APPLIED INFORMATION TECHNOLOGY AND COMPUTER SCIENCE
e-ISSN: 2773-5141

AITCS

Vol. 6 No. 2 (2025) 1338-1355
https://publisher.uthm.edu.my/periodicals/index.php/aitcs

J&J Vegetables Wholesale Ordering System

Serena Ng Yen Xin[1], Rabatul Aduni Sulaiman[1]*

[1] *Fakulti Sains Komputer dan Teknologi Maklumat,*
Universiti Tun Hussein Onn Malaysia, Parit Raja, Batu Pahat, 86400, MALAYSIA

*Corresponding Author: rabatul@uthm.edu.my
DOI: https://doi.org/10.30880/aitcs.2025.06.02.072

Article Info

Received: 16 June 2025
Accepted: 20 November 2025
Available online: 30 November 2025

Keywords

E-commerce, Web-based System, Ordering System, Order Management

Abstract

J&J Vegetables Marketing supplies fresh produce to various sectors. Currently, the company relies on phone calls and WhatsApp for orders, leading to inefficiencies, miscommunication, and disorganised data. The objectives of this project are to design and develop the J&J Vegetables Wholesale Ordering System using an object-oriented and web-based approach, followed by comprehensive system testing and user acceptance testing. The project adopts the prototype methodology as its software development lifecycle model. The system is developed using Visual Studio Code, Laravel framework, PHP, MySQL, XAMPP, JavaScript, HTML and CSS, and is hosted on a web server to ensure accessibility across devices. It supports key user groups, including administrators, staff, drivers, and customers within Johor Bahru and surrounding areas. This user-friendly platform streamlines the ordering process, improves data management, reduces errors, enhances customer experience, and supports the company's expansion into door-to-door delivery services.

1. Introduction

Vegetable wholesaling plays a critical role in supplying fresh produce to retailers, restaurants, and consumers, ensuring a steady supply of quality vegetables. J&J Vegetables Marketing, based in Permas Jaya, Johor Bahru, operates as a vegetable wholesaler serving a diverse range of clients, including restaurants, markets, hotels, and individual customers. The company sources its products directly from farms, ensuring they meet the quality standards before distributing them through a logistics network. However, the company's current operations rely heavily on processing orders via phone calls and WhatsApp, leading to inefficiencies, miscommunication, and disorganised data.

This study focuses on the development of a web-based ordering system for J&J Vegetables Marketing to address the current challenges in order management. The system's goals are to streamline the ordering process, improve operational efficiency, and enhance customer experience. This project focuses on three key objectives: (1) designing the J&J Vegetables Wholesale Ordering System with an object-oriented approach, (2) developing it as a web-based application, and (3) conducting system testing and user acceptance testing to evaluate the system's functionality and usability. The system development adopts the prototyping methodology as the Software Development Lifecycle (SDLC) model. The system is intended for four key user groups, including administrators, staff, drivers, and customers. The expected outcomes include minimising errors, improving customer satisfaction, and supporting the company's expansion into door-to-door services.

2. Related Work

This section will discuss the literature review that has been conducted in relation to the ordering system.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 1338-1355

2.1 Background of the Case Study

J&J Vegetables Marketing primarily operates as a business-to-business (B2B) entity. B2B commerce involves companies doing business with each other [1]. However, since the outbreak of COVID-19 in 2020, the business operations of many of their customers have been significantly impacted. Consequently, J&J Vegetables Marketing has expanded their services to include general consumers seeking their products. The company aspires to reach a broader consumer base through the implementation of an ordering platform, or a business-to-consumer (B2C) e-commerce. B2C e-commerce is a commerce between companies and the general public [2].

In the current system, customers place orders by calling or messaging the company via WhatsApp. Once an order is received, the company's staff promptly enter the order details into their system and begin preparing the orders. The order list containing customer information, address, and order specifics is then compiled and handed over to the driver. By the following day, deliveries are made to the customers, who receive their orders along with an invoice. At the end of the week, the owner visits customers at their addresses to collect payments. Fig. 1 shows the current ordering process (as-is model) of J&J Vegetables Marketing.

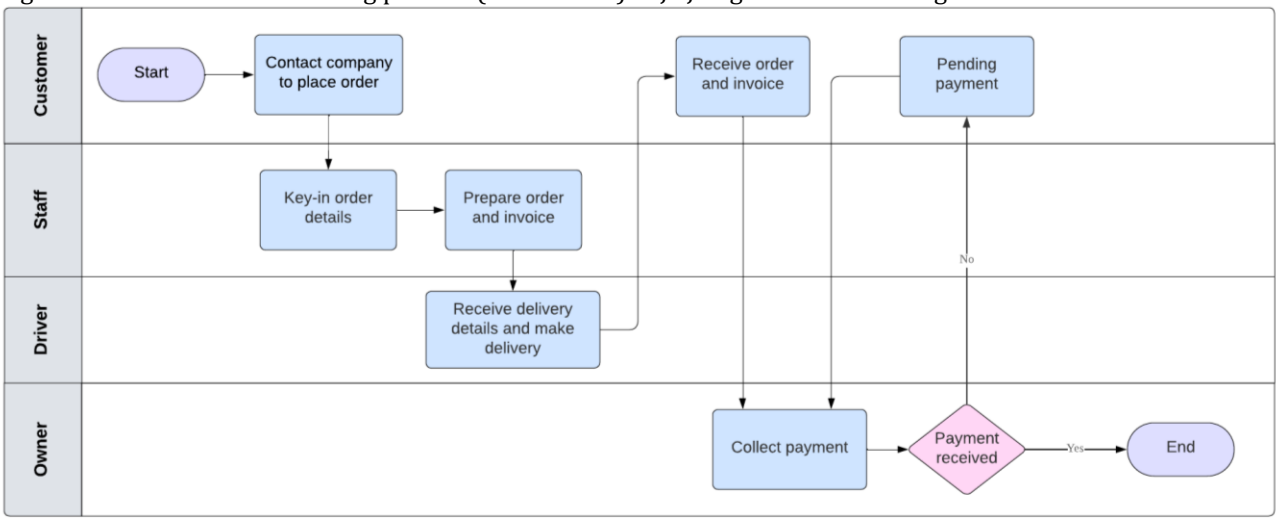


Fig. 1 *Current Ordering Process (as-is Model)*

2.2 Study of Existing Systems

A study of three existing web-based ordering systems, Veggies.my [3], e-Petani [4], and Organic4U [5], was conducted to identify their features and compare them with the proposed system.

2.2.1 Veggies.my

Veggies.my is operated by Green Frontiers Sdn. Bhd. and serves customers in Klang Valley area. The platform provides an online catalogue for customers to browse products and offers guidance through a "how to order" page detailing delivery areas, schedules, and charges. Its user interface features a slide-out shopping cart, and customers can select a delivery date during checkout. A notable feature is its customer loyalty program, which allows users to earn "veggies points" that can be redeemed for discounts. However, the system lacks integrated messaging and referral functionality.

2.2.2 e-Petani

e-Petani is an e-commerce platform supporting three branches of e-Petani Malaysia stores located in Kuala Lumpur and Petaling Jaya. Their services are provided to customers in the Klang Valley area. e-Petani allows product browsing, cart management, and both guest and registered checkouts. Delivery or self-pickup options are available at checkout. A drawback of this system is that product availability is only shown during the transaction process, which may inconvenience users. Additionally, e-Petani does not offer customer loyalty programme, integrated messaging or referral features.

2.2.3 Organic4U

Organic4U, an online ordering platform operated by Organic For You Trading in Kuala Lumpur, provides services across Klang Valley. Customers must register and login before placing orders. Once logged in, they can proceed to checkout, modify shipping details, and review order information. Organic4U also offers a customer dashboard where users can view account details, purchases, and a redemption menu, although the redemption section appears to be inactive or incomplete. While the system does include a loyalty programme, it does not feature integrated messaging or referral functionality.

2.2.4 System Comparison

The results of the comparison between these three systems and the proposed system are shown in Table 1. N/A indicates the relevant features may not be available or accessible due to access control.

Table 1 *System Comparison*

Features/System	Veggies.my	e-Petani	Organic4U	J&J Vegetables Wholesale Ordering System
Account Registration	Yes	Yes	Yes	Yes
Login	Yes	Yes	Yes	Yes
User Management	N/A	N/A	N/A	Yes
Product Management	Yes	Yes	Yes	Yes
Cart Management	Yes	Yes	Yes	Yes
Order Management	Yes	Yes	Yes	Yes
Payment	Yes	Yes	Yes	Yes
Delivery	N/A	N/A	N/A	Yes
Generate Report	N/A	N/A	N/A	Yes
Integrated Messaging	No	No	No	Yes
Customer Loyalty Programmes	Yes	No	Yes	Yes
Referral	No	No	No	Yes (QR Code Referral)

In the system comparison, the table shows that all four systems support basic e-commerce functionalities such as account registration, login, product browsing, cart management, and checkout processes. However, none of the three existing systems implement integrated messaging or referral features, while e-Petani lacks a customer loyalty programme. These findings highlight opportunities for the proposed system to stand out by incorporating integrated messaging, customer loyalty programme, and QR code referral functionality.

3. Methodology

This section will discuss the methodology used in the development of the ordering system.

3.1 Prototype Model

The Software Development Life Cycle (SDLC) is a methodology consisting of well-defined steps to develop high-quality software that boasts cost-effectiveness and reliability [6]. The standard phases within an SDLC process include planning, designing, implementing, testing, deploying, and maintaining [7]. In this project, the SDLC model chosen for development is the prototyping model. Prototyping is a great approach for developing complex and extensive systems that lack established manual processes or existing systems for requirement determination [8]. The prototype model prioritises the development of the actual software over extensive documentation, which allows for earlier release of the software [9]. The prototype model is beneficial due to the active participation of users during the development of the system, which helps in the early identification of absent functionalities and errors [10]. Fig. 2 shows the prototyping model, consisting of six phases, which are the requirements gathering and analysis phase, quick design phase, prototype development phase, user evaluation phase, prototype refinement phase, and implementation and testing phase.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 1338-1355

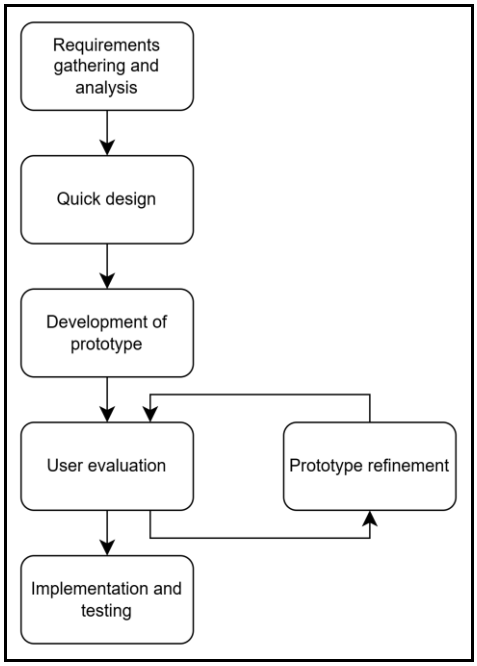


Fig. 2 *Prototype Methodology* [11]

4. System Analysis and Design

This section will discuss the analysis and design of the proposed system.

4.1 System Requirements Analysis

The system requirements analysis will cover user requirements, functional requirements, and non-functional requirements. User requirements consist of the end-users' and stakeholders' needs, expectations, and specifications for the system. The requirements are the functionalities or features required by users of the system. Table 2 shows the user requirements for each key users involved in the system.

Table 2 *User Requirements*

User	Requirements
Administrator	• Able to login to the system. • Able to add and delete users. • Able to manage products. • Able to generate reports.
Staff	• Able to login to the system. • Able to manage products. • Able to manage orders. • Able to generate reports. • Able to manage messages.
Driver	• Able to login to the system. • Able to manage deliveries.
Customer	• Able to register an account. • Able to login to the system. • Able to view products. • Able to add products to cart. • Able to place order. • Able to make payments. • Able to view order details. • Able to obtain points from completed orders. • Able to claim rewards with points. • Able to apply referral codes to earn points. • Able to message customer service.

Functional requirements convey the system's capability in performing functions and answer to what is essential in the system [12]. Table 3 shows the functional requirements of the system.

Table 3 *Functional Requirements*

Modules	User	Requirements
Login Module	All	• The system shall allow users to login to the system.
Account Registration Module	Customer	• The system shall allow customer to input and submit account registration details • The system shall be able to verify the account registration details. • The system shall be able to store customer account details in system database.
User Management Module	Administrator	• The system shall allow administrator to view, add, edit, and delete employee details. • The system shall allow administrator to view customer details. • The system shall allow administrator to ban or unban customers.
	Customer	• The system shall allow customers to view and edit their personal details.
Product Management Module	Administrator, Staff	• The system shall allow users to create, edit, and delete products. • The system shall allow users to input and submit product details. • The system shall be able to store, update, or remove product details in system database.
Cart Management Module	Customer	• The system shall allow customers to view their cart. • The system shall allow users to add or remove items from cart. • The system shall allow users to proceed to checkout using cart. • The system shall be able to store, update, or remove cart details in system database.
Order Management Module	Staff	• The system shall allow staff to manage customer orders. • The system shall allow staff to update order statuses. • The system shall allow staff to view order history. • The system shall be able to store or update order details in system database.
	Customer	• The system shall allow customers to manage orders. • The system shall allow customers to view order history. • The system shall be able to store or update order details in system database.
Payment Processing Module	Customer	• The system shall handle payment transactions.
Delivery Module	Driver	• The system shall display pending deliveries to drivers. • The system shall allow drivers to manage deliveries. • The system shall allow drivers to update delivery statuses. • The system shall be able to store or update delivery details in system database.
Report Module	Administrator, Staff	• The system shall generate reports by retrieving relevant sales and order data from database.
Messages Module	Staff, Customer	• The system shall allow a two-way communication between staff and customers. • The system shall be able to store or update message details in system database.
Rewards Management Module	Customer	• The system shall calculate points acquired with each completed order. • The system shall allow customers to earn points via referral. • The system shall allow customers to redeem rewards with points.

Non-functional requirements define a system's quality attributes and are non-dismissible as their absence could lead to user's dissatisfaction [13]. Table 4 shows the non-functional requirements of the system.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 1338-1355

Table 4 *Non-Functional Requirements*

Requirements	Description
Performance	• The system shall have a reasonable response time with each instance of interaction between the user and the system.
Operational	• The system shall be able to work with any web browsers.
Usability	• The user interface shall be intuitive and easy to be navigated by users. • The user interface shall have a clear and consistent layout across all pages.
Security	• The password should have a minimum length of 8 characters, combining letters, numbers, and special characters. • The system shall deny access to login credentials that do not match with the database.
Integrity	• The password stored in system database should be encrypted.
Maintainability	• The system shall have an organised and maintainable codebase.
Availability	• The system shall always be available for use

4.2 Unified Modelling Language (UML)

Unified Modelling Language (UML) diagrams serve as visual models to portray object-oriented design models, which help in the requirements and scope identification of a system [13]. Fig. 3 shows the use case diagram for the proposed system. There are four key actors involved in the use case diagram of the system. Administrator is involved in login, manage users, manage products, and generate report. Staff is involved in login, manage products, manage orders, generate reports, and message. Driver is involved in login and manage delivery. Customer is involved in register account, login, manage profile, manage order, make payments, manage rewards, and message.

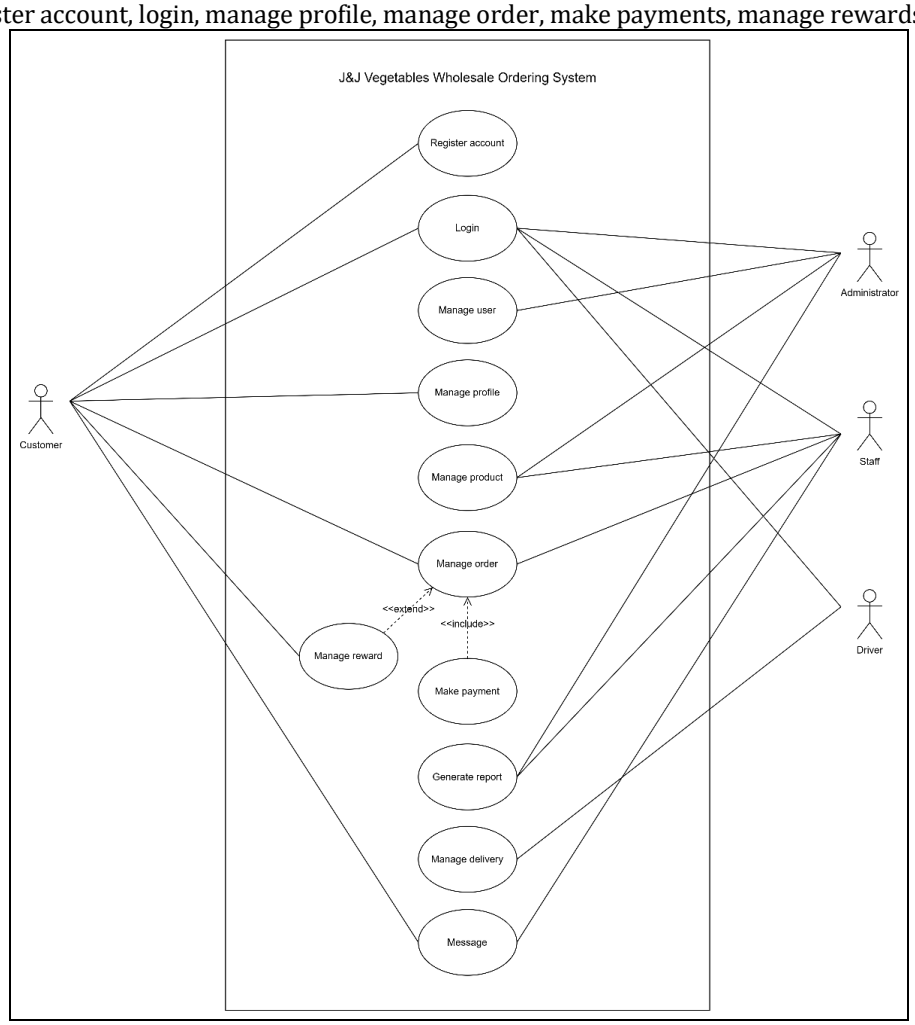


Fig. 3 *Use Case Diagram*

UML class diagrams enable declarative modelling for the static structure of a system, including concepts and their relationship [14]. The class diagram for the proposed system is presented in Fig. 4.

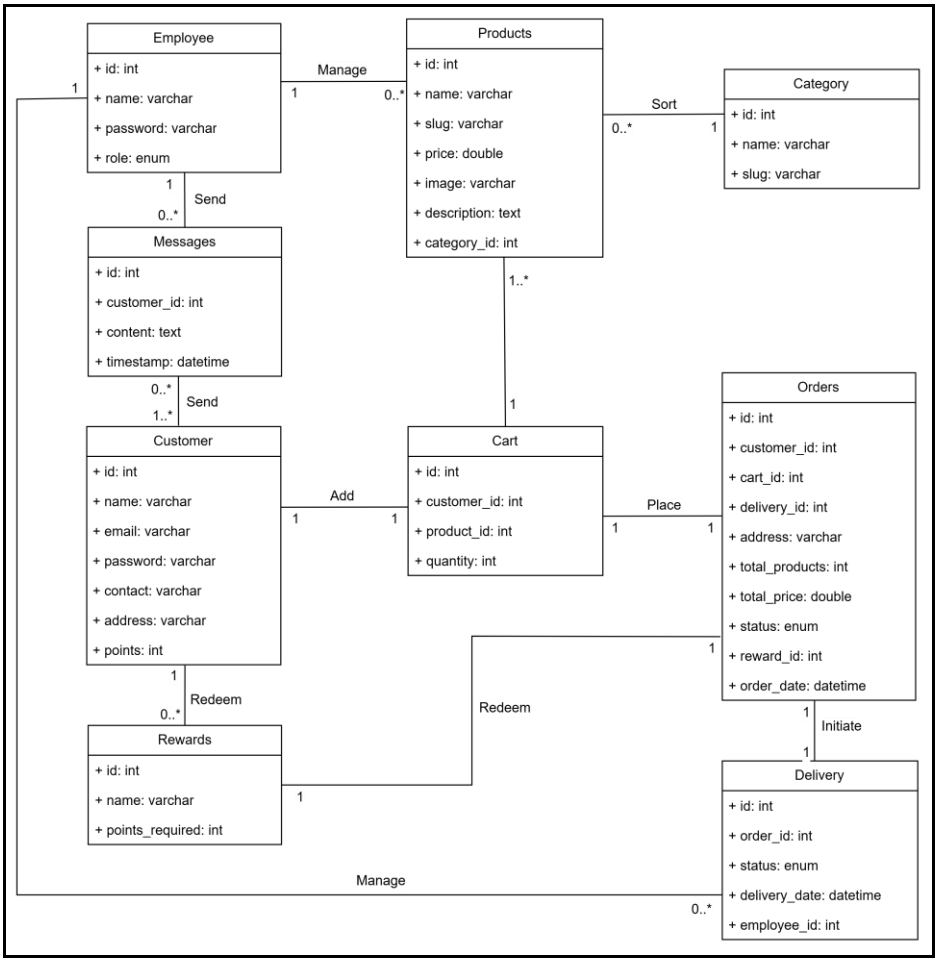


Fig. 4 *Class Diagram*

4.3 Database Design

Database schema is used as guidance to the design and organisation of the system's database. In the proposed system, the database involves data elements such as employee, customer, products, category, cart, orders, delivery, rewards, and messages. The database schema of the proposed system is as following:

i. `Employee (id, name, email, password, role)`
ii. `Customer (id, name, email, password, contact, address, points)`
iii. `Products (id, category_id, name, slug, price, image, description)`
iv. `Category (id, name, slug)`
v. `Cart (id, customer_id, product_id, quantity)`
vi. `Orders (id, customer_id, cart_id, delivery_id, address, total_products, total_price, status, order_date)`
vii. `Delivery (id, employee_id, order_id, status, delivery_date)`
viii. `Rewards (id, name, points_required)`
ix. `Messages (id, customer_id, content, timestamp)`

5. Implementation and Testing

This section will discuss the implementation and testing of the developed system.

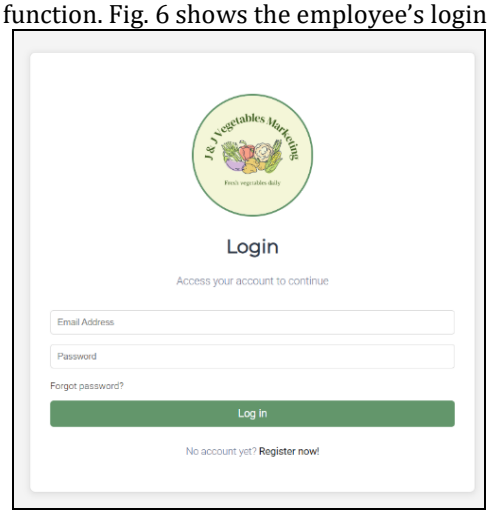
5.1 Implementation

A total of 11 modules were developed during the implementation phase, including login, account registration, user management, product management, cart management, order management, payment processing, delivery, report, message, and rewards. To develop the system, Visual Studio Code was used as the Integrated Development Environment (IDE), along with XAMPP and Laravel framework. The development was carried out using PHP, JavaScript, HTML, and CSS, while MySQL was utilised as the database management system.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 1338-1355

5.1.1 Login Module

In the Login Module, all users are required to input their email and password to login into the system and access their accounts. Fig. 5 shows the customer's login interface and the code segment of customer's login function. Fig. 6 shows the employee's login interface and the code segment of employee's login function.




```php
class UserController extends Controller
{
    1 reference | 0 overrides
    public function login(Request $request): RedirectResponse {
        $incomingFields = $request->validate(rules: [
            'loginemail' => 'required',
            'loginpassword' => 'required'
        ]);

        $user = User::where(column: 'email', operator: $incomingFields['loginemail'])->first();

        if ($user && $user->banned_at) {
            return back()->withErrors(provider: ['loginemail' => 'Your account is banned.']);
        }

        if (Auth::attempt(credentials: ['email' => $incomingFields['loginemail'], 'password' =>
            $request->session()->regenerate();
            return redirect(to: '/');
        }

        return back()->withErrors(provider: [
            'loginemail' => 'The provided credentials do not match our records.',
        ]);
    }
```

(a) **(b)**

Fig. 5 *(a) Customer's Login Interface; (b) Code Segment for Customer's Login Function*

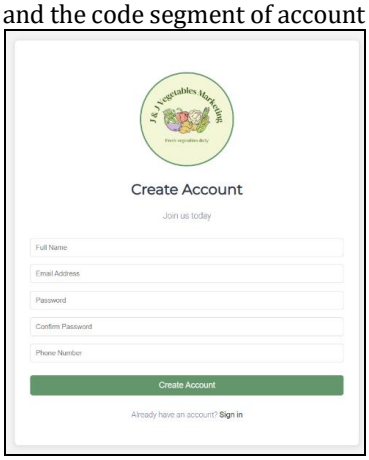

```php
public function login(Request $request): mixed|RedirectResponse
{
    $credentials = $request->only(keys: 'email', 'password');

    if (Auth::guard(name: 'employee')->attempt(credentials: $credentials)) {
        $role = Auth::guard(name: 'employee')->user()->role;
        if (Auth::guard(name: 'employee')->attempt(credentials: $credentials)) {
            $role = Auth::guard(name: 'employee')->user()->role;
            if ($role === 'admin') {
                return redirect()->route(route: 'admin.dashboard');
            } elseif ($role === 'staff') {
                return redirect()->route(route: 'staff.dashboard');
            } elseif ($role === 'driver') {
                return redirect()->route(route: 'driver.dashboard');
            }
        }
    }
    return redirect()->route(route: 'admin.login')->withErrors(provider: ['email' => 'Invalid
}
```

(a) **(b)**

Fig. 6 *(a) Employee's Login Interface; (b) Code Segment for Employee's Login Function*

5.1.2 Account Registration Module

In the Account Registration Module, customers can create an account to access the system. To create an account, customers can reach the registration page and fill out the required details such as full name, email address, password, and phone number to complete the registration process. Fig. 7 shows the account registration interface and the code segment of account registration function.



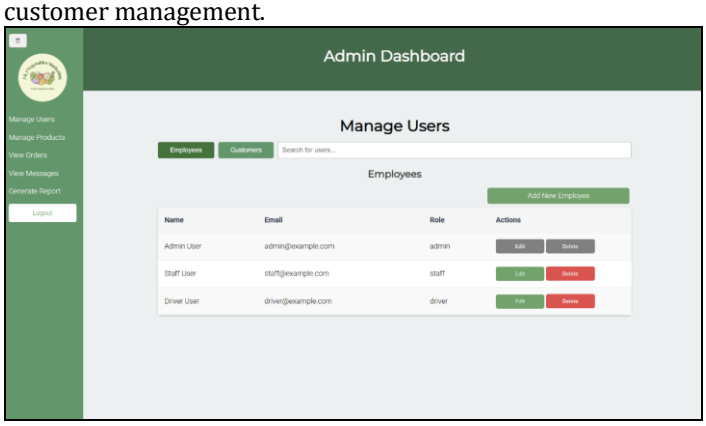

```php
public function register(Request $request): RedirectResponse {
    $incomingFields = $request->validate(rules: [
        'name' => ['required', 'min:3', 'max:25', Rule::unique(table: 'users',
        'email' => ['required', 'email', Rule::unique(table: 'users', column:
        'password' => ['required', 'min:8', 'confirmed'],
        'contact' => ['required', 'regex:/^\+?[0-9]{10,15}$/'],
    ]);

    $incomingFields['password'] = bcrypt(value: $incomingFields['password']);
    $user = User::create(attributes: $incomingFields);

    Auth::login(user: $user);
    return redirect(to: '/');

}
```

(a) **(b)**

Fig. 7 *(a) Account Registration Interface; (b) Code Segment for Account Registration Function*

5.1.3 User Management Module

In the User Management Module, admin can manage users, including the ability to manage employee details, view customer details, and ban or unban customers. Fig. 8 shows the employee management interface and code segment for employee management. Fig. 9 shows the customer management interface and code segment for customer management.

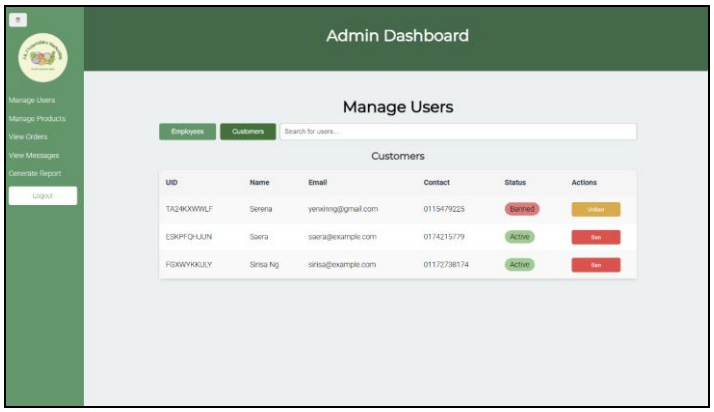


(a)

```php
class EmployeeController extends Controller
{
    0 references | 0 overrides
    public function index(): View
    {
        $employees = Employee::all();
        $customers = User::all();
        return view(view: 'admin.manage-users', data: compact(var_name: 'employees',
    }

    0 references | 0 overrides
    public function create(): View
    {
        return view(view: 'admin.employees.create-employee');
    }

    0 references | 0 overrides
    public function store(Request $request): RedirectResponse
    {
        $request->validate(rules: [
            'name' => 'required|string|max:255',
            'email' => 'required|string|email|max:255|unique:employees',
            'password' => 'required|string|min:8|confirmed',
            'role' => 'required|in:admin,staff,driver',
        ]);
```

(b)

Fig. 8 *(a) Employee Management Interface; (b) Code Segment for Employee Management Function*



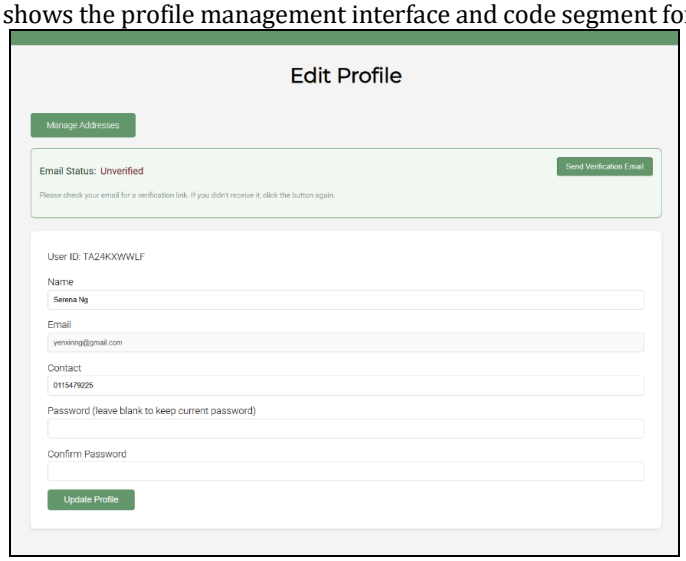
(a)

```php
public function denyCustomer(User $user): JsonResponse|m
{
    try {
        $user->banned_at = now();
        $user->save();

        return response()->json(data: [
            'success' => true,
            'message' => 'Customer banned successfully'
        ]);
    } catch (\Exception $e) {
        return response()->json(data: [
            'success' => false,
            'message' => $e->getMessage()
        ], status: 500);
    }
}
```

(b)

Fig. 9 *(a) Customer Management Interface; (b) Code Segment for Customer Management Function*

From the customer's perspective, User Management Module allows customer to manage their profile, including viewing and editing their personal details, such as name, email, phone number, password, and addresses. Fig. 10 shows the profile management interface and code segment for profile management functions.



(a)

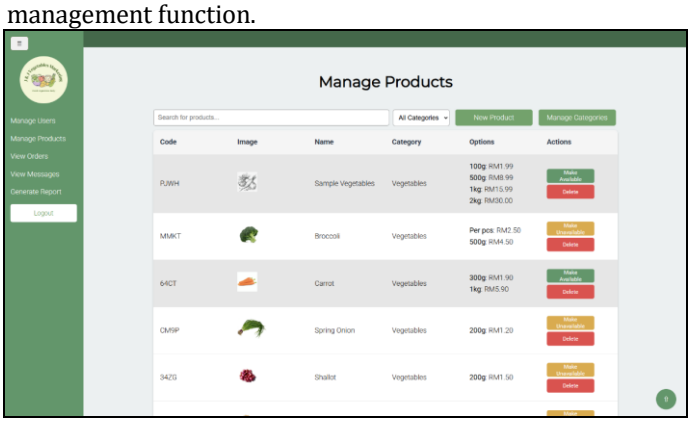
```php
public function edit(): View
{
    $user = Auth::user();
    return view(view: 'profile', data: compact(var_name: 'user'));
}

1 reference | 0 overrides
public function update(Request $request): RedirectResponse
{
    $user = Auth::user();

    $request->validate(rules: [
        'name' => 'required|string|min:3|max:25',
        'contact' => 'required|string|regex:/^\+?[0-9]{10,15}$/',
        'password' => 'nullable|string|min:8|confirmed',
    ]);

    $user->name = $request->name;
    $user->contact = $request->contact;

    if ($request->password) {
        $user->password = Hash::make(value: $request->password);
    }

    $user->save();

    return redirect()->route(route: 'profile.edit')->with(key: 'success',
}
```

(b)

Fig. 10 *(a) Profile Management Interface; (b) Code Segment for Profile Management Function*

5.1.4 Product Management Module

In the Product Management Module, admin and staff can manage the product inventory, which includes the actions to add, update, and remove products and categories. Each product has product code, image, name, category, options, and availability. Fig. 11 shows the product management interface and the code segment of product management function.



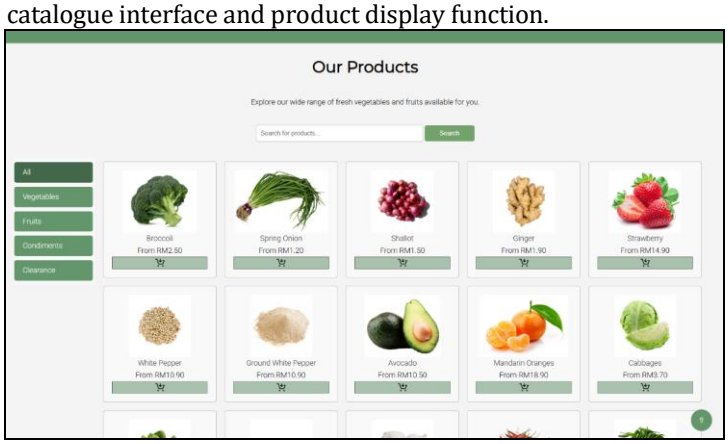
(a)

```php
class ProductController extends Controller
{
    0 references | 0 overrides
    public function index(Request $request): View
    {
        $categoryId = $request->query(key: 'category');
        $categories = Category::all();

        $query = Product::with(relations: ['category', 'options']);

        if ($categoryId) {
            $query->where(column: 'category_id', operator: $categoryId);
        }

        $products = $query->get();

        return view(view: 'staff.manage-products', data: compact(var_name: 'products', var_
    }

    0 references | 0 overrides
    public function create(): View
    {
        $categories = Category::all();
        return view(view: 'staff.create-product', data: compact(var_name: 'categories'));
    }
```

(b)

Fig. 11 *(a) Product Management Interface; (b) Code Segment for Product Management Function*

From the customer's perspective, they can view products and proceed to add to cart. Fig. 12 shows the product catalogue interface and product display function.



(a)

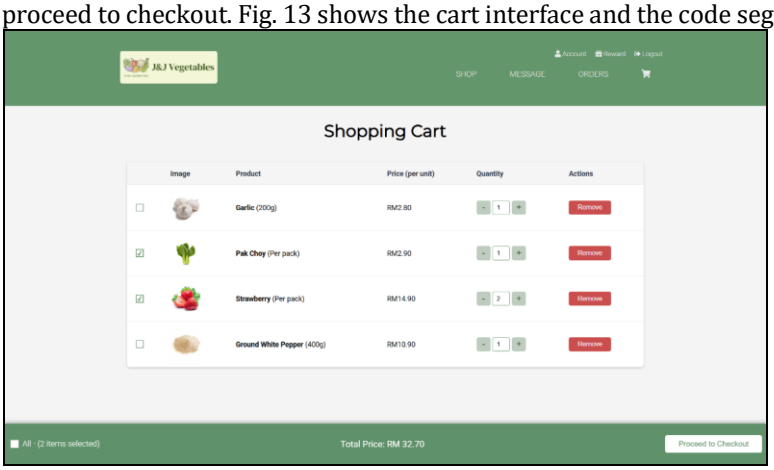
```php
public function showProducts(Request $request): View
{
    $categorySlug = $request->query(key: 'category');
    $category = Category::where(column: 'slug', operator: $categorySlug)->first();
    $categories = Category::all();

    $query = Product::available()->with(relations: 'options');

    if ($category) {
        $query->where(column: 'category_id', operator: $category->id);
    }

    $products = $query->get();

    return view(view: 'product', data: compact(var_name: 'products', var_names: 'categories',
}

1 reference | 0 overrides
public function search(Request $request): View
{
    $query = $request->input(key: 'query');
    $products = Product::where(column: 'name', operator: 'LIKE', value: "%$query%")
        ->available()
        ->with(relations: 'options')
        ->get();
    $categories = Category::all();

    return view(view: 'product', data: compact(var_name: 'products', var_names: 'categories'))
}
```

(b)

Fig. 12 *(a) Product Catalogue Interface; (b) Code Segment for Product Display Function*

5.1.5 Cart Management Module

In the Cart Management Module, customers can view their cart, add or remove items from the cart, and proceed to checkout. Fig. 13 shows the cart interface and the code segment of cart management function.



(a)

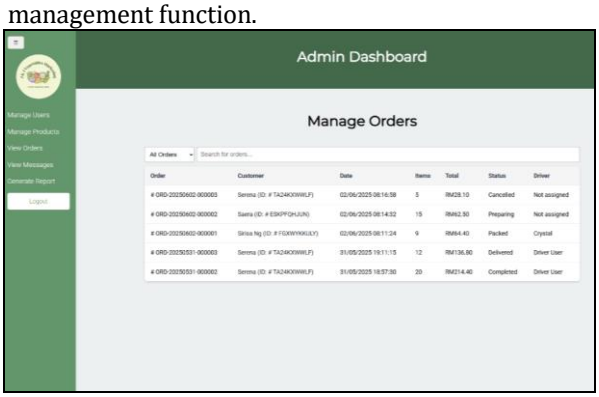
```php
class CartController extends Controller
{
    1 reference | 0 overrides
    public function index(): View
    {
        $user_id = Auth::id();
        $cart = Cart::firstOrCreate(attributes: ['user_id' => $user_id]);

        $cartItems = $cart->items()
            ->with(relations: ['product', 'option'])
            ->orderBy(column: 'updated_at', direction: 'DESC')
            ->get();

        return view(view: 'cart', data: compact(var_name: 'cartItems'));
    }

    1 reference | 0 overrides
    public function add(Request $request, $id): RedirectResponse
    {
        $product = Product::findOrFail(id: $id);
        $user_id = Auth::id();
        $cart = Cart::firstOrCreate(attributes: ['user_id' => $user_id]);

        if ($product->options->isNotEmpty()) {
            $request->validate(rules: [
                'option_id' => 'required|exists:product_options,id',
            ]);
        }
```

(b)

Fig. 13 *(a) Cart Interface; (b) Code Segment for Cart Management Function*

5.1.6 Order Management Module

In the Order Management Module, admin can view customers' orders, and staff can manage customers' orders, order status, and order history. Fig. 14 shows the order management interface and the code segment of order management function.



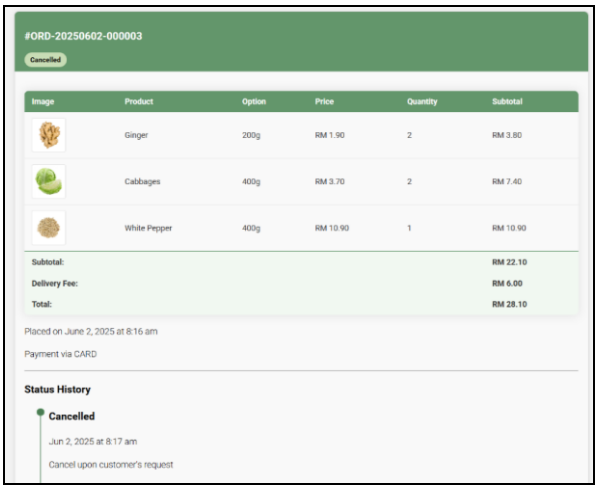
(a)

```php
class OrderController extends Controller
{
    3 references | 0 overrides
    public function index(): View
    {
        $user = auth(guard: 'employee')->user();

        $orders = Order::with(relations: ['user', 'items.product', 'driver'])
                    ->when(value: request(key: 'status'), callback: function(Builder<Order> $query):
                        $query->where(column: 'status', operator: request(key: 'status'));
                    })
                    ->when(value: $user->role === 'driver', callback: function(Builder<Order> $query)
                        $query->where(column: 'driver_id', operator: $user->id)
                            ->whereIn(column: 'status', values: ['packed', 'delivering', 'delivered']
                    })
                    ->latest()
                    ->paginate(perPage: 15);

        $drivers = $user->role === 'staff' ? Employee::where(column: 'role', operator: 'driver')->get

        return view(view: 'admin.orders.index', data: compact(var_name: 'orders', var_names: 'drivers
    }
```

(b)

Fig. 14 *(a) Order Management Interface; (b) Code Segment for Order Management Function*

From the customer's perspective, they can view their order details and track order progress. Fig. 15 shows the customer's order interface and the code segment of customer order function.



(a)

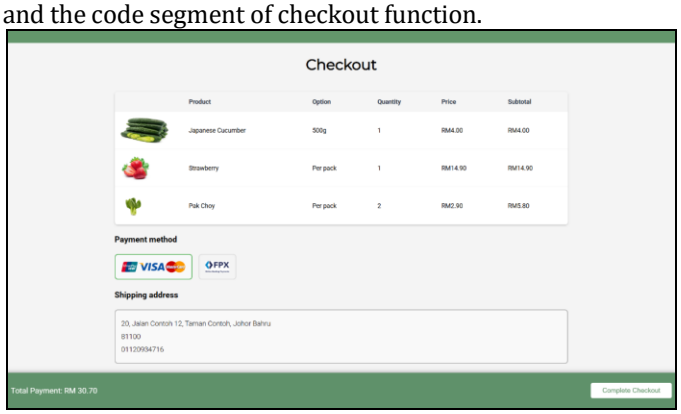
```php
class OrderController extends Controller
{
    1 reference | 0 overrides
    public function index(): View
    {
        $orders = Order::where(column: 'user_id', operator: Auth::id())
            ->with(relations: ['items.product', 'statusHistory'])
            ->latest()
            ->paginate(perPage: 10);

        return view(view: 'orders.index', data: compact(var_name: 'orders'));
    }

    1 reference | 0 overrides
    public function show(Order $order): View
    {
        if ($order->user_id !== Auth::id()) {
            abort(code: 403);
        }

        $order->load(relations: ['items.product', 'items.option', 'address',

        return view(view: 'orders.show', data: compact(var_name: 'order'));
    }
```

(b)

Fig. 15 *(a) Customer's Order Interface; (b) Code Segment for Customer Order Function*

5.1.7 Order Management Module

In the Payment Processing Module, the checkout procedure is handled. Customers are required to input their payment method, shipping address, and vouchers (if applicable) in this page. Fig. 16 shows the checkout interface and the code segment of checkout function.



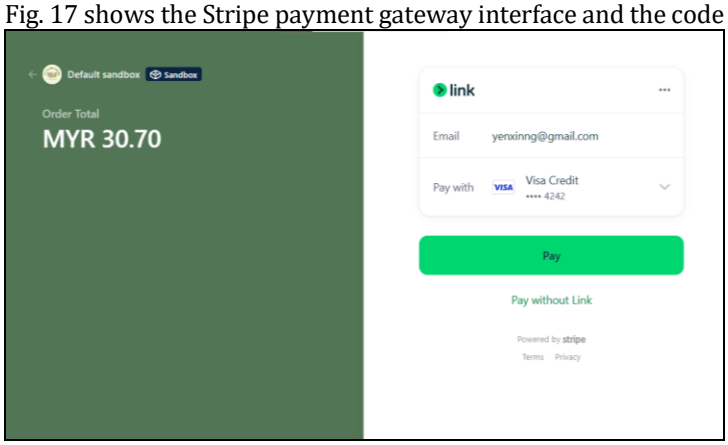
(a)

```php
class CheckoutController extends Controller
{
    1 reference | 0 overrides
    public function index(Request $request): RedirectResponse|View
    {
        $selectedItems = $request->input(key: 'selected_items', default: []);

        if (empty($selectedItems)) {
            return redirect()->route(route: 'cart.index')->with(key: 'error', value: 'No items
        }

        $cartItems = CartItem::whereHas(relation: 'cart', callback: function(Builder<TRelatedM
                $query->where(column: 'user_id', operator: Auth::id());
            })
            ->whereIn(column: 'id', values: $selectedItems)
            ->with(relations: ['product', 'option'])
            ->get();

        if ($cartItems->isEmpty()) {
            return redirect()->route(route: 'cart.index')->with(key: 'error', value: 'Selected
        }

        $totalPrice = $cartItems->sum(callback: function (CartItem $item): float|int {
            return $item->option->price * $item->quantity;
        });
```

(b)

Fig. 16 *(a) Checkout Interface; (b) Code Segment for Checkout Function*

Stripe Payment Application Programming Interface (API) is implemented in the Payment Processing Module. Fig. 17 shows the Stripe payment gateway interface and the code segment of Stripe's payment processing function.



(a)

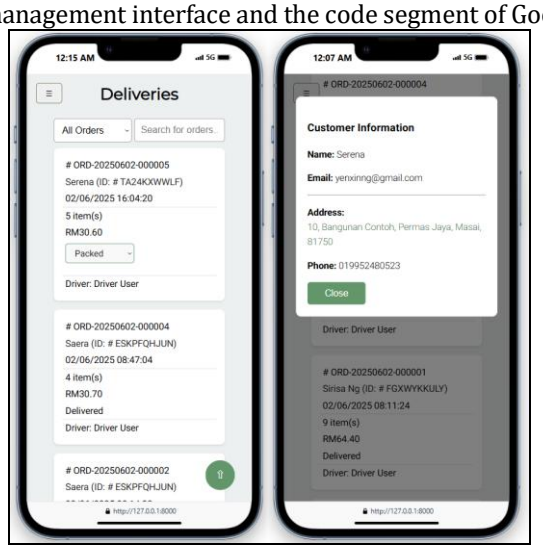
```php
class StripeController extends Controller
{
    1 reference | 0 overrides
    public function payment(Request $request): mixed|RedirectResponse
    {
        Stripe::setApiKey(apiKey: env(key: 'STRIPE_SECRET'));

        $request->validate(rules: [
            'price' => 'required|numeric',
            'payment_method' => 'required|in:card,fpx',
            'address_id' => 'required|exists:addresses,id,user_id,'.Auth::id(),
            'selected_items' => 'required|array',
            'selected_items.*' => 'exists:cart_items,id',
            'voucher_code' => 'nullable|exists:vouchers,code'
        ]);

        // Store checkout data in session
        session(key: [
            'checkout_data' => [
                'address_id' => $request->address_id,
                'payment_method' => $request->payment_method,
                'selected_items' => $request->selected_items,
                'delivery_fee' => 6.00,
                'grand_total' => $request->price,
                'voucher_code' => $request->voucher_code ?? null
            ]
        ]);
```

(b)

Fig. 17 *(a) Stripe Payment Gateway Interface; (b) Code Segment for Stripe's Payment Processing Function*

5.1.8 Delivery Management Module

In the Delivery Module, drivers can manage the orders that are packed by staff. Drivers can view customer information and delivery details, whereby the address is linked to Google Maps. Fig. 18 shows the delivery management interface and the code segment of Google Maps link function.



(a)

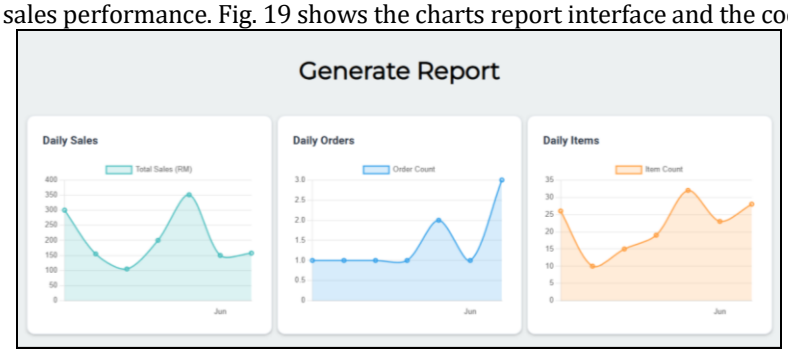
```javascript
document.addEventListener('DOMContentLoaded', function() {
    // Customer info modal
    const customerLinks = document.querySelectorAll('.customer-name-link');
    const customerModal = new bootstrap.Modal(document.getElementById('customerInfoModal'));

    customerLinks.forEach(link => {
        link.addEventListener('click', function(e) {
            e.preventDefault();

            // Get the address components
            const address = this.dataset.orderAddress;
            const postal = this.dataset.orderPostal;

            // Set the modal content
            document.getElementById('customer-name').textContent = this.dataset.customerName;
            document.getElementById('customer-email').textContent = this.dataset.customerEmail;
            document.getElementById('order-address').textContent = address;
            document.getElementById('order-postal').textContent = postal;
            document.getElementById('order-phone').textContent = this.dataset.orderPhone;

            // Set the Google Maps link
            const mapLink = document.getElementById('map-link');
            const fullAddress = encodeURIComponent(`${address}, ${postal}`);
            mapLink.href = `https://www.google.com/maps/search/?api=1&query=${fullAddress}`;

            // Show the modal
            customerModal.show();
        });
    });
});
```

(b)

Fig. 18 *(a) Delivery Management Interface; (b) Code Segment for Google Maps Link Function*

5.1.9 Delivery Management Module

In the Report Module, admin and staff can view daily sales, orders, and items chart to get an overview of recent sales performance. Fig. 19 shows the charts report interface and the code segment of chart generation function.



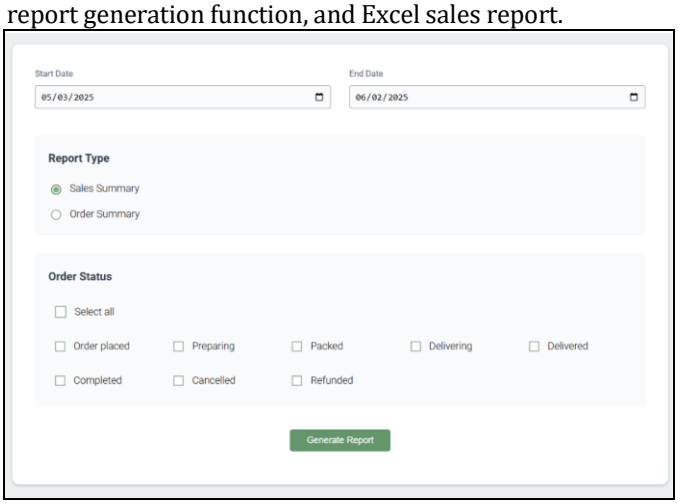
(a)

```javascript
<script src="https://cdn.jsdelivr.net/npm/chart.js"></script>
<script>
    document.addEventListener('DOMContentLoaded', function () {
        // Chart data from controller
        const labels = @json(value: $labels);
        const salesData = @json(value: $salesData);
        const orderCountData = @json(value: $orderCountData);
        const itemCountData = @json(value: $itemCountData);

        const chartConfig = (id, label, data, color, isCurrency = false) => {
            const ctx = document.getElementById(id).getContext('2d');
            return new Chart(ctx, {
                type: 'line',
                data: {
                    labels: labels,
                    datasets: [{
                        label: label,
                        data: data,
                        borderColor: color,
                        backgroundColor: color.replace('1)', '0.2)'),
                        borderWidth: 2,
                        tension: 0.3,
                        fill: true
```

(b)

Fig. 19 *(a) Charts Report Interface; (b) Code Segment for Chart Generation Function*

In addition to report in charts, admin and staff can generate Excel (.xlsx) sales and order reports by selecting the start date, end date, and statuses. Fig. 20 shows the Excel report generation interface, the code segment of Excel report generation function, and Excel sales report.



(a)

```php
public function generate(Request $request): BinaryFileResponse
{
    $request->validate(rules: [
        'start_date' => 'required|date',
        'end_date' => 'required|date|after_or_equal:start_date',
        'statuses' => 'required|array',
        'statuses.*' => 'in:order_placed,preparing,packed,delivering,delivered,completed,cancel
        'report_type' => 'required|in:summary,detailed',
    ]);

    $startDate = Carbon::parse(time: $request->start_date)->startOfDay();
    $endDate = Carbon::parse(time: $request->end_date)->endOfDay();
    $statuses = $request->statuses;

    $orders = Order::whereBetween(column: 'created_at', values: [$startDate, $endDate])
                ->whereIn(column: 'status', values: $statuses)
                ->with(relations: ['user', 'items.product', 'items.option'])
                ->get();

    if ($request->report_type === 'detailed') {
        return Excel::download(
            export: new DetailedOrderReportExport(orders: $orders, startDate: $startDate, endDa
            fileName: 'detailed_orders_report_' . now()->format(format: 'Ymd_His') . '.xlsx'
        );
    }

    return Excel::download(
        export: new SalesReportExport(orders: $orders, startDate: $startDate, endDate: $endDate
        fileName: 'sales_report_' . now()->format(format: 'Ymd_His') . '.xlsx'
    );
}
```

(b)

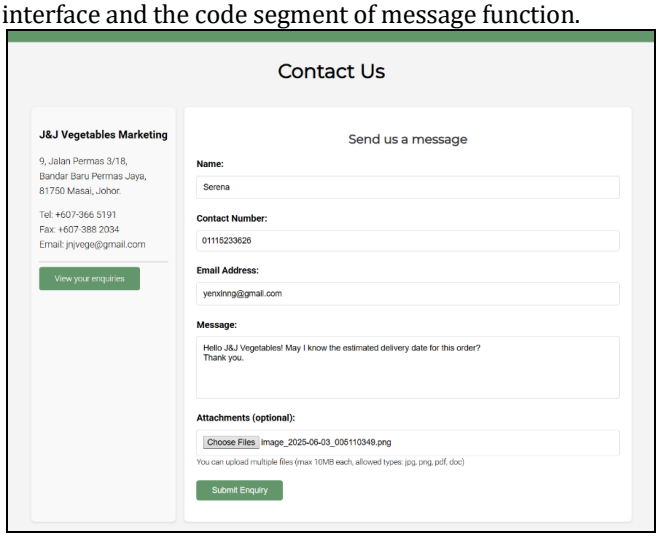
	A	B	C	D	E	F	G	H	I
1	**Order Number**	**Customer Name**	**Order Date**	**Status**	**Subtotal (RM)**	**Delivery Fee (RM)**	**Discount (RM)**	**Total (RM)**	**Item Count**
2	ORD-20250531-000002	Serena	2025-05-31 18:57:30	Completed	208.4	6	0	214.4	20
3	ORD-20250531-000003	Serena	2025-05-31 19:11:15	Delivered	130.8	6	0	136.8	12
4	ORD-20250602-000001	Sirisa Ng	2025-06-02 08:11:24	Delivered	58.4	6	0	64.4	9
5	ORD-20250602-000002	Saera	2025-06-02 08:14:32	Delivered	56.5	6	0	62.5	15
6	ORD-20250602-000003	Serena	2025-06-02 08:16:58	Cancelled	22.1	6	0	28.1	5
7	ORD-20250602-000004	Saera	2025-06-02 08:47:04	Delivered	24.7	6	0	30.7	4

(c)

Fig. 20 *(a) Excel Report Generation Interface; (b) Code Segment for Excel Report Generation Function; (c) Excel Sales Report*

5.1.10 Message Module

In the Message Module, customers can send an enquiry to the system. Fig. 21 shows the customer's message interface and the code segment of message function.



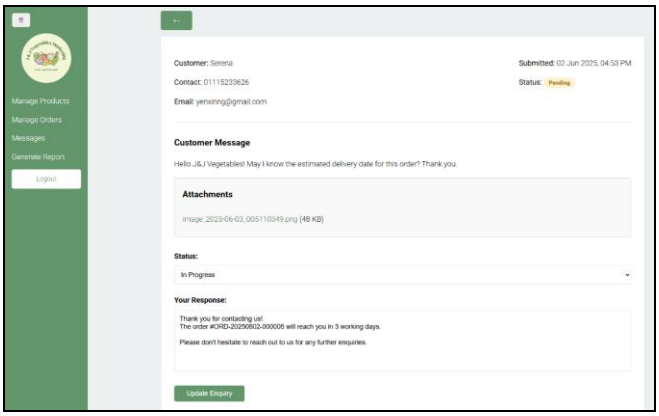
(a)

```php
class ContactController extends Controller
{
    1 reference | 0 overrides
    public function index(): View
    {
        return view(view: 'contact.index');
    }

    1 reference | 0 overrides
    public function store(Request $request): RedirectResponse
    {
        $request->validate(rules: [
            'name' => 'required|string|max:255',
            'phone' => 'required|string|max:20',
            'email' => 'required|email|max:255',
            'message' => 'required|string|max:1000',
            'attachments.*' => 'sometimes|file|max:10240|mimes:jpg,jpeg,png,pdf,doc,
        ]);

        $enquiryData = [
            'user_id' => Auth::id(),
            'enquiry_number' => Enquiry::generateEnquiryNumber(),
            'name' => $request->name,
            'contact_number' => $request->phone,
            'email' => $request->email,
            'message' => $request->message,
        ];

        $enquiry = Enquiry::create(attributes: $enquiryData);
```

(b)

Fig. 21 *(a) Customer's Message Interface; (b) Code Segment for Message Function*

Staff can respond to the customer's enquiry via message management. Fig. 22 shows the staff's message interface and the code segment of message management function.



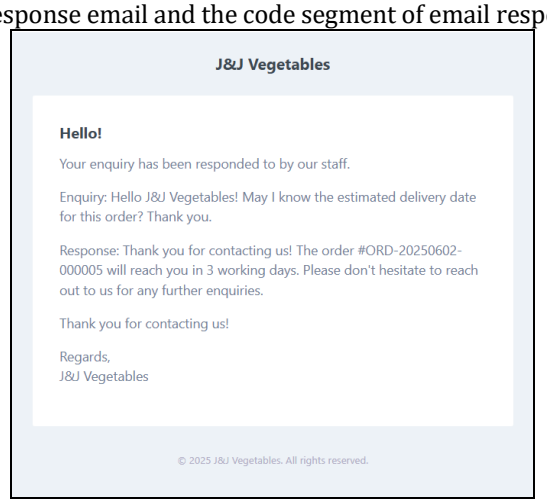
(a)

```php
class StaffContactController extends Controller
{
    2 references | 0 overrides
    public function index(): View
    {
        $enquiries = Enquiry::latest()->paginate(perPage: 10);
        return view(view: 'staff.enquiries.index', data: compact(var_name: 'enq
    }

    2 references | 0 overrides
    public function show(Enquiry $enquiry): View
    {
        // Only assign to staff if user is staff
        if (auth(guard: 'employee')->user()->role === 'staff' && is_null(value:
            $enquiry->update(attributes: ['staff_id' => auth(guard: 'employee')
        }

        return view(view: 'staff.enquiries.show', data: [
            'enquiry' => $enquiry,
            'isAdmin' => auth(guard: 'employee')->user()->role === 'admin'
        ]);
    }
```

(b)

Fig. 22 *(a) Staff's Message Interface; (b) Code Segment for Message Management Function*

The staff's response will be notified to customers via email to serve as a notification. Fig. 23 shows the response email and the code segment of email response function.



(a)

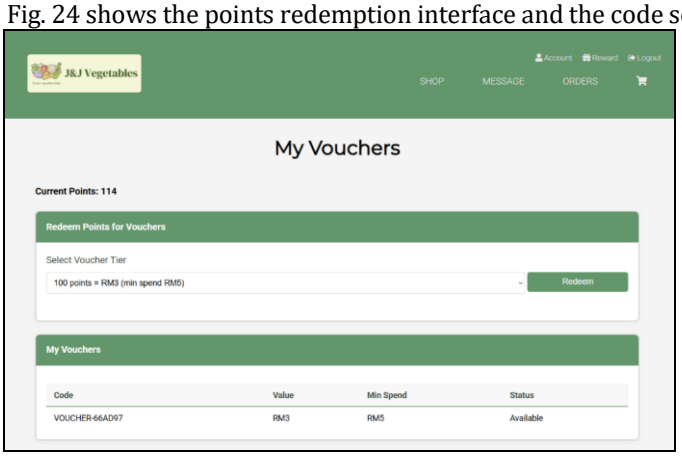
```php
class EnquiryResponse extends Notification implements ShouldQueue
{
    use Queueable;

    5 references
    public $enquiry;

    1 reference | 0 overrides
    public function __construct(Enquiry $enquiry)
    {
        $this->enquiry = $enquiry;
    }

    0 references | 0 overrides
    public function via($notifiable): array
    {
        return ['mail'];
    }

    0 references | 0 overrides
    public function toMail($notifiable): MailMessage
    {
        return (new MailMessage)
            ->subject(subject: 'Response to Your Enquiry #' . $this->enquiry->id)
            ->line(line: 'Your enquiry has been responded to by our staff.')
            ->line(line: 'Enquiry: ' . $this->enquiry->message)
            ->line(line: 'Response: ' . $this->enquiry->response)
            ->action(text: 'View Enquiry', url: route(name: 'enquiries.show', parameters: $this->enquiry))
            ->line(line: 'Thank you for contacting us!');
    }
}
```

(b)

Fig. 23 *(a) Response Email; (b) Code Segment for Email Response Function*

5.1.11 Rewards Module

In the Rewards Module, customers can view their reward points history and exchange points for vouchers. These points are accumulated from completed purchases, in which every RM1 will be converted to 1 point value. Fig. 24 shows the points redemption interface and the code segment for points redemption function.



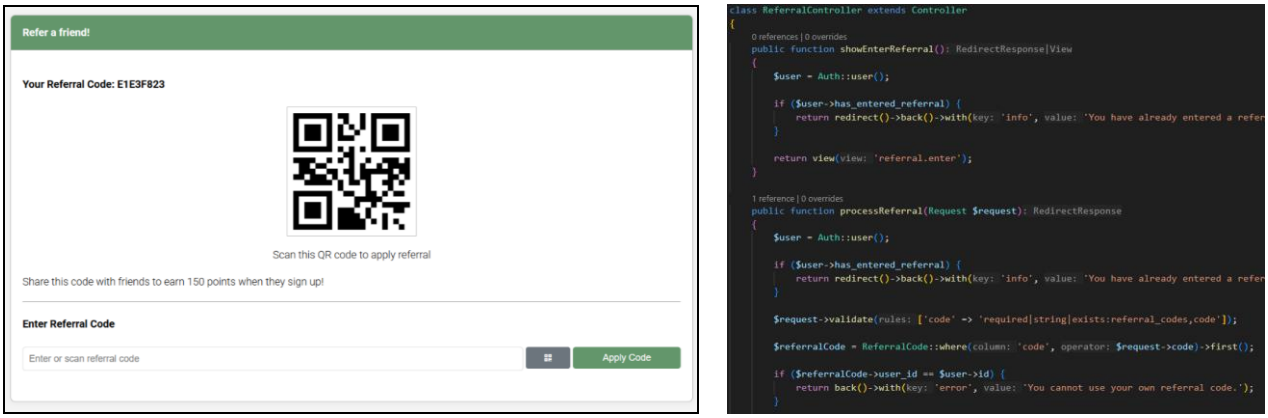
(a)

```php
public function redeem(Request $request): RedirectResponse
{
    $user = Auth::user();

    $tiers = [
        100 => ['discount' => 3, 'min_spend' => 5],
        250 => ['discount' => 8, 'min_spend' => 10],
        500 => ['discount' => 20, 'min_spend' => 22],
    ];

    $points = $request->input(key: 'points');

    if (!array_key_exists(key: $points, array: $tiers)) {
        return back()->with(key: 'error', value: 'Invalid points redemption tier.');
    }

    if ($user->points < $points) {
        return back()->with(key: 'error', value: 'You don\'t have enough points for this voucher
    }

    $voucher = Voucher::create(attributes: [
        'user_id' => $user->id,
        'code' => Voucher::generateUniqueCode(),
        'points_required' => $points,
        'discount_amount' => $tiers[$points]['discount'],
        'minimum_spend' => $tiers[$points]['min_spend'],
    ]);
```

(b)

Fig. 24 *(a) Points Redemption Interface; (b) Code Segment for Points Redemption Function*

Alternatively, customers can earn points by entering another user's referral code or by having others use their own referral code. Fig. 25 shows the referral interface and the code segment for referral processing function.



(a) **(b)**

Fig. 25 *(a) Points Redemption Interface; (b) Code Segment for Points Redemption Function*

5.2 Testing

Two testing methods are involved in the testing phase, which are system testing and user acceptance testing.

5.2.1 System Testing

System testing is conducted to evaluate the overall functionality and performance of the developed J&J Vegetables Wholesale Ordering System. Test cases are created to determine if the system satisfies the specified requirements and functions. There is a total of 11 modules, with 55 test cases that have been conducted to test the system. Table 5 presents the overall results of the test cases.

Table 5 *Overall Results of Test Cases*

Test Case ID	Module	Total Test Cases	Total Success	Total Failed
TC_100	Login	6	6	0
TC_200	Account Registration	5	5	0
TC_300	User Management	7	7	0
TC_400	Product Management	3	3	0
TC_500	Cart Management	7	7	0
TC_600	Order Management	6	6	0
TC_700	Payment Processing	5	5	0
TC_800	Delivery	6	6	0
TC_900	Report	4	4	0
TC_1000	Message	3	3	0
TC_1100	Rewards	3	3	0

5.2.2 User Acceptance Testing

User acceptance testing (UAT) is conducted to evaluate whether the developed system meets end-users' and stakeholders' needs and business requirements. UAT is divided into three main sections, which are user's satisfaction on system functionality, system usability, and system user interface (UI). Fig. 26 shows the result of UAT in terms of user's satisfaction on system functionality. The survey on User's Satisfaction on System Functionality indicates positive feedback. Most users agree that the system allows them to place orders without errors or issues, all essential features are working as expected, the system responds promptly to their actions, and they can manage and track orders efficiently.

Applied Information Technology and Computer Science Vol. 6 No. 2 (2025) p. 1338-1355

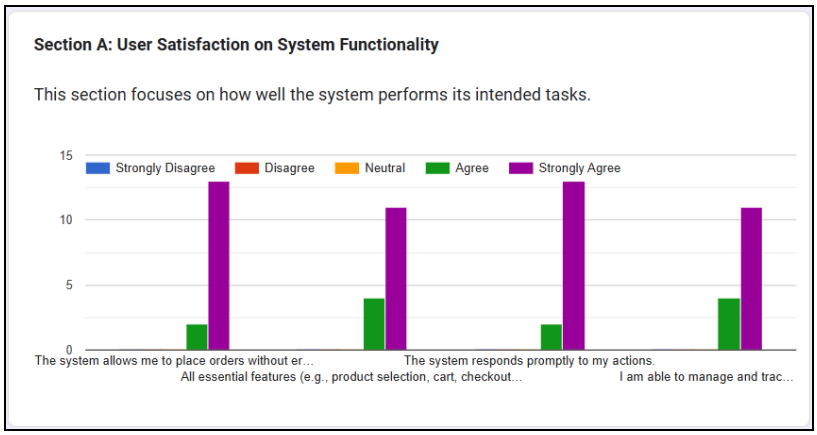


Fig. 26 *User's Satisfaction on System Functionality*

Fig. 27 shows the result of UAT in terms of user's satisfaction on system usability. The survey on User's Satisfaction on System Functionality indicates positive feedback. Most users agree that the system is easy to learn and use, navigation through the system is intuitive, users can complete tasks without requiring assistance, and the steps required to complete an action is minimal.

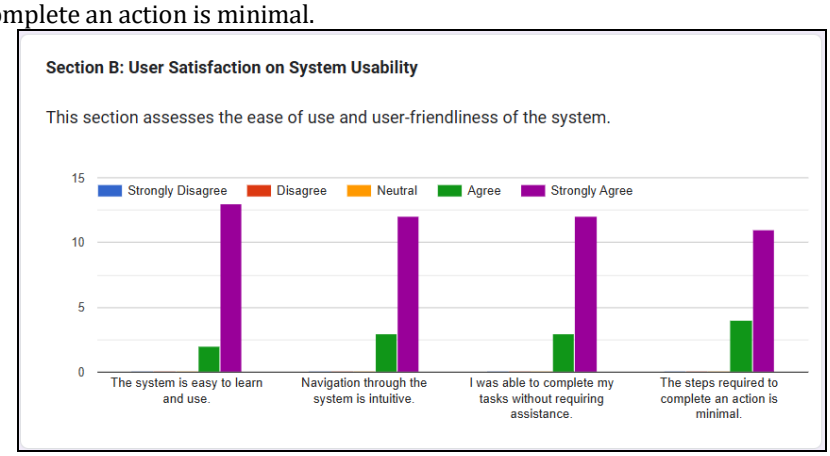


Fig. 27 *User's Satisfaction on System Usability*

Fig. 28 shows the result of UAT in terms of user's satisfaction on system user interface. The survey on User's Satisfaction on System Functionality indicates positive feedback. Most users agree that the system is easy to learn and use, navigation through the system is intuitive, users can complete tasks without requiring assistance, and the steps required to complete an action is minimal. The survey on User's Satisfaction on System User Interface (UI) indicates positive feedback. Most users agree that the system interface is visually appealing, UI elements are clear and easy to understand, the layout of information is organised and uncluttered, and the design is consistent across different pages or modules.

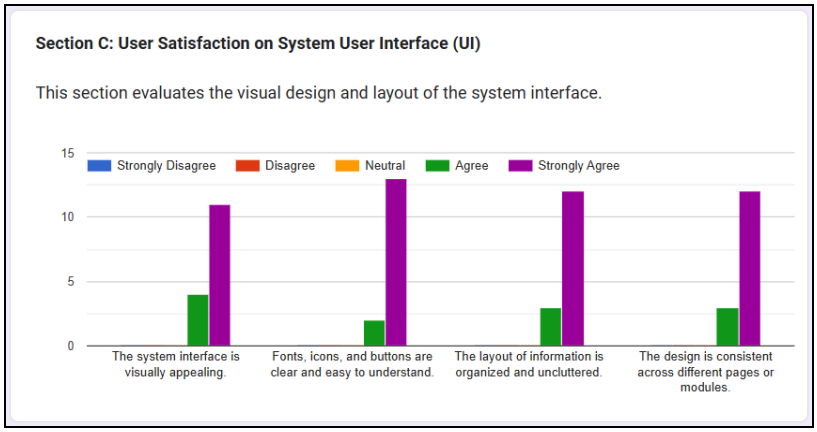


Fig. 28 *User's Satisfaction on System User Interface (UI)*

An open-ended question was included in the User Acceptance Testing (UAT) response form for additional suggestions and feedback from the users. Fig. 29 shows the additional suggestions and feedback from the User Acceptance Testing (UAT) conducted.

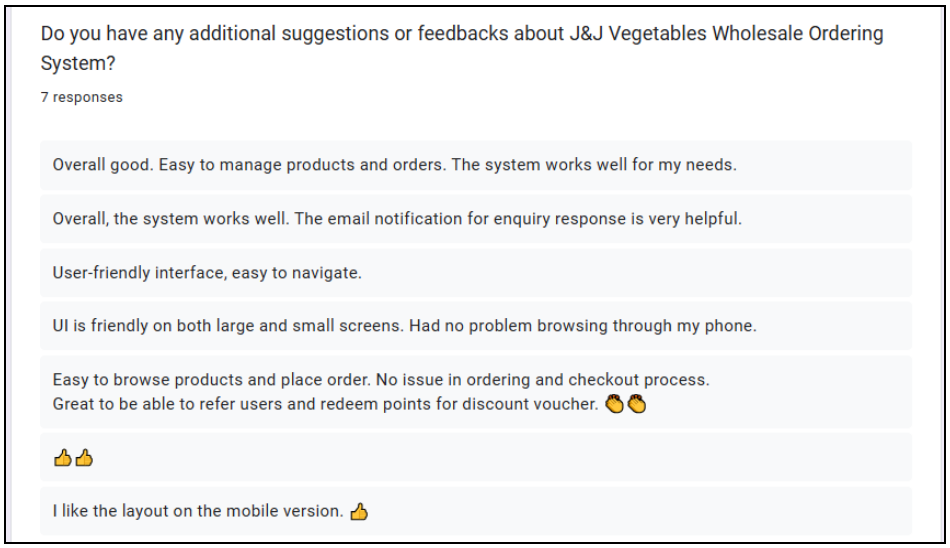


Fig. 29 *Additional Suggestions and Feedback*

Based on the results of the User Acceptance Testing (UAT), the general sentiment of the responses is very positive, indicating a high level of satisfaction with the system. Users highlighted the system's ease of use, mobile-friendly interface, and efficient ordering process. Positive remarks about helpful features like email notifications and referral rewards suggest that the system is well-received and functional.

6. Conclusion

This project highlights the development of the J&J Vegetables Wholesale Ordering System. All project objectives have been successfully achieved, effectively addressing the challenges faced by the organisation. The system provides numerous advantages, particularly in streamlining the ordering and management processes. By replacing manual and fragmented communication methods such as phone calls and WhatsApp with a centralised, web-based ordering platform, the system enhances efficiency and accuracy. The adoption of the prototype methodology and object-oriented approach ensured that the system was iteratively improved based on user feedback and testing outcomes.

The system caters to four key roles, which are administrators, staff, drivers, and customers, by offering secure login, order tracking, integrated messaging, rewards system, and report generation. These features support J&J Vegetables Marketing's goals of improving operational workflow and expanding into door-to-door delivery services.

While the current implementation has achieved its core objectives, certain limitations such as payment options, single-language support, and the lack of e-invoicing integration with the Inland Revenue Board of Malaysia (LHDN) present opportunities for future enhancement. Addressing these limitations in future updates will help increase the system's usability, accessibility, and regulatory compliance.

Overall, the system is well-positioned to support J&J Vegetables Marketing in achieving greater efficiency and customer satisfaction.

Acknowledgement

The authors would like to thank the Faculty of Computer Science and Information Technology, Universiti Tun Hussein Onn Malaysia for its support.

Conflict of Interest

Authors declare that there is no conflict of interests regarding the publication of the paper.

Author Contribution

The authors confirm contribution to the paper as follows: ***study conception and design:*** *Serena Ng Yen Xin, Rabatul Aduni Binti Sulaiman;* ***data collection:*** *Serena Ng Yen Xin, Rabatul Aduni Binti Sulaiman;* ***analysis and interpretation of results:*** *Serena Ng Yen Xin, Rabatul Aduni Binti Sulaiman;* ***draft manuscript preparation:*** *Serena Ng Yen Xin, Rabatul Aduni Binti Sulaiman. All authors reviewed the results and approved the final version of the manuscript.*

References

[1] A. Manzoor, *E-commerce: an introduction*, Amir Manzoor, 2010.
[2] A. Gupta, "E-Commerce: Role of E-Commerce in today's business," *Int. J. Comput. and Corp. Res.*, vol. 4, no. 1, pp. 1-8, 2014.
[3] Veggies.my. [Online]. Available: https://veggies.my/. [Accessed: Oct. 25, 2024].
[4] e-Petani Malaysia. [Online]. Available: https://epetanimalaysia.com/. [Accessed: Oct. 25, 2024].
[5] Organic4U. [Online]. Available: https://www.organic4u.com.my. [Accessed: Oct. 25, 2024].
[6] A. Gupta, A. Rawal, and Y. Barge, "Comparative Study of Different SDLC Models," *Int. J. Res. Appl. Sci. Eng. Technol.*, vol. 9, no. 11, pp. 73-80, 2021.
[7] S. Pargaonkar, "A Comprehensive Research Analysis of Software Development Life Cycle (SDLC) Agile & Waterfall Model Advantages, Disadvantages, and Application Suitability in Software Quality Engineering," *Int. J. Sci. and Res. Publ. (IJSRP)*, vol. 13, no. 08, pp. 345-358, 2023.
[8] S. S. Kute and S. D. Thorat, "A review on various software development life cycle (SDLC) models," *Int. J. Res. Comput. and Commun. Technol.*, vol. 3, no. 7, pp. 778-779, 2014.
[9] R. G. Sabale and A. R. Dani, "Comparative study of prototype model for software engineering with system development life cycle," *IOSR J. Eng.*, vol. 2, no. 7, pp. 21-24, 2012.
[10] R. Arora and N. Arora, "Analysis of SDLC models," *Int. J. Curr. Eng. and Technol.*, vol. 6, no. 1, pp. 268-272, 2016.
[11] H. P. Maryani, F. L. Gaol, and A. N. Hidayanto, "Comparison of the system development life cycle and prototype model for software engineering," *Int. J. Emerg. Technol. Adv. Eng.*, vol. 12, no. 4, pp. 155-162, 2022.
[12] C. Haskins and A. M. Fet, *Systems Engineering*, Springer, 2023.
[13] H. Koç, A. M. Erdoğan, Y. Barjakly, and S. Peker, "UML diagrams in software engineering research: a systematic literature review," *Proceedings*, vol. 74, no. 1, p. 13, Mar. 2021.
[14] D. Berardi, D. Calvanese, and G. De Giacomo, "Reasoning on UML class diagrams," *Artificial Intelligence*, vol. 168, no. 1-2, pp. 70-118, 2005.