ITE 18 – Application Development and Emerging Technologies

Web/Mobile Application - Final Project Submission

Student Name(s): **Jan Franscine P. Herbolingo**

Project Title: **CarSUcart:** A Web-Based Campus E-Commerce System for Caraga State University

Date of Submission: **12/24/2025**

Course/Section: **ITE-18/ EJ1**

1. Project Overview

1.1 Project Description

CarSUcart is a specialized, web-based e-commerce platform developed specifically for the academic community of **Caraga State University (CSU)**. In the current campus ecosystem, student entrepreneurs and student organizations often rely on fragmented channels—such as social media groups, physical bulletin boards, or pop-up stalls—to sell products and services. This decentralized approach results in information asymmetry, limited market reach, and inefficient transaction tracking.

CarSUcart solves these issues by providing a centralized digital marketplace. It serves as a bridge between campus buyers (students, faculty, and staff) and sellers (student entrepreneurs and organizations). The system facilitates product listing, inventory management, secure ordering, and sales analytics within a controlled, campus-centric environment. By digitizing these transactions, it promotes the university's entrepreneurial spirit while ensuring a streamlined, paperless, and efficient shopping experience.

1.2 Target Users

The system is designed to accommodate three distinct user roles, each with specific technical needs and system interactions:

1. **Campus Buyers (Students & Faculty):**

Needs: Ease of access to product information, efficient search and filtering, secure cart management, and transparent order tracking.

Interaction: Front-end browsing, cart manipulation, and order history review.

2. **Sellers (Student Entrepreneurs & Organizations)**:

Needs: Tools to manage inventory (CRUD operations), track orders, and view financial performance.

Interaction: Seller dashboard access for product uploads, stock adjustments, and sales reporting.

3. **System Administrators**:

Needs: Oversight of the entire platform to ensure compliance with campus policies, user management, and category maintenance.

Interaction: Admin panel for user moderation, category management, and system-wide analytics.

1.3 Key Features

- **Role-Based Access Control (RBAC)**: Distinct interfaces and permission sets for Buyers, Sellers, and Admins to ensure security and workflow segregation.
- **Dynamic Product Catalog**: A searchable and filterable listing of products, categorized by academic and lifestyle needs (e.g., School Supplies, Food, Uniforms).
- **Shopping Cart System**: A persistent storage mechanism allowing users to accumulate items, adjust quantities, and view subtotal calculations prior to checkout.
- **Order Lifecycle Management**: A comprehensive workflow handling order creation, status updates (Pending, Processing, Completed, Cancelled), and history tracking.
- **Seller Analytics Dashboard**: Visual representation of sales data, including monthly revenue and order volume, aiding in business decision-making.
- **Review and Rating System**: A feedback loop allowing buyers to rate products, fostering trust and quality control within the marketplace.
- **Secure Authentication**: Implementation of token-based authentication (Laravel Sanctum) to protect user sessions and data.

1.4 Technology Stack

The selection of technologies was driven by the need for scalability, security, and modern user experience standards.

- **Frontend: React 19 (via Vite) & Tailwind CSS**
  - Justification: React provides a component-based architecture allowing for a reactive, Single Page Application (SPA) experience. Vite ensures rapid development build times. Tailwind CSS was chosen for its utility-first approach, enabling rapid UI prototyping and consistent styling compliant with modern web standards.
- **Backend: Laravel 12 (PHP)**
  - Justification: Laravel is an enterprise-grade PHP framework known for its elegant syntax and robust security features (CSRF protection, SQL injection prevention). It follows the Model-View-Controller (MVC) pattern, which structures the backend logic efficiently.
- **Database: MySQL**
  - Justification: A relational database management system (RDBMS) is essential for maintaining complex relationships between users, orders, and products. MySQL offers the reliability and ACID compliance required for transactional e-commerce systems.
- **Development Tools:**
  - **Visual Studio Code:** Primary IDE.
  - **Postman:** For API endpoint testing and documentation.
  - **XAMPP:** Local development environment server.
  - **Git:** Version control for source code management.

2. App Plan

2.1 Project Scope

In-Scope

- User registration and authentication system with role selection.
- Product management (Create, Read, Update, Delete) for sellers.
- Category management for administrators.
- Cart functionality (Add, Remove, Update Quantity, Clear).
- Checkout process and Order creation.
- Order status tracking and history.
- Basic sales analytics for sellers.
- Product reviews and ratings.

Out-of-Scope:

- **Third-party Payment Gateway Integration**: The system uses "Cash on Pickup" or manual reference checking for digital payments to avoid sandbox complexities and fees during the academic pilot.
- **Real-time Shipping API**: Logistics are restricted to campus pickup or internal delivery services; integration with couriers like FedEx/DHL is excluded.
- **Live Chat:** Real-time messaging between buyer and seller is reserved for future iterations.

2.2 Objectives & Goals

1. **Centralization:** To reduce the time spent by students searching for products by 40% through a unified platform.
2. **Efficiency:** To automate the order calculation and tracking process, eliminating manual errors associated with paper-based or chat-based ordering.
3. **Security:** To achieve 100% data integrity for user accounts and transaction records using industry-standard encryption and authentication protocols.
4. **Usability**: To achieve a System Usability Scale (SUS) score of at least 75 in user acceptance testing.

2.3 User Stories & Use Cases

User Story 1: Buyer Purchasing

- **Story:** "As a student buyer, I want to add multiple items to my cart and checkout in one go so that I can save time purchasing school supplies."
    - **Acceptance Criteria:**
        - ✓ System must validate stock availability before adding to cart.
        - ✓ Cart subtotal must update instantly upon quantity change.
        - ✓ Checkout must create a unique Order ID and deduct stock from the database.

User Story 2: Seller Inventory Management

- **Story:** "As a student seller, I want to update the stock level of my products so that buyers do not order out-of-stock items."
    - **Acceptance Criteria:**
        - ✓ Seller must be able to edit the 'stock' field of their own products.
        - ✓ Changes must be reflected immediately on the public product listing.
        - ✓ System must prevent setting negative stock values.

2.4 System Architecture

CarSUcart utilizes a **Client-Server Architecture** employing RESTful APIs.

- **Client Layer:** The React application runs in the user's browser, handling the presentation logic and user input.
- **API Layer:** The Laravel application acts as the API provider, receiving JSON requests, processing business logic, and returning JSON responses.
- **Data Layer:** MySQL stores persistent data.

This separation allows for independent scaling of the frontend and backend and facilitates future development.

2.5 Development Timeline

- **Phase 1: Requirement Analysis (Weeks 1-2):** Gathering user requirements and defining database schema.
- **Phase 2: Database & Backend Implementation (Weeks 3-6):** Setting up Laravel, migrations, models, and API controllers.
- **Phase 3: Frontend Development (Weeks 7-10):** Building React components, integrating API calls, and styling with Tailwind.
- **Phase 4: Integration & Testing (Weeks 11-12):** Connecting frontend to backend, performing unit and integration tests.
- **Phase 5: Deployment & Documentation (Weeks 13-14):** Finalizing documentation and deploying to the local production environment.

3. UI/UX Design

3.1 Design Philosophy

The interface design follows the **"Mobile-First"** and **"Clean UI"** philosophies. Given that the primary demographic (students) accesses the internet predominantly via smartphones, the layout utilizes responsive grids (Tailwind CSS) that adapt to various screen sizes. Unnecessary clutter is removed to focus the user's attention on products and call-to-action buttons.

3.2 Color Scheme

The color palette reflects the branding of Caraga State University (Green/Gold/White) while maintaining modern aesthetic standards.

- **Primary Green (#059669 - Emerald 600):** Used for primary buttons, active states, and branding. Symbolizes growth and the university identity.
- **Background White (#F3F4F6 - Gray 100):** Used for the main application background to reduce eye strain and provide contrast.
- **Text Dark (#1F2937 - Gray 800):** Used for primary text to ensure high readability ratios.
- **Alert Red (#EF4444 - Red 500):** Used for error messages and destructive actions (e.g., "Delete Item").

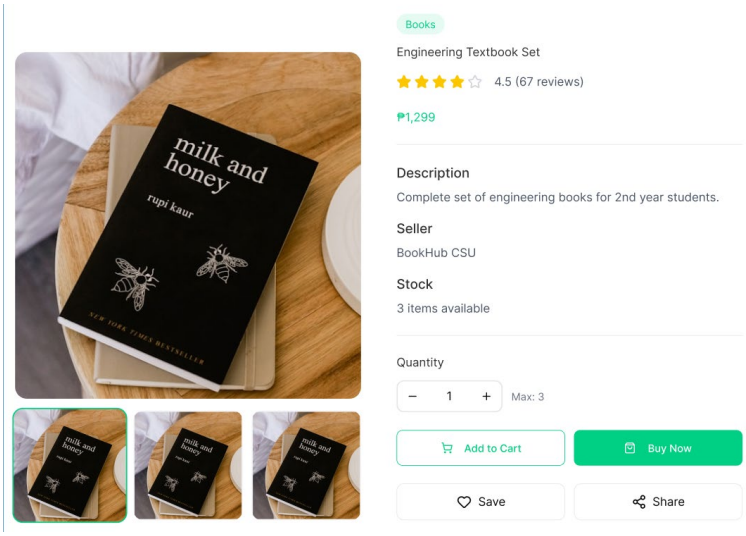
3.3 Typography

Font Family: Inter or **Roboto (San-serif).**

- **Justification:** Sans-serif fonts offer superior legibility on digital screens compared to serif fonts. The variable font weights allow for clear hierarchy (e.g., Bold for Headers, Regular for body text).

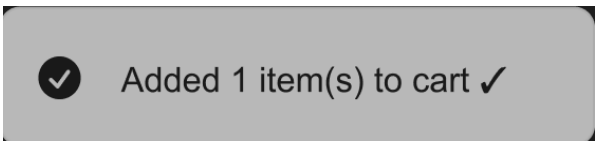
3.4 UI Components

- **Product Card:** A self-contained component displaying product image, title, price, and an "Add to Cart" button. It features a hover effect to indicate interactivity.



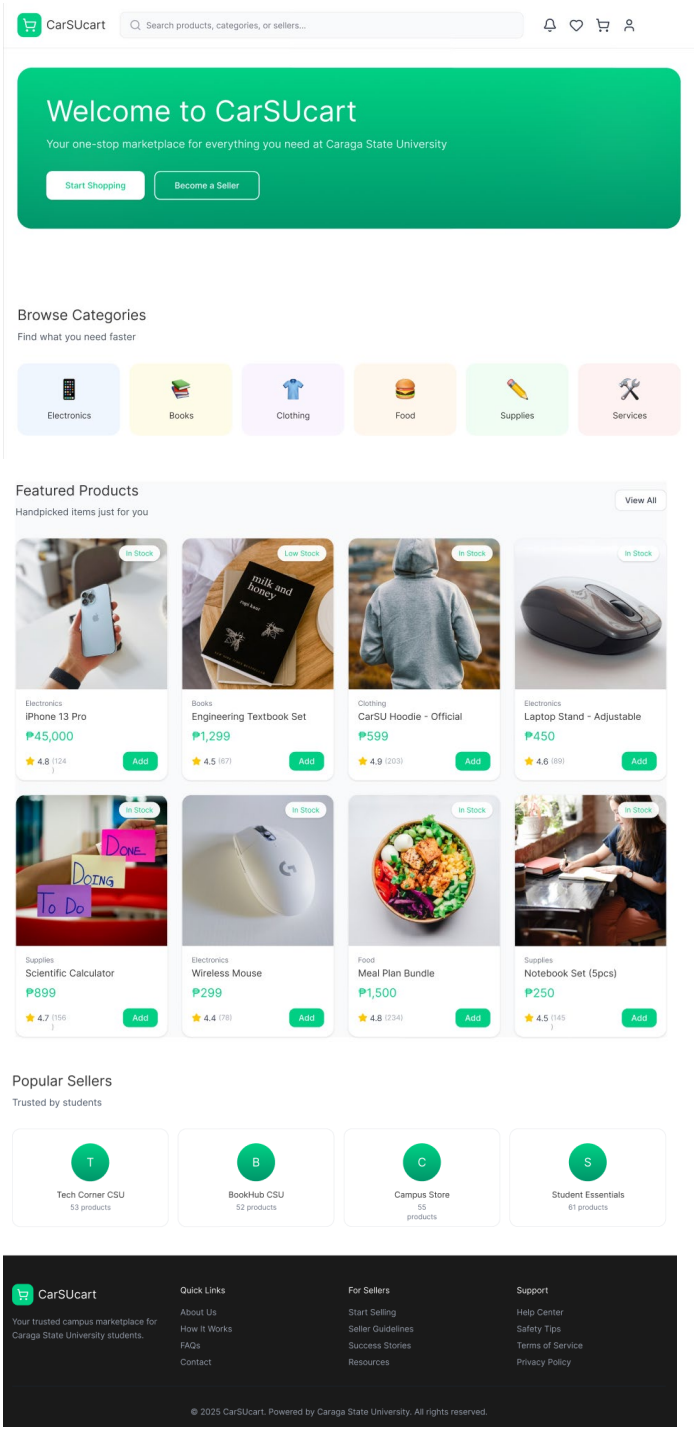
- **Modal Dialogs:** Used for critical confirmations (e.g., "Confirm Order Placement") to prevent accidental actions.

- **Toast Notifications:** Non-intrusive pop-ups that provide immediate feedback (e.g., "Added 1 item(s) to cart") without blocking the user flow.

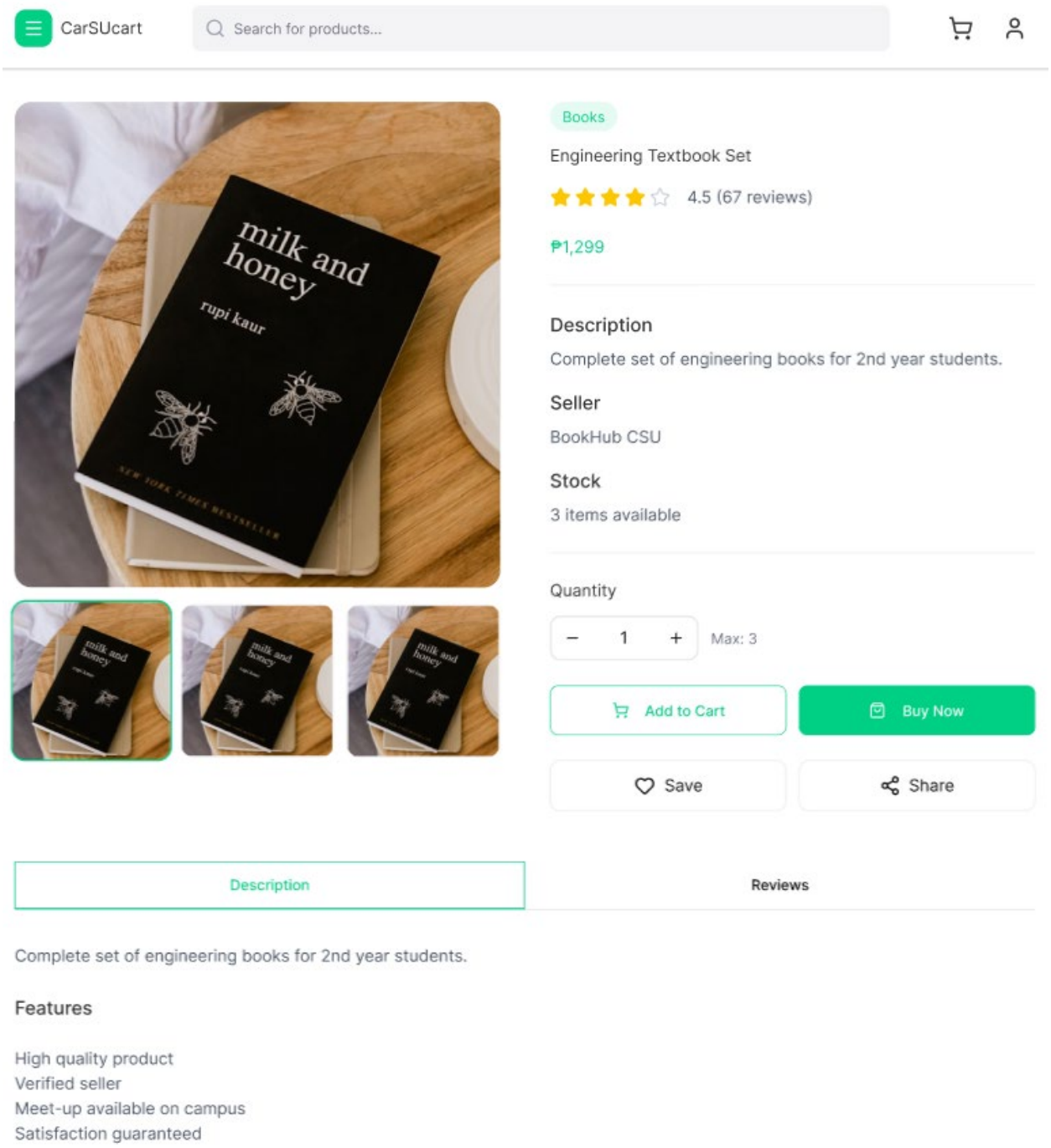


3.5 Wireframes & Mockups

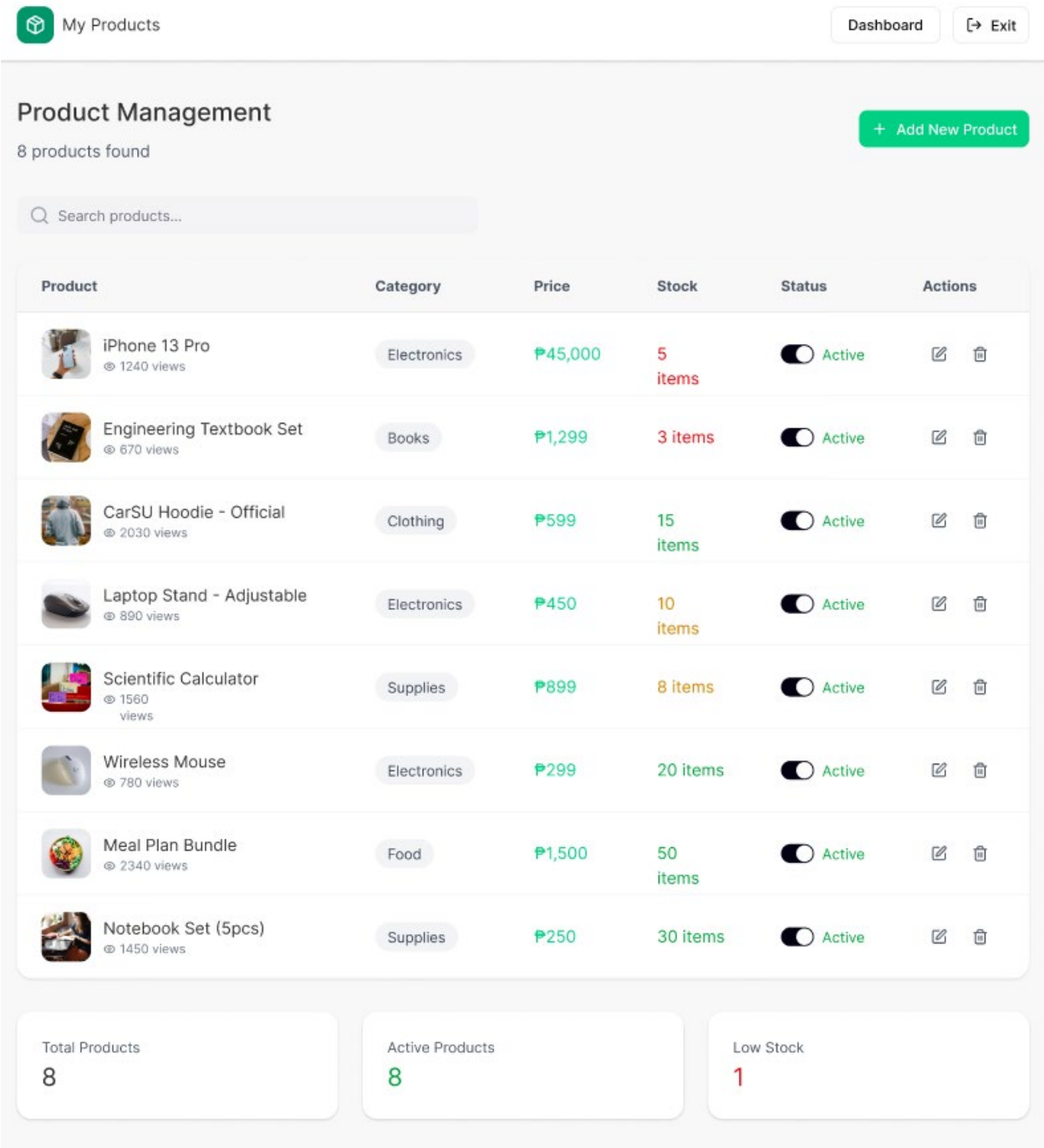
- **Home/Landing Page:** Features a hero banner, a search bar, and a grid of "Featured Products."



- **Product Details Page:** Displays high-resolution images, description, stock count, and seller information.



- **Seller Dashboard**: A table view of listed products with "Edit" and "Delete" actions, alongside a graphical chart of monthly sales.



3.6 User Flows

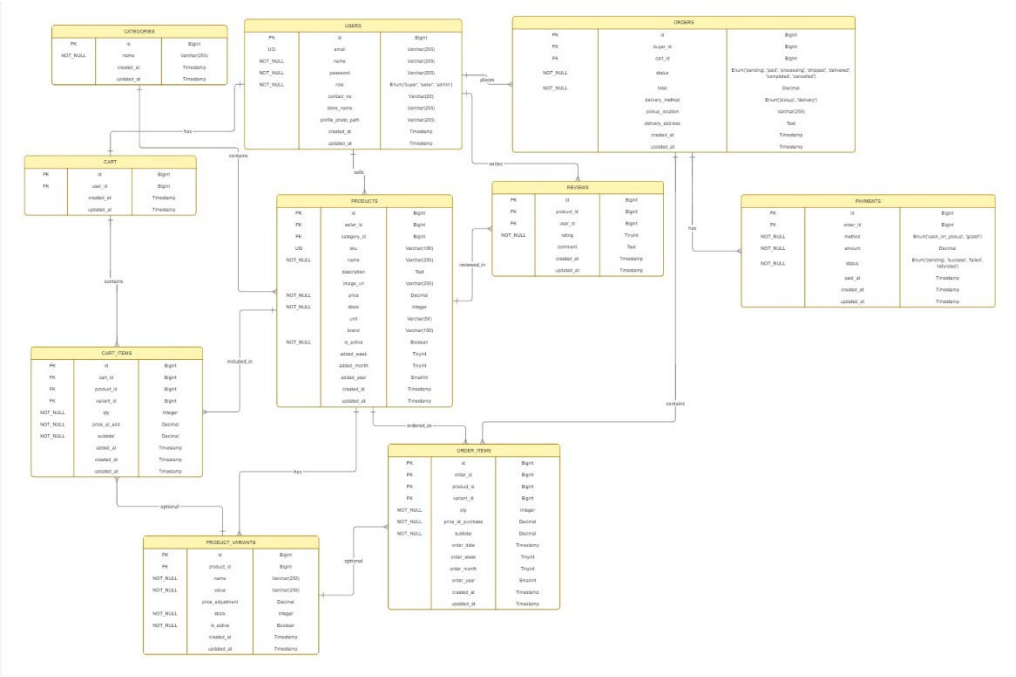
Purchase Flow:

1. **Login:** User authenticates via `/login`.
2. **Browse:** User filters products by category on the Home page.
3. **Selection:** User clicks a product, views details, and clicks "Add to Cart".
4. **Review:** User navigates to the Cart page to review total price.
5. **Checkout:** User clicks "Checkout", selects payment method, and confirms.
6. **Confirmation:** System generates an Order ID and shows a success message.

4. Database Architecture (ERD)

4.1 Entity Relationship Diagram

The database is built on a relational model centered around the `users` and `products` tables. The `orders` table acts as a transactional junction, linking buyers to the products they purchase via `order_items`.



4.2 Entity Descriptions

Table: User

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each user.
name	VARCHAR	NOT NULL	Full name of the user.
email	VARCHAR	NOT NULL, UNIQUE	Unique email address used for login.
password	VARCHAR	NOT NULL	Hashed password for authentication.
role	ENUM	NOT NULL	User role (buyer, seller, admin).
contact_no	VARCHAR	NULL	Contact number of the user.
store_name	VARCHAR	NULL	Name of the shop (only applicable to sellers).
profile_photo_path	VARCHAR	NULL	Path to the user's profile photo.
created_at	TIMESTAMP	NOT NULL	Date and time the account was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the account was last updated.

Table: Categories

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each category.
name	VARCHAR	NOT NULL	Name of the product category.
created_at	TIMESTAMP	NOT NULL	Date and time the category was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the category was last updated.

Table: Products

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each product.
seller_id	BIGINT	NOT NULL, FOREIGN KEY → users(id)	References the seller who owns the product.
category_id	BIGINT	NOT NULL, FOREIGN KEY → categories(id)	References the product category.
sku	VARCHAR	UNIQUE	Stock Keeping Unit identifier.
name	VARCHAR	NOT NULL	Product title or name.
description	TEXT	NULL	Detailed description of the product.
image_url	VARCHAR	NULL	Image URL of the product.
price	DECIMAL	NOT NULL, CHECK (price ≥ 0)	Unit price of the product.
stock	INTEGER	NOT NULL	Available stock quantity.
unit	VARCHAR	NULL	Unit of measurement (e.g., piece, pack).
brand	VARCHAR	NULL	Brand of the product.
is_active	BOOLEAN	DEFAULT TRUE	Indicates if the product is active or visible.
added_week	TINYINT	NULL	Week when the product was added.
added_month	TINYINT	NULL	Month when the product was added.
added_year	SMALLINT	NULL	Year when the product was added.
created_at	TIMESTAMP	NOT NULL	Date and time the product was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the product was last updated.

Table: Product_Variants

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each product variant.
product_id	BIGINT	NOT NULL, FOREIGN KEY → products(id)	References the parent product.
name	VARCHAR	NOT NULL	Variant name (e.g., Size, Color).
value	VARCHAR	NOT NULL	Variant value (e.g., Large, Red).
price_adjustment	DECIMAL	DEFAULT 0	Additional price adjustment for the variant.
stock	INTEGER	NOT NULL	Available stock for the variant.
is_active	BOOLEAN	DEFAULT TRUE	Indicates if the variant is active.
created_at	TIMESTAMP	NOT NULL	Date and time the variant was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the variant was last updated.

Table: Cart

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each cart.
user_id	BIGINT	NOT NULL, FOREIGN KEY → users(id)	References the user who owns the cart.
created_at	TIMESTAMP	NOT NULL	Date and time the cart was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the cart was last updated.

Table: Cart_items

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each cart item.
cart_id	BIGINT	NOT NULL, FOREIGN KEY → cart(id)	References the cart.
product_id	BIGINT	NOT NULL, FOREIGN KEY → products(id)	References the product added to the cart.
variant_id	BIGINT	NULL, FOREIGN KEY → product_variants(id)	References the selected product variant (optional).
qty	INTEGER	NOT NULL	Quantity of the product added.
price_at_add	DECIMAL	NOT NULL	Price of the product at the time it was added.
subtotal	DECIMAL	NOT NULL	Subtotal amount for the cart item.

added_at	TIMESTAMP	NOT NULL	Date and time the item was added to the cart.
created_at	TIMESTAMP	NOT NULL	Date and time the record was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the record was last updated.

Table: Orders

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each order.
buyer_id	BIGINT	NOT NULL, FOREIGN KEY → users(id)	References the buyer who placed the order.
cart_id	BIGINT	NOT NULL, FOREIGN KEY → cart(id)	References the cart used for checkout.
status	ENUM	NOT NULL	Order status (pending, paid, processing, shipped, delivered, completed, cancelled).
total	DECIMAL	NOT NULL, CHECK (total ≥ 0)	Total cost of the order.
delivery_method	ENUM	NOT NULL	Delivery method (pickup, delivery).
pickup_location	VARCHAR	NULL	Pickup location if pickup is selected.
delivery_address	TEXT	NULL	Delivery address if delivery is selected.
created_at	TIMESTAMP	NOT NULL	Date and time the order was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the order was last updated.

Table: Orders_items

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each order item.
order_id	BIGINT	NOT NULL, FOREIGN KEY → orders(id)	References the order.
product_id	BIGINT	NOT NULL, FOREIGN KEY → products(id)	References the purchased product.
variant_id	BIGINT	NULL, FOREIGN KEY → product_variants(id)	References the product variant (optional).
qty	INTEGER	NOT NULL	Quantity ordered.
price_at_purchase	DECIMAL	NOT NULL	Price of the product at the time of purchase.
subtotal	DECIMAL	NOT NULL	Subtotal amount for the order item.
order_date	TIMESTAMP	NOT NULL	Date and time the order was placed.
order_week	TINYINT	NULL	Week when the order was placed.
order_month	TINYINT	NULL	Month when the order was placed.
order_year	SMALLINT	NULL	Year when the order was placed.
created_at	TIMESTAMP	NOT NULL	Date and time the record was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the record was last updated.

Table: Payment

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each payment.
order_id	BIGINT	NOT NULL, FOREIGN KEY → orders(id)	References the related order.
method	ENUM	NOT NULL	Payment method (cash_on_pickup, gcash).
amount	DECIMAL	NOT NULL	Amount paid for the order.
status	ENUM	NOT NULL	Payment status (pending, success, failed, refunded).
paid_at	TIMESTAMP	NULL	Date and time the payment was made.
created_at	TIMESTAMP	NOT NULL	Date and time the payment record was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the payment record was last updated.

Table: Reviews

Field	Data Type	Constraints	Description
id	BIGINT	PRIMARY KEY, AUTO_INCREMENT	Unique identifier for each review.
product_id	BIGINT	NOT NULL, FOREIGN KEY → products(id)	References the reviewed product.
user_id	BIGINT	NOT NULL, FOREIGN KEY → users(id)	References the user who wrote the review.
rating	TINYINT	NOT NULL	Rating score given by the user.
comment	TEXT	NULL	Written feedback or comment.
created_at	TIMESTAMP	NOT NULL	Date and time the review was created.
updated_at	TIMESTAMP	NOT NULL	Date and time the review was last updated.

4.3 Relationships

- **Users 1: N Products:** A single seller can list multiple products.
- **Users 1:1 Cart**: Each user has one shopping cart.
- **Users 1: N Orders:** A single buyer can place multiple orders.
- **Categories 1: N Products:** A category contains multiple products.
- **Products 1: N Product Variants:** A product can have multiple variants.
- **Cart 1: N Cart Items:** A cart can contain multiple items.
- **Orders 1: N Order Items:** An order is composed of multiple line items.
- **Orders 1: N Payments:** An order may have one or more payment records.
- **Users 1: N Reviews:** A user can write multiple reviews.
- **Products 1: N Reviews:** A product can receive multiple reviews.

4.4 Database Normalization

The database schema adheres to the **Third Normal Form (3NF):**

- **1NF:** All columns contain atomic values.
- **2NF:** All non-key attributes are fully functional dependent on the primary key.
- **3NF:** There are no transitive dependencies (e.g., product category names are stored in a separate `categories` table, referenced only by ID in the `products` table, preventing data redundancy).

5. Application Features & Functionality

5.1 User Authentication & Authorization

The system implements a comprehensive authentication and authorization mechanism using **Laravel Sanctum** for token-based API authentication. This ensures secure access to protected resources and role-based functionality.

Authentication Features:

- **Token-Based Authentication:** Upon successful login, the system issues a unique API token via Laravel Sanctum, which must be included in subsequent API requests via the `Authorization` header.
- **User Registration:** Supports registration for three distinct roles: `buyer`, `seller`, and `admin`. Each role has specific registration requirements (e.g., sellers must provide a `store_name`).
- **Password Security:** All passwords are hashed using Bcrypt before storage, ensuring that plain-text passwords are never stored in the database.

- **Session Management:** Tokens can be revoked upon logout, providing secure session termination

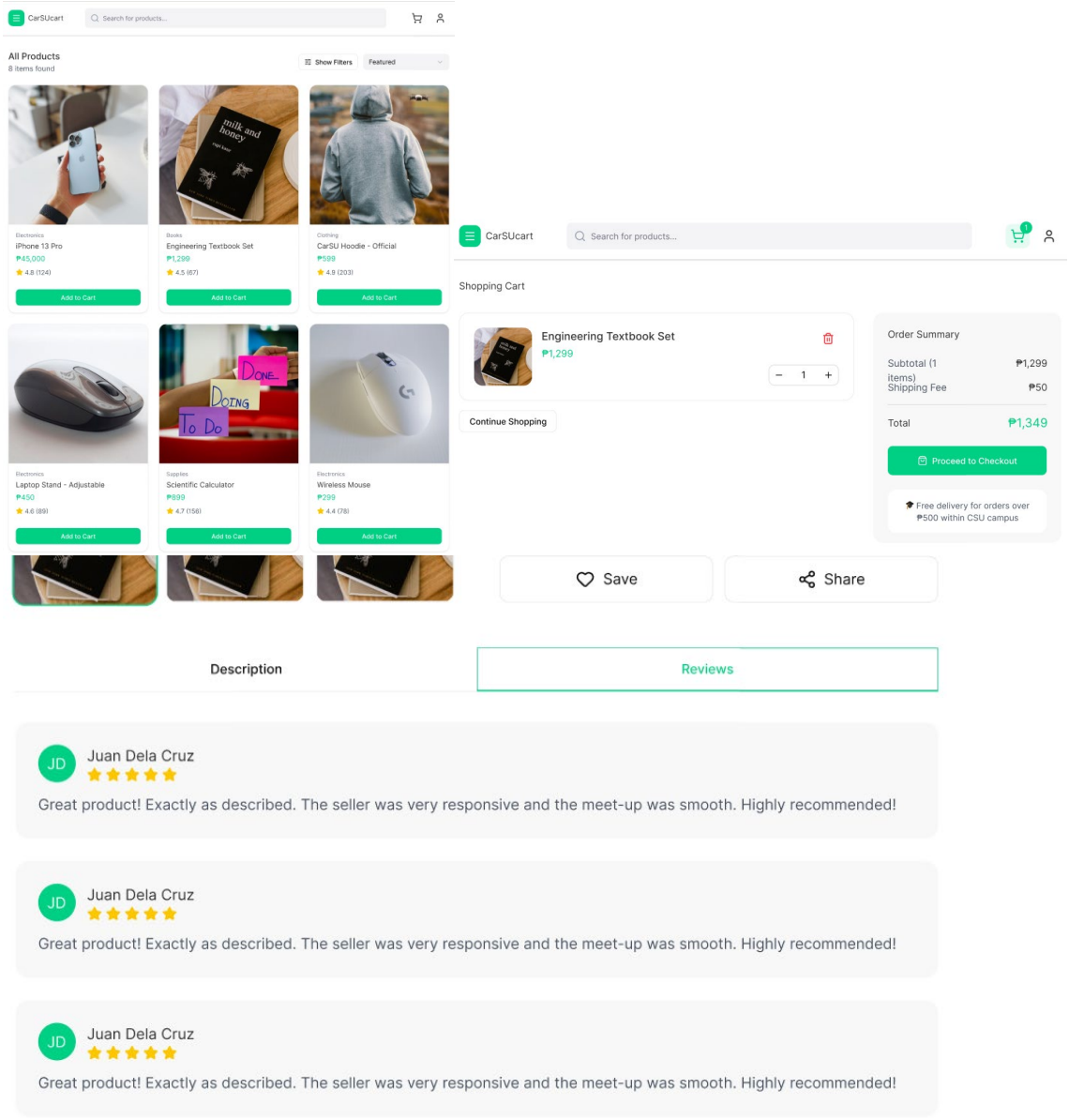
Authorization & Middleware:

- `auth:sanctum`: Verifies if the token is valid and the user is authenticated before allowing access to protected routes.
- `role:seller`: Checks if the authenticated user has the 'seller' role before allowing product creation, editing, or deletion operations.
- `role:admin`: Restricts access to administrative functions such as user management, category management, and system-wide analytics.

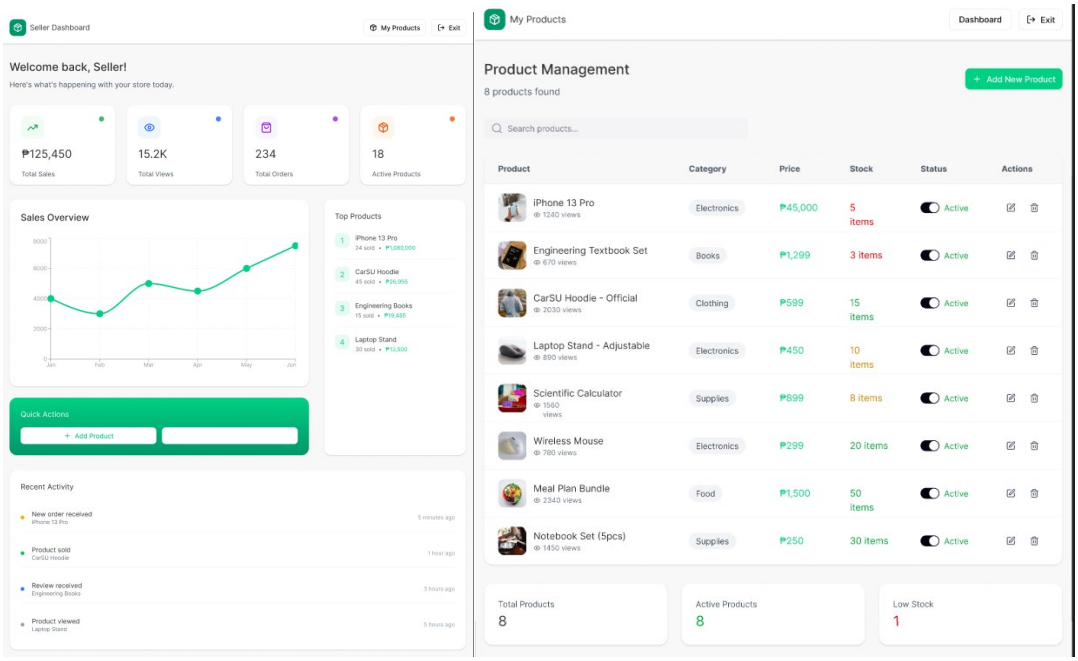
Resource Ownership: Sellers can only modify their own products, and buyers can only access their own cart and order history, enforced through ownership checks in controllers.

Role-Based Access Control (RBAC):

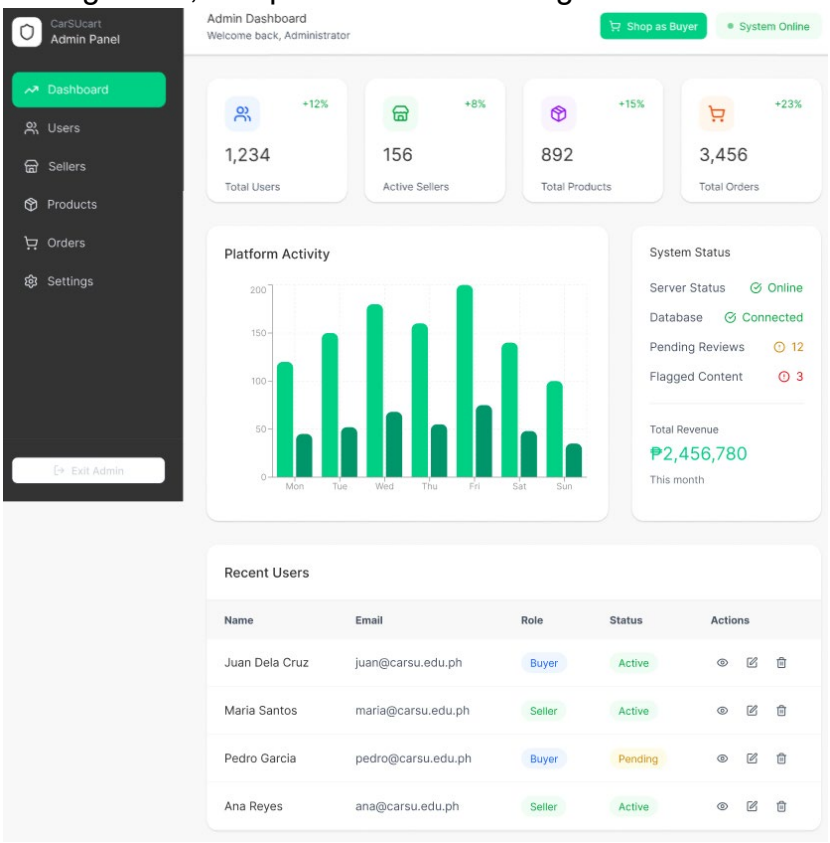
- **Buyers:** Can browse products, manage cart, place orders, and write reviews.



- **Sellers:** Can manage their product inventory, view sales analytics, and track orders for their products.



- **Admins:** Have full system access including user management, category management, and platform-wide oversight.



5.2 Product Management (CRUD)

The system provides comprehensive product management capabilities for sellers, enabling them to create, read, update, and delete products with full inventory control.

Product Creation:

- **Automatic Seller Assignment:** When a product is created, the system automatically assigns the `seller_id` based on the currently logged-in user, ensuring data ownership security and preventing unauthorized product creation.
- **Category Association:** Products must be associated with a category from the predefined category list, facilitating organized product browsing and filtering.
- **Product Variants Support:** Sellers can create product variants (e.g., size, color) with individual pricing and stock levels, allowing for flexible product configurations.
- **Image Upload:** Products support image uploads stored in the `public/storage` directory, with URLs stored in the `image_url` field for display purposes.

Product Information Fields:

- **Basic Information:** Name, description, SKU (Stock Keeping Unit), brand, and unit of measurement.
- **Pricing & Inventory:** Price (decimal with 2 decimal places), stock quantity (integer), and active status flag.

Product Validation:

- **Input Validation:** All inputs are validated before database insertion (e.g., price must be numeric and > 0, stock must be non-negative integer, name is required).
- **Business Logic:** The system prevents setting negative stock values and ensures SKU uniqueness across all products.
- **Active Status:** Products can be marked as active or inactive, allowing sellers to temporarily hide products without deletion.

Product Updates & Deletion:

- **Ownership Verification:** Sellers can only update or delete products they own, enforced through middleware and controller logic.
- **Cascade Protection:** Product deletion is protected by foreign key constraints, ensuring referential integrity with orders and reviews.
- **Stock Management:** Real-time stock updates are reflected immediately in the product catalog, preventing overselling.

5.3 Order Processing System

When a user initiates checkout:

1. **Transaction Start:** A database transaction begins to ensure atomicity.
2. **Stock Check:** System iterates through cart items to verify sufficient stock.
3. **Deduction:** Stock is decremented from the `products` table.
4. **Creation**: Records are created in `orders` and `order_items`.
5. **Cleanup**: The user's `cart` is cleared.
6. **Transaction Commit:** Changes are saved. If any step fails, the transaction rolls back.

6. Security & Error Handling

6.1 Security Measures

- **Password Hashing:** All passwords are hashed using **Bcrypt** before storage.
- **CSRF Protection:** Laravel's built-in CSRF tokens protect against cross-site request forgery.
- **XSS Protection**: React automatically escapes content before rendering, neutralizing Cross-Site Scripting attacks.
- **SQL Injection:** Use of **Eloquent ORM** and PDO bindings ensures that raw SQL queries are parameterized, preventing injection attacks.

6.2 Error Handling

- **Backend:** Global exception handlers catch errors and return standardized JSON responses (e.g., HTTP 404 for Not Found, 422 for Validation Errors) instead of exposing stack traces.
- **Frontend:** `Try-Catch` blocks in async API calls ensure that network failures trigger user-friendly error messages (e.g., "Failed to load products. Please check your connection.") via Toast notifications.

7. Installation & Setup Instructions

7.1 Prerequisites

- **PHP:** Version 8.2 or higher.
- **Node.js**: Version 18 or higher.
- **Composer:** PHP Dependency Manager.
- **MySQL:** Database Server.

7.2 Installation Steps

1. **Clone the Repository:**

```bash
git clone https://github.com/your-repo/carsucart.git
cd carsucart
```

2. **Backend Dependencies:**

```bash
composer install
```

3. **Environment Configuration:**
   - Copy `.env.example` to `.env`.
   - Update `DB_DATABASE`, `DB_USERNAME`, and `DB_PASSWORD` in `.env`.
   - Run `php artisan key:generate`.

4. **Database Setup:**

```bash
php artisan migrate --seed
```

(This creates tables and populates them with sample data).

5. **Frontend Dependencies:**

```bash
cd frontend
npm install
```

6. **Running the Application:**
   - Terminal 1 (Backend): `php artisan serve`
   - Terminal 2 (Frontend): `npm run dev`

8. Testing

8.1 Test Cases

Feature	Test Steps	Expected Result	Status
Login	Enter valid email and password, then submit.	Authentication token received; user redirected to Dashboard.	Passed
Add to Cart	Click **Add to Cart** on a product with available stock.	Cart badge updates and item appears in the cart.	Passed
Stock Check	Attempt to add quantity greater than available stock.	System displays error message: *"Insufficient stock."*	Passed
Checkout	Proceed to checkout with items in the cart.	Order is created, cart is cleared, and product stock is reduced.	Passed

8.2 Known Issues & Limitations

- **Image Hosting:** Currently, images are stored in the local `public/storage` folder. In a production cloud environment, this should be moved to an S3 bucket or similar object storage.
- **Concurrent Orders:** While transactions are used, extremely high concurrency (e.g., flash sales) might require row-level locking which is not yet optimized.

9. Code Quality & Documentation

9.1 Code Structure

The project follows Laravel's standard directory structure with a separate React frontend. Organized as follows:

Backend Structure (Laravel)

```
carsucart/
├── app/
│   ├── Http/
│   │   ├── Controllers/
│   │   │   ├── Api/                    # API Controllers (RESTful)
│   │   │   │   ├── AuthController.php           # Authentication (login, register, logout)
│   │   │   │   ├── AdminAuthController.php      # Admin authentication
│   │   │   │   ├── ProductController.php        # Product CRUD operations
│   │   │   │   ├── CartController.php           # Shopping cart management
│   │   │   │   ├── OrderController.php          # Order processing & tracking
│   │   │   │   ├── ReviewController.php         # Product reviews & ratings
│   │   │   │   ├── PaymentController.php        # Payment recording
│   │   │   │   ├── DashboardController.php      # Statistics & analytics
│   │   │   │   ├── CategoryController.php       # Category management
│   │   │   │   ├── UserController.php           # User profile management
│   │   │   │   └── SimpleApiController.php      # Simple API endpoints
│   │   │   └── Web/                    # Web Controllers (Blade templates)
│   │   │       ├── AuthController.php
│   │   │       ├── HomeController.php
│   │   │       ├── ProductController.php
│   │   │       ├── CartController.php
│   │   │       ├── OrderController.php
│   │   │       ├── CategoryController.php
│   │   │       └── DashboardController.php
│   │   ├── Middleware/
│   │   │   ├── RoleMiddleware.php      # Role-based access control (buyer, seller, admin)
│   │   │   └── AdminMiddleware.php     # Admin authentication middleware
│   │   └── Resources/                  # API Resources (JSON transformation)
│   │       ├── ProductResource.php
│   │       ├── OrderResource.php
│   │       ├── CartResource.php
│   │       ├── UserResource.php
│   │       ├── CategoryResource.php
│   │       ├── ReviewResource.php
│   │       ├── PaymentResource.php
│   │       └── ...
│   ├── Models/                         # Eloquent Models
│   │   ├── User.php                    # User model (buyer/seller/admin)
│   │   ├── Admin.php                   # Admin model (separate authentication)
│   │   ├── Product.php                 # Product model
│   │   ├── Category.php                # Category model
│   │   ├── Cart.php                    # Shopping cart model
│   │   ├── CartItem.php                # Cart items model
│   │   ├── Order.php                   # Order model
│   │   ├── OrderItem.php               # Order line items model
│   │   ├── Payment.php                 # Payment model
│   │   ├── Review.php                  # Review model
│   │   └── ProductVariant.php          # Product variants model
│   └── Providers/
│       └── AppServiceProvider.php      # Service container bindings
│
├── database/
│   ├── migrations/                     # Database schema migrations
│   │   ├── create_users_table.php
│   │   ├── create_products_table.php
│   │   ├── create_categories_table.php
│   │   ├── create_cart_table.php
│   │   ├── create_cart_items_table.php
│   │   ├── create_orders_table.php
│   │   ├── create_order_items_table.php
│   │   ├── create_payments_table.php
│   │   ├── create_reviews_table.php
│   │   └── ...
│   ├── seeders/                        # Database seeders (sample data)
│   │   ├── DatabaseSeeder.php          # Main seeder orchestrator
│   │   ├── UserSeeder.php
│   │   ├── CategorySeeder.php
│   │   ├── ProductSeeder.php
│   │   ├── OrderSeeder.php
│   │   └── ...
│   └── factories/                      # Model factories (testing)
│       ├── UserFactory.php
│       ├── ProductFactory.php
│       └── ...
│
├── routes/
│   ├── api.php                         # API routes (RESTful endpoints)
│   ├── web.php                         # Web routes (Blade templates)
│   └── console.php                     # Artisan command routes
│
├── resources/
│   ├── views/                          # Blade templates
│   │   ├── web/                        # Web views
│   │   │   ├── partials/
│   │   │   │   ├── header.blade.php
│   │   │   │   └── footer.blade.php
│   │   │   └── ...
│   │   └── welcome.blade.php
│   ├── css/
│   │   └── app.css
│   └── js/
│       ├── app.js
│       └── bootstrap.js
│
├── config/                             # Configuration files
│   ├── app.php
│   ├── auth.php
│   ├── database.php
│   ├── sanctum.php                     # Laravel Sanctum configuration
│   └── ...
│
└── public/                             # Public web root
    ├── index.php                       # Application entry point
    ├── products/                       # Product image uploads
    └── profile-photos/                 # User profile photo uploads
```

Frontend [Typescript + React]

```
frontend/

├── src/
│   ├── components/                 # Reusable UI components
│   │   ├── Header.tsx              # Global navigation header
│   │   ├── ProtectedRoute.tsx      # Route protection wrapper
│   │   ├── FilterSidebar.tsx       # Product filter sidebar
│   │   ├── FavoriteButton.tsx      # Wishlist button
│   │   ├── StarRating.tsx          # Rating display component
│   │   ├── ReviewCard.tsx          # Review display card
│   │   ├── ReviewModal.tsx         # Review submission modal
│   │   ├── MessagingModal.tsx      # Seller messaging modal
│   │   ├── OrderTrackingModal.tsx  # Order tracking modal
│   │   ├── StockUpdateModal.tsx    # Stock update modal (seller)
│   │   └── VoucherCard.tsx         # Voucher display card
│   │
│   ├── pages/                      # Page components (routes)
│   │   ├── auth/
│   │   │   ├── LoginPage.tsx       # User login page
│   │   │   └── RegisterPage.tsx    # User registration page
│   │   ├── HomePage.tsx            # Product listing/home page
│   │   ├── ProductsPage.tsx        # Products listing page
│   │   ├── ProductDetailsPage.tsx  # Single product details
│   │   ├── CartPage.tsx            # Shopping cart page
│   │   ├── CheckoutPage.tsx        # Checkout process page
│   │   ├── MyOrdersPage.tsx        # User order history
│   │   ├── WishlistPage.tsx        # User wishlist
│   │   ├── SellerDashboard.tsx     # Seller management dashboard
│   │   ├── SellerStorePage.tsx     # Seller store view
│   │   ├── ProductComparePage.tsx  # Product comparison
│   │   ├── MessagesPage.tsx        # Messaging page
│   │   ├── VouchersPage.tsx        # Vouchers page
│   │   └── Placeholder.tsx         # Placeholder component
│   │
│   ├── contexts/                   # React Context providers
│   │   └── AuthContext.tsx         # Authentication context
│   │
│   ├── api.ts                      # API client functions (axios)
│   ├── types.ts                    # TypeScript type definitions
│   ├── App.tsx                     # Main application component
│   ├── App.css                     # Application styles
│   ├── index.css                   # Global styles
│   └── main.tsx                    # Application entry point
│
├── public/                         # Static assets
│   └── vite.svg
│
├── tailwind.config.js              # Tailwind CSS configuration
├── vite.config.ts                  # Vite build configuration
├── tsconfig.json                   # TypeScript configuration
└── package.json                    # NPM dependencies
```

9.2 Code Standards

- **Backend**: Adheres to **PSR-12** coding standards (indentation, naming conventions).
- **Frontend**: Uses **ESLint** configurations to enforce JavaScript best practices and **Prettier** for consistent formatting. Variables use `camelCase`; Classes use `PascalCase`.

9.3 API Endpoints

Sample Request: Login

- **Endpoint:** `POST /api/login`
- **Body:** `{ "email": "admin@csu.edu.ph", "password": "password" }`
- **Response:**

```
```json
{
   "token": "1|laravel_sanctum_token_string...",
   "user": { "id": 1, "name": "Admin", "role": "admin" }
}
```
```

Sample Request: Get Products

- **Endpoint:** `GET /api/products`
- **Response:**

```
```json
[
   {
      "id": 1,
      "name": "Math 101 Textbook",
      "price": 450.00,
      "category": { "id": 2, "name": "Books" }
   }
]
``
```

10. References & Resources

1. **Laravel Documentation.** (2025). **Laravel 12.x Framework**. Retrieved from https://laravel.com/docs/12.x
2. **React Documentation.** (2025). **React 19 Library.** Retrieved from https://react.dev/
3. **Tailwind CSS.** (2025). **Utility-First CSS Framework**. Retrieved from https://tailwindcss.com/
4. **Caraga State University. (2025). Institutional Branding Guidelines**.

11. Appendix

11.1 Additional Diagrams

11.2 Supplementary Information

Student/Team Signature: JAN FRANSCINE P. HERBOLINGO Date: **Dec 24, 2025**
```