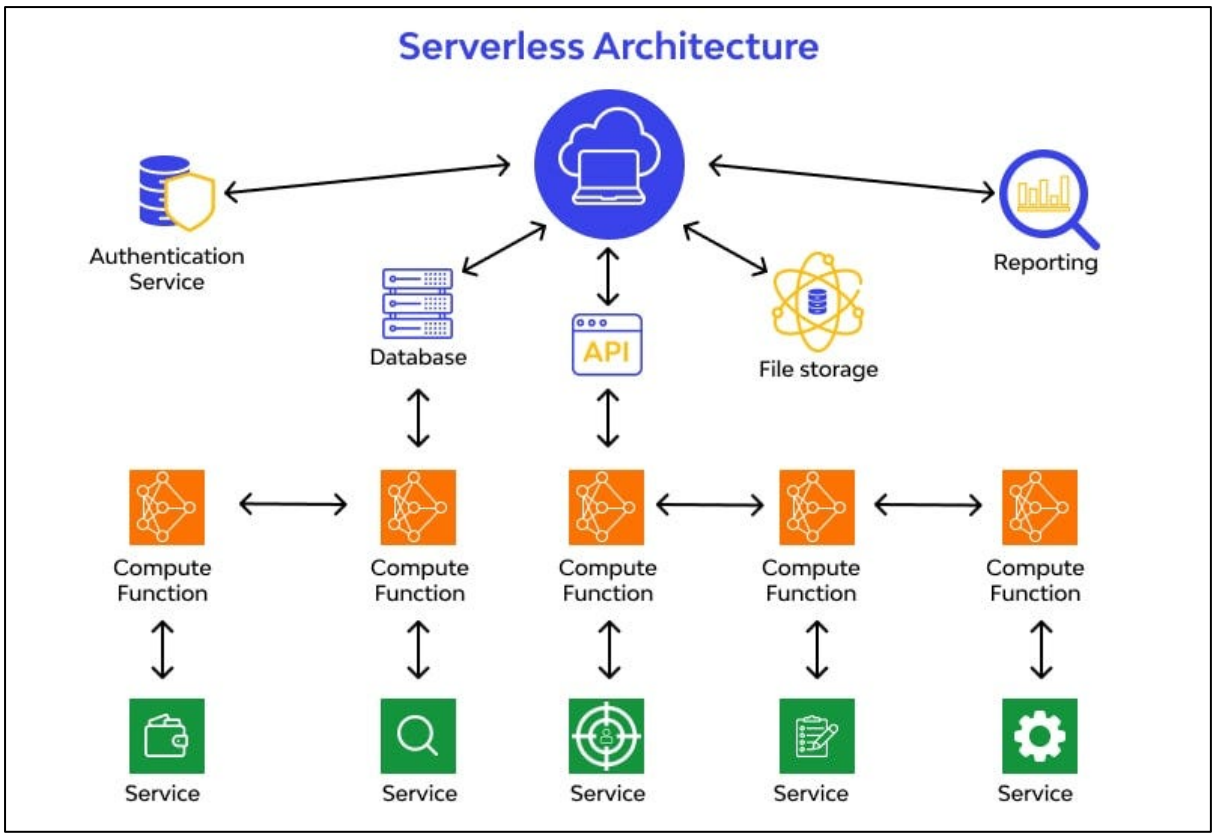


Figure 4: Illustration of Serverless Architecture (Dineshchandgr, 2022)

Serverless architecture is considered as the latest trend in mobile backend architecture as it is a cloud computing execution model where the cloud provider manages server allocation and provisioning like Amazon Web Services or AWS (AWS Amazon, n.d.). Using this model as shown in the figure above, developers will no longer require to manage their server and focus primarily on writing code for their application functionalities. Cloud providers such as AWS, Google Cloud Platform, Microsoft Azure provides serverless functions that execute code in response to events which can benefit in terms of cost efficiency and scalability significantly. Developers will only pay for what they used and the rest will be handled by cloud providers.

8.3. API-based Backend Systems

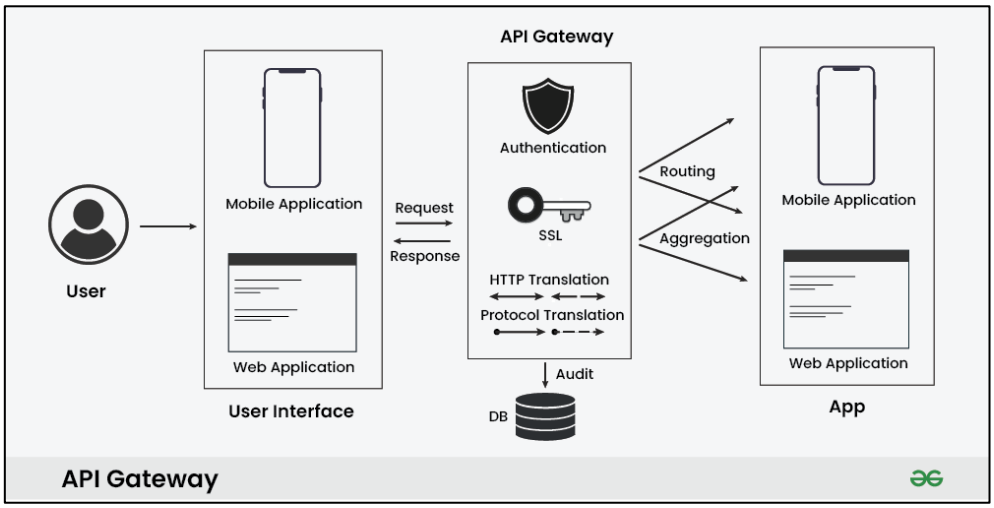


Figure 5: Example of API-based Backend Systems (Geeksforgeeks, 2024)

API-based backend systems, also known as API-driven architectures, plays an important role in modern software development. It provides a structured way to develop and interact with the backend services through predefined protocols and endpoints, illustrated in the figure above.

As stated by Chiu (2024), there are several advantages for having an API, like for instance, separation of concerns where the API helps in separating the client and server, allowing both front-end and back-end to work independently and boost efficiency. Other than that, APIs can facilitate scaling individual components of an application independently in which heavy or resource-consuming algorithms can be implemented in backend services while front-end will have more spaces to work on its user interface design. Besides that, APIs also allow developers to reuse the code for different applications such as a single authentication or authourization API can be used in multiple applications, reducing code redundancy and saving time and effort.

Since APIs are all about calling requests and responses, it can easier integrate with thrid-party services and systems which promotes a robust ecosystem with seamless communication. API-based backend systems are least likely to disrupt client-side code as business models or business processes are only included in the backend, thus it can be tested using external platforms like Postman to validate its responses or packages. Last but not least, having APIs will encourage standardization throughout the codebase because APIs mostly follow standards like REST, GraphQL, or SOAP and improve the overall performance of the system using techniques like caching, load balancing, and asynchronous processing.

8.4. Role of Mobile Backend as Service Environment

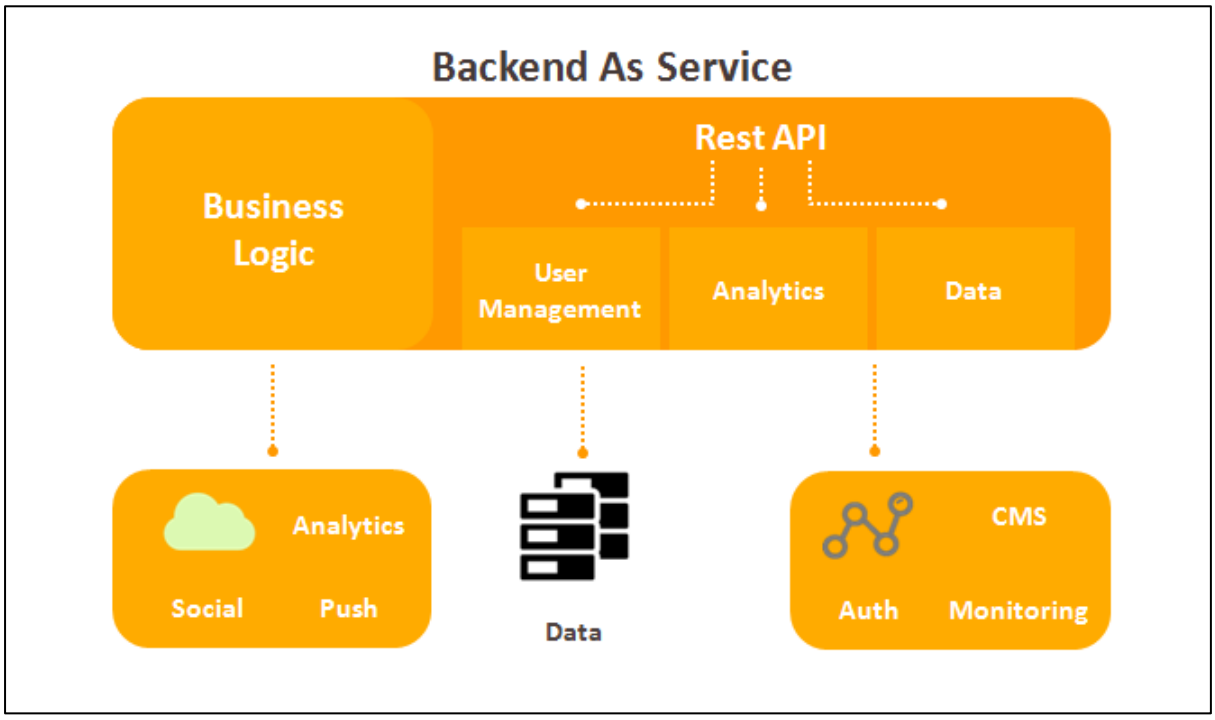


Figure 6: Illustration of MBaaS (Yoosuf, 2015)

Mobile backend as service environment, MBaaS in short, is an online cloud-based service or infrastructure which is designed to be an all-in-one solution for modern application development (Fanchi, 2023). For instance, Firebase, AWS Amplify, Backendless and Kinvey are examples of MBaaS providers. These providers can cater multiple requirements independently and help developers to organize the backend development process in a single platform. It hides the complexity of backend development and provide developers with pre-built templates or services, allowing developers to spend more time on the front-end without worrying about the server and database issues.

By leveraging MBaaS, developers can considerably reduce the time required to build, deploy and manage backend services because MBaaS contains ready-to-use features such as authentication, data storage, and push notifications, which can be integrated into the application with no more than two lines of code written. MBaaS platforms are designed in such a way that it can fit a variety of application usage, from small-scale to enterprise levels while ensuring smooth performance and reliability such as minimal downtime and backups without the need for manual intervention. Since it varies based on the application deployed, it is also cost-effective due to the pay-as-you-go model and eliminates the cost needed for physical servers and infrastructure.

Aside from that, MBaaS providers uses advanced security measures to protect data and backend services including data encryption, secure authentication mechanisms, and compliance with industry standards and regulations. This helps in protecting sensitive user data and ensuring secure application operations. Moreover, it also come with integrated analytics and monitoring tools that help developers to track the application usage, performance metrics, and user behaviour, which is pivotal for application optimization, improvement in user engagements, and make major business decisions.

Lastly, most MBaaS offer real-time data synchronization capabilities, allowing changes made on one device to be instantaneously uploaded and reflected on all other connected devices. This will be extremely important and useful for applications that require real-time collaboration or instant data updates like stock markets or productivity tools.

8.5. Justification on Adopted Mobile Backend Systems

Based on the previous chapter, it is crucial to understand what the requirements are needed for the application to be impactful in the market. There are a few primary factors in adopting a mobile backend system which includes architecture selection and technology stack to support the data and functionality of a mobile application.

Firstly, scalability is the ability to handle increasing amounts of data and user requests is crucial as the application grows. For instance, backend-as-a-service (BaaS) platforms like Firebase or AWS Amplify provide automatic scaling capabilities, which can manage fluctuations in user load without manual configuration. Secondly, high performance is necessary to ensure quick response time and smooth user experience. By combining RESTful APIs and GraphQL, it can optimize data retrieval by ensuring that only necessary data is fetched and reducing latency.

Thirdly, protecting user data and preventing unauthorized access are important for any mobile application. It can be solved by adopting services that offer built-in security features such as OAuth for authentication, encryption for data storage, and regular security updates. Fourthly, managing costs is critical, especially for startups and small businesses. Serverless architectures or BaaS platforms reduce the need for managing infrastructure, leading to lower operational costs. Pay-as-you-go pricing models further allow businesses to pay only for the resources developers use.

Faster development cycles can bring products to market more quickly and allow for rapid iteration based on user feedback. Push notifications and real-time databases can accelerate the development process by using a backend platform with pre-built services like user management. The ease of integration between backend system and other tools and services should be considered. For example, choosing a backend that supports integration with third-party APIs and various databases ensures that the application can leverage existing tools and expand its functionality easily.

The backend system should allow for custom solutions tailored to specific business needs. Using cloud platforms like AWS and Azure, which offer a wide range of services and the ability to build custom logic using serverless functions or microservices. Lastly, being able to track user behaviour, application performance, and backend health is crucial for ongoing improvement and long-terms support. Platforms that provide built-in analytics and monitoring tools, such as Firebase Analytics, allow developers to gain insight and troubleshoot issues.

9. References

Adityaraj1711. (2022). *Django Backend Architecture*. Retrieved from GitHub: https://github.com/Adityaraj1711/django-backend-architecture

Awati, R., & Wigmore, I. (May, 2022). *Monolithic Architecture*. Retrieved from TechTarget: https://www.techtarget.com/whatis/definition/monolithic-architecture

AWS Amazon. (n.d.). *Building Applications with Serverless Architectures*. Retrieved from AWS Amazon: https://aws.amazon.com/lambda/serverless-architectures-learn-more/#:~:text=What%20is%20a%20serverless%20architecture,management%20is%20done%20by%20AWS.

Chiu, T. (26 March, 2024). *Demystifying the Relationship between APIs and Backend Systems*. Retrieved from Medium: https://medium.com/@tiokachiu/demystifying-the-relationship-between-apis-and-backend-systems-4060b92e8546#:~:text=API%20and%20Backend%20Integration&text=The%20backend%20system%20processes%20the,back%20to%20the%20client%20application.

Dineshchandgr. (23 July, 2022). *Do you know everything about Serverless Architecture?* Retrieved from Medium: https://medium.com/javarevisited/do-you-know-everything-about-serverless-architecture-f0cd06c81329

Dubey, A. (15 Jun, 2021). *The story of a backend — the beginning*. Retrieved from Medium: https://medium.com/geekculture/the-story-of-a-backend-the-beginning-e51a31ea83d5

Fanchi, C. (20 January, 2023). *What Is Mobile Backend As A Service (MBaaS)?* Retrieved from Backendless: https://backendless.com/what-is-mobile-backend-as-a-service-mbaas/

Flexiple. (n.d.). *Guide to Backend Development*. Retrieved from Flexiple: https://flexiple.com/backend/deep-dive

Geeksforgeeks. (8 March, 2024). *What is API Gateway | System Design ?* Retrieved from Geeksforgeeks: https://www.geeksforgeeks.org/what-is-api-gateway-system-design/

Google Cloud. (n.d.). *What is Microservices Architecture?* Retrieved from Google Cloud: https://cloud.google.com/learn/what-is-microservices-architecture#:~:text=A%20microservices%20architecture%20is%20a,architecture%20diagrams%20and%20services%20independently.

Ozkaya (a), M. (14 February, 2023). *When to use Monolithic Architecture*. Retrieved from Medium: https://medium.com/design-microservices-architecture-with-patterns/when-to-use-monolithic-architecture-57c0653e245e

Ozkaya (b), M. (17 February, 2023). *Microservices Architecture for Enterprise Large-Scaled Application*. Retrieved from Medium: https://medium.com/design-microservices-architecture-with-patterns/microservices-architecture-for-enterprise-large-scaled-application-825436c9a78a

Yoosuf. (30 December, 2015). *What, When and Why MBaaS?* Retrieved from To The New: https://www.tothenew.com/blog/what-when-and-why-mbaas/