Table of Contents

1. Problem statement

In today's digital era, social media has become an integral part of modern business operations, with over 28 million Malaysians actively engaging on platforms such as Instagram, TikTok, Facebook, and YouTube. Many entrepreneurs and **small businesses rely heavily on these platforms to promote and sell their products**. However, despite the convenience of social media selling, the lack of a dedicated e-commerce platform limits these sellers' ability to build trust, manage their brand identity, and sustain long-term customer relationships. Moreover, depending solely on social media platforms exposes businesses to **unpredictable algorithm and policy changes** that can significantly affect their visibility and sales performance.

Currently, existing e-commerce solutions such as Shopify and Wix present significant challenges for small-scale entrepreneurs. These platforms often require **high subscription fees** and **demand technical knowledge** that most social media sellers lack. For instance, Shopify's annual cost exceeds RM1000, which is a considerable investment for small businesses. Additionally, the complex setup process and overwhelming configuration options discourage many sellers from transitioning to their own online store. This gap creates a need for a more user-friendly, affordable, and efficient solution that enables social media sellers to establish their own e-commerce presence quickly and effectively.

Miles aims to address this problem by providing a **streamlined e-commerce builder designed** specifically for social media sellers in Malaysia and the Southeast Asia region. The platform focuses on simplicity, affordability, and automation, allowing users to create a fully functional e-commerce store within minutes without requiring technical expertise. By integrating essential features such as Google OAuth login, payment gateway, and business analytics, Miles offers a comprehensive and user-friendly solution for digital entrepreneurs. In doing so, it bridges the gap between social media selling and professional e-commerce ownership, empowering small businesses to grow sustainably in the digital economy.

2. Project objective

The primary objective of the Miles project is to develop a user-friendly e-commerce builder that empowers social media sellers to create and manage their online stores efficiently, affordably, and without technical barriers. The project focuses on addressing the key pain points faced by small-scale sellers.

The specific objectives of this project are as follows:

1. **To design and develop an intuitive e-commerce builder** that allows users to generate a functional online store within five minutes through a guided, form-based setup process.
2. **To implement secure and convenient user authentication** using Google OAuth to ensure a fast registration and login process while maintaining high-level data protection.
3. **To integrate reliable payment gateways** ensuring that sellers can process payments seamlessly.

4. **To develop a business analytics module** that provides insights into sales trends, top-performing products, helping sellers make data-driven decisions.
5. **To ensure scalability and performance efficiency** by leveraging cloud hosting solutions and open-source frameworks that minimize operational costs and maximize system reliability.
6. **To enhance user engagement and accessibility** through a responsive and modern UI/UX design built with Figma, emphasizing ease of use and local market familiarity.

Collectively, these objectives aim to provide a comprehensive solution that transforms how small businesses transition from social media selling to full-fledged e-commerce operations, enabling them to compete effectively in the rapidly growing digital marketplace.

3. Methodology/Technique/Approach

The development of Miles follows the **Agile methodology**, which emphasizes iterative progress, continuous feedback, and adaptive planning to achieve project objectives efficiently. This approach allows the development team to make improvements throughout the project while ensuring that user needs are consistently met.

3.1 Data Collection and Requirement Analysis

At the start of the project, the team conducted informal research and collected feedback from social media sellers to identify the common challenges they face when creating online stores. This helped determine essential system requirements such as fast setup, simple navigation, and affordable pricing. The gathered insights were then translated into clear functional requirements that guided the development process.

3.2 Design and Planning

Using **Figma**, the team created interactive prototypes to visualize the user journey and page layouts. Each design iteration was reviewed internally to ensure that it met usability goals, emphasizing minimal user input and an intuitive interface. The planning phase also included dividing project tasks into smaller, manageable modules to be developed in short sprints.

3.3 Development Approach:

The project was developed incrementally through multiple Agile sprints. Each sprint included planning, development, and testing cycles. The **frontend** was built using **React**, focusing on fast and responsive user interactions. The **backend** was developed using **Laravel PHP**, handling authentication, database operations, and server-side logic. The team used **MySQL** for data management and integrated **Google OAuth** to align with the project's objectives of speed and simplicity.

3.4 Testing and Evaluation

Testing was carried out at the end of each sprint to ensure that each module performed as intended. Unit testing was used to verify individual functions such as login, product upload, and payment gateway integration. Integration testing ensured smooth communication between the frontend and backend. User feedback was gathered from early testers to refine the user experience and fix potential issues before deployment.

3.5 Deployment and Maintenance:

The completed system was deployed using **Vercel** for the frontend and **Railway** for the backend and database. These platforms were chosen for their scalability, automatic deployment, and low maintenance cost. Post-deployment monitoring and updates are conducted to maintain performance and add new features based on user feedback.

By applying Agile practices and continuous iteration, the project team successfully achieved an efficient workflow that ensured Miles remained user-cantered, scalable, and aligned with the project's goals of simplicity, affordability, and speed.

4. Project Management

The pages below show the Work Breakdown Structure as well as the Gantt Chart prepared for this project.

DIP2009 Work Breakdown Structure

4.1 Work Breakdown Structure

Phase	WBS Code	Task/Deliverable	Activities/Details	Duration (Days)	Start Date	End Date	Precedence/Dependencies
1	1	**Requirements Gathering & Planning**		4	17/09/2025	22/09/2025	Project Start
	1.1	Project Scope Definition	• Define project objectives and success criteria • Identify target users (store owners, customers) • Document project constraints and assumptions	1	17/09/2025	17/09/2025	Project Start
	1.2	Functional Requirements	• Document user authentication requirements • Document store and product management requirements • Document order processing requirements • Document payment processing requirements • Document analytics and reporting requirements	1	18/09/2025	18/09/2025	1.1
	1.3	Non-Functional Requirements	• Define security requirements • Define performance requirements • Define scalability requirements	1	19/09/2025	19/09/2025	1.2
	1.4	Technology Research & Planning	• Research Stripe payment integration • Finalize technology stack (Laravel, React, MySQL, Stripe) • Create project timeline and milestones	1	22/09/2025	22/09/2025	1.3
2	2	**System Design & Architecture**		4	23/09/2025	26/09/2025	1.4
	2.1	Database Design	• Design database schema (users, stores, products, customers, orders, order_items) • Define table relationships and foreign keys • Create Entity-Relationship (ER) diagram • Plan data validation rules	2	23/09/2025	24/09/2025	1.4
	2.2	API Design	• Design RESTful API endpoints • Plan authentication strategy (Laravel Sanctum) • Define authorization rules • Plan Stripe webhook integration	1	25/09/2025	25/09/2025	2.1
	2.3	Frontend Architecture	• Create wireframes for key pages (store creation, product management, storefront, checkout) • Design user flows (admin, guest customer, registered customer) • Plan component structure and reusability • Plan state management strategy (Context API for cart) • Design responsive layouts	1	26/09/2025	26/09/2025	2.2
3	3	**Backend Development**		17	29/09/2025	21/10/2025	2.3
	3.1	Environment Setup	• Install Laravel framework • Configure database connection • Install Laravel Sanctum for authentication • Install Laravel Socialite for Google OAuth • Configure CORS settings	1	29/09/2025	29/09/2025	2.3
	3.2	Database Implementation	• Create database migrations for all tables • Run migrations and verify structure • Test relationships and constraints	1	30/09/2025	30/09/2025	3.1
	3.3	User Authentication	• Implement user registration and login • Implement Google OAuth integration • Implement logout functionality • Generate and manage Sanctum tokens • Add validation and error handling	3	01/10/2025	03/10/2025	3.2
	3.4	Store Management	• Create Store model and controller • Implement store CRUD operations • Handle image uploads (logo, background) • Implement publish/unpublish functionality • Add authorization middleware	2	06/10/2025	07/10/2025	3.3
	3.5	Product Management	• Create Product model and controller • Implement product CRUD operations • Handle product image uploads • Implement stock and category management • Add validation and authorization	3	08/10/2025	10/10/2025	3.4
	3.6	Customer & Public Store	• Create Customer model and authentication • Implement public store API (fetch by slug) • Return published stores and products only	1	13/10/2025	13/10/2025	3.5
	3.7	Order Management	• Create Order and OrderItem models • Implement order creation (guest and registered customers) • Generate unique order numbers • Calculate order totals • Store product snapshots in order items • Implement order retrieval for store owners	3	14/10/2025	16/10/2025	3.6
	3.8	Analytics	• Create analytics endpoint • Calculate revenue and order statistics • Identify top-selling products	1	17/10/2025	17/10/2025	3.7
	3.9	Payment Integration	• Install Stripe PHP SDK • Configure Stripe API keys • Implement payment intent creation • Implement Stripe webhook handler • Handle payment success (create order, update status, reduce stock) • Handle payment failures with logging	2	20/10/2025	21/10/2025	3.8
4	4	**Frontend Development**		19	29/09/2025	23/10/2025	2.3
	4.1	Environment Setup	• Create React application • Install dependencies (React Router, Material-UI, Axios, Stripe) • Setup project folder structure • Configure routing • Create API service utilities	1	29/09/2025	29/09/2025	2.3
	4.2	Authentication Pages	• Create navigation bar component • Create login page with validation • Create registration page with validation • Implement Google OAuth button • Create OAuth callback handler • Implement token storage and session management • Create protected route component	2	30/09/2025	01/10/2025	4.1
	4.3	Store Management	• Create store creation form with image uploads • Implement store editing functionality • Create store dashboard page • Display product grid on dashboard • Add admin control bar (navigation to products, orders, analytics) • Implement publish/unpublish toggle • Handle loading and error states	3	02/10/2025	06/10/2025	4.2

Phase	WBS Code	Task/Deliverable	Activities/Details	Duration (Days)	Start Date	End Date	Precedence/Dependencies
	4.4	Product Management	• Create product form (add/edit) • Implement image upload handling • Implement product CRUD operations • Create product card component • Display stock level indicators • Add delete confirmation dialog • Handle validation and errors	3	07/10/2025	09/10/2025	4.3
	4.5	Customer Storefront	• Create public store page (accessible via slug) • Create store navigation bar • Display store hero section with branding • Create product grid for customers • Implement shopping cart (Context API) • Create cart drawer/sidebar • Implement cart actions (add, remove, update quantity) • Persist cart to localStorage • Add cart badge to navigation • Handle out-of-stock scenarios	3	10/10/2025	14/10/2025	4.4
	4.6	Checkout & Payment	• Install Stripe React libraries • Create checkout page with cart summary • Create customer information form • Add optional account creation during checkout • Integrate Stripe Elements • Implement payment submission flow • Handle 3D Secure authentication • Create payment success page • Create payment failure page with retry • Clear cart after successful payment • Add comprehensive error handling	4	15/10/2025	20/10/2025	4.5
	4.7	Admin Orders & Analytics	• Create analytics dashboard page • Display key metrics (revenue, order count, top products) • Create orders list page • Implement order detail view • Display order items and customer information • Style with Material-UI components	2	21/10/2025	22/10/2025	4.6
	4.8	Styling & Responsiveness	• Ensure consistent design across all pages • Make all pages responsive (mobile, tablet, desktop) • Add loading spinners and notifications • Polish button styles and transitions • Fix spacing and alignment	1	23/10/2025	23/10/2025	4.7
5	**5**	**Integration & Testing**		7	24/10/2025	03/11/2025	3.9 and 4.8
	5.1	Stripe Configuration	• Create Stripe account • Obtain test API keys • Install Stripe CLI for webhook testing • Configure webhook endpoint • Add API keys to environment files	1	24/10/2025	24/10/2025	3.9 and 4.8
	5.2	API Testing	• Test authentication endpoints (register, login, OAuth) • Test store CRUD operations • Test product CRUD operations • Test order creation and retrieval • Test payment intent creation • Test analytics endpoint • Verify authorization rules • Verify validation rules	2	27/10/2025	28/10/2025	5.1
	5.3	Payment Testing	• Test successful payment flow • Test declined payment scenarios • Test 3D Secure authentication • Test webhook delivery • Verify order creation after payment • Verify stock reduction • Verify payment reference storage	2	29/10/2025	30/10/2025	5.2
	5.4	End-to-End Testing	• Test complete admin workflow • Test guest checkout workflow • Test registered customer workflow • Test across different browsers • Test responsive design on devices • Test edge cases (empty cart, out of stock, session expiration)	1	31/10/2025	31/10/2025	5.3
	5.5	Bug Fixing	• Document identified bugs • Prioritize bugs by severity • Fix critical and high-priority bugs • Retest all fixes	1	03/11/2025	03/11/2025	5.4
6	**6**	**Documentation**		3	04/11/2025	06/11/2025	5.5
	6.1	Technical Documentation	• Create comprehensive README file • Document installation steps for backend • Document installation steps for frontend • Document environment variables setup • Document Stripe configuration • Create API documentation with examples • Update ER diagram	1	04/11/2025	04/11/2025	5.5
	6.2	User Documentation	• Create admin user guide with screenshots • Create customer user guide with screenshots • Document common troubleshooting steps	1	05/11/2025	05/11/2025	6.1
	6.3	Project Report	• Write project introduction and objectives • Describe system architecture • Explain technology stack choices • Include feature screenshots • Document testing methodology • Discuss challenges and solutions • Suggest future enhancements	1	06/11/2025	06/11/2025	6.2
7	**7**	**Deployment & Presentation**		2	07/11/2025	10/11/2025	6.3
	7.1	Demo Preparation	• Clear test data from database • Create database seeder • Generate realistic demo data (stores, products, orders) • Run seeder • Configure Stripe in test mode • Test all workflows with demo data • Prepare demonstration script • Prepare test payment cards for demo	1	07/11/2025	07/11/2025	6.3

Phase	WBS Code	Task/Deliverable	Activities/Details	Duration (Days)	Start Date	End Date	Precedence/Dependencies
	7.2	Presentation	• Create presentation slides • Practice live demonstration • Prepare backup materials (screenshots/video) • Prepare responses for Q&A	1	10/11/2025	10/11/2025	7.1
		TOTAL PROJECT DURATION		**39 days**	**17/09/2025**	**10/11/2025**	

Tasks

Name	Begin date	End date
Project Scope Definition • *Define project objectives and success criteria* • *Identify target users (store owners, customers)* • *Document project constraints and assumptions*	17/09/2025	17/09/2025
Functional Requirements • *Document user authentication requirements* • *Document store and product management requirements* • *Document order processing requirements* • *Document payment processing requirements* • *Document analytics and reporting requirements*	18/09/2025	18/09/2025
Non-Functional Requirements • *Define security requirements* • *Define performance requirements* • *Define scalability requirements*	19/09/2025	19/09/2025
Technology Research & Planning • *Research Stripe payment integration* • *Finalize technology stack (Laravel, React, MySQL, Stripe)* • *Create project timeline and milestones*	22/09/2025	22/09/2025
Database Design • *Design database schema (users, stores, products, customers, orders, order_items)* • *Define table relationships and foreign keys* • *Create Entity-Relationship (ER) diagram* • *Plan data validation rules*	23/09/2025	24/09/2025
API Design • *Design RESTful API endpoints* • *Plan authentication strategy (Laravel Sanctum)* • *Define authorization rules* • *Plan Stripe webhook integration*	25/09/2025	25/09/2025
Frontend Architecture • *Create wireframes for key pages (store creation, product management, storefront, checkout)* • *Design user flows (admin, guest customer, registered customer)* • *Plan component structure and reusability* • *Plan state management strategy (Context API for cart)* • *Design responsive layouts*	26/09/2025	26/09/2025
B.E Environment Setup • *Install Laravel framework* • *Configure database connection* • *Install Laravel Sanctum for authentication* • *Install Laravel Socialite for Google OAuth* • *Configure CORS settings*	29/09/2025	29/09/2025
B.E Database Implementation • *Create database migrations for all tables* • *Run migrations and verify structure* • *Test relationships and constraints*	30/09/2025	30/09/2025

Tasks

Name	Begin date	End date
B.E User Authentication • *Implement user registration and login* • *Implement Google OAuth integration* • *Implement logout functionality* • *Generate and manage Sanctum tokens* • *Add validation and error handling*	01/10/2025	03/10/2025
B.E Store Management • *Create Store model and controller* • *Implement store CRUD operations* • *Handle image uploads (logo, background)* • *Implement publish/unpublish functionality* • *Add authorization middleware*	06/10/2025	07/10/2025
B.E Product Management • *Create Product model and controller* • *Implement product CRUD operations* • *Handle product image uploads* • *Implement stock and category management* • *Add validation and authorization*	08/10/2025	10/10/2025
B.E Customer & Public Store • *Create Customer model and authentication* • *Implement public store API (fetch by slug)* • *Return published stores and products only*	13/10/2025	13/10/2025
B.E Order Management • *Create Order and OrderItem models* • *Implement order creation (guest and registered customers)* • *Generate unique order numbers* • *Calculate order totals* • *Store product snapshots in order items* • *Implement order retrieval for store owners*	14/10/2025	16/10/2025
B.E Analytics • *Create analytics endpoint* • *Calculate revenue and order statistics* • *Identify top-selling products*	17/10/2025	17/10/2025
B.E Payment Integration • *Install Stripe PHP SDK* • *Configure Stripe API keys* • *Implement payment intent creation* • *Implement Stripe webhook handler* • *Handle payment success (create order, update status, reduce stock)* • *Handle payment failures with logging*	20/10/2025	21/10/2025
F.E Environment Setup • *Create React application* • *Install dependencies (React Router, Material-UI, Axios, Stripe)* • *Setup project folder structure* • *Configure routing* • *Create API service utilities*	29/09/2025	29/09/2025

Tasks

Name	Begin date	End date
F.E Authentication Pages • *Create navigation bar component* • *Create login page with validation* • *Create registration page with validation* • *Implement Google OAuth button* • *Create OAuth callback handler* • *Implement token storage and session management* • *Create protected route component*	30/09/2025	01/10/2025
F.E Store Management • *Create store creation form with image uploads* • *Implement store editing functionality* • *Create store dashboard page* • *Display product grid on dashboard* • *Add admin control bar (navigation to products, orders, analytics)* • *Implement publish/unpublish toggle* • *Handle loading and error states*	02/10/2025	06/10/2025
F.E Product Management • *Create product form (add/edit)* • *Implement image upload handling* • *Implement product CRUD operations* • *Create product card component* • *Display stock level indicators* • *Add delete confirmation dialog* • *Handle validation and errors*	07/10/2025	09/10/2025
F.E Customer Storefront • *Create public store page (accessible via slug)* • *Create store navigation bar* • *Display store hero section with branding* • *Create product grid for customers* • *Implement shopping cart (Context API)* • *Create cart drawer/sidebar* • *Implement cart actions (add, remove, update quantity)* • *Persist cart to localStorage* • *Add cart badge to navigation* • *Handle out-of-stock scenarios*	10/10/2025	14/10/2025
F.E Checkout & Payment • *Install Stripe React libraries* • *Create checkout page with cart summary* • *Create customer information form* • *Add optional account creation during checkout* • *Integrate Stripe Elements* • *Implement payment submission flow* • *Handle 3D Secure authentication* • *Create payment success page* • *Create payment failure page with retry* • *Clear cart after successful payment* • *Add comprehensive error handling*	15/10/2025	20/10/2025

Tasks

Name	Begin date	End date
F.E Admin Orders & Analytics *• Create analytics dashboard page • Display key metrics (revenue, order count, top products) • Create orders list page • Implement order detail view • Display order items and customer information • Style with Material-UI components*	21/10/2025	22/10/2025
F.E Styling & Responsiveness *• Ensure consistent design across all pages • Make all pages responsive (mobile, tablet, desktop) • Add loading spinners and notifications • Polish button styles and transitions • Fix spacing and alignment*	23/10/2025	23/10/2025
Stripe Configuration *• Create Stripe account • Obtain test API keys • Install Stripe CLI for webhook testing • Configure webhook endpoint • Add API keys to environment files*	24/10/2025	24/10/2025
API Testing *• Test authentication endpoints (register, login, OAuth) • Test store CRUD operations • Test product CRUD operations • Test order creation and retrieval • Test payment intent creation • Test analytics endpoint • Verify authorization rules • Verify validation rules*	27/10/2025	28/10/2025
Payment Testing *• Test successful payment flow • Test declined payment scenarios • Test 3D Secure authentication • Test webhook delivery • Verify order creation after payment • Verify stock reduction • Verify payment reference storage*	29/10/2025	30/10/2025
End-to-End Testing *• Test complete admin workflow • Test guest checkout workflow • Test registered customer workflow • Test across different browsers • Test responsive design on devices • Test edge cases (empty cart, out of stock, session expiration)*	31/10/2025	31/10/2025

Tasks

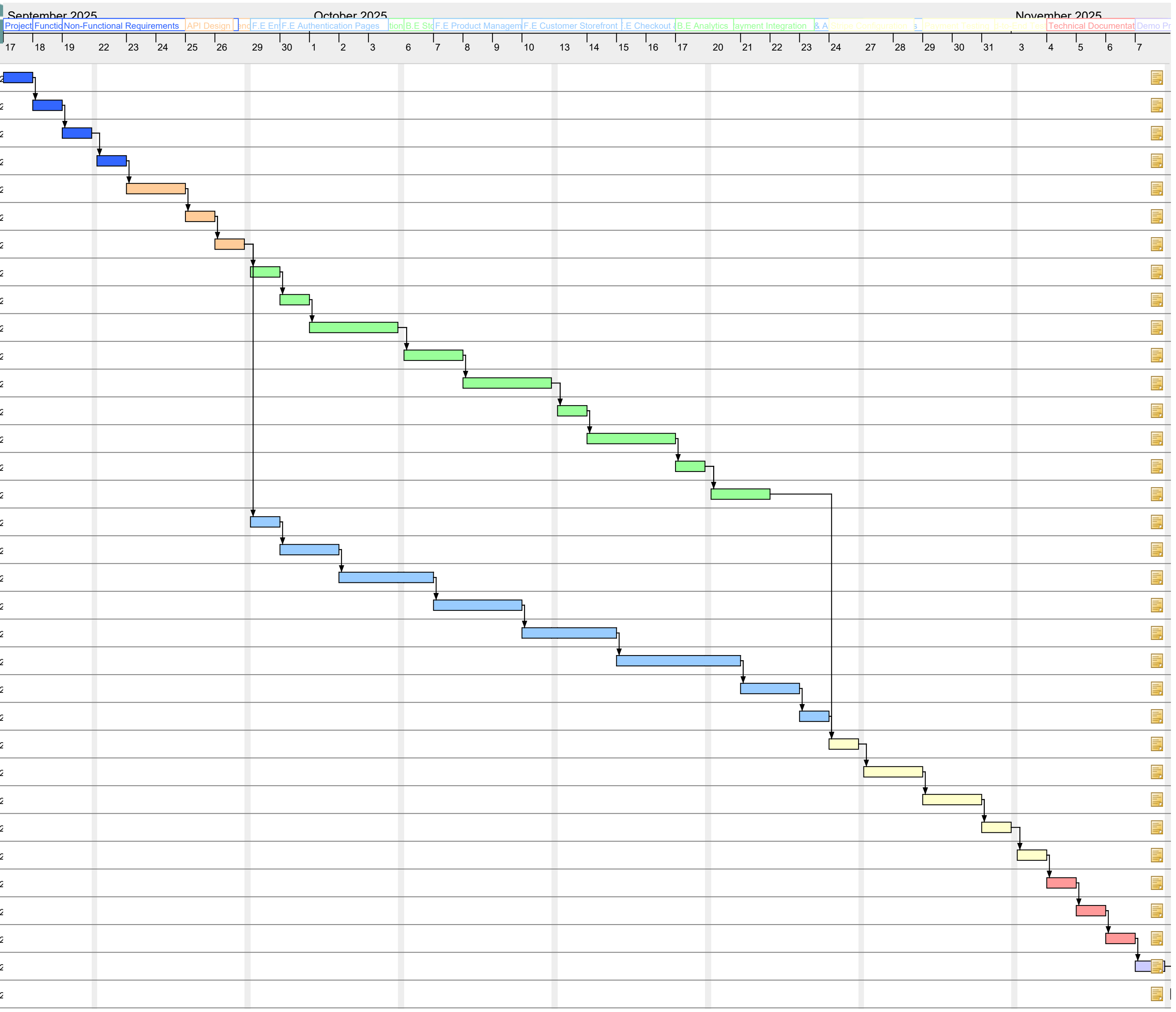
Name	Begin date	End date
Bug FIxing • *Document identified bugs* • *Prioritize bugs by severity* • *Fix critical and high-priority bugs* • *Retest all fixes*	03/11/2025	03/11/2025
Technical Documentation • *Create comprehensive README file* • *Document installation steps for backend* • *Document installation steps for frontend* • *Document environment variables setup* • *Document Stripe configuration* • *Create API documentation with examples* • *Update ER diagram*	04/11/2025	04/11/2025
User Documentation • *Create admin user guide with screenshots* • *Create customer user guide with screenshots* • *Document common troubleshooting steps*	05/11/2025	05/11/2025
Project Report • *Write project introduction and objectives* • *Describe system architecture* • *Explain technology stack choices* • *Include feature screenshots* • *Document testing methodology* • *Discuss challenges and solutions* • *Suggest future enhancements*	06/11/2025	06/11/2025
Demo Presentation • *Clear test data from database* • *Create database seeder* • *Generate realistic demo data (stores, products, orders)* • *Run seeder* • *Configure Stripe in test mode* • *Test all workflows with demo data* • *Prepare demonstration script* • *Prepare test payment cards for demo*	07/11/2025	07/11/2025
Presentation • *Create presentation slides* • *Practice live demonstration* • *Prepare backup materials (screenshots/video)* • *Prepare responses for Q&A*	10/11/2025	10/11/2025

DIP2009 Gantt Project

24 Oct 2025

Gantt Chart

Name	Begin date	End date
Project Scope Definition	17/09/2025	17/09/202
Functional Requirements	18/09/2025	18/09/202
Non-Functional Requirements	19/09/2025	19/09/202
Technology Research & Planning	22/09/2025	22/09/202
Database Design	23/09/2025	24/09/202
API Design	25/09/2025	25/09/202
Frontend Architecture	26/09/2025	26/09/202
B.E Environment Setup	29/09/2025	29/09/202
B.E Database Implementation	30/09/2025	30/09/202
B.E User Authentication	01/10/2025	03/10/202
B.E Store Management	06/10/2025	07/10/202
B.E Product Management	08/10/2025	10/10/202
B.E Customer & Public Store	13/10/2025	13/10/202
B.E Order Management	14/10/2025	16/10/202
B.E Analytics	17/10/2025	17/10/202
B.E Payment Integration	20/10/2025	21/10/202
F.E Environment Setup	29/09/2025	29/09/202
F.E Authentication Pages	30/09/2025	01/10/202
F.E Store Management	02/10/2025	06/10/202
F.E Product Management	07/10/2025	09/10/202
F.E Customer Storefront	10/10/2025	14/10/202
F.E Checkout & Payment	15/10/2025	20/10/202
F.E Admin Orders & Analytics	21/10/2025	22/10/202
F.E Styling & Responsiveness	23/10/2025	23/10/202
Stripe Configuration	24/10/2025	24/10/202
API Testing	27/10/2025	28/10/202
Payment Testing	29/10/2025	30/10/202
End-to-End Testing	31/10/2025	31/10/202
Bug FIxing	03/11/2025	03/11/202
Technical Documentation	04/11/2025	04/11/202
User Documentation	05/11/2025	05/11/202
Project Report	06/11/2025	06/11/202
Demo Presentation	07/11/2025	07/11/202
Presentation	10/11/2025	10/11/202



5. Analysis & Design (Chan Zi Hao B2400169)

5.1 User Requirements (Backend Perspective)

Our platform serves three core user types which require comprehensive backend APIs to meet all users needs. The first user type are store owners or sellers who want complete store management capabilities; the second user type are administrators who require clear system overview; the last user type are customers who demand a seamless and enjoyable shopping experience.

Store owners require APIs to create the storefront, manage product inventory, process orders, and analyse order or sales data. Customers expect secure authentication, smooth product browsing experience and a simple cart management along with payment processing APIs.

To support these users, the backend will provide RESTful APIs that handle business logic, third-party integrations and data persistence. The API layer allows the frontend application to interact with stores, products, orders, and payment systems while maintaining data consistency and security through layers of validation, authentication, and authorization.

5.2 Functional Requirements

FR-1: Authentication & Authorization

The backend API must be built with secure user authentication to protect store data and enforce the rule that users can only access their own resources. We decided to implement 2 authentication methods (traditional method & Google OAuth) to accommodate different user preferences.

FR-1.1: User Registration

Store owners need to be able to create an account using their own email address along with a secure password. The function will activate a new account and generate the required authentication token so users can access the platform upon successful registration.

Key Requirements:

- Each email address can only be registered once
- Passwords meet basic security standards (>8 characters)
- Account activation occurs instantly upon successful registration
- Authentication token generated automatically by the platform for seamless login experience

FR-1.2: User Login

Users that have registered must be able to log into their accounts securely using email and password or Google OAuth credentials. Each login generates a new authentication token without destroying the previous token thus allowing users to be logged in from multiple devices at the same time.

Key Requirements:

- Controller verifies whether email exists and password matches stored hash password
- New token is generated on every login
- Previous token remain valid (to support multi-device)
- Failed login should not reveal whether email exists for security purposes

FR-1.3: Google OAuth Integration
Our platform will support Google OAuth login to improve user experience and reduce friction by allowing users to register or log in using their existing Google accounts without creating a new password.

Key Requirements:
- First-time Google users get accounts created automatically
- Returning Google users are logged in instantly
- Google ID is stored for future authentication
- No password required or stored for OAuth users

Integration Requirements:
- Installation of Laravel Socialite
- Configuration of Google OAuth credentials (Client ID & Secret)
- Callback URL defined in Google Developer Console

FR-1.4: User Logout
Users must be able to log out securely, we will do so by revoking the user's current authentication token. This is crucial for security purposes, especially on shared devices.

Key Requirements:
- Current token must be deleted from database
- User has to re-authenticate to access resources again
- Other active sessions from different devices remain logged in
- No data is lost during and after logout

FR-1.5: Authorization & Access Control
Besides authentication, the platform must have authorization rules enforced to ensure users can only access and modify their own data. This will prevent unauthorized access to other stores'.

Key Requirements:
- All protected endpoints must verify for valid authentication token
- Users can only view and edit their own store and products belonging to their own stores
- Store owners can only see orders for their own stores
- Public browsing endpoints (for the customer facing public storefront) do not require authentication

Implementation Approach:
- Laravel Sanctum middleware will protect sensitive endpoints
- Authorization checks verify resource ownership before allowing any operations

FR-2: Store Management
Each user can create and manage their own online stores with customizable branding. The store acts as a hub for products, orders and sales analytics.

FR-2.1: Create Store
Users need the ability to create their own online stores with their own branding which includes their name, logo, background image and descriptive information. This will be their first step in setting up their e-commerce presence to extend their reach over social media.

Key Requirements:

- Store name and slug (URL-friendly identifier) must be provided
- Store slug must be unique across platform for public URLs
- Logo and background images are set to be optional but recommended for professionalism
- New stores are not immediately publicly visible
- Stores are automatically associated to the creating user
- Contact information (email, address) stored for customer communication

FR-2.2 View Store Details
Users need access to all information about their store to review or edit it based on their requirements.

Key Requirements:

- Return all store data including images
- User can only view their own store

FR-2.3 Update Store Information
Store owners must be able to modify their own store's information as their business grows, changes or pivots.

Key Requirements:

- Only store owner can update store information
- Store owner can only update specific field
- Old images are deleted upon being replaced
- Store slug cannot be changed once store is created (to maintain public URL consistency)

FR-2.4 Publish/Unpublish Store
Store owners should have the right to decide when to make their store publicly visible or when to temporarily close their store for any reasons, whether it is to prepare or update inventory, or update store information.

Key Requirements:

- Two states: "published" (publicly visible) or "draft" (owner access only)
- Only published stores appear in public listings and search
- Products in draft stores are not publicly accessible
- Owner can toggle store status whenever
- Store must have at least one product before publishing

FR-2.5 Delete Store
Users can permanently remove their store if they no longer want it.

Key Requirements:

- Complete deletion of store data and all associated data
- Cascading delete which deleted all products and orders related to store
- Images deleted from storage
- Action is permanent and irreversible
- Must require double confirmation on frontend to prevent accidents

FR-3: Product Management
Store owners have full control over their product inventory, they can add new items, update information, manage stock levels, and remove discontinued products

FR-3.1 Add New Product
Store owners must be able to add products to their inventory with all information required for customers to make an informed decision on their purchase

Key Requirements:

- Product must be associated with a particular store
- User must own the store to add a new product
- Required information: name, category, price, initial stock quantity
- Optional information: description, product image
- Price must be more than zero
- Stock quantity cannot be negative
- Images stored securely

FR-3.2: View Products
Store owners need to be able to see their product inventory with filtering and search capabilities for store owner's effective management and customer convenience.

Key Requirements:

- View all products owned
- Filter by category
- Search by product name
- Only show products from user's own store
- Includes stock levels for store owner's convenience

FR-3.3: View Product Details
Users need access to full product information for review or editing purposes

Key Requirements:

- Show all product data including pricing, stock, and image
- User can only view products from their own store

FR-3.4 Update Product
Product information needs to be updated as prices change, stock replenishes, or descriptions are refined.

Key Requirements:

- Only product owner (store owner) can update
- All fields can be modified
- Old product image will be deleted when replaced
- Supports partial modification of product information

FR-3.5 Remove Product
Discontinued products need to be removed from product inventory

Key Requirements:

- Deletion of product is permanent
- Product image deleted from storage
- User can only delete their own product

FR-3.6 Automatic Stock Management
Stock quantities must be updated as soon as orders are placed to prevent overselling and maintain inventory accuracy

Key Requirements:

- Stock only reduced after payment is successful
- Triggered automatically by payment webhook
- Quantity reduced must match the order quantity (to maintain data integrity)
- Stock cannot go below zero
- Product with zero stock should display as unavailable to customers

FR-4: Order Management
The system must handle the entire order lifecycle, from order creation to payment and all the way until completed. It will support both guest customers and registered customers.

FR-4.1: Create Order
When customers complete their checkout, the system will create an order record with all the order details and relevant customer information.

Key Requirements:

- Supports both registered customers and guest checkouts
- For guests: Customer information must be filled up manually
- For registered customers: Customer information is auto-filled
- Automatically generates unique order number for tracking
- Calculate total amount
- Order status default to "pending" until payment completed
- Validates all products exist and have sufficient stock

FR-4.2: View Store Orders
Store owners can see all orders placed for their store for them to manage order fulfilment and track sales.

Key Requirements:
- Shows all orders for owner's store
- Optional filtering by order status
- Ordered by newest order

FR-4.3: View Order Details
Store owners need full order information to process order and handle customer's query

Key Requirements:
- Show all order information: customer details, items ordered, payment info
- Show order status and timestamps
- User can only view orders from their own store

FR-4.4: Update Order Status
Store owners can update order status as they process and ship orders

Key Requirements:
- Allow changing of order status
- Valid transitions: pending -> paid -> shipped -> completed
- Cannot modify completed orders
- Only store owner can update their order status

FR-5: Payment Processing
The platform integrates with Stripe to safely process online payments. We use Stripe's payment intent flow for PCI compliance, a set of security standards set by the Payment Card Industry Security Standards Council (PCI SSC) to protect cardholder data from theft and fraud.

FR-5.1: Create Payment Intent
When a customer reaches checkout, the backend of our platform creates a Stripe payment intent and returns a client secret for the frontend to complete the payment.

Key Requirements:
- Accepts payment amount and order data from checkout
- Creates payment intent in Stripe
- Amount is converted to smallest currency unit (sen)
- Order data is stored in payment intent metadata for webhook processing
- Returns client_secret to frontend for payment confirmation
- Default currency is set to MYR

Integration Requirements:
- Installation and configuration of Stripe PHP SDK
- Stripe API keys in the backend environment configuration (.env)
- Minimum amount of RM0.50 for validation

FR-5.2: Webhook Even Handler

Stripe will send a webhook notification when a payment event occurs. The system must receive and process these events to update the order status

Key Requirements:

- Receive POST requests from Stripe webhook endpoint
- Verifies webhook signature to ensure requests is from Stripe
- Processes relevant events whether payment success or failed
- Responds in less than 10 seconds to avoid retry storms
- Handles duplicate events gracefully

FR-5.3: Process Successful Payments
When Stripe confirms that payment is successful, the backend must create or update the order and reduce the product's stock count.

Key Requirements:

- Extract payment intent ID and metadata with order information
- Create order record from order information in metadata
- Update order status
- Store payment reference in order
- Reduce product stock count for all ordered products
- Log successful payment

FR-5.4: Process Failed Payments
When payment fails, the system should log the failure for debugging purposes

Key Requirements:

- Extract payment intent ID and failure reason
- Log failure details with error code
- Mark order as pending
- Do not reduce stock count

FR-5.5: Webhook Signature Verification
All incoming webhook requests must be verified to ensure the request actually comes from Stripe and hasn't been meddled with.

Key Requirements:

- Extract Stripe-Signature header from request
- Use Stripe's webhook signing secret to check for HMAC signature
- Reject requests with missing signatures
- Use Stripe SDK's built-in verification method

FR-6: Analytics
The platform will provide comprehensive sales analytics with performance metrics for store owners to see.

Key Metrics:

- Total Revenue: Sum of all successful orders

- Total Orders: Sum of all orders
- Order Status Breakdown: Total orders in each status ('pending', 'paid', 'shipped', 'completed', 'cancelled')
- Top Selling Products: Five best-performing products

5.3 Non-Functional Requirements
NFR-1: Security
The most crucial non-functional requirement in this project is security, as our platform has authentication and authorization, we are storing personal information which are private and confidential. Thus, we must ensure all protected endpoints require a valid authentication token before we return any information from the backend. Not only that, user passwords must be hashed before storing in the database to keep password secure. With that, even if the database is leaked or compromised, the actual password cannot be found. Additionally, it is important that users can only access their own stores, products, and orders. Users must not be able to access other resources. Plus, as we use Stripe to accept payment, we must ensure Stripe webhook signatures are verified before proceeding with the payment flow. Last but not least, OAuth tokens for Google account login users have to be validated before a session is created to prevent unauthorized access.

The other critical part of security that requires attention will be data protection. All input must be valid and sanitized to prevent SQL injection or Cross-Site Scripting (XSS) attacks. Furthermore, payment credentials should only be stored in the environment variables (.env). Besides that, Cross-Origin Resource Sharing (CORS) must be configured to only allow the frontend domain requests and reject all other requests from any other domain or port. Last but not least, no card data should ever be stored on our server and it should be fully handled by Stripe.

By enforcing the security best practices, we can protect user accounts, prevent unauthorized access, and ensure we are PCI compliant.

NFR-2: Performance
The response time targets for different criterias are as follows. We aim to complete simple queries such as login under 200ms. For complex queries such as listing entire stores and products, we aim to have it done under 500ms. As for analytics calculation, we target to complete it in under 1000ms. For image uploads, our aim is under 3000ms.

We can optimize query response time by using database indexes such as user_id, store_id, product_id as the database is able to zoom in directly on the specific row that matches the query instead of doing a full table scan. Not only that, we can eager load to prevent N+1 query problems using built in functions in Laravel PHP to retrieve all related data in a single, optimized query rather than triggering multiple additional database calls for each record. Additionally, images are stored in filesystems instead of the database which significantly decreases response time. Furthermore, we delete old images when they are replaced to save storage.

The end goal is to support over 50 concurrent users without degradation while ensuring responsive user experience and efficient resource usage.

NFR-3: Scalability
All APIs are stateless with no server-side session storage which enables horizontal scaling. Besides that, our database has a capacity to support up to 1,000 users, 10,000 products, and 50,000 orders. Even with that many records, it will only be around 100MB which is definitely sustainable. Not only that, our service is independent of other third-party services. Hence, even if Stripe webhook is delayed or Google OAuth is unavailable, it is still able to run without any major issues. This design sets the platform up for growth as the user base continues to increase.

NFR-4: Reliability
In order to design a reliable system, comprehensive error handling is mandatory. All errors must be logged with appropriate details and the error messages should be user-friendly. It should not expose any SQL queries or stack traces. Plus, it should have a consistent JSON error format as best practice.

Data integrity is also a crucial factor to ensure reliability of a system. For example, cascading deletion is important so that when a store is removed, products and orders associated with it are also deleted. Furthermore, stock quantities should never turn negative as that is physically impossible. Last but not least, foreign key constraints must be established to maintain data integrity throughout the database.

Our platform is aiming for 99% availability and uptime during business hours. Additionally, daily database backups will ensure most data can be recovered even if any breach or compromise occurs. Overall, all of the above will ensure our platform maintains data consistency and system stability.

NFR-5: Maintainability
The entire Laravel PHP codebase follows Laravel coding standards and best practices which guarantees readability of code. Besides that, meaningful naming conventions ensure anyone that sees it can immediately understand what it means. Additionally, the entire database schema is documented with the Entity-Relationship Diagram (ERD). Furthermore, all environment variables are stored in the .env file to ensure it never gets pushed to the cloud and is secure. Last but not least, the entire repository is managed in Git with meaningful commit messages for version control. All of these efforts allow future development and debugging to be easy for anyone working on it.

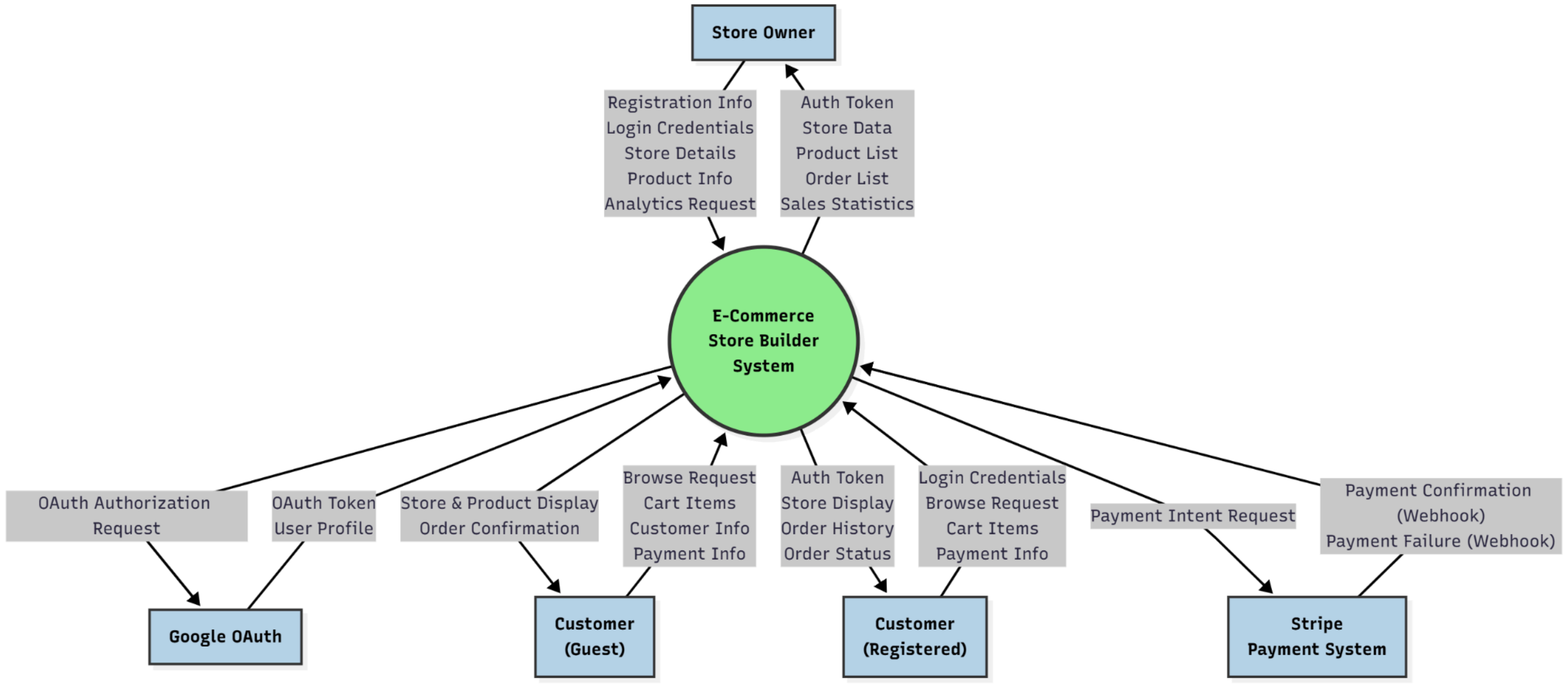
NFR-6: API Standards
All APIs follow RESTful API design. This includes GET (retrieve), POST (create), PUT (update), and DELETE (remove), along with standard HTTP status codes such as 200, 201, 401, 403, 404, 422, and 500.

In addition to that, all response formats for APIs are in JSON format which is considered the golden standard. Responses will always follow a consistent structure which starts with the message, then the data or errors. These API standards are crucial in ensuring predictable API behaviour for frontend integration.

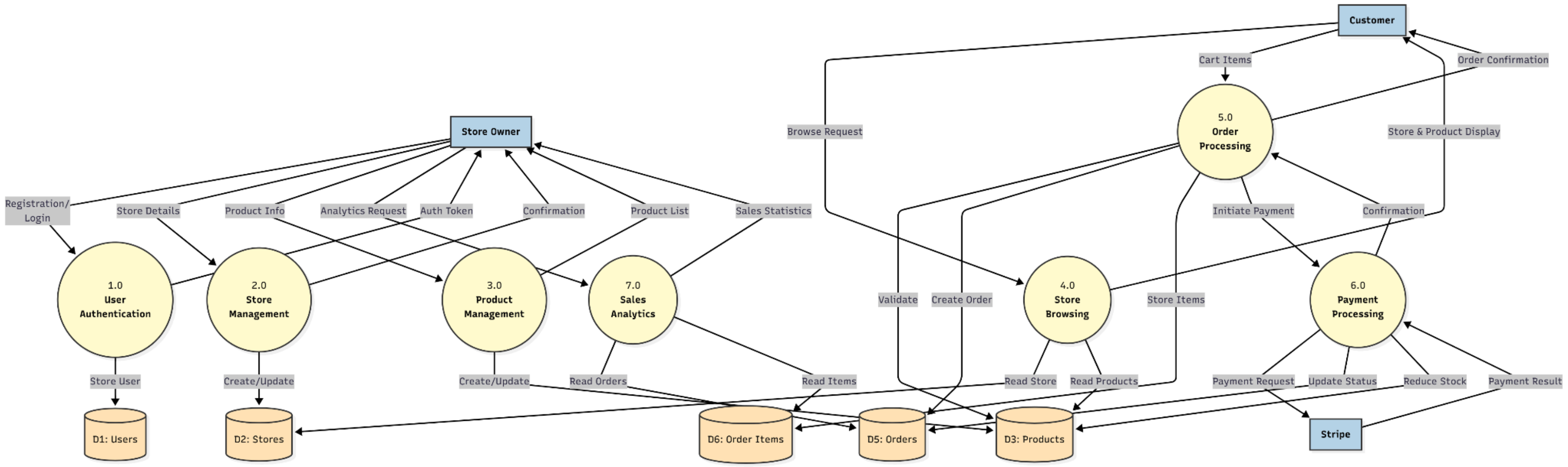
NFR-7: Compliance
We are dedicated to maintaining PCI DSS compliance by adhering to a strict no card storage policy when integrating Stripe as our payment gateway. Additionally, we abide by standard data privacy policies such as ensuring only store owners have access to customer data. Last but not least, we follow web standards such as UTF-8 encoding and proper Content-Type headers.

5.4 Logical Process Model
Before beginning our implementation, we designed two diagrams to assist us in understanding how data should flow through our system. With that, we managed to identify core business processes and their relationships without being overwhelmed by technical details.

The logical model consists of two layers, the Context Diagram gives a higher level overview of the entire system, showing how each entity interacts with our platform; the Data Flow Diagram goes deeper into the seven main processes which shows the important dependencies which require considerations. Below are both diagrams representing our system’s logical architecture:



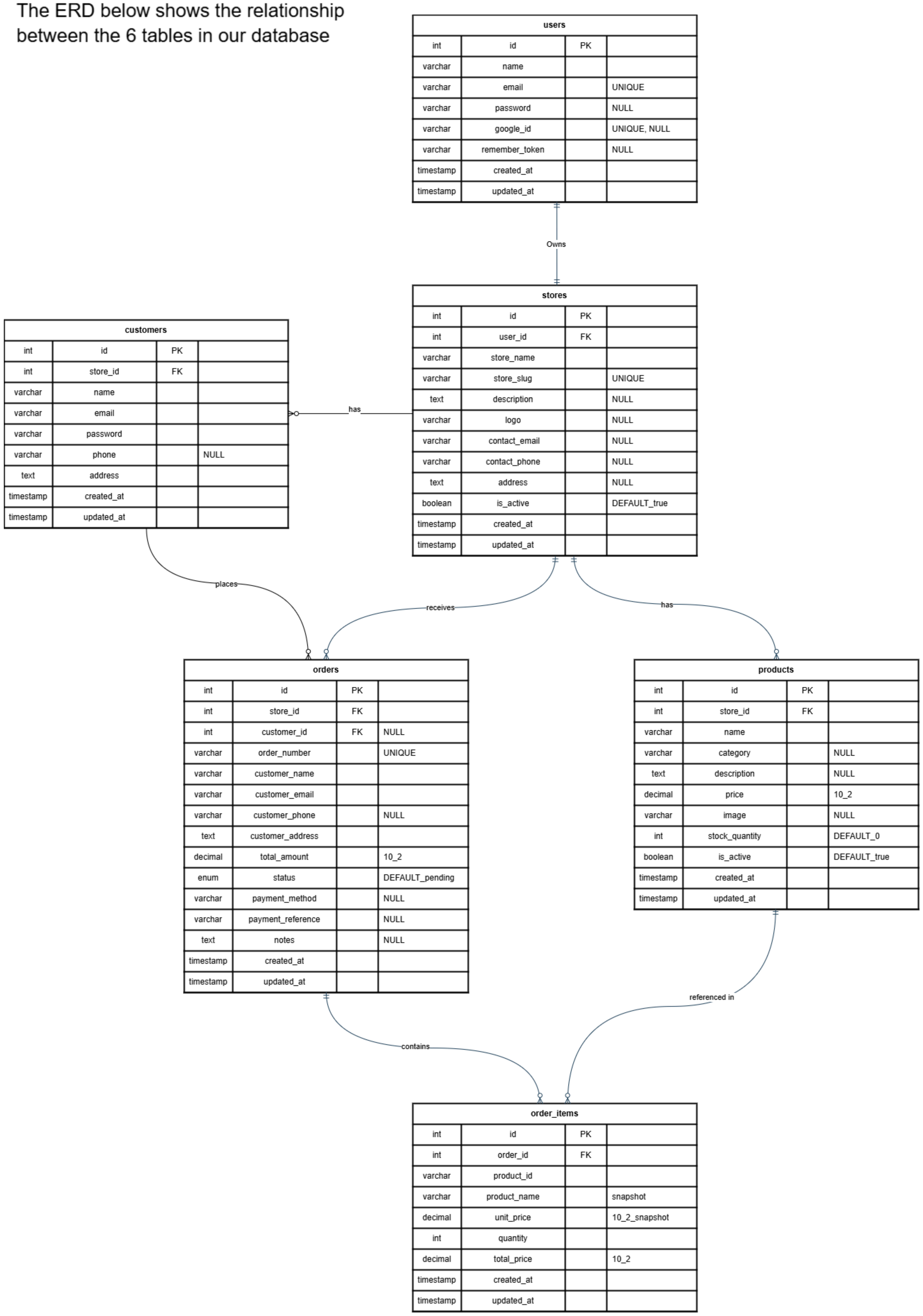
5.4.1 Context Diagram



5.4.2 Data Flow Diagram

5.5 Entity Relationship Diagram (ERD)

The ERD below shows the relationship between the 6 tables in our database



users			
int	id	PK	
varchar	name		
varchar	email		UNIQUE
varchar	password		NULL
varchar	google_id		UNIQUE, NULL
varchar	remember_token		NULL
timestamp	created_at		
timestamp	updated_at		

stores			
int	id	PK	
int	user_id	FK	
varchar	store_name		
varchar	store_slug		UNIQUE
text	description		NULL
varchar	logo		NULL
varchar	contact_email		NULL
varchar	contact_phone		NULL
text	address		NULL
boolean	is_active		DEFAULT_true
timestamp	created_at		
timestamp	updated_at		

customers			
int	id	PK	
int	store_id	FK	
varchar	name		
varchar	email		
varchar	password		
varchar	phone		NULL
text	address		
timestamp	created_at		
timestamp	updated_at		

orders			
int	id	PK	
int	store_id	FK	
int	customer_id	FK	NULL
varchar	order_number		UNIQUE
varchar	customer_name		
varchar	customer_email		
varchar	customer_phone		NULL
text	customer_address		
decimal	total_amount		10_2
enum	status		DEFAULT_pending
varchar	payment_method		NULL
varchar	payment_reference		NULL
text	notes		NULL
timestamp	created_at		
timestamp	updated_at		

products			
int	id	PK	
int	store_id	FK	
varchar	name		
varchar	category		NULL
text	description		NULL
decimal	price		10_2
varchar	image		NULL
int	stock_quantity		DEFAULT_0
boolean	is_active		DEFAULT_true
timestamp	created_at		
timestamp	updated_at		

order_items			
int	id	PK	
int	order_id	FK	
varchar	product_id		
varchar	product_name		snapshot
decimal	unit_price		10_2_snapshot
int	quantity		
decimal	total_price		10_2
timestamp	created_at		
timestamp	updated_at		

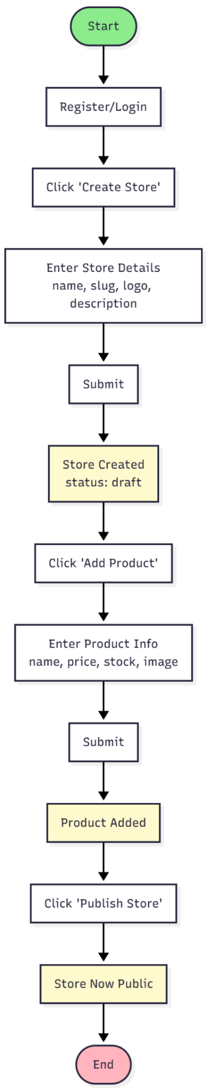


Diagram 5.6.1

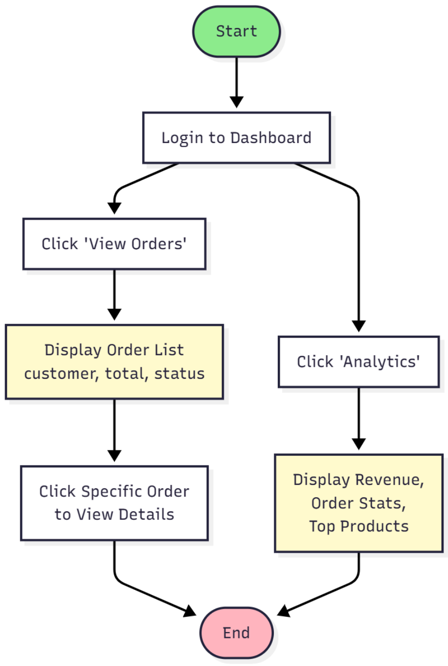


Diagram 5.6.2

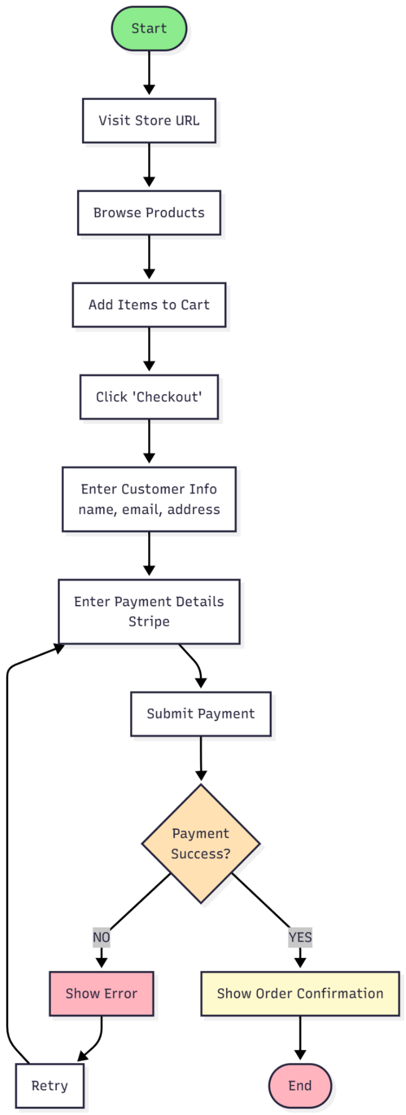


Diagram 5.6.3

5.6 User Flow

The three flowcharts above show the user flow when using our platform. **Diagram 5.6.1** is the store owner setup flow which presents how a store owner builds their own e-commerce store with our platform. **Diagram 5.6.2** is the user flow of store owners viewing their orders or analytics. Last but not least, **Diagram 5.6.3** is the customer browsing and checkout flow which shows how customers can see the e-commerce stores and buy from it.

5.7. Hardware and Software Specifications

5.7.1 Backend Hardware Requirements

Below shows the minimum hardware requirements to run our platform:
Processor: Dual-core 2.0 GHz
RAM: 4GB
Storage: 10GB of available space
Network: Stable internet connection for Stripe API

Below shows the recommended hardware specifications to run our platform smoothly:
Processor: Quad-core 2.5GHz+
RAM: 8GB+
Storage: 20GB SSD
Network: High-speed internet (10Mbps+)

5.7.2 Backend Software Requirements

The table below shows the individual components of the backend along with the corresponding framework or library version and some description on its usage.

Component	**Specification**	**Purpose**
Backend Framework	Laravel 12.x	PHP framework for API development
Programming Language	PHP 8.2+	Server-side language
Database	MySQL 8.0+	Relational database for data storage
Authentication	Laravel Sanctum 4.0	Token-based API authentication
OAuth Provider	Laravel Socialite 5.23	Google OAuth integration
Payment SDK	Stripe PHP SDK 13.x	Payment processing integration
Dependency Manager	Composer 2.x	PHP package management

5.8 Physical Process Model

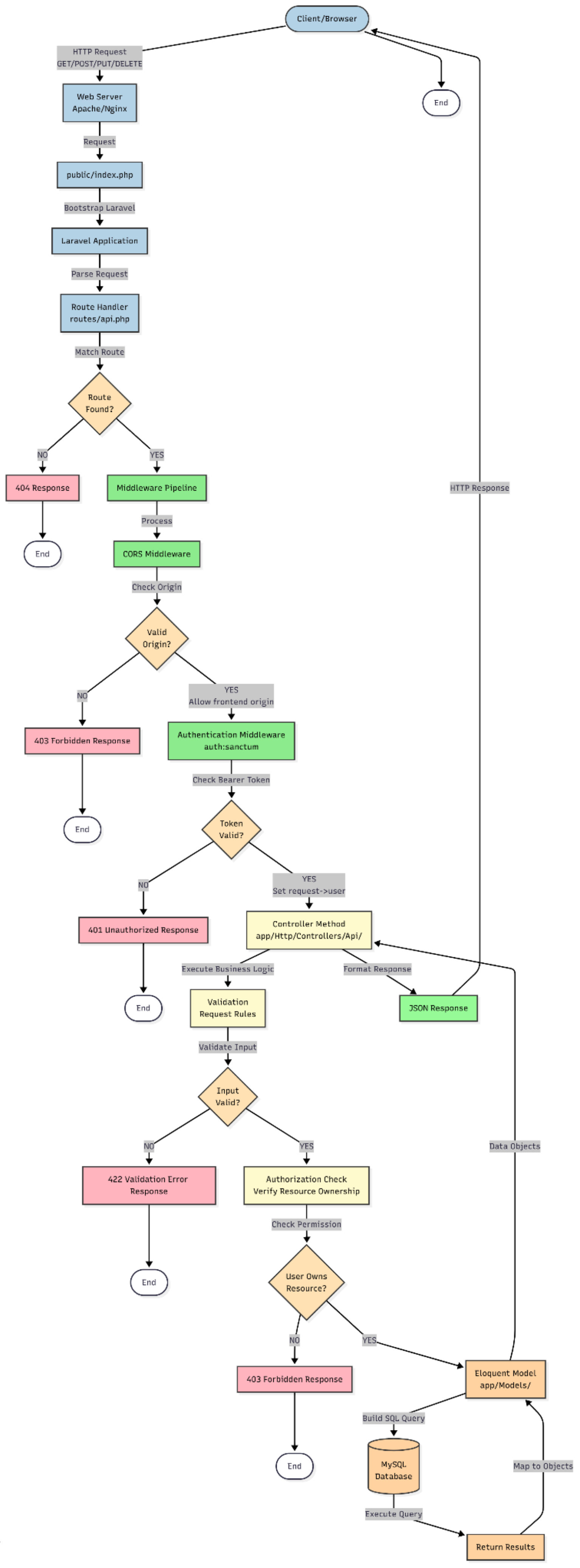
The diagram presented is a Laravel Request Lifecycle which shows the entire process of how a request sent to the backend is processed.

When the frontend client sends a HTTP Request, it goes through the web server to request the index.php. However, since we are using Laravel, it will send the request to our routes/api.php to parse the request. If the route is defined, it will proceed, if it does not exist, the server will return a 404 response to the client.

After the route is found, it will go through the CORS Middleware to ensure that this origin is allowed, if it is not, the server will return a 403 Forbidden Response. If it is the allowed frontend origin, the request will go through auth:sanctum, an authentication middleware to check whether the bearer token found in the header of the request is valid.

An invalid bearer token will return a 401 Unauthorized Response while a valid token will bring the request all the way to the controller methods to process. Then, depending on the method, it will go through input validation to ensure the request body is providing the desired input. An invalid input will result in a 422 Validation Error response while a valid input will then proceed.

The last check will be ensuring the user requesting actually owns the resource it is trying to get. If they do not own it, a 403 Forbidden Response will be returned. If the user owns it, Laravel's built in Eloquent Model will query the MySQL database, return the results to the method, and finally a JSON response will be sent back to the client.



6. Technical Implementation (Chan Zi Hao B2400169)

6.1 Overview

Our project is an e-commerce builder for social media sellers and I am in charge of developing the backend API for this project. I have decided to use Laravel 12.0 as the framework and PHP 8.4 as the language because I have experience using them during my internship and I believe Laravel is a comprehensive framework capable of fulfilling the requirements of this project. Overall, the backend APIs are all implemented using the RESTful API architecture.

The system is designed to allow store owners to create and manage their own e-commerce store, as well as a built in customer authentication for their own store too. The backend follows an API-first design pattern so the development of the frontend and backend can be done simultaneously with everyone on the same page. For authentication and authorization, I utilized Laravel Sanctum for token-based authentication and Google OAuth using Laravel Socialite for social login.

For the backend APIs, I have implemented a comprehensive CRUD operation for my stores, products, orders, and customers table. The database uses SQLite for prototyping with Eloquent ORM (Laravel's built in feature) for data management which simplifies query building for developers. The database can be easily migrated to MySQL or PostgreSQL as the project moves to the production phase.

6.2 Architecture Pattern

The development of all APIs follow the RESTful API architecture strictly. All APIs are resources that can be accessed through a Unique Resource Identifier (URI). For example, to login, the frontend must call '/register'. Besides that, all APIs developed are stateless, which means the server (backend) does not store any context between requests from the client. Every single client request must contain all necessary information for the controller to process the request. That is why in every request from the frontend for protected resources such as '/stores', the header must contain a bearer token, as the server will not remember the last time you showed it one. Additionally, our project follows a client-server architecture where the frontend React.js is the client and my Laravel backend is the server. Hence, both frontend and backend can be developed simultaneously. Last but not least, I follow standardized operations using HTTP methods such as GET and POST to perform CRUD (Create, Read, Update, Delete) operations on all resources and always return responses in JSON format.

As I am using Laravel PHP for the backend of this project, the backend follows a MVC (Model, View, Controller) design pattern. Models in Laravel are the simple way I can interact with my database using Laravel's Eloquent Object-Relational Mapper (ORM) which promotes code readability and maintainability. View is how you would display GUI using Laravel PHP only but in this project, we do not use it as we are using React.js for the GUI as Laravel Blade is static and rendered server-side while React.js is dynamic and client-side which is more ideal. Controller is where you write all the CRUD operations that process requests from the client and return them a response in JSON format. MVC architecture is a great way to separate between data (Model, the GUI (View), and the logic (Controller).

Module	**Description**	**Technologies Used**	**Key Features**
User Authentication	Handles store owner registration, login, JWT-based authentication, and Google OAuth integration	- Laravel Sanctum 4.0 - Laravel Socialite 5.23 - Bcrypt password hashing	● Token-based API authentication ● Google OAuth social login ● Stateless authentication ● Password hashing (bcrypt, 12 rounds) ● Token revocation on logout
Customer Authentication	Separate authentication system for shop customers with store-scoped isolation	- Laravel Sanctum - Hash facade	● Per-store customer accounts ● Store-scoped email uniqueness ● Independent token system ● Customer registration & login per store
Store Management	Allows CRUD operations on stores with image uploads, publishing, and customization	- Laravel Storage - File validation - Eloquent relationships	● Store creation with logo & background images ● Store slug-based public URLs ● Store activation/publishing (requires ≥1 active product) ● Authorization checks (user can only manage own store) ● Automatic old image deletion on update
Product Management	Enables product catalog management with categories, images, and stock tracking	- Eloquent ORM - Storage facade - Query filtering	● Product CRUD operations ● Image upload & storage (public disk) ● Category-based organization ● Stock quantity management ● Active/inactive status toggle ● Filtering by category and status

Order Processing	Manages order lifecycle from creation to completion with status tracking	- Eloquent relationships - Order status enum	● Order creation with customer details ● Unique order number generation ● Order status lifecycle (pending→paid→shipped→completed/cancelled) ● Payment method & reference tracking ● Order items with product snapshots ● Historical pricing preservation
Public Store Display	Provides public-facing store pages accessible via slug without authentication	- Route parameters - Query optimization	● Slug-based store lookup ● Active products display ● Store activation check ● No authentication required
File & Image Management	Handles upload, storage, and deletion of store and product images	- Laravel Storage (public disk) - File validation	● Image upload (JPEG, PNG, JPG, GIF, WebP) ● Max 2MB file size validation ● Automatic old file cleanup ● Public disk storage for frontend access
Data Validation	Comprehensive input validation across all endpoints	- Laravel Validation - Form Requests	● Email uniqueness validation ● Price & numeric validation ● File type/size validation ● Custom error messages ● Unique constraint enforcement

6.3 Tools & Technologies Used

Backend Framework:

- Laravel 12.0 (PHP Framework)
- PHP 8.2+

Datebase:

- SQLite (Development)
- MySQL/PostgreSQL (Production)
- Eloquent Object-Relational Mapping (ORM) for database abstraction

API & Authentication:

- RESTful API design
- Laravel Sanctum 4.0 (Token-based API authentication)
- Laravel Socialite 5.23 (Google OAuth integration)
- Bcrypt password hashing (12 rounds)

Key Dependencies:

- laravel/sanctum: API authentication via tokens
- laravel/socialite: OAuth social login (Google)
- laravel/tinker: Laravel's built-in CLI based debugging tool

Security Features:

- CORS middleware (configured to allow frontend domain only)
- Password hashing with Bcrypt
- Token-based authentication
- Authorization checks on all protected routes (sanctum)
- Automatic file delete on update

6.4 API Endpoints

The tables below show the API endpoints implemented segregated by each module.

6.4.1 Store Owner Authentication Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
POST	/api/register	Store owner registration	No	name, email, password, password_confirmation	User object with token

POST	/api/login	Store owner login	No	email, password	User object with token
POST	/api/logout	Store owner logout	Yes	None	Success message
GET	/api/user	Get authenticated store owner details	Yes	None	User object
GET	/api/auth/google	Google OAuth redirect	No	None	Redirect to Google
GET	/api/auth/google/callback	Google OAuth callback	No	code (query param)	User object with token

6.4.2 Customer Authentication Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
POST	/api/customer/register	Customer registration per store	No	store_id, name, email, password, password_confirmation	Customer object with token
POST	/api/customer/login	Customer login per store	No	store_id, email, password	Customer object with token

POST	/api/customer/logout	Customer logout	Yes	None	Success message

6.4.3 Store Management Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
GET	/api/stores	List user's store	Yes	None	Store object
POST	/api/stores	Create new store with image upload	Yes	name, description, logo (file), background_image (file)	Store object
GET	/api/stores/{id}	View specific store	Yes	id (path param)	Store object with products
PUT	/api/stores/{id}	Update store with images	Yes	name, description, logo (file), background_image (file), is_published	Updated store object
DELETE	/api/stores/{id}	Delete store	Yes	id (path param)	Success message

6.4.4 Product Management Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary

GET	/api/products	List products with filtering	Yes	category, is_active (query params)	Array of products
POST	/api/products	Create product with image	Yes	store_id, name, description, price, stock_quantity, category, image (file)	Product object
GET	/api/products/{id}	View specific product	Yes	id (path param)	Product object
PUT	/api/products/{id}	Update product with image	Yes	name, description, price, stock_quantity, category, image (file), is_active	Updated product object
DELETE	/api/products/{id}	Delete product	Yes	id (path param)	Success message

6.4.5 Order Management - Store Owner Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
GET	/api/orders	List all orders for store with filtering	Yes	status, date_from (query params)	Array of orders
GET	/api/orders/{id}	View order details with items	Yes	id (path param)	Order object with items

PUT	/api/orders/{id}/status	Update order status	Yes	status (paid, shipped, completed, cancelled)	Updated order object
DELETE	/api/orders/{id}	Cancel/delete order	Yes	id (path param)	Success message
GET	/api/orders/statistics	Order statistics summary	Yes	None	Statistics object with counts

6.4.6 Order Management - Customer Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
POST	/api/customer/orders	Create order/checkout	Yes	store_id, customer_name, customer_email, customer_phone, items[], payment_method	Order object with order number
GET	/api/customer/orders	List customer's order history	Yes	None	Array of orders
GET	/api/customer/orders/{id}	View specific order details	Yes	id (path param)	Order object with items

PUT	/api/customer/orders/{id}/cancel	Cancel pending order only	Yes	id (path param)	Updated order object

6.4.7 Payment Processing Module - Stripe

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
POST	/api/payments/stripe/create-intent	Create payment intent	Yes	order_id, amount	Payment intent object with client_secret
POST	/api/payments/stripe/confirm	Confirm payment	Yes	payment_intent_id, order_id	Payment confirmation object
GET	/api/payments/{reference}/status	Check payment status	Yes	reference (path param)	Payment status object
POST	/api/webhooks/stripe	Stripe webhook handler	No	Stripe signature verification	Success response

6.4.8 Payment Processing Module - FPX

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary

POST	/api/payments/fpx/create	Create FPX payment	Yes	order_id, bank_code, amount	FPX payment object with redirect URL
POST	/api/payments/fpx/callback	FPX payment callback	No	FPX response data	Success response
POST	/api/webhooks/fpx	FPX webhook handler	No	FPX signature verification	Success response

6.4.9 Analytics & Dashboard Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
GET	/api/analytics/sales	Sales statistics	Yes	period, date_from, date_to (query params)	Sales data object
GET	/api/analytics/revenue	Revenue breakdown	Yes	period (query param)	Revenue data object
GET	/api/analytics/products/top-selling	Top selling products	Yes	limit, date_from (query params)	Array of top products
GET	/api/analytics/orders/summary	Order summary statistics	Yes	None	Order statistics object

GET	/api/analytics/customers/count	Customer statistics	Yes	None	Customer count object
GET	/api/analytics/dashboard	Overall dashboard	Yes	None	Aggregated dashboard data

6.4.10 Public Endpoints Module

Method	Endpoint	Description	Auth	Parameters/Body	Response Summary
GET	/api/public/stores/{slug}	View published store with active products	No	slug (path param)	Store object with products
GET	/api/public/stores/{slug}/products/{id}	View single product details	No	slug, id (path params)	Product object
GET	/api/public/stores/{slug}/categories	List all categories in store	No	slug (path param)	Array of categories
GET	/api/public/products/search	Search products across stores	No	q, category, min_price, max_price (query params)	Array of products

GET	/api/public/stores/{slug}/products	Filter/search products within store	No	slug (path param), category, min_price, max_price, sort (query params)	Array of products

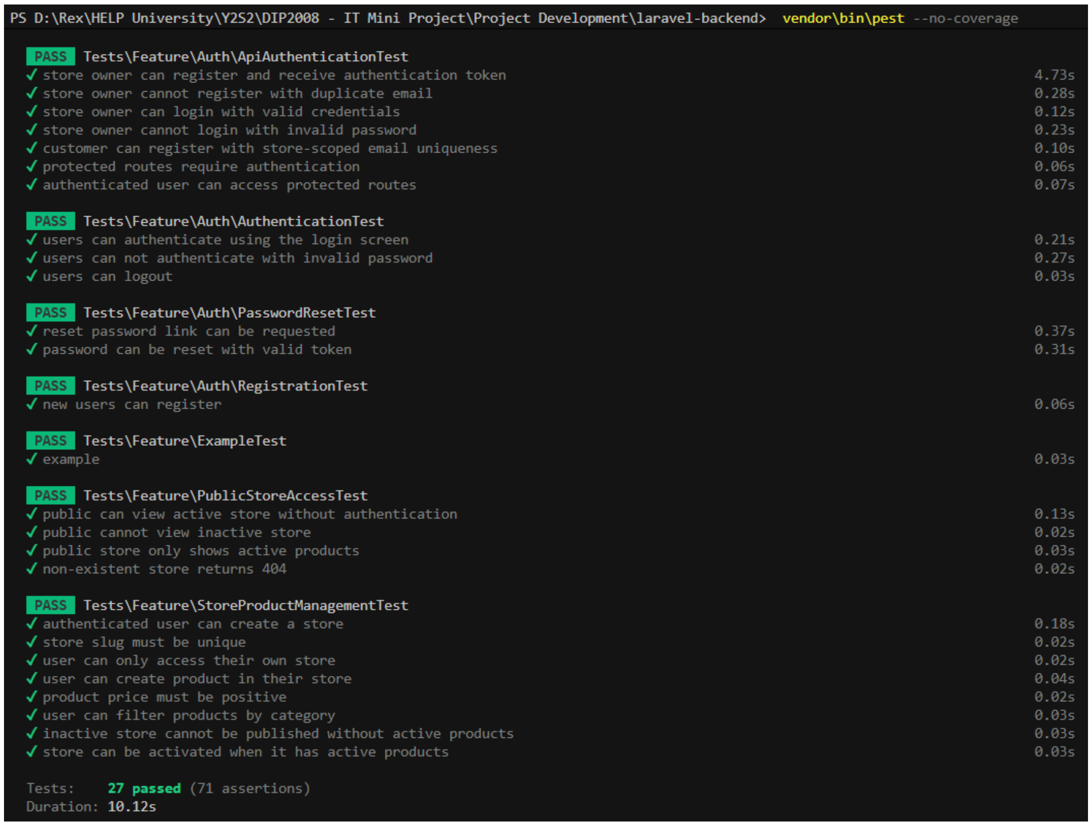
7. Testing (Chan Zi Hao B2400169)

7.1 Overview

This project is a complex multi-role platform, thus it is essential that business rules do not break as the project is being developed. Hence, it is crucial to implement a robust testing strategy that reduces time taken and errors of manual testing. As such, I have implemented automated testing using Pest PHP 4.0 to ensure API functions as expected and prevent regression.

My testing strategy focuses on three crucial areas. The first area is authentication and authorization. I must ensure that the program validates that only the authenticated users can access the protected resources and that users only have access to their own data. The second area of focus is the CRUD operations for store owners. They should be able to create, read, update and delete stores and products. Plus, all requests should be properly validated. The last area of focus is public API access for published stores. All customers should be able to view the published stores without any authentication.

In order for all tests to be done in parallel and to ensure tests do not interfere with each other, I have used an isolated SQLite in-memory database that refreshes for each test making the tests run very quickly. Below shows the test execution results.



```
PS D:\Rex\HELP University\Y2S2\DIP2008 - IT Mini Project\Project Development\laravel-backend>  vendor\bin\pest --no-coverage

  PASS  Tests\Feature\Auth\ApiAuthenticationTest
  ✓ store owner can register and receive authentication token                        4.73s
  ✓ store owner cannot register with duplicate email                                 0.28s
  ✓ store owner can login with valid credentials                                     0.12s
  ✓ store owner cannot login with invalid password                                   0.23s
  ✓ customer can register with store-scoped email uniqueness                         0.10s
  ✓ protected routes require authentication                                          0.06s
  ✓ authenticated user can access protected routes                                   0.07s

  PASS  Tests\Feature\Auth\AuthenticationTest
  ✓ users can authenticate using the login screen                                    0.21s
  ✓ users can not authenticate with invalid password                                 0.27s
  ✓ users can logout                                                                 0.03s

  PASS  Tests\Feature\Auth\PasswordResetTest
  ✓ reset password link can be requested                                             0.37s
  ✓ password can be reset with valid token                                           0.31s

  PASS  Tests\Feature\Auth\RegistrationTest
  ✓ new users can register                                                           0.06s

  PASS  Tests\Feature\ExampleTest
  ✓ example                                                                          0.03s

  PASS  Tests\Feature\PublicStoreAccessTest
  ✓ public can view active store without authentication                              0.13s
  ✓ public cannot view inactive store                                                0.02s
  ✓ public store only shows active products                                          0.03s
  ✓ non-existent store returns 404                                                   0.02s

  PASS  Tests\Feature\StoreProductManagementTest
  ✓ authenticated user can create a store                                            0.18s
  ✓ store slug must be unique                                                        0.02s
  ✓ user can only access their own store                                             0.02s
  ✓ user can create product in their store                                           0.04s
  ✓ product price must be positive                                                   0.02s
  ✓ user can filter products by category                                             0.03s
  ✓ inactive store cannot be published without active products                       0.03s
  ✓ store can be activated when it has active products                               0.03s

  Tests:    27 passed (71 assertions)
  Duration: 10.12s
```

Results of the automated testing shows that all 27 tests ran are successful, meaning the APIs are working as intended, while 71 assertions have been made to test whether the output matches the expected result.

7.2 Test Implementation Details

Below is a table which shows the Store & Product Management Test Plan.

Test ID	**Test Case**	**Input**	**Expected Result**	**Status**
CRUD-01	Authenticated user able to create store	store_name, slug, email	201, Created + store object in response	Pass
CRUD-02	Store slug must be unique	Existing slug	422 Validation Error	Pass
CRUD-03	User can only access their own store	Different user's store_id	403 Forbidden	Pass
CRUD-04	User can create product in their store	Product data + store_id	201 Created + product is returned in response	Pass
CRUD-05	Product price must be positive	price = -10.00	422 Validation Error	Pass
CRUD-06	User can filter products by category	?category = electronics	200 OK + filtered results only	Pass
CRUD-07	Inactive store cannot be published without any product	is_active = true, 0 products	422 Validation Error	Pass
CRUD-08	Store with active products can be activated	Is_active = true, ≥1 product	200 OK + store activated	Pass

7.2.1 Detailed Test Analysis on CRUD-03

The reason why "Users can only access their own store" (CRUD-03) is important is because the most crucial security concern in a multi-tenant system like this is a data leak. This test validates whether a user can intentionally or accidentally access another store that is not theirs. In an e-commerce platform, many store owners are using the same system. I must prevent unauthorized access that could be detrimental to businesses. For example, if a business gets their hands on their competitors' data, it could lead to proprietary knowledge or customer data being exposed which could create legal issues. Hence, this test is to ensure this does not occur.

Test Implementation:

```
test('user can only access their own store', function () {
    $user1 = User::factory()->create();
    $user2 = User::factory()->create();
    $store1 = Store::factory()->create(['user_id' => $user1->id]);

    $response = $this->actingAs($user2, 'sanctum')
                     ->getJson("/api/stores/{$store1->id}");

    $response->assertStatus(403);
});
```

First, I use the factory method to create 2 users. Then, I create a store using user 1's id. Following that, I request to access the store using user 2's id to see whether I can gain access to it as another user. To put it in simple words, user 2 attempts to access user 1's store. Last but not least, the status returned should be 403 Forbidden. In the test, 403 is returned, hence this test has passed the requirements given.

This test ensures that my StoreController has proper authorization logic that checks whether the user id that belongs to the store is equivalent to the user id given in the request. If these two user_id do not match, the controller must return a 403 error, it cannot return the store data.

This automated test which runs in less than a second can catch this vulnerability immediately, thus preventing unauthorized access to sales data and customer data theft. This small test that takes only a fraction of a second can protect millions of dollars in business data and intelligence.

7.2.2 Detailed Test Analysis on CRUD-07 & CRUD-08

These two test cases are paired tests with CRUD-07 being a negative test case while CRUD-08 is the positive test case.

```
test('inactive store cannot be published without active products',
function () {
    $user = User::factory()->create();
    $store = Store::factory()->create([
        'user_id' => $user->id,
        'is_active' => false,
    ]);

    // No products yet
    $response = $this->actingAs($user, 'sanctum')
                     ->putJson("/api/stores/{$store->id}", [
                         'store_name' => $store->store_name,
                         'store_slug' => $store->store_slug,
                         'contact_email' => $store->contact_email,
                      'is_active' => true, // Try to activate
                  ]);

    $response->assertStatus(422);
});

test('store can be activated when it has active products', function () {
    $user = User::factory()->create();
    $store = Store::factory()->create([
        'user_id' => $user->id,
        'is_active' => false,
    ]);

    // Create an active product
    Product::factory()->create([
        'store_id' => $store->id,
        'is_active' => true,
    ]);

    $response = $this->actingAs($user, 'sanctum')
                  ->putJson("/api/stores/{$store->id}", [
                      'store_name' => $store->store_name,
                      'store_slug' => $store->store_slug,
                      'contact_email' => $store->contact_email,
                      'is_active' => true,
                  ]);

    $response->assertStatus(200);
    $this->assertDatabaseHas('stores', [
        'id' => $store->id,
        'is_active' => true,
    ]);
});
```

What is happening in these two test cases is, I am creating a user, followed by a store using the user_id. Then, for the negative test case, I try to update the is_active to true without creating a single product. While for the positive test case, I create a product before updating the is_active to true. In the StoreController, there is a logic that is supposed to check whether the store has a product before allowing the is_active field to be updated to true. Thus, for the negative test case, it should return 422 Validation Error while the positive test case returns 200 OK.

The negative test case is important so only stores with active products can be activated. The reason why this business rule is important is so that customers do not have a bad experience of viewing an empty store with no products. The positive test case is important so we are sure that stores with products can be activated with no issues. Overall, both test cases passed as shown in the terminal test screenshot thus assuring all logic is working as intended.

7.2.3 Importance of Quality Assurance

I believe testing is the most overlooked yet important part of software development. For starters, constant testing can prevent security breaches such as store owners accessing other store's data.

Besides that, it allows us to refactor our code with confidence. For example, when we migrate from SQLite to MySQL or we optimize a query or upgrade Laravel versions, we can just run the tests in seconds to check whether anything has been broken. Furthermore, tests also act as a form of documentation. Other engineers are able to identify business rules, or validation requirements by just looking at the tests without going through the entire codebase just to understand it. Additionally, with automated testing, all 27 test cases can be run in just 10 seconds which means I can run the test frequently to immediately get feedback on whether my new code has broken any of my old code. Bugs can be caught early and fixed early before it is too late. Last but not least, these automated tests are the best quality assurance for preventing regressions. The moment a bug is found and resolved, I can add a new test that will ensure it never happens again.

In a nutshell, I have tested my code thoroughly to ensure it meets the user requirements and all features are functional.

5. Analysis & Design (Aidel B2400370)

5.1 User Requirements

The frontend of **Miles** focuses on ensuring sellers and customers can easily interact with the system through an intuitive, responsive interface.
User requirements for the frontend are summarized below:

- The **developer** should be able to access the main landing page and monitor system information.
- The **seller** should be able to:
  - Register and log in to their account.
  - Access their dashboard after logging in.
  - Fill in the store details form to create their personalized online store.
  - Add products, view orders, and check analytics through their dashboard.
  - Publish their store live to make it visible to customers.
- The **customer** should be able to:
  - View a seller's online store.
  - Register and log in as a customer.
  - Browse products, search, and apply filters.
  - Add products to the cart and proceed to checkout through Stripe.

5.2 Functional Requirements

- The system should provide a **responsive landing page** for developers and visitors.
- The seller registration and login pages must authenticate users via API calls to the backend.
- The frontend should allow sellers to **submit store details** through dynamic forms and display them on their personalized store pages.

- The **seller dashboard** should provide tabs for:
    - **Add products:** Add item name, price, quantity, description, and product image
    - **Analytics:** View total revenue, top selling item
    - **View order:** Display all customer orders.
- The **customer interface** must allow:
    - Browsing, searching, and filtering products.
    - Adding products to the cart and viewing the total before checkout.
    - Redirecting to **Stripe's secure checkout page** for payment

5.3 Non-Functional Requirements

- **Usability:**
  UI design must follow **Material UI** standards for consistent visual hierarchy, typography, and color schemes.
- **Performance:**
  Pages should load within 3 seconds under a stable internet connection.
- **Accessibility:**
  Clear fonts, color contrast, and intuitive navigation are ensured for all users.
- **Maintainability:**
  The frontend is structured with reusable React components and well-organized folder hierarchy.

5.4 Overview of Project Architecture

General Architecture

The system adopts a **client–server architecture**, where the **frontend (client)** communicates with the **backend server** through RESTful APIs.

- **Frontend:** Developed using React and styled with Material UI.
- **Backend:** Laravel (handled by partner).
- **Database:** MySQL (backend-managed).
- **Integration:** Stripe API for payment, handled through secure redirects.

Components / Modules (Frontend Focus)

- **Landing Page Module:** Displays system overview and call-to-action for sellers.
- **Authentication Module:** Handles seller and customer login/registration through interactive forms.
- **Seller Navigation Bar:** Allows sellers to access add product, analytics, and view order.

- **Store Registration Module:** Collects store information and generates personalized store pages dynamically.
- **Product Display Module:** Shows available products to customers with filtering and search options.
- **Cart & Checkout Module:** Handles cart functions and redirects to Stripe for payment processing.

5.5 Frontend Module (UI/UX Design)

5.5.1 Wireframes

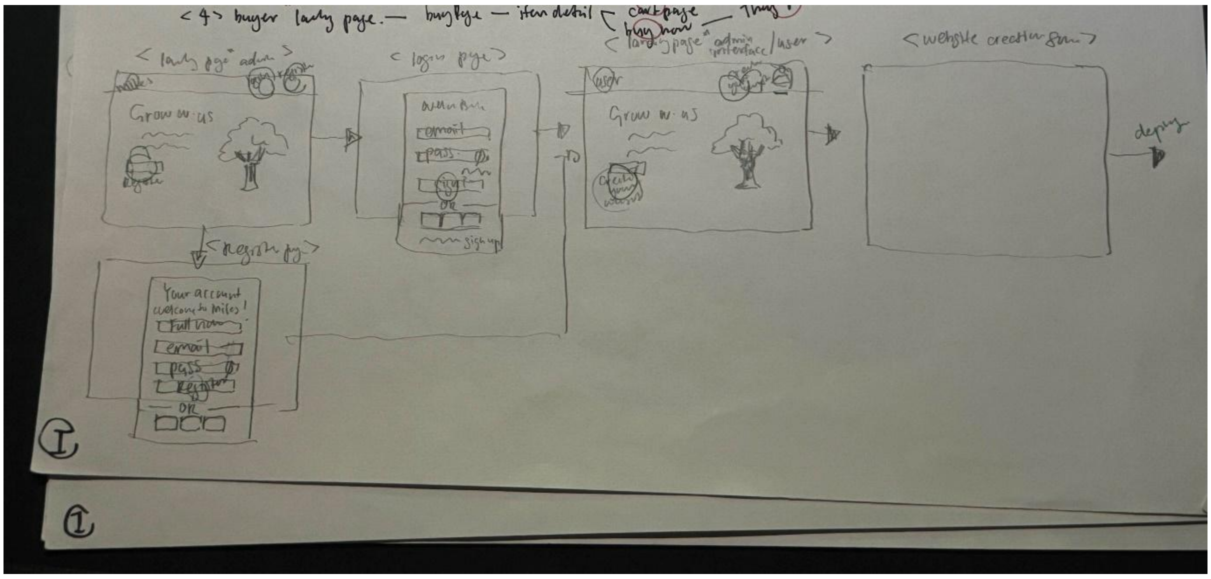


Figure 1: Developer Landing Page and Seller Dashboard

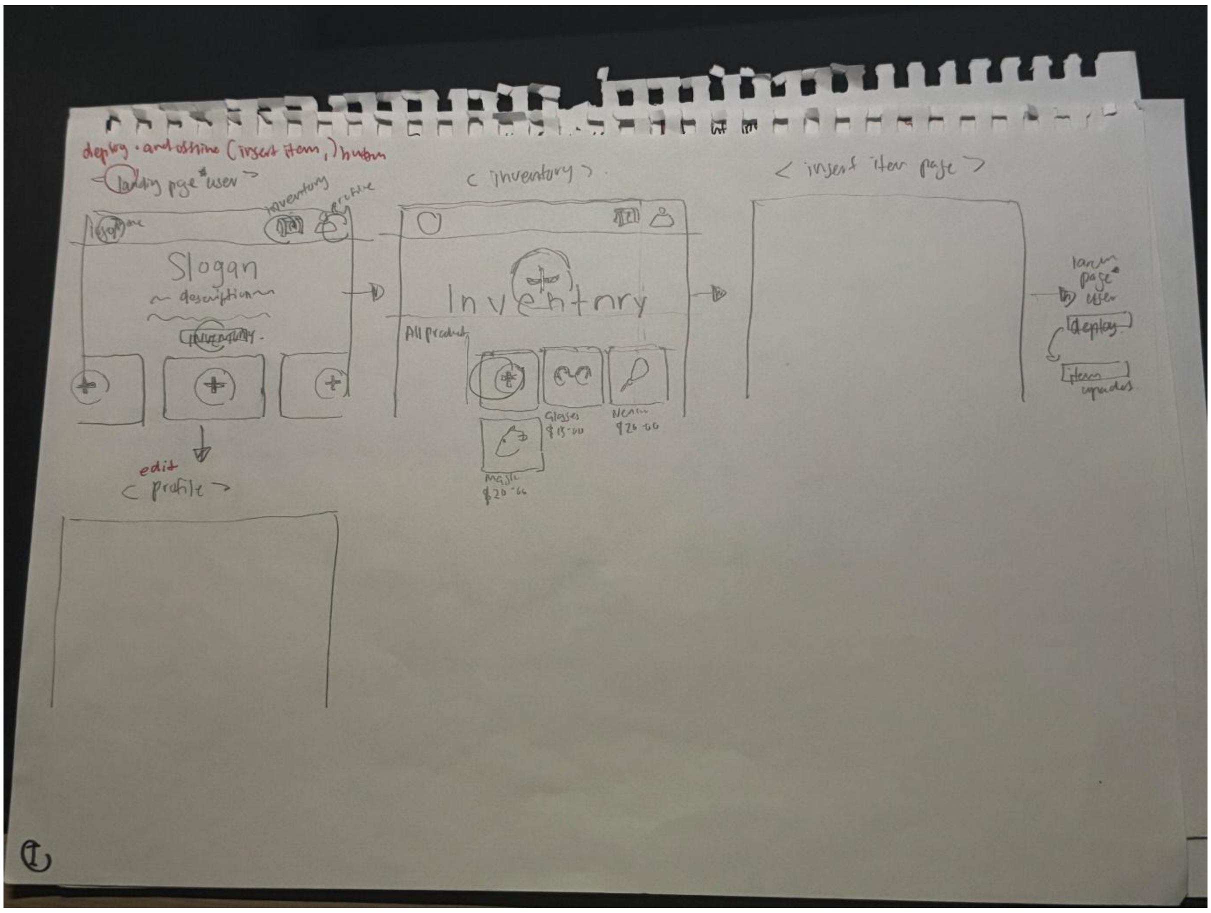


Figure 2: Seller Dashboard with Store Detail

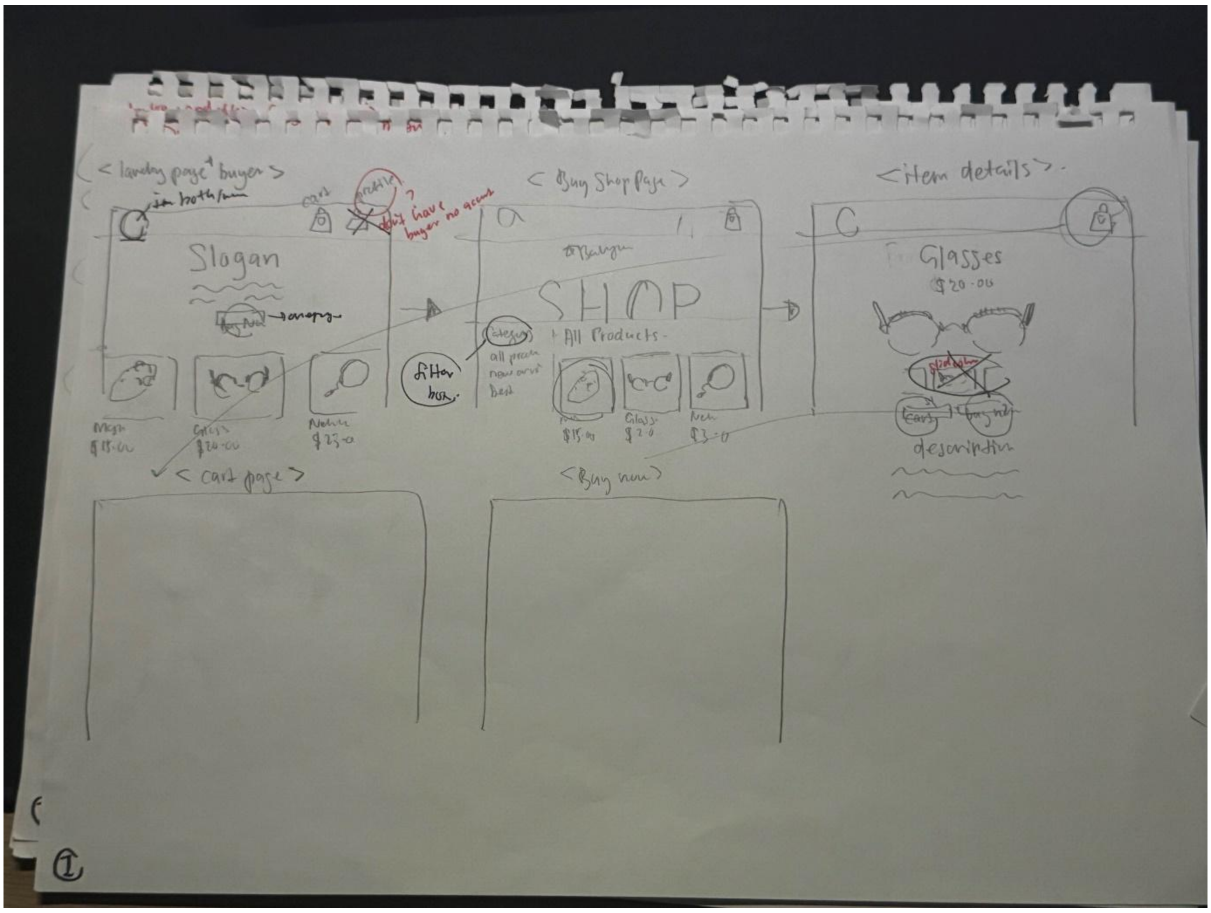


Figure 3: Customer View

5.5.2 Mockups

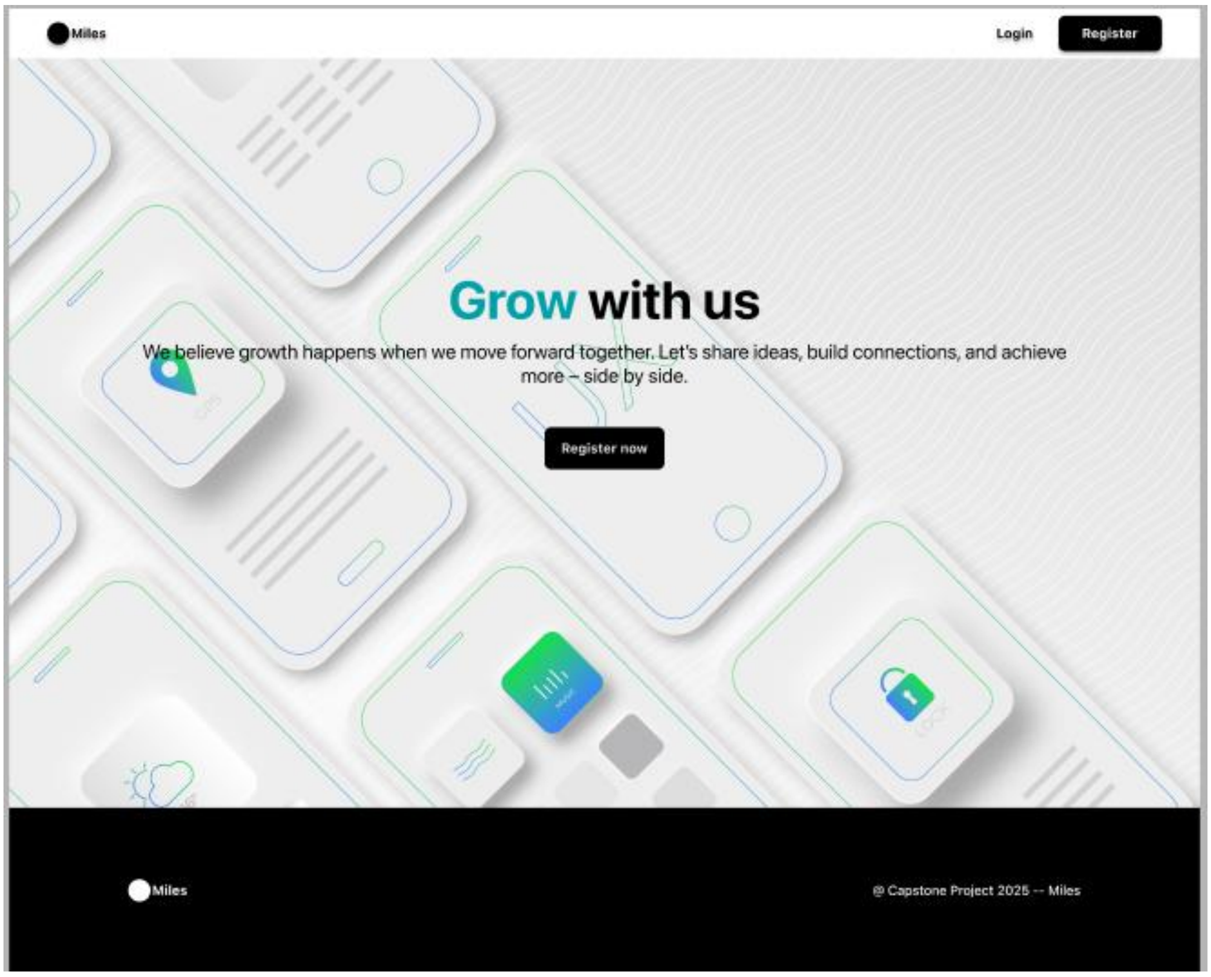

Miles
Login
Register
Grow with us
We believe growth happens when we move forward together. Let's share ideas, build connections, and achieve more – side by side.
Register now
Miles
@ Capstone Project 2025 -- Miles

Figure 4: Developer Landing Page

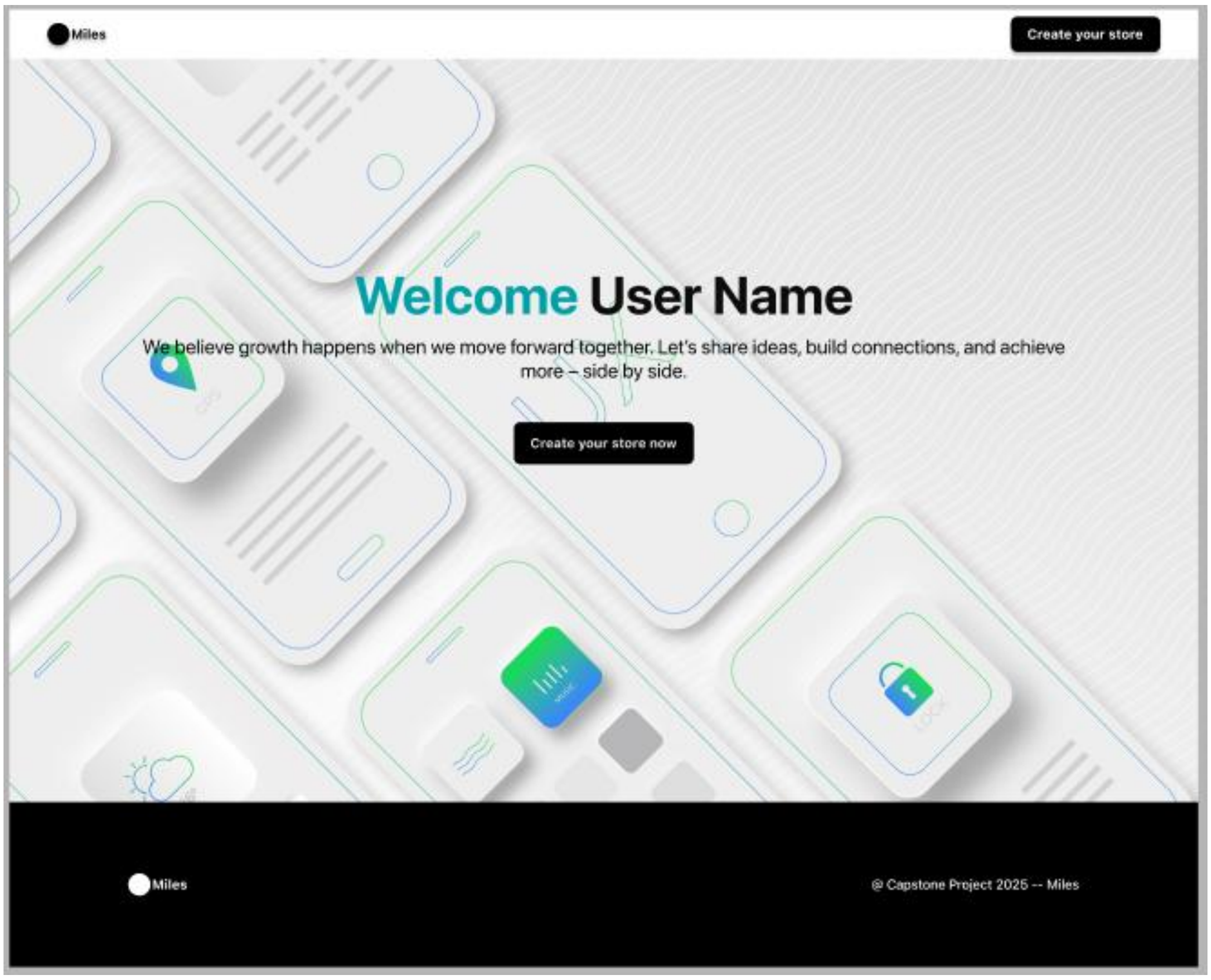
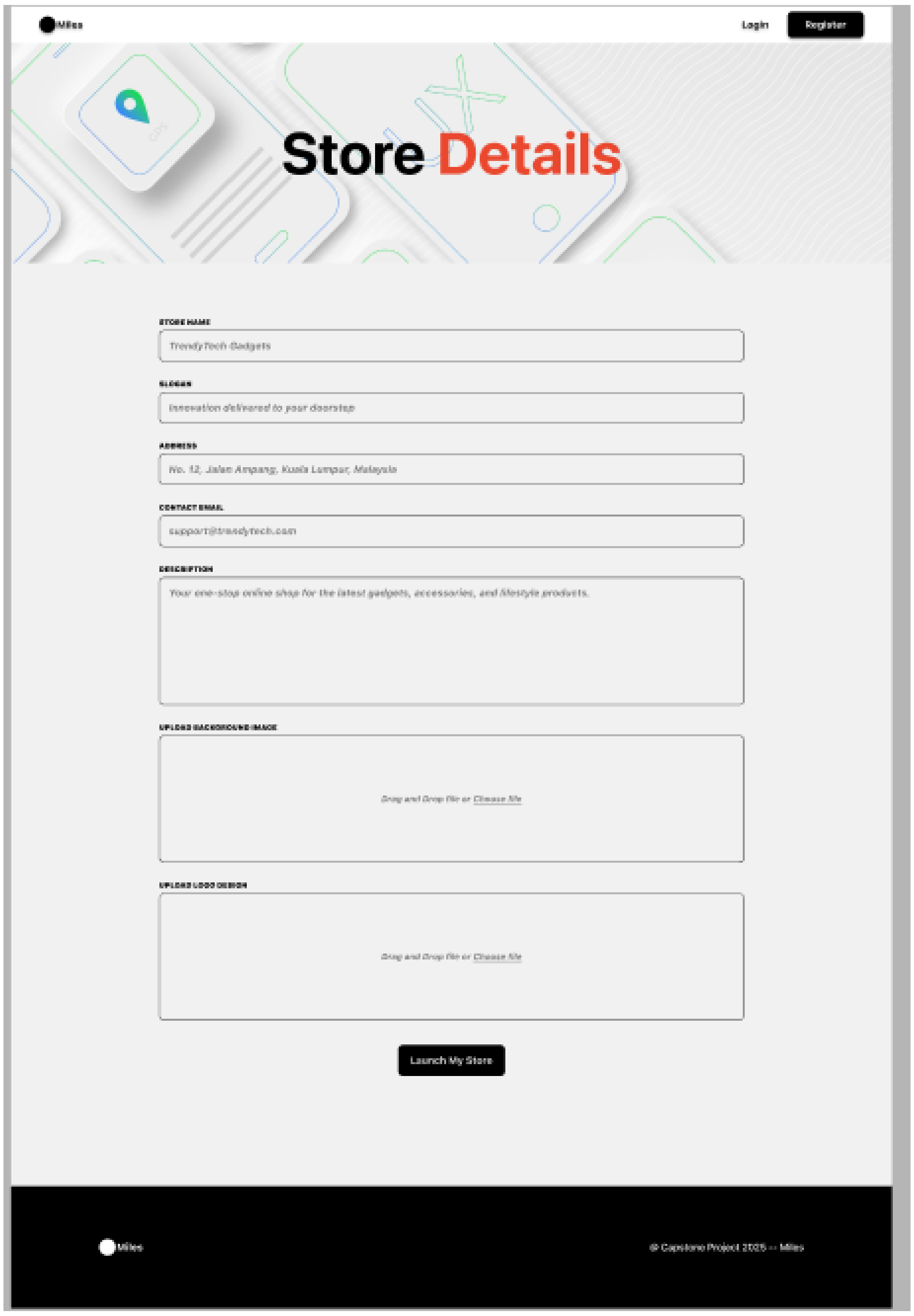

Miles
Create your store
Welcome User Name
We believe growth happens when we move forward together. Let's share ideas, build connections, and achieve more – side by side.
Create your store now
Miles
© Capstone Project 2025 -- Miles

Figure 5: Seller Dashboard


Miles

Login

Register

Store Details

STORE NAME

TrendyTech Gadgets

SLOGAN

Innovation delivered to your doorstep

ADDRESS

No. 12, Jalan Ampang, Kuala Lumpur, Malaysia

CONTACT EMAIL

support@trendytech.com

DESCRIPTION

Your one-stop online shop for the latest gadgets, accessories, and lifestyle products.

UPLOAD BACKGROUND IMAGE

Drag and Drop file or Choose file

UPLOAD LOGO DESIGN

Drag and Drop file or Choose file

Launch My Store

Miles

© Capstone Project 2025 — Miles

Figure 6: Store Details Form

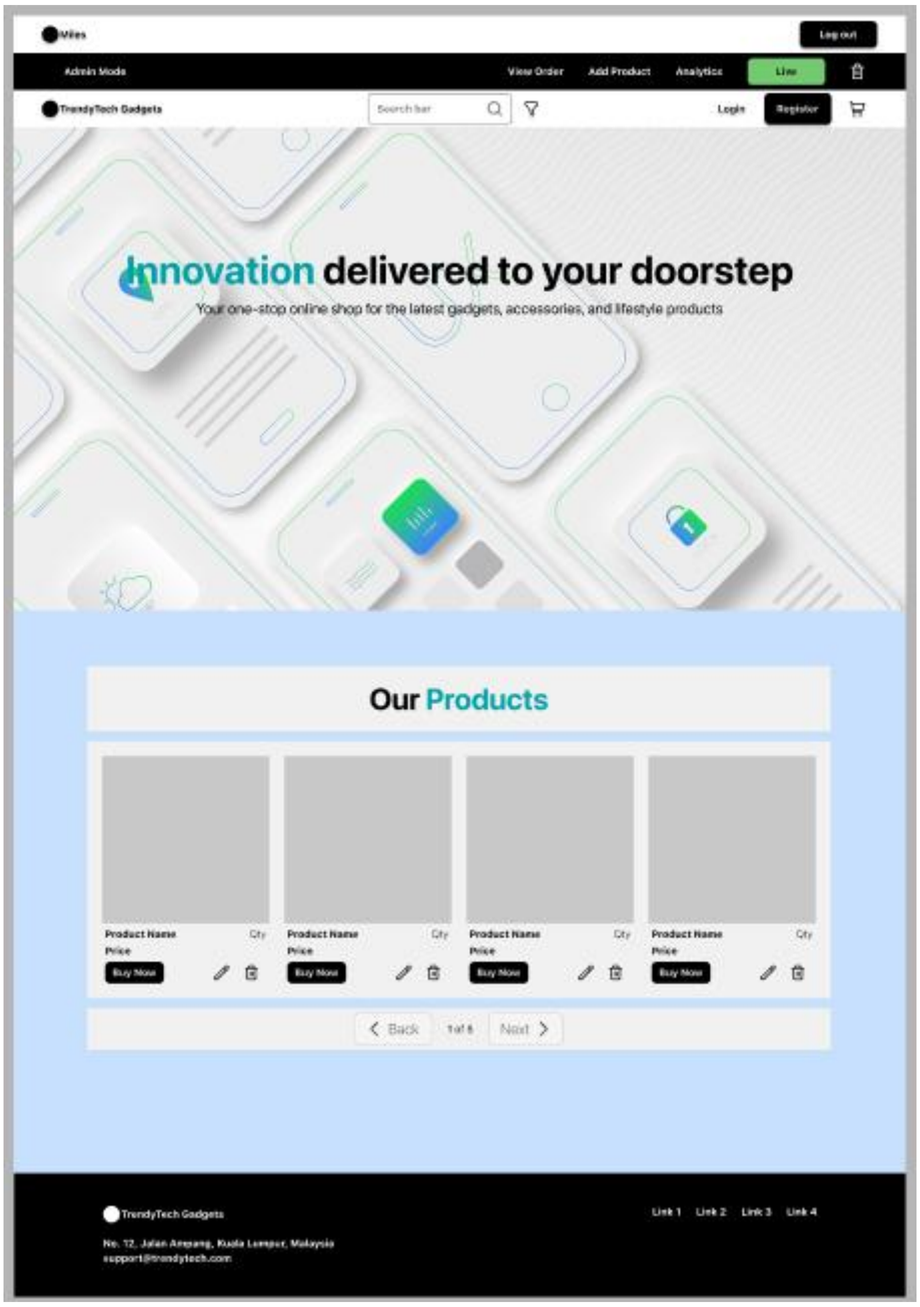
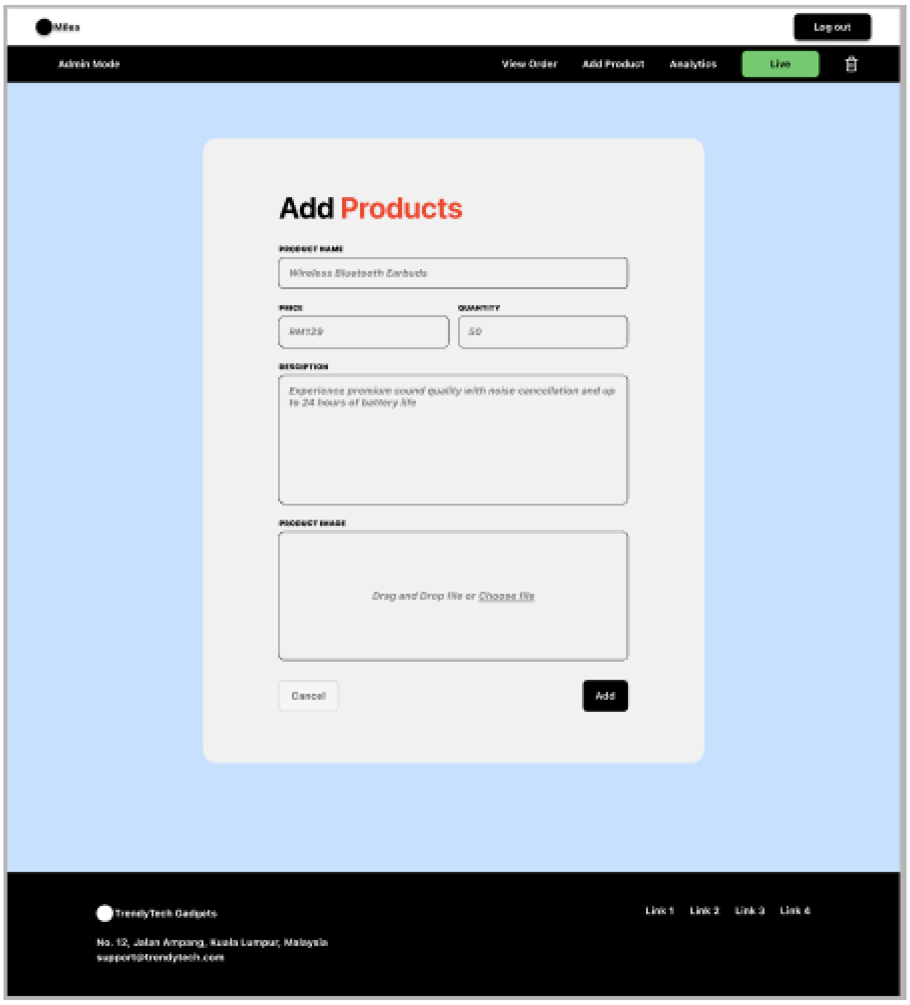


Figure 7: Seller Dashboard with Store Details



Add Products

PRODUCT NAME

Wireless Bluetooth Earbuds

PRICE

RM129

QUANTITY

50

DESCRIPTION

Experience premium sound quality with noise cancellation and up to 24 hours of battery life

PRODUCT IMAGE

Drag and Drop file or Choose file

No. 12, Jalan Ampang, Kuala Lumpur, Malaysia
support@trendytech.com

Figure 8: Add Product Page

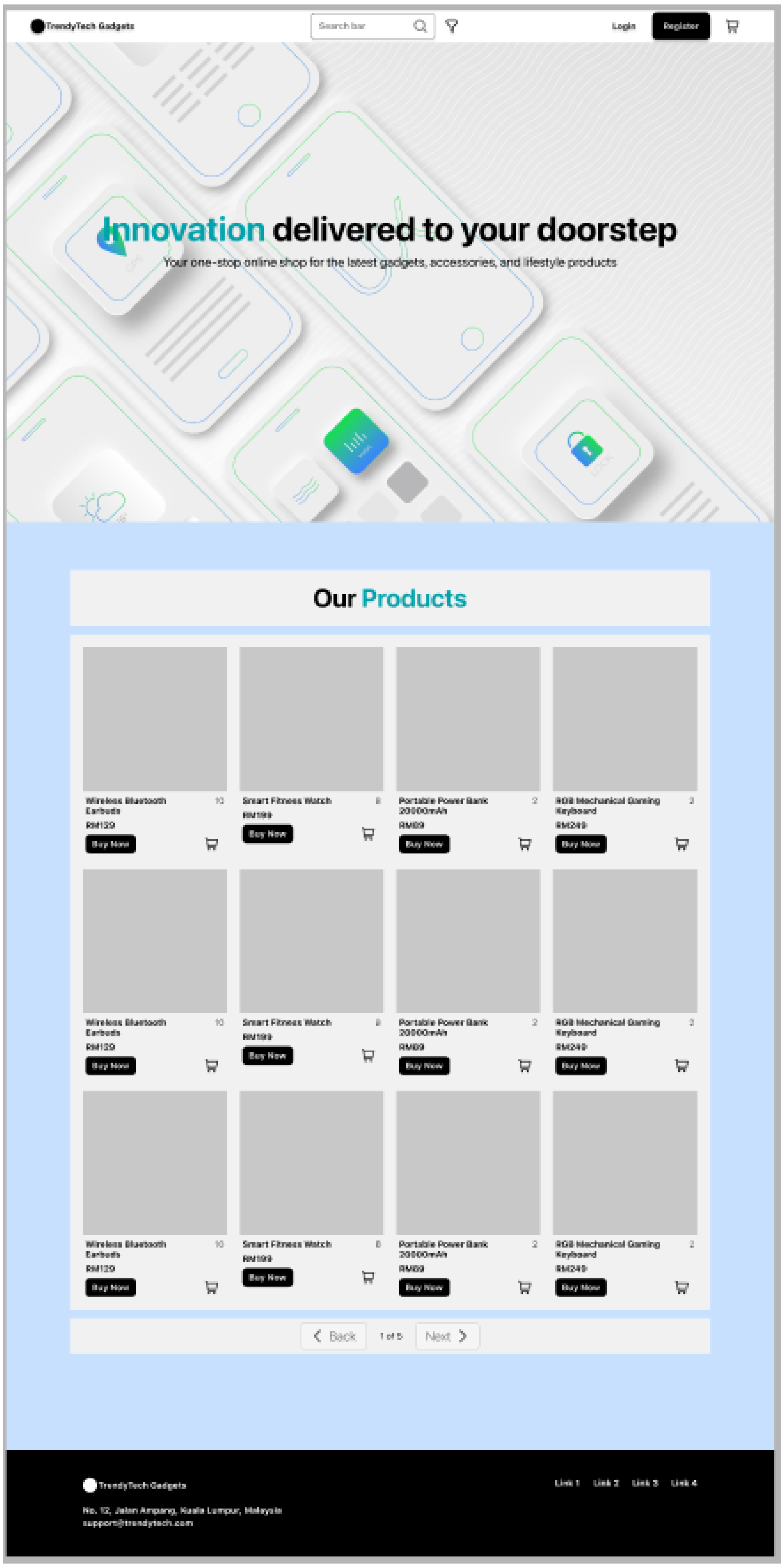
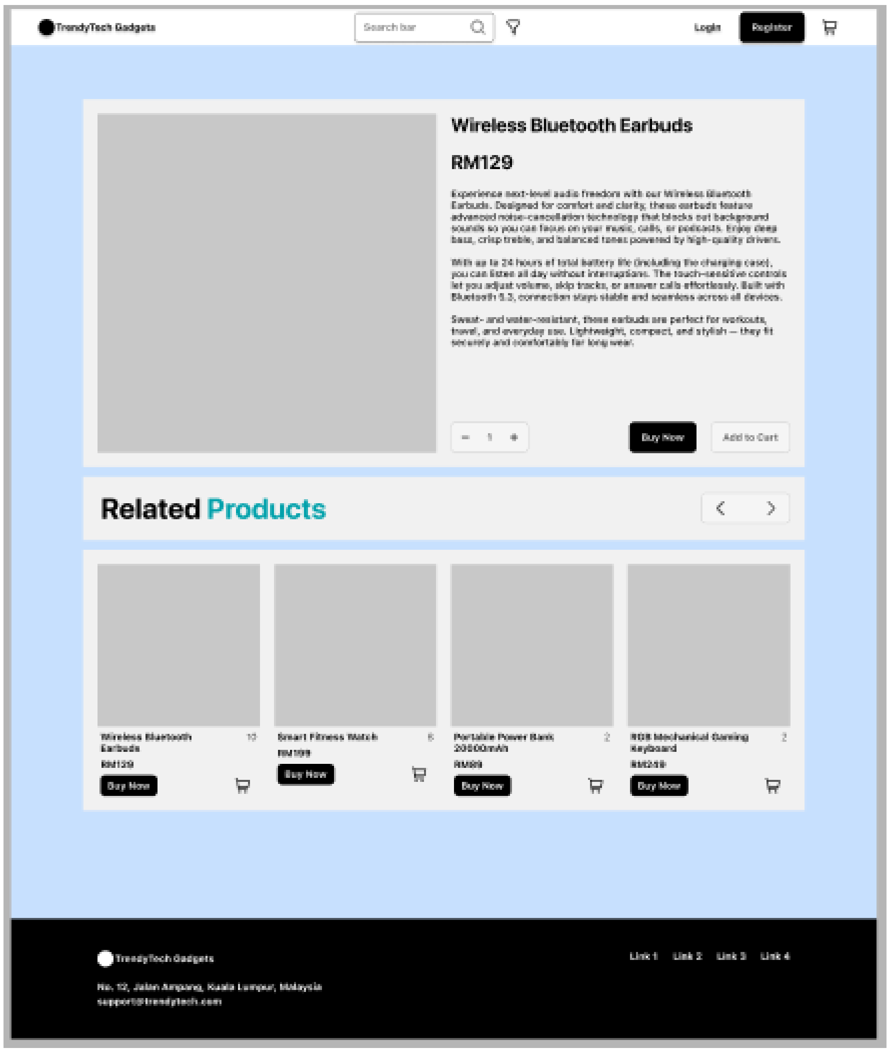

TrendyTech Gadgets
Search bar
Login
Register
Innovation delivered to your doorstep
Your one-stop online shop for the latest gadgets, accessories, and lifestyle products
Our Products
Wireless Bluetooth Earbuds
10
RM129
Buy Now
Smart Fitness Watch
8
RM199
Buy Now
Portable Power Bank 20000mAh
2
RM89
Buy Now
RGB Mechanical Gaming Keyboard
2
RM249
Buy Now
Back
1 of 5
Next
TrendyTech Gadgets
No. 12, Jalan Ampang, Kuala Lumpur, Malaysia
support@trendytech.com
Link 1
Link 2
Link 3
Link 4

Figure 9: Customer view Page



Experience next-level audio freedom with our Wireless Bluetooth Earbuds. Designed for comfort and clarity, these earbuds feature advanced noise-cancellation technology that blocks out background sounds so you can focus on your music, calls, or podcasts. Enjoy deep bass, crisp treble, and balanced tones powered by high-quality drivers.

With up to 24 hours of total battery life (including the charging case), you can listen all day without interruptions. The touch-sensitive controls let you adjust volume, skip tracks, or answer calls effortlessly. Built with Bluetooth 5.3, connection stays stable and seamless across all devices.

Sweat- and water-resistant, these earbuds are perfect for workouts, travel, and everyday use. Lightweight, compact, and stylish — they fit securely and comfortably for long wear.

Figure 10: View Product Page

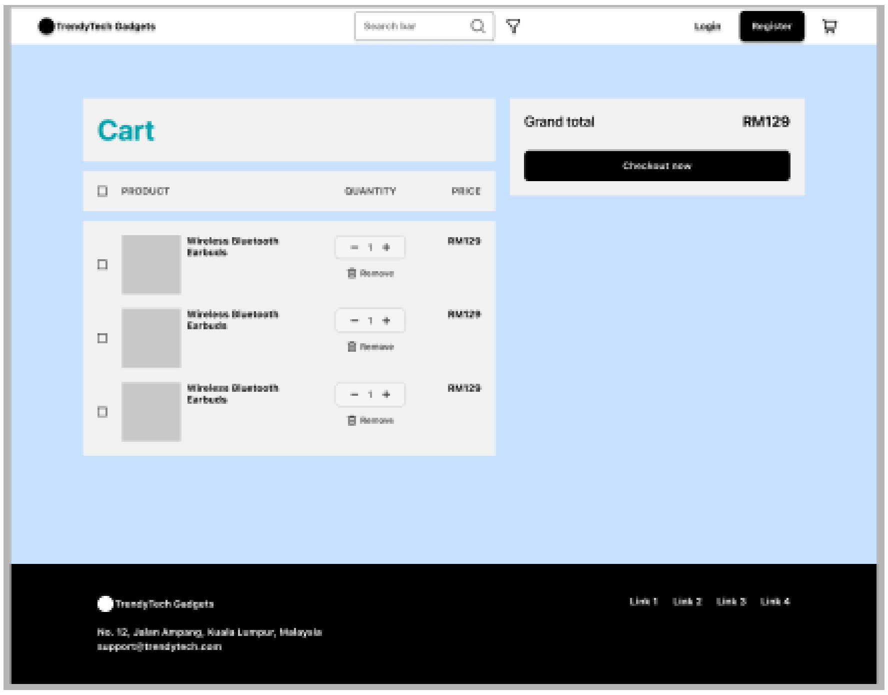


Figure 11: Cart Page

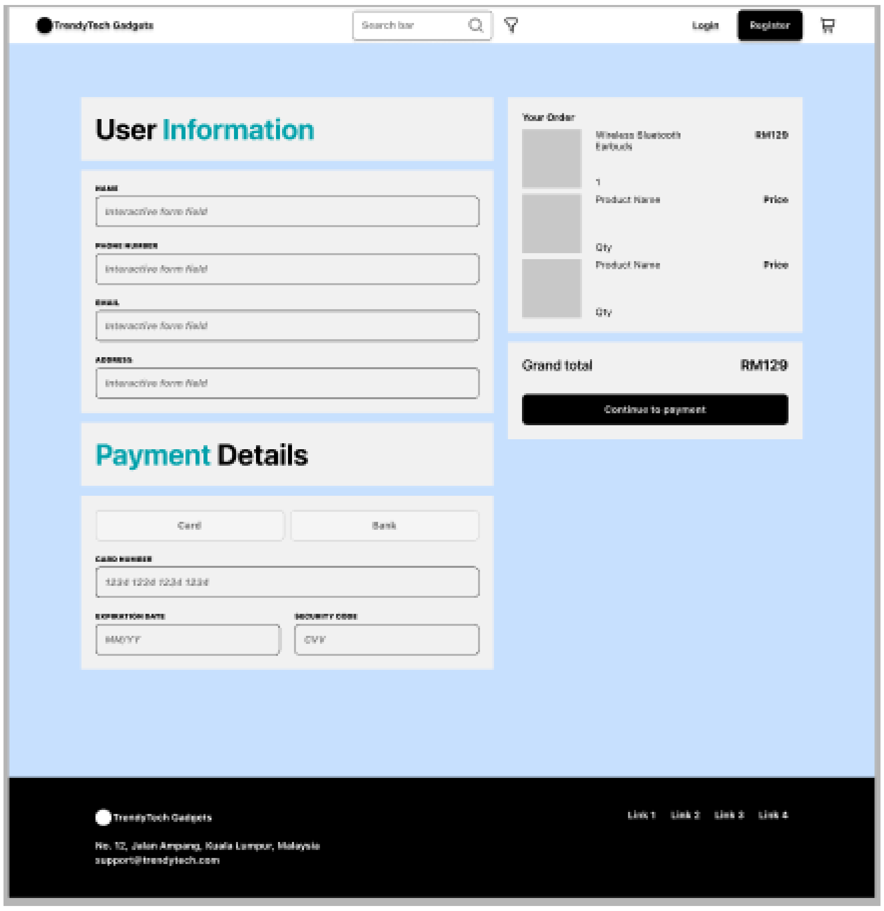


Figure 12: Checkout Page

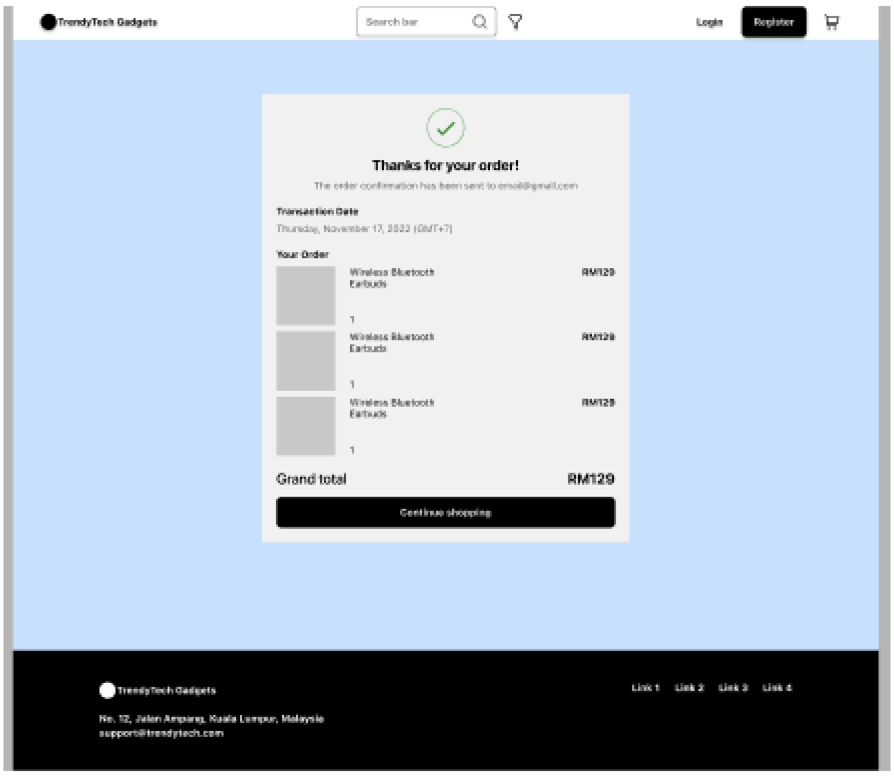

TrendyTech Gadgets
Search bar
Login
Register
Thanks for your order!
The order confirmation has been sent to email@gmail.com
Transaction Date
Thursday, November 17, 2022 (GMT+7)
Your Order
Wireless Bluetooth Earbuds
RM129
1
Wireless Bluetooth Earbuds
RM129
1
Wireless Bluetooth Earbuds
RM129
1
Grand total
RM129
Continue shopping
TrendyTech Gadgets
No. 12, Jalan Ampang, Kuala Lumpur, Malaysia
support@trendytech.com
Link 1
Link 2
Link 3
Link 4

Figure 13: Order Complete Page

5.5.3 Component Hierarchy Diagram

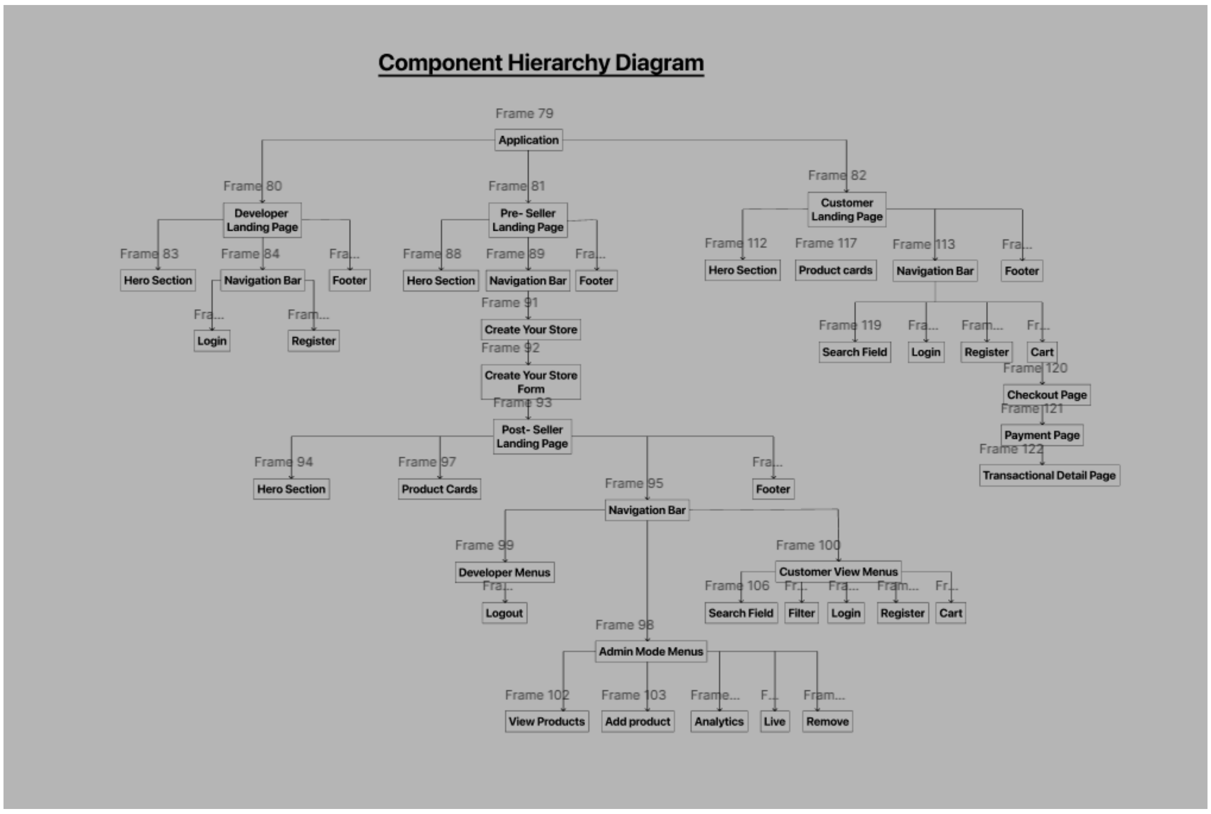


Figure 14: Component Hierarchy Diagram

5.5.4 State Diagram

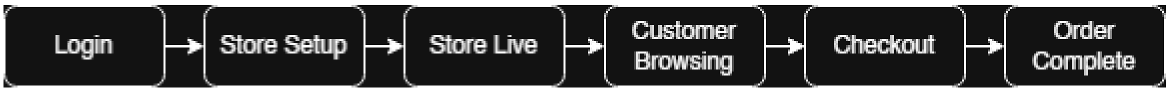


Figure 15: State Diagram

5.5.5 User Flow Diagram

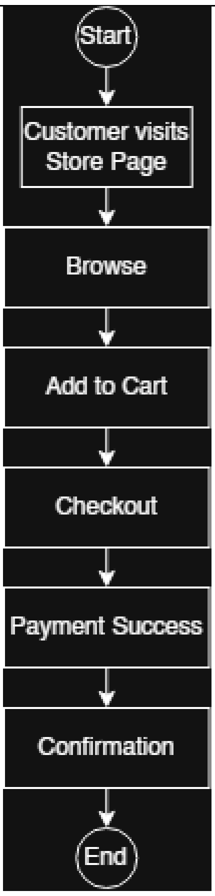


Figure 16: Customer Navigation Journey

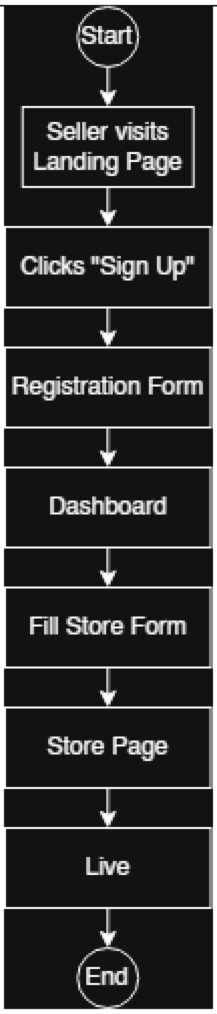


Figure 17: Seller Navigation Journey

6. Technical implementation (Aidel B2400370)

6.1 Overview of Technical Implementation

Summary

The frontend of **Miles** is implemented using **React**, providing a responsive and interactive user experience for sellers and customers. The main purpose of the frontend is to handle all user-facing interactions — from registration and store creation to browsing and purchasing products. It is designed to be intuitive, fast, and consistent with Material UI's visual design system.

Architecture Explanation

The system follows a **client–server architecture** where the **frontend (React)** communicates with the **backend (Laravel)** through **RESTful APIs**. The frontend handles all user interface rendering, input validation, and navigation, while the backend manages authentication, data storage, and payment operations.

The **frontend structure** is component-based, meaning every UI element (such as buttons, forms, and product cards) is built as an independent reusable React component. Routing between pages is managed using **React Router**, allowing smooth transitions between seller and customer views without reloading the page.

Data from the backend is fetched using **Axios API calls** and stored temporarily in React states for display. The entire frontend is styled using **Material UI**, ensuring consistency, responsiveness, and accessibility across all devices.

Technology Description

- **React:** Core framework for building the single-page application interface.
- **Material UI:** Provides pre-built, responsive, and accessible UI components.
- **Axios:** Used to make HTTP requests to backend APIs.
- **React Router:** Manages navigation between pages like Dashboard, Store Details Form, and Customer View.
- **JavaScript (ES6+):** Implements dynamic features and logic.
- **HTML5 & CSS3:** Define the basic structure and layout styling.
- **Vercel Hosting:** Used for deployment, providing continuous integration and delivery for the frontend.

6.2 Core Modules & Features

Module	Description	Technologies Used
Landing Page	Displays an overview of Miles and encourages sellers to sign up. Includes navigation bar and CTA buttons.	React, Material UI, React Router
Authentication Module	Handles seller and customer registration and login through interactive forms. Uses API integration to communicate with the backend.	React, Material UI, Axios
Store Registration Module	Allows sellers to fill in store name, description, and preferences. On submission, data is sent to the backend and dynamically displayed on their store page.	React Forms, Axios, Material UI
Seller Navigation Bar	Displays analytics, products, and order summaries. Navigation tabs are built using Material UI's layout components.	React Router, Material UI Tabs, Axios

Module	Description	Technologies Used
Product Management Interface	Enables sellers to view and manage their product listings. Products are fetched and displayed as Material UI cards.	React, Axios, Material UI
Customer View Page	Public storefront displaying seller's products. Includes search bar, filter options, cart button, and redirect to Stripe for payment.	React, React Router, Axios, Stripe Redirect
Navigation & Routing	Handles page transitions without reloads	React Router DOM

6.3 Tools & Technologies Used

Category	Tools / Technologies	Purpose
Frontend Framework	React (JavaScript)	Build dynamic and modular single-page application.
UI Framework	Material UI	Provides pre-designed components and ensures responsive design.
Routing	React Router DOM	Manages page navigation and routes within the app.
API Communication	Axios	Sends and receives data from backend endpoints.
Language	HTML5, CSS3, JavaScript (ES6+)	Defines structure, styling, and interactivity.
Version Control	GitHub	Tracks project changes and support team collaboration.
Design Tool	Figma	Used for creating UI mockups, wireframes, and prototypes.
Deployment	Vercel	Hosts the frontend with automatic deployment and scalability.
Testing Tool	Postman (for frontend API testing)	Used to verify API endpoints and data responses.

7. Testing (Aidel B2400370)

7.1 Introduction

This Test Plan describes the testing strategy, scope, and schedule for verifying the frontend of Miles, a web-based e-commerce builder. The purpose of testing is to ensure that all pages and user interactions work as intended, particularly for sellers and customers, and that the interface meets usability and performance standards.

7.2 Objectives

- Verify that all core frontend features (navigation, forms, and payment redirection) function correctly.
- Ensure that the user interface is responsive, consistent, and intuitive.
- Validate integration between frontend modules and backend APIs.
- Confirm system readiness for public release.

7.3 Scope

In Scope:

- Website navigation and routing
- Seller registration and login form validation
- Store details submission
- Product management interface
- Customer browsing, search, and cart functions
- Stripe payment gateway redirection

Out of Scope:

- Backend database logic
- Stripe transaction verification (handled by Stripe)
- Admin management panel

7.4 Test Items (Features to Be Tested)

Feature	Description
Landing Page	Ensure page loads and navigation links work correctly
Authentication Page	Test registration and login input validation
Store Setup Form	Verify form data submission and field validation
Seller Navigation Bar	Ensure seller dashboard tabs (Add Product, View Order, Analytics) display properly
Customer Storefront	Confirm products load, filters, and search work
Cart & Checkout	Verify adding to cart and Stripe redirection

7.5 Test Approach

Testing will include:

- **Unit Testing:** Verifying individual frontend components (e.g., form input validation).
- **Integration Testing:** Ensuring data flow between React frontend and Laravel backend APIs works correctly.
- **System Testing:** Testing the full navigation flow from seller registration to checkout.
- **Usability Testing:** Reviewing the interface layout and ease of use.
- **Release Testing:** Confirming readiness for live deployment on Vercel.

7.6 Test Environment

Component	**Details**
Browser	Chrome, Firefox, Edge
Framework	React (Material UI)
API Tools	Postman (for endpoint testing)
Hosting	Vercel (frontend test build)

7.7 Test Data

Field	**Example Data**
Seller Email	seller@example.com
Password	Test1234
Store Name	TrendyTech Gadgets
Product Name	Wireless Bluetooth Earbuds
Price	RM129
Customer Email	customer@example.com

7.8 Test Cases Overview

Test Case ID	**Description**	**Type**	**Expected Result**
TC_001	Verify landing page loads successfully	Functional	Homepage visible with navigation links

Test Case ID	Description	Type	Expected Result
TC_002	Test seller registration with valid inputs	Functional	Account created successfully
TC_003	Test invalid email format on registration	Validation	“Please enter a valid email” message shown
TC_004	Verify store details form submission	Functional	Redirects seller to dashboard
TC_005	Verify product listing displays correctly	Functional	Product cards rendered properly
TC_006	Test adding product to cart	Integration	Product appears in cart panel
TC_007	Verify Stripe payment redirection	Integration	Stripe checkout page opens
TC_008	Check responsive layout on mobile	Usability	Layout adapts correctly
TC_009	Verify navigation between dashboard tabs	Functional	Tabs switch without page reloads

7.9 Example Detailed Test Cases

7.9.1 Functional Test Cases

Test Case ID	Test Objective	Preconditions	Steps	Test Data	Expected Result
TC_FT_001	Verify seller registration form works	Website online	1. Go to Registration Page 2. Enter email/password 3. Click Register	seller@example.com / Test1234	Redirects to dashboard with success message
TC_FT_002	Verify dashboard tabs display correctly	Logged in as seller	1. Navigate to Dashboard 2. Click on each tab	N/A	Each tab loads correct section content
TC_FT_003	Verify customer can view products	Seller store live	1. Visit store link 2. Scroll through products	N/A	Product list loads correctly

7.9.2 Validation Test Cases

Test Case ID	Test Objective	Preconditions	Steps	Test Data	Expected Result
TC_VT_001	Validate empty fields in registration	On registration page	1. Leave fields blank 2. Click Register	N/A	“Fields cannot be empty” error message
TC_VT_002	Validate store form field input types	On store setup form	1. Enter invalid characters in phone field 2. Click Submit	Phone = “abc”	Error shown: “Please enter a valid phone number”

7.9.3 Integration / Interaction Test Cases

Test Case ID	Test Objective	Preconditions	Steps	Test Data	Expected Result
TC_IT_001	Verify frontend connects to backend API	Backend online	1. Register new seller 2. Inspect API call	seller@example.com	API returns success status (200 OK)
TC_IT_002	Verify Stripe redirect works	Store live with cart	1. Add product to cart 2. Click Checkout	Product ID = 101	Stripe checkout page opens in new tab

7.9.4 Usability Test Cases

Test Case ID	Test Objective	Preconditions	Steps	Expected Result
TC_UU_001	Verify buttons and labels are readable	Website loaded	1. Open landing page 2. Observe button text	Buttons are clearly visible and descriptive
TC_UU_002	Verify responsive design	Website loaded	1. Resize window or test on mobile	Layout adjusts and remains usable

7.10 Test Schedule

Phase	Start Date	End Date
Test Planning	25 Oct 2025	27 Oct 2025
Test Case Development	28 Oct 2025	30 Oct 2025

Phase	**Start Date**	**End Date**
Test Execution	31 Oct 2025	3 Nov 2025
Defect Fix & Retesting	4 Nov 2025	6 Nov 2025
Final Review	7 Nov 2025	8 Nov 2025

8. Future Development

This project is only in the prototype phase and there is huge room for improvement and advancement. This platform targets Malaysian social media sellers who want a fast establishment of presence online beyond Instagram, TikTok and Facebook. The expansions and enhancements stated below will address the most crucial gaps in serving the e-commerce market, focusing on features that can help Malaysian sellers tremendously while meeting Malaysian customers' expectations.

8.1 WhatsApp Business Integration

For many small social media businesses, they tend to accept orders through text messages as they might not have Stripe or PayPal setup. Not just that, almost all Malaysians use WhatsApp daily. This includes both sellers and customers, so by integrating WhatsApp into the platform, we are making it even more convenient for sellers to handle orders through WhatsApp and allowing customers to communicate directly with the seller which builds trust.

The key feature will be a newly added button to "Order via WhatsApp" that customers can click on which will instantly open their WhatsApp with a pre-filled message to the seller. Example message will be "Hello! I'm interested in buying [Product Name] Price: RM39 Link: [link to product]." Then, customers can just click send to have the order sent directly to the seller's WhatsApp business number. With this, there is no payment gateway included in the process, which means it is entirely up to the seller to arrange the payment manually. This option could increase a business's revenue as there are many customers who distrust or are unfamiliar with online payments.

After a customer sends the message, sellers will be notified on WhatsApp and if there are any order updates, an auto-notification will be sent to the customer on WhatsApp. The entire process will feel extremely comfortable to users as WhatsApp is an app they are familiar with. Additionally, when a product is low on stock, a WhatsApp reminder could be automatically sent to the seller.

All of these features and automation can be made using WhatsApp Business API. For the button, all we need to do is construct a link to WhatsApp with a pre-filled text. For example, "https://wa.me/<phone_number>?text=<url_encoded _message>". The most important development that has to be done will be webhook integration for the system to receive notifications from WhatsApp Business API and using Meta's Graph API to send messages to customers or sellers via WhatsApp.

8.2 Malaysian Payment Methods

Offering key local payment options such as Touch 'n Go eWallet and Cash on Delivery (COD) can greatly increase conversion rate. This is because most Malaysians use e-wallets for online shopping and many people prefer COD for their first purchase. By expanding the payment methods available, it will encourage even more customers to purchase on our platform.

How we can implement these payment options into our system is by using Malaysian payment gateways like iPay88 or Senangpay. These payment gateways offer all local methods in just one integration. They can handle TNG, GrabPay, Boost, ShopeePay, FPX. Besides the integration of one of these payment gateways, implementation of a webhook to listen for payment confirmation will be required. Implementing this feature is highly feasible and critical to removing the barrier for customers.

8.3 Multi-Language Product Listing

Currently, there is no multilingual support in the prototype, however multi-language support is crucial in a country like Malaysia. The solution is to implement a feature where sellers only have to use one language for their store and products, and automatically provide in all three languages.

The key feature is sellers only add products once, whether in Malay, English or Chinese. Then, based on the customer's browser default language, the platform will auto-select that language as the default. How we can do this is by detecting the browser language on the frontend using navigator.language. Additionally, we can provide a manual language toggle for users to switch between languages. For example, "BM | EN | 中文".

How this can work is by utilising AI to translate the provided language into the other 2 languages. The LLM (Large Language Model) used will be GPT-4. For example, if the store and product is written in Malay, use AI to translate it into English and Chinese. After all three languages are shown, sellers can review and edit it in case it is inaccurate. The key skill required here will be prompt engineering, the prompt given to the AI must be precise and fulfill all requirements for the output to be accurate as well as reliable. If translation is missing, the default shown will be the language provided by the seller.

This matters even more for Malaysia as we are a multiracial country. Sellers want to reach all communities to capture the maximum amount of revenue. Furthermore, customers are always more willing to buy from a store with a language they are familiar with.

8.4 Mobile Store Management App

Once the platform has established itself as a reliable e-commerce builder, the next step will be to introduce a mobile app for sellers to create and manage their stores on their phone without turning on their desktop.

What we can develop is a fully native mobile app for iOS and Android users that provide the full store management functions just like what we have on the website. The key feature will make full use of the mobile phone. Sellers just have to open the app, take a photo of their product, fill in the details, and tap "Publish" and their store is live in less than a minute.

Besides that, when new orders are placed, we can introduce a push notification to immediately notify sellers. Furthermore, all the current features such as sales dashboard will be available in the mobile app. Additionally, sellers will be able to post products to Instagram or Facebook with just a few clicks with the integration of Meta Business API.

This matters in Malaysia because most sellers are mobile-first. Competitors like Shopee and Lazada have solid mobile apps with really good user experience. Thus, it is crucial that this platform has a mobile app that at least matches the quality of the competitors.

By using React Native, we can develop a mobile app for both iOS and Android in one codebase. The core functionalities will require camera access and image uploading. For push notification, we will use Firebase Cloud Messaging to send updates to the seller. Last but not least, we will introduce biometric login using fingerprint or face ID to make it secure and convenient for sellers.

8.5 Integrated Logistics

The last and most important feature to implement will be integrating shipping logistics to make it extremely convenient for sellers. My solution is creating a one-click shipping integration with Malaysian courier services such as J&T, PosLaju, and Ninja Van. Currently, social media sellers do this manually, thus having this feature will be a massive time saver for them.

The key feature will be as customers click checkout, the system will automatically calculate shipping cost for customers based on different courier service rates for customers to choose. After customers choose their preferred courier service, the amount is added to the order total. Then, the seller receives the order and just has to click “Book J&T Pickup” and our system will automatically fill in the required information such as sender and receiver address.

Following that, we can implement an auto-update feature that updates customers through WhatsApp when a package is picked up, delivering, and delivered. Plus, a link to the real-time tracking on J&T website can be attached to the message too.
Why this matters for Malaysian sellers is because shipping takes up the most time in the whole business process. Manual booking process can take up to 15 minutes per order which compounds as sellers get more orders. With integration of logistics, it could easily save hours of sellers time everyday making our platform much more valuable.

How we can integrate these courier services is by using their APIs such as J&T Express API, PosLaju API, and Ninja Van API. Then, we develop webhooks for tracking the shipping progress and use it to notify sellers as well as customers. Last but not least, delivery rates can be calculated using weight of package along with postcode making this a feasible implementation with huge value.

8.6 Implementation Roadmap

The below is a table of the implementation roadmap for the features suggested above.

Timeline	Features	Expected Impact
Month 1-3	WhatsApp Integration + Malaysian Payment Methods	Introduce local communication and payment options that build trust
Month 4-6	Mobile Management App + Logistics Integration	Allow sellers to scale their operations and help sellers save a significant amount of time

Month 6-9	Multi-Language Support	Expand to all demographics in Malaysia

8.7 Conclusion

These five enhancements will be able to address all the specific challenges in the Malaysian social media seller market. Currently, Shopee and Lazada are great platforms for sellers but charge commission and are a marketplace mess with all sorts of products. Additionally, Shopify is the best commercial e-commerce builder out there but is too expensive for Malaysian social media sellers to use.

That is where our platform has the advantage. Our platform will be the first solution for Malaysian social media sellers that provide local payments, integrate with WhatsApp, work with local courier services, support all Malaysian languages, provide a mobile-first design and provide all of these at an affordable pricing which appeals to sellers.
By implementing these features, our platform positions itself as the best solution for social media sellers that want to grow beyond social media into professional online businesses.

References

1. Agile Alliance. (2023). *What is Agile software development?* https://www.agilealliance.org/agile101/
2. Laravel. (2024). *Laravel 12.x documentation: Eloquent ORM*. https://laravel.com/docs/12.x/eloquent
3. Laravel. (2024). *Laravel Sanctum documentation: API token authentication*. https://laravel.com/docs/12.x/sanctum
4. Composer. (2024). *Composer documentation: Dependency management for PHP*. https://getcomposer.org/doc/
5. Mozilla Developer Network. (2024). *HTTP request methods and RESTful API design*. https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods
6. React. (2024). *React documentation: Getting started*. https://react.dev/learn
7. npm. (2024). *npm documentation: Package management*. https://docs.npmjs.com/
8. Google Developers. (2024). *OAuth 2.0 for client-side web applications*. https://developers.google.com/identity/protocols/oauth2
9. JSON Web Tokens. (2024). *Introduction to JWT authentication*. https://jwt.io/introduction
10. OWASP Foundation. (2024). *OWASP top ten web application security risks*. https://owasp.org/www-project-top-ten/
11. Stripe. (2024). *Payment Intents API: Accept online payments*. https://stripe.com/docs/payments/payment-intents
12. Pest PHP. (2024). *Pest testing framework documentation*. https://pestphp.com/docs/
13. Postman. (2024). *API testing and documentation guide*. https://learning.postman.com/docs/
14. GitHub. (2024). *Git documentation: Version control basics*. https://git-scm.com/doc
15. Figma. (2024). *Figma design documentation and best practices*. https://help.figma.com/hc/en-us
16. Malaysian Communications and Multimedia Commission. (2023). *Internet Users Survey 2022*. https://www.mcmc.gov.my/skmmgovmy/media/General/IUS-2022.pdf
17. Statista. (2024). *E-commerce statistics and market data Malaysia*. https://www.statista.com/topics/5776/e-commerce-in-malaysia/

Project Title: E-commerce Website Builder for Social Media Sellers

Student Name: Chan Zi Hao

Supervisor Name: Mr Koon Kim Peh

Date: 17/09/2025

---

1. Summary of Progress

- First meeting with supervisor. Business proposal has been completed and submitted before meeting so supervisor has understanding of what our project is. Initial project setup has also been done to kickstart project.

2. Tasks Completed This Week

Task	Description	Date Completed	Comments
[Task 1]	Business Proposal Completed	07/09/2025	Business Proposal has been completed and submitted to turnitin
[Task 2]	Project Repository & Github setup	16/09/2025	Laravel backend and React frontend repository setup and connected to remote repository on GitHub

3. Tasks in Progress

Task	Description	Expected Completion Date	Comments
[Task 1]	Complete database migration	24/09/2025	Finalizing required tables with partner and scoping out required fields for the project.
[Task 2]	Research on Implementation of OAuth for registration	21/09/2025	Research on how to implement Laravel Socialite to enable Google Login for store owners.

4. Planned Tasks for Next Week

Task	Description	Expected Start Date	Expected Completion Date
[Task 1]	Setup AuthController	18/09/2025	23/09/2025
[Task 2]	Finalise Authentication Flow in Figma	18/09/2025	23/09/2025

5. Challenges and Issues

- Briefly describe any challenges or issues encountered this week.

  Validating market demand in social media sellers and figuring out tech stack for the project

- Discuss how these were addressed or propose potential solutions.

  Research on market size, competitors and gaps in between this market has proved market demand exists. As for the tech stack, we settled on Laravel PHP for the backend as I have experienced from my internship, and we decided on React JS for the frontend as my partner has experience in React from his internship.

6. Supervisor Feedback and Comments

- Record any feedback or suggestions provided by the supervisor during the meeting.

  Supervisor asked many questions regarding our project target market to ensure we are clear as to who are the people we are targeting, and what is our unique selling point compared to our competitors.

Student's Signature

[illegible]

Name: Chan Zi Hao

Date: 17/09/2025

Supervisor's Signature

[illegible]

Name: Koon Kim Peh

Date: 17 September 2025

Project Title: E-commerce Website Builder for Social Media Sellers

Student Name: Chan Zi Hao

Supervisor Name: Mr Koon Kim Peh

Date: 03/10/2025

---

1. Summary of Progress

- Provide a brief summary of the project's current status and any progress made since the last meeting.

2. Tasks Completed This Week

Task	Description	Date Completed	Comments
[Task 1]	AuthController for Store Owner	03/10/2025	Successfully tested API using Postman and also implemented OAuth using Laravel Socialite
[Task 2]	Database Migration for all tables	03/10/2025	Successfully migrated all tables into a SQLite database

3. Tasks in Progress

Task	Description	Expected Completion Date	Comments
[Task 1]	Implementation of StoreController	05/10/2025	Implemented Create and Read function for store table with proper validation, left with Update and Delete function
[Task 2]	Setting up of API routes	05/10/2025	Defining all routes for all backend APIs required for frontend to call

4. Planned Tasks for Next Week

Task	**Description**	**Expected Start Date**	**Expected Completion Date**
[Task 1]	Completion of StoreController and ProductController	03/10/2025	08/10/2025
[Task 2]	Comprehensive Entity Relationship Diagram (ERD) which includes all tables	03/10/2025	08/10/2025

5. Challenges and Issues

- Briefly describe any challenges or issues encountered this week.

  Refreshing my knowledge on setting up Controllers and Migration Schemas in Laravel PHP as well as method to setup Laravel Socialite.

- Discuss how these were addressed or propose potential solutions.

  Looking up Laravel PHP documentation online helped me remember how to make the Controllers and Migration.

6. Supervisor Feedback and Comments

- Record any feedback or suggestions provided by the supervisor during the meeting.

  Supervisor advised to make ERD diagram to see a clearer picture of database relationship as well as to speed up development to make it in time for presentation.

Student’s Signature	Supervisor’s Signature
czihao.	[illegible]
Name: Chan Zi Hao	Name: Koon Kim Peh
Date: 03/10/2025	Date: 3 October 2025

Project Title: E-commerce Website Builder for Social Media Sellers

Student Name: Chan Zi Hao

Supervisor Name: Mr Koon Kim Peh

Date: 08/10/2025

---

1. Summary of Progress

- Provide a brief summary of the project's current status and any progress made since the last meeting.

2. Tasks Completed This Week

Task	Description	Date Completed	Comments
[Task 1]	Completion of StoreController and ProductController	08/10/2025	The CRUD functionalities for both Stores and Products table is ready to be called by the frontend
[Task 2]	Comprehensive Entity Relationship Diagram (ERD) which includes all tables	08/10/2025	The ERD diagram is made and ready for review by supervisor

3. Tasks in Progress

Task	Description	Expected Completion Date	Comments
[Task 1]	Make Product and Store API POST request from frontend	10/10/2025	Must submit post request body in formData, cannot submit body in JSON as we are also sending photo files which cannot be stored in JSON.

[Task 2]	Make Product and Store API GET request from frontend	10/10/2025	Send GET request along with the token in header as the API routes require user to be authorized if not the GET request will fail.

4. Planned Tasks for Next Week

Task	**Description**	**Expected Start Date**	**Expected Completion Date**
[Task 1]	Migrate a new Customers table and add relevant foreign keys in other tables	11/10/2025	18/10/2025
[Task 2]	Make CustomerAuthController for customers to register and login to the e-commerce stores	11/10/2025	18/10/2025

5. Challenges and Issues

- Briefly describe any challenges or issues encountered this week.

  Encountered challenge in sending image files to the backend server as conventional JSON body does not work for image files.

- Discuss how these were addressed or propose potential solutions.

  Figured out that we can append all data into a formData before submitting the whole formData object to the backend which works.

6. Supervisor Feedback and Comments

- Record any feedback or suggestions provided by the supervisor during the meeting.

  Supervisor suggests adding login and registration for e-commerce stores thus we are implementing authentication for customers as well.

Student's Signature	Supervisor's Signature [illegible]
czihao.	
Name: Chan Zi Hao	Name: Koon Kim Peh
Date: 08/10/2025	Date: 8 October 2025

Appendix A: Weekly Project Progress Report Template

Project Title: E-commerce Website Builder for Social Media Sellers

Student Name: Muhammad Aidel Bin Mustapha Kamal

Supervisor Name: Mr. Koon

Date: 17/9/2025

---

1. Summary of Progress

Team officially began the Miles Frontend Development project by presenting and discussing the **project proposal** with the supervisor. The meeting focused on defining project objectives, identifying the main modules, and outlining individual responsibilities between team members.

2. Tasks Completed This Week

Task	Description	Date Completed	Comments
Project Proposal Discussion	Presented and refined the Miles project idea with the supervisor	16 Sept 2025	Supervisor approved concept and provided feedback for improvements
Wireframe Design	Created wireframes for landing page, registration, store setup, and dashboard	16 Sept 2025	Finalized wireframe layout using Figma for frontend reference

3. Tasks in Progress

Task	Description	Expected Completion Date	Comments
Wireframe Adjustments	Refining layout consistency and navigation flow	24 Sept 2025	

4. Planned Tasks for Next Week

Task	Description	Expected Start Date	Expected Completion Date
Wireframe Adjustments	Refining layout consistency and navigation flow	17 Sept 2025	24 Sept 2025
Navbar and Authentication Page	Implement navigation bar and login/register forms	17 Sept 2025	24 Sept 2025

Appendix A: Weekly Project Progress Report Template

5. Challenges and Issues

- Briefly describe any challenges or issues encountered this week.
  The main challenge during the first week was aligning the wireframe design with project functionality and ensuring each page supports smooth navigation once implemented.
- Discuss how these were addressed or propose potential solutions.
  Discuss about layout hierarchy and navigation logic with the backend team before coding and continue refining the wireframes to maintain consistency across all modules.

6. Supervisor Feedback and Comments

- Record any feedback or suggestions provided by the supervisor during the meeting.
  The supervisor approved the overall project proposal and appreciated the initial wireframe design.

Student’s Signature	Supervisor’s Signature
Name:	Name:
Date:	Date:

Appendix A: Weekly Project Progress Report Template

Project Title: E-commerce Website Builder for Social Media Sellers

Student Name: Muhammad Aidel Bin Mustapha Kamal

Supervisor Name: Mr. Koon

Date: 3/10/2025

---

1. Summary of Progress

The frontend work focused on establishing the core structure and navigation of the Miles platform. The developer landing page, navigation bar, and user authentication pages (registration and login) were successfully completed, forming the foundation of the seller module. Progress was also made toward developing seller-specific features such as store details and product management pages, which are planned for further development in the coming week.

2. Tasks Completed This Week

Task	Description	Date Completed	Comments
Developer Landing Page	Designed and implemented the homepage layout for developers and sellers	1 Oct 2025	Ensures clear navigation and visual consistency
Navigation Bar	Created responsive navigation bar linking all main sections	1 Oct 2025	Create 3 types of navigation bar, developer mode, admin mode, and customer view mode
Registration and Login Page	Developed authentication interface for sellers	30 Sept 2025	Form validation and layout completed; awaiting API integration

Appendix A: Weekly Project Progress Report Template

3. Tasks in Progress

Task	Description	Expected Completion Date	Comments
Store Details Page	Designing form for sellers to input their store information	3 Oct 2025	
Add Product Page	Developing product upload form and layout	3 Oct 2025	
Seller Dashboard	Building main seller dashboard with overview and navigation	3 Oct 2025	

4. Planned Tasks for Next Week

Task	Description	Expected Start Date	Expected Completion Date
Store Details Page	Designing form for sellers to input their store information	24 Sept 2025	3 Oct 2025
Add Product Page	Developing product upload form and layout	24 Sept 2025	3 Oct 2025
Seller Dashboard	Building main seller dashboard with overview and navigation	24 Sept 2025	3 Oct 2025

5. Challenges and Issues

- Briefly describe any challenges or issues encountered this week.
  One challenge this week was maintaining proper navigation flow between different. Some route configurations in React Router caused page reload issues during navigation. Another issue was ensuring UI consistency across pages while using Material UI's grid system.
- Discuss how these were addressed or propose potential solutions.
  Adjust React Router configuration to use nested routes for smoother navigation. Continue refining Material UI styles for consistent spacing, color palette, and typography across all pages

6. Supervisor Feedback and Comments

- Record any feedback or suggestions provided by the supervisor during the meeting.
  The supervisor encouraged focusing on completing all seller-related pages to ensure the full seller flow is functional.

Student's Signature	Supervisor's Signature
Name:	Name:
Date:	Date:

Appendix A: Weekly Project Progress Report Template

Project Title: E-commerce Website Builder for Social Media Sellers

Student Name: Muhammad Aidel Bin Mustapha Kamal

Supervisor Name: Mr. Koon

Date: 8/10/2025

---

1. Summary of Progress

The frontend continued to make progress in designing and implementing the seller interface for the Miles platform. Key focus areas included refining the seller dashboard layout, completing the store form page, and improving the user interface to ensure consistency across all pages. And began developing the add product form, which allows sellers to upload and manage their product details easily.

2. Tasks Completed This Week

Task	**Description**	**Date Completed**	**Comments**
Store Details Form Page	Designed and coded form for sellers to input store information	6 Oct 2025	Integrated with navigation and validated form fields
Add Product Form	Created add product page with input fields and image upload	7 Oct 2025	Ready for backend API integration
Seller Dashboard	Developed dashboard page displaying store overview and navigation tabs	7 Oct 2025	

3. Tasks in Progress

Task	Description	Expected Completion Date	Comments
Product Page Design	Designing customer-facing product listing page	15 Oct 2025	
Checkout Page	Implement UI for checkout process and test Stripe redirection	15 Oct 2025	

4. Planned Tasks for Next Week

Task	Description	Expected Start Date	Expected Completion Date
Product Page Design	Designing customer-facing product listing page	9 Oct 2025	15 Oct 2025
Checkout Page	Implement UI for checkout process and test Stripe redirection	9 Oct 2025	15 Oct 2025

5. Challenges and Issues

- Briefly describe any challenges or issues encountered this week.
  One main challenge this week was ensuring **layout consistency and responsiveness** across different pages using Material UI components. Some spacing and alignment issues occurred when switching between mobile and desktop views.
- Discuss how these were addressed or propose potential solutions.
  Continue fine-tuning the CSS grid and Material UI breakpoints to improve responsive behavior.

6. Supervisor Feedback and Comments

- Record any feedback or suggestions provided by the supervisor during the meeting.

  The supervisor advised maintaining design consistency across all seller modules and recommended focusing on user experience improvements for the checkout page layout.

Student's Signature	Supervisor's Signature
Name:	Name:
Date:	Date: