# TSE2101
# Individual Report

## for

# Nursery Plant Shopping System

**Version <1.0>**

**Tutorial Section:** **TT3L**

**Group No.:** **Group 3**

**TOH EE LIN** **1221303203**

**Date:** **11/02/2024**

# Table of Contents

**1 System Overview ..... 4**
1.1 Description ..... 4
1.2 Use Cases ..... 4
1.3 Assumptions and Dependencies ..... 4
**2 Requirements ..... 5**
2.1 Use Case Diagram ..... 5
2.2 Class Diagram ..... 6
2.3 State Diagrams ..... 8
2.4 Data Flow Diagrams ..... 10
**3 Design ..... 12**
3.1 Use Cases ..... 12
3.2 Data Dictionary ..... 25
3.3 Data Structures ..... 33
3.4 Subsystem Architecture ..... 38
3.5 Subsystem Screens ..... 39
3.6 Subsystem Components ..... 45
**4 Implementation ..... 52**
4.1 Development Environment ..... 52
4.2 Main Program Codes ..... 53
4.3 Sample Screens ..... 72
**5 Testing ..... 76**
5.1 Test Data ..... 76
5.2 Acceptance Test ..... 80
5.3 Test Results ..... 80
**6 Conclusion ..... 115**
6.1 Project Achievements ..... 115
6.2 Quality Assurance ..... 115
6.3 Problems Encountered ..... 115
6.4 Remarks/Comments ..... 116

# Revisions

| Version | Primary Author(s) | Description of Version | Date Completed |
|---|---|---|---|
| Version 1.0 | Toh Ee Lin | This version includes the demonstration of essential functionalities, testing strategies, and a thorough update of various diagrams including use case diagram, class diagram, and data flow diagrams. Furthermore, it provides insights into the development environment, showcases sample screens from the implementation phase, and discusses the test data alongside acceptance tests and results. The document concludes with reflections on project achievements and problem encountered. | 11/02/2024 |

# 1 System Overview

## 1.1 Description

In our nursery plant shopping system, every user, including guests, is welcomed by the home screen named TLET Nursery Plant Shopping System. From here, registered users can click on the “Login” button and fill in their email and password. This will lead them to their distinct dashboards according to their specific roles—be it Administrator, Customer, or Delivery Man.

For Administrator, the system is designed to allow him to manage the plants. This includes create a new plant, update an existing plant, and delete a plant. Administrator can also search for plants by plant name. Besides, administrator can oversee customer orders, which allows him to view detailed order information and update order statuses as needed. Administrator can also assign deliveries to delivery men.

## 1.2 Use Cases

Table 1.1 shows the Administrator and his corresponding use cases in our nursery plant shopping system. Administrator is responsible for managing plants and orders as well as assigning deliveries.

Table 1.1: Administrator and Respective Use Cases

| Actor | Use Cases |
|---|---|
| Administrator | An administrator manages the plants.<br>● Create a plant. (sub-use case)<br>● Update a plant. (sub-use case)<br>● Delete a plant. (sub-use case) |
| | An administrator searches a plant by plant name. |
| | An administrator views order details and updates order status. |
| | An administrator assigns deliveries to delivery men. |

## 1.3 Assumptions and Dependencies

In this project, there are several assumptions related to administrator that we have made:

1. The nursery plant shopping system is designed to have only one administrator responsible for its management and operation.

2. Regardless of the number of times an assigned delivery order is declined by delivery men, eventually, there will be a delivery man who accepts and successfully completes the order.

# 2 Requirements

## 2.1 Use Case Diagram

Figure 2.1 is a use case diagram for Administrator, which is one of the actors in our nursery plant shopping system. This use case diagram depicts the interactions between the administrator and the system. The Administrator can manage the plants, and this manage plant use case includes three sub-use cases, which are create a plant, update a plant and delete a plant. For search a plant by plant name use case, the administrator can initiate a search by entering a plant name in the provided search bar. For views order details and updates order status use case, the administrator can view and update the status of ongoing orders for different customers. For pickup orders, the administrator can update their status from “Waiting” to “Ready” and to “Completed, whereas for delivery orders, the administrator can update their status from “Waiting” to “Ready”. Lastly, the administrator can assign deliveries to delivery men.

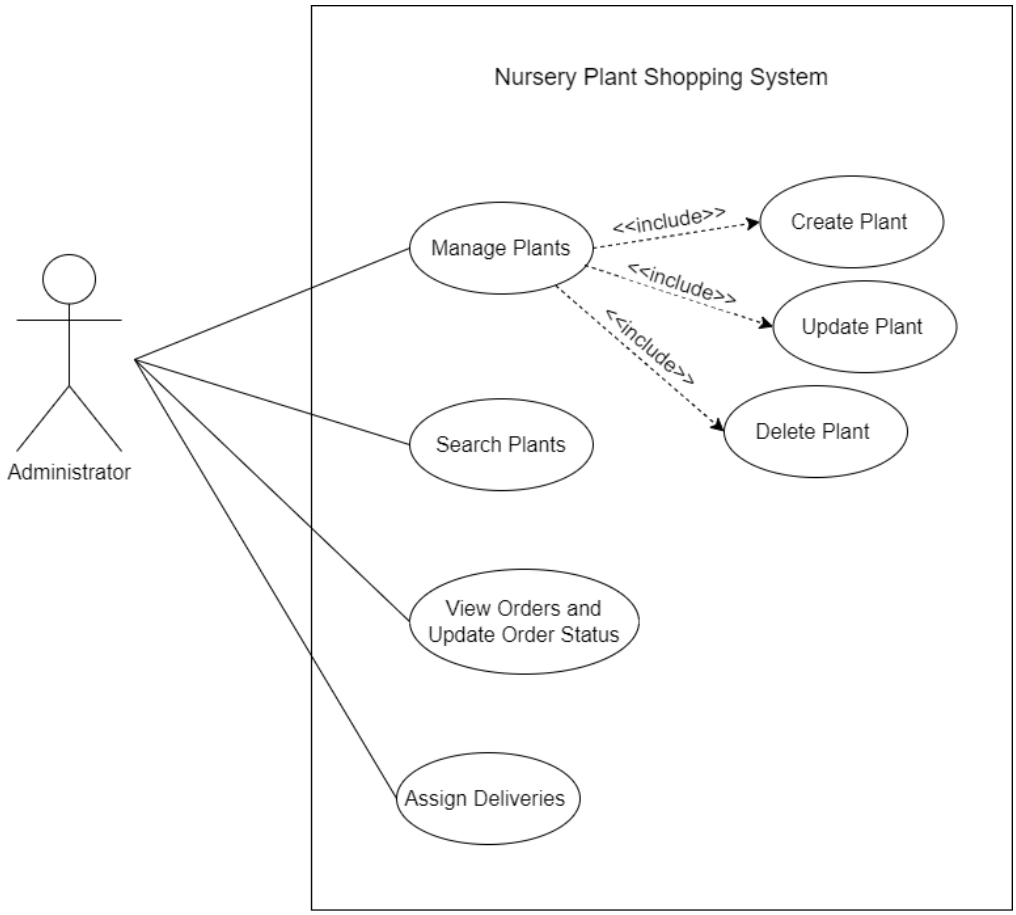



Figure 2.1: Administrator Use Case Diagram

## 2.2 Class Diagram

Figure 2.2 shows the partial nursery plant shopping system class diagram, which only related to Administrator. Administrator class, which is an extension of the User class, is equipped with methods for administrative actions like managing plants, orders, and deliveries.

Administrator class is connected via a one-to-one association to the User class, which includes attributes like user_id and email, along with role indicators (is_customer, is_deliveryman, is_admin) that are fundamental to authentication and role determination within the system. Besides, Administrator class is linked to Order class while Order class is related to OrderItem class by composition. This relationship allows the administrator to see the details for each order. Plant class is linked to OrderItem class, which means each order item is linked to a specific plant, representing the purchase of that plant. On the other hand, OrderItem class is connected to Review class, which allows customers to provide feedback on individual items.

Furthermore, Customer class is linked to Order class, which allows a single customer to place multiple orders and helps in tracking a customer's order within the system. DeliveryMan class is also linked to Order class, which streamlines the assignment and management of delivery orders. Payment class, which records financial transactions details, is having a one-to-one relationship with Order class. This ensures that each order has a corresponding payment record for financial tracking.

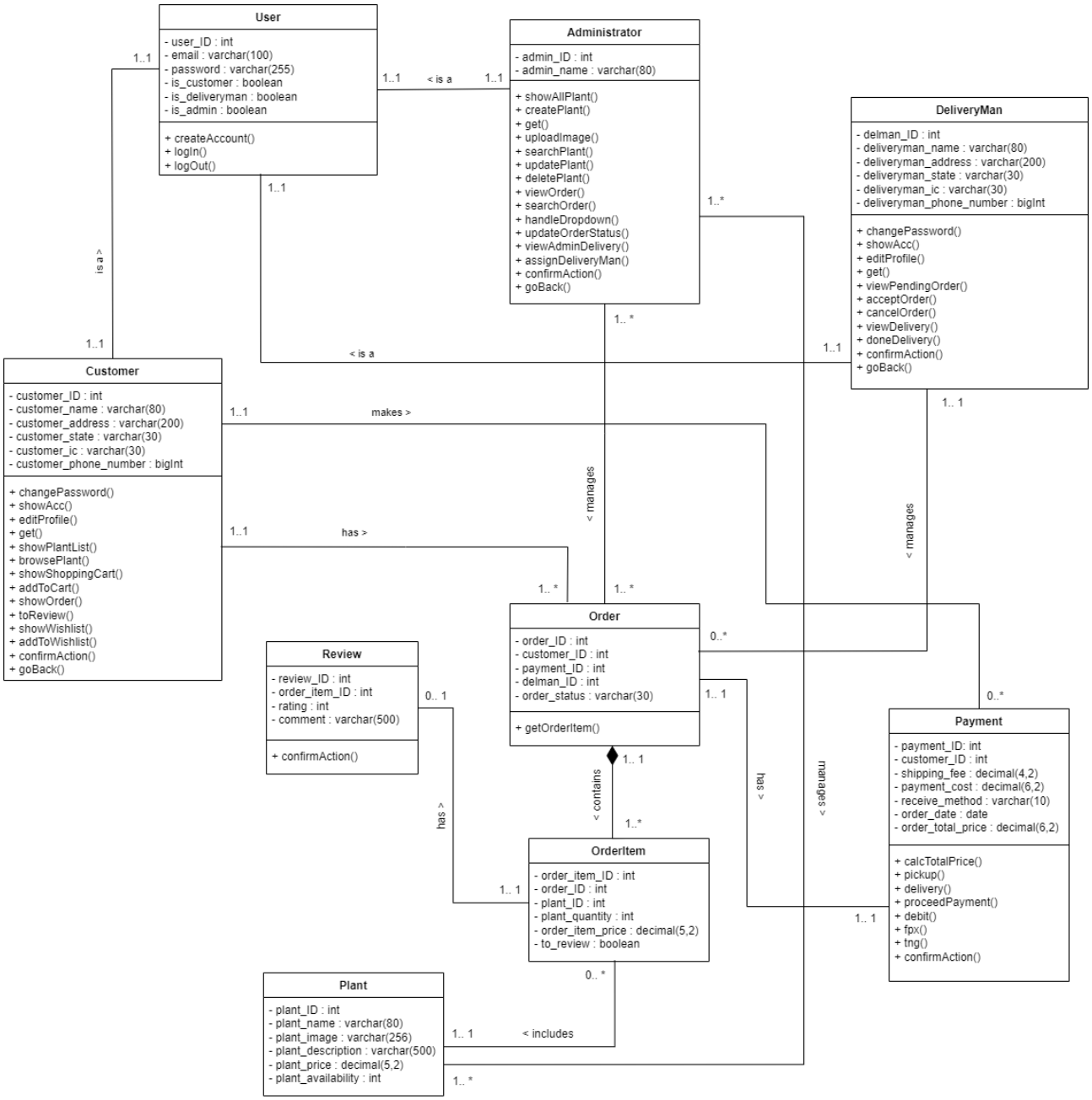



Figure 2.2: Nursery Plant Shopping System Class Diagram (Related to Administrator)

## 2.3 State Diagrams

### 2.3.1 Nursery Plant Shopping System State Diagram

Figure 2.3 illustrates the State Diagram for TLET Nursery Plant Shopping System. The user will begin with the “TLET Nursery Plant Shopping System Screen” as the initial state, from which users can navigate to the “Login Screen”. Upon successful login, the system transitions to different home screens based on the user's role which are “Administrator Home Screen” for administrator login, “Customer Home Screen” for customer login, and “Delivery Man Home Screen” for delivery personnel login. Each home screen provides a “Logout” transition leading back to the “Login Screen”.

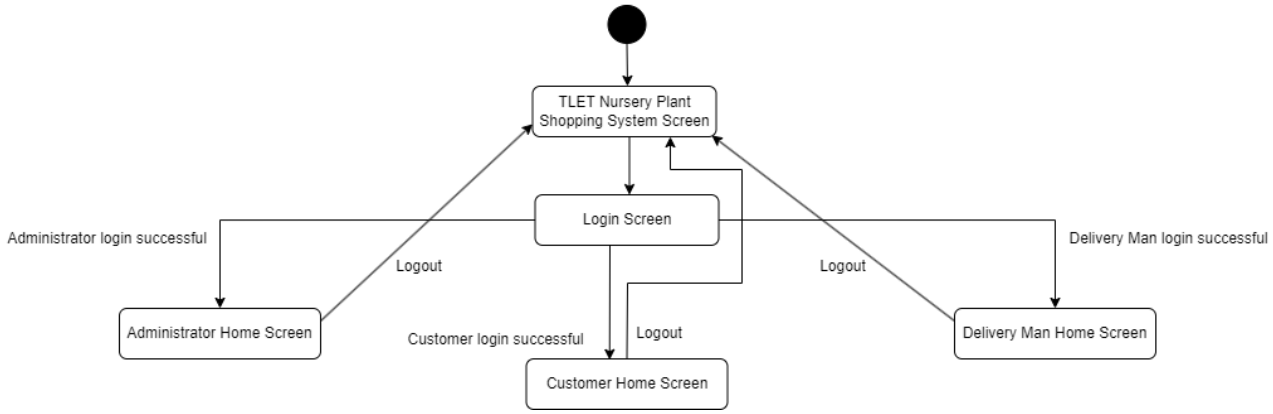



Figure 2.3: State Diagram for TLET Nursery Plant Shopping System

### 2.3.2 Administrator State Diagram

Figure 2.4 illustrates the State Diagram for Administrator. From the TLET Nursery Plant Shopping System Screen, a user clicks on the “Login” button to go to the Login Screen. After the user filling in the email and password and is identified as administrator, he/she is presented with the Administrator Home Screen. From here, the administrator has multiple pathways to manage the system effectively:

If the “Plant Management” section is chosen, the administrator will enter the Plant Management Screen. Here, he/she can opt to create a new plant by clicking the “Create New Plant” button, which leads to the Create New Plant Screen. Upon successful creation, the system transitions back to the Plant Management Screen. Alternatively, the administrator can search for existing plants or delete them within this same screen. If the administrator decides to update a plant's details, he/she can click on the “Edit” button. This will take them to the Update Plant Screen, where after successful updates, they will be redirected back to the Plant Management Screen.

When the administrator selects the “Order Management” section, he/she will be led to the Order Management Screen. In this screen, the administrator can search for orders by entering the orderID in the search bar. He/she also can view the order details and update the status of ongoing orders for each customer, streamlining the order processing and tracking phases of the system's operation.

Furthermore, when the “Delivery Management” section is chosen, the Delivery Management Screen will be displayed. The administrator can use the search bar to search for delivery orders and assign

these delivery orders to delivery men. This ensures that the delivery workflow is maintained efficiently.

At any point, the administrator can return to the Administrator Home Screen by selecting the “Admin Dashboard” section. Besides, he/she can sign out of the system by clicking “Logout”, which will then redirect them to the TLET Nursery Plant Shopping System Screen.

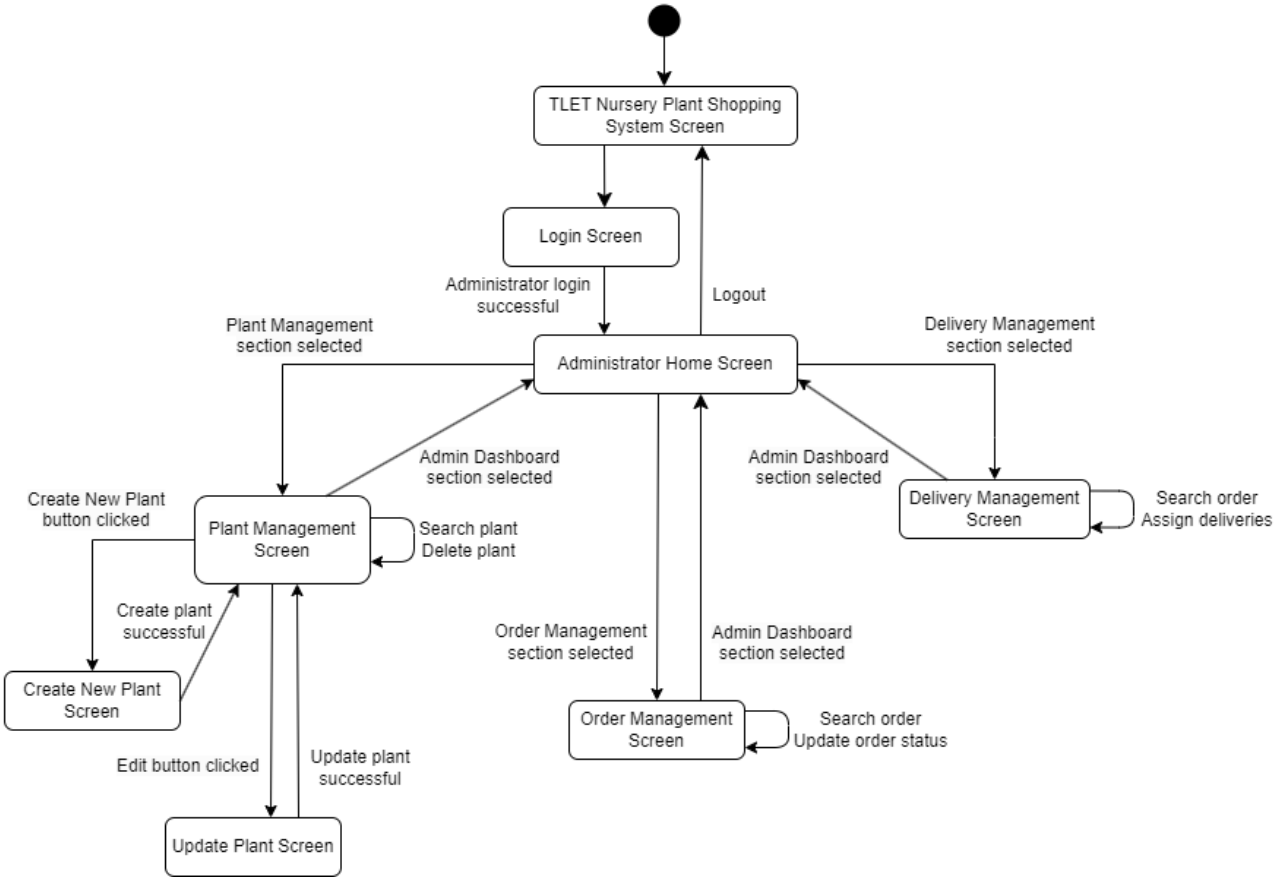



Figure 2.4: State Diagram for Administrator

## 2.4 Data Flow Diagrams

Figure 2.5 illustrates the Data Flow Diagram for Nursery Plant Shopping System. When creating a plant, the Administrator inputs essential details such as the plant ID, plant name, plant description, plant image, plant price, and plant availability, which are subsequently stored in the Plant Database. This database becomes the primary source for all plant-related information within the system. When Administrator wishes to update a plant's details, he/she can modify the existing records such as the plant's name, description, image, price, and availability in order to ensure the Plant Database reflects the most current information. In cases where a plant needs to be removed from the system, the Administrator uses the plant ID to delete the respective data from the Plant Database. Besides plant management, Administrator can search for specific plants by plant name. The system will then pull the relevant details from the Plant Database to assist in quick retrieval or editing.

Additionally, the Administrator plays a crucial role in the order management process. He/she can view the order details sourced from the Order Database and are responsible for updating the order status to reflect its progress. For example, Administrator can mark both pickup orders and delivery orders as “Ready” as well as mark the pickup orders as “Completed” after customer pickup.

The Administrator also holds the responsibility of assigning delivery orders to specific delivery man within the nursery plant shopping system. This is done by passing the delman ID to the corresponding order, effectively linking the delivery order with the designated delivery man. Consequently, this allocated order becomes visible on the assigned delivery man's pending order screen, allowing them to see and manage their delivery tasks effectively.

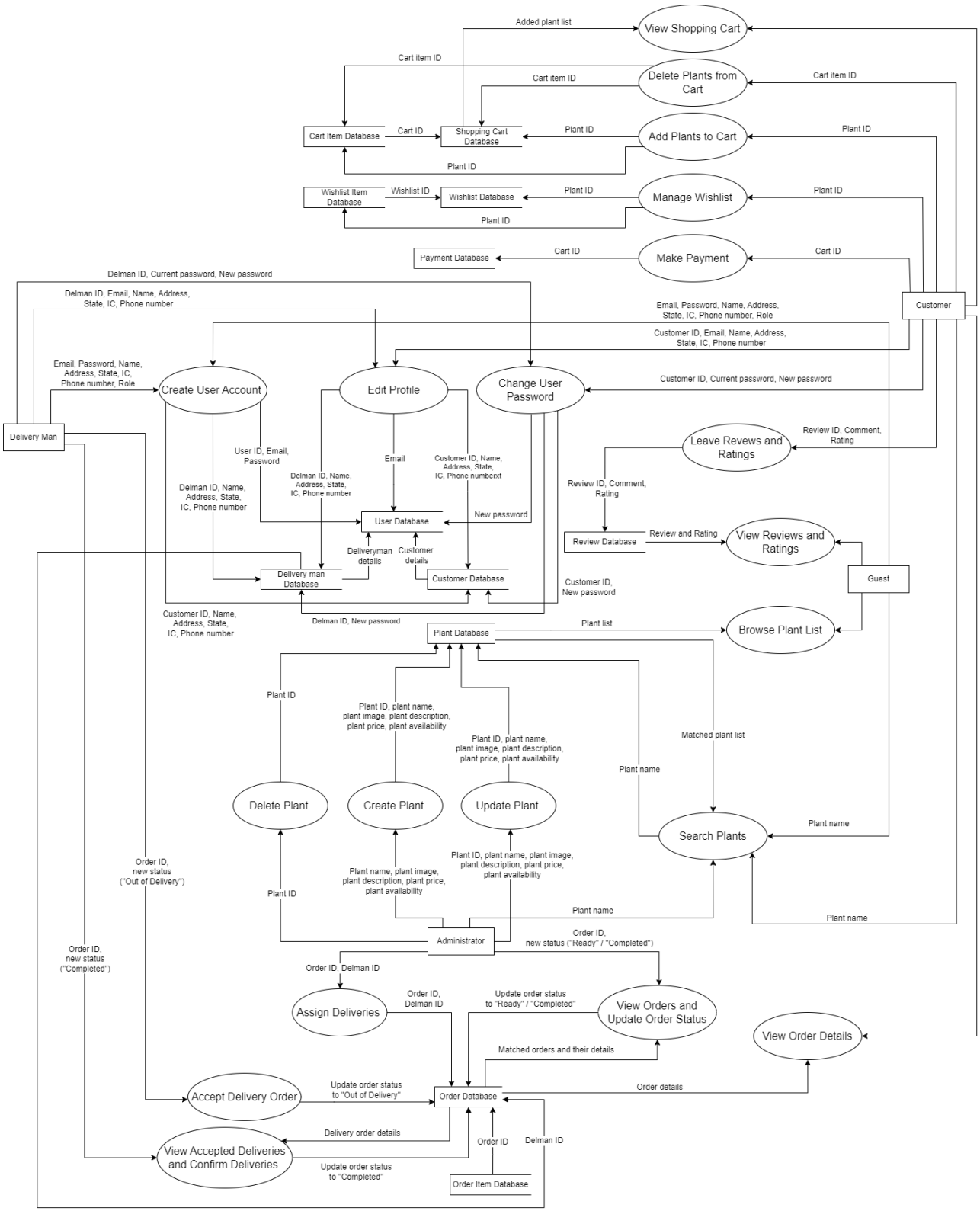



Figure 2.5: Data Flow Diagram for Nursery Plant Shopping System

# 3 Design

## 3.1 Use Cases

### 3.1.1 An administrator manages the plant.

Table 3.1: An Administrator Manages Plant Use Case Description

| | |
|---|---|
| Use Case Name: | An administrator creates a plant. |
| Description: | The administrator can create a new plant by navigating through a step-by-step process, such as entering details and uploading images for the new plant. In case of cancellation and validation failure, the system will return the existing plant list. |
| Primary Actor: | Administrator |
| Precondition | ● The administrator is logged into the system. |
| Postcondition | ● The new plant is successfully created, and the administrator can view it in the updated plant list. |
| Main Success Scenario: | S01: The administrator clicks on the "Plant Management" section.<br>S02: The system displays a list of existing plants.<br>S03: The administrator clicks on the “Create New Plant” button.<br>S04: The system presents a form for entering plant details, including plant name, descriptions, quantity and pricing.<br>S05: The administrator fills in the required information in the form such as plant name, descriptions, quantity and pricing.<br>S06: The administrator clicks on the "Choose File" button to upload the plant image.<br>S07: The system displays the photos that are stored inside the file of the administrator computer.<br>S08: The administrator selects the images that he/she wants to upload and clicks the “Upload Image” button.<br>S09: The administrator clicks on the “Confirm” button.<br>S10: The system validates the entered plant name to ensure they are new and do not exist in the system data storage.<br>S11: If validation succeeds, the system creates the new plant and auto-generates an ID as a unique identifier for the plant.<br>S12: The system displays a confirmation message, which shows the successful creation of the plant. |

| | |
|---|---|
| | S13: The system displays the updated list of plants. |
| Alternative Scenario: | - |
| Exception Scenario: | A01: @S04 The administrator decides not to proceed with the creation of a plant.<br>A01.05: The administrator clicks on the “Back” button to cancel the creation process.<br>A01.06: The system displays back the list of existing plants.<br>A01.07: END<br><br>A02: @S10 The system validation fails.<br>A02.11: The system displays an error message, which shows the same plant exists in the system.<br>A02.12: The system displays back the list of existing plants.<br>A02.13: END |
| Use Case Name: | An administrator updates a plant. |
| Description: | The administrator can update the plant by clicking the "Edit" button to modify its information. After the administrator clicks the "Update" button, the system confirms the successful update and displays the updated list of plants. |
| Primary Actor: | Administrator |
| Precondition | ● The administrator is logged into the system.<br>● There are existing plants in the system.<br>● The administrator has the necessary permissions to update the plant. |
| Postcondition | ● The selected plant is successfully updated with the modified information or images. |
| Main Success Scenario: | S01: The administrator clicks on the "Plant Management" section.<br>S02: The system displays a list of existing plants.<br>S03: The administrator clicks the "Edit" button on a particular plant.<br>S04: The system presents the details of the selected plant, including plant name, image, descriptions, quantity and pricing.<br>S05: The administrator makes the necessary changes to the plant information.<br>S06: The administrator clicks on the "Update" button. |

| | |
|---|---|
| | S07: The system displays a confirmation message indicating the successful update of the plant.<br>S08: The system returns the updated list of plants. |
| Alternative Scenario: | - |
| Exception Scenario: | A01: @S04 The administrator decides not to update the plant.<br>A01.05: The administrator clicks on the “Back” button.<br>A01.06: The system displays back the list of existing plants.<br>A01.07: END<br><br>A02: @S06 The administrator wants to update the image for the plant.<br>A02.07: The administrator clicks on the "Choose File" button.<br>A02.08: The system displays the photos that are stored inside the file of the administrator computer.<br>A02.09: The administrator selects the images that he/she wants to upload and clicks the “Upload Image” button.<br>A02.10: RESUME @S06 |
| Use Case Name: | An administrator deletes a plant. |
| Description: | The administrator can select a specific plant for deletion. Upon clicking the "Delete" button, the system prompts the administrator to confirm the action. In the event of a cancellation, the system returns to the plant list without making any changes. |
| Primary Actor: | Administrator |
| Precondition | ● The administrator is logged into the system.<br>● There are existing plants in the system. |
| Postcondition | ● The selected plant is successfully deleted, and the administrator can review the updated list of plants. |
| Main Success Scenario: | S01: The administrator clicks on the "Plant Management" section.<br>S02: The system displays a list of existing plants.<br>S03: The administrator clicks the "Delete" button on a particular plant.<br>S05: The system displays a confirmation message, which prompts the administrator to confirm the deletion.<br>S06: The administrator clicks on the "Delete" button to confirm the deletion.<br>S07: The system deletes the selected plant and updates the list of plants.<br>S08: The system displays a confirmation message, indicating the successful deletion of the plant. |

| | |
|---|---|
| | S09: The system displays the updated list of plants. |
| Alternative Scenario: | - |
| Exception Scenario: | A01: @S05 The administrator decides not to proceed with the deletion.<br>A01.06: The administrator clicks on the "Cancel" button to cancel the deletion process.<br>A01.07: The system displays back the plant list without making any changes.<br>A01.08: END |

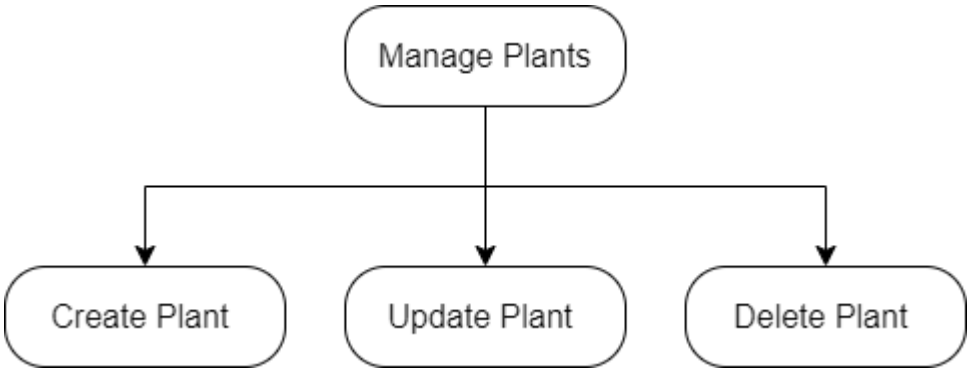



Figure 3.1: Sub-use cases for Manage Plants Use Case

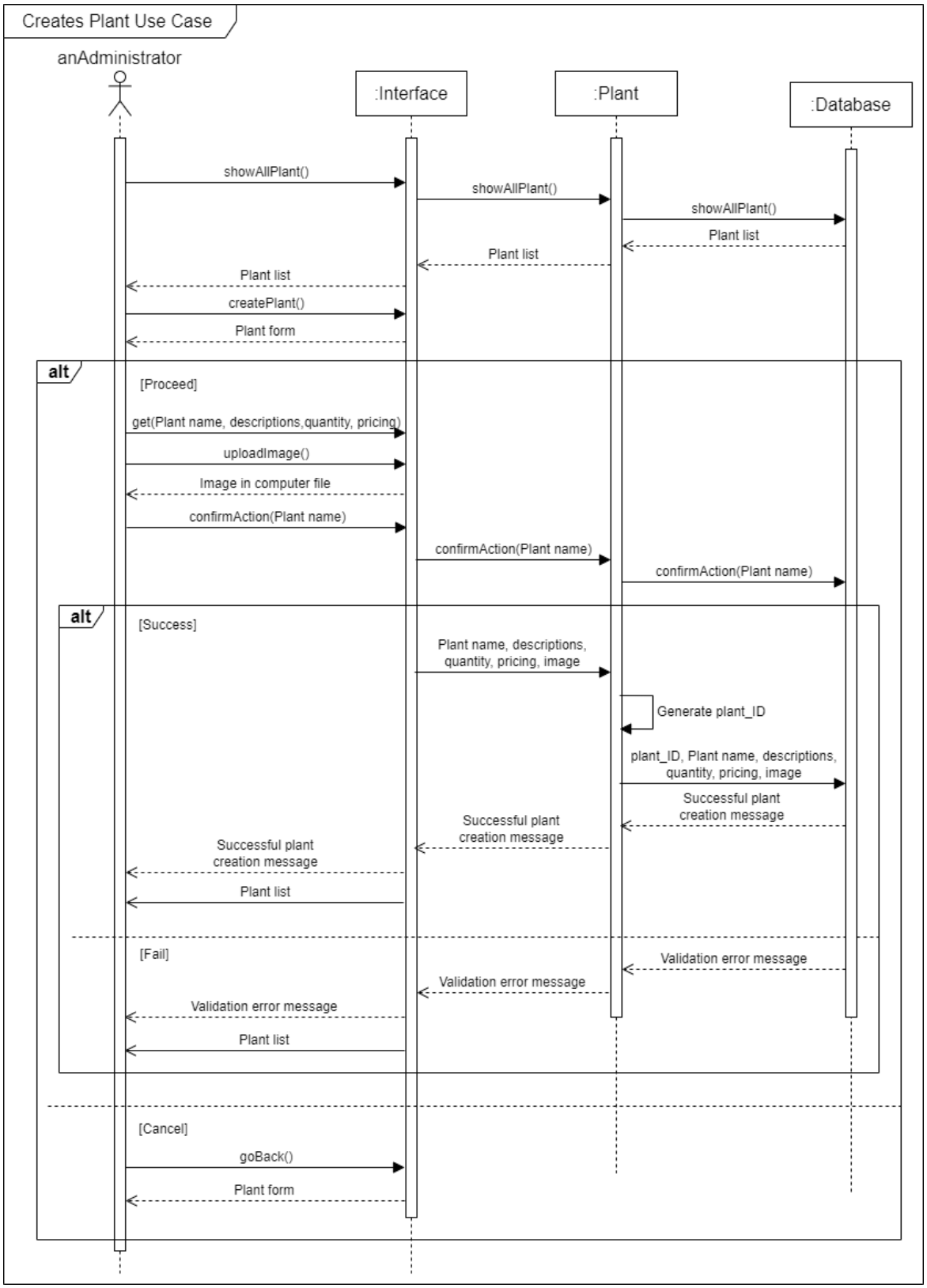



Figure 3.2: Create Plant Sequence Diagram

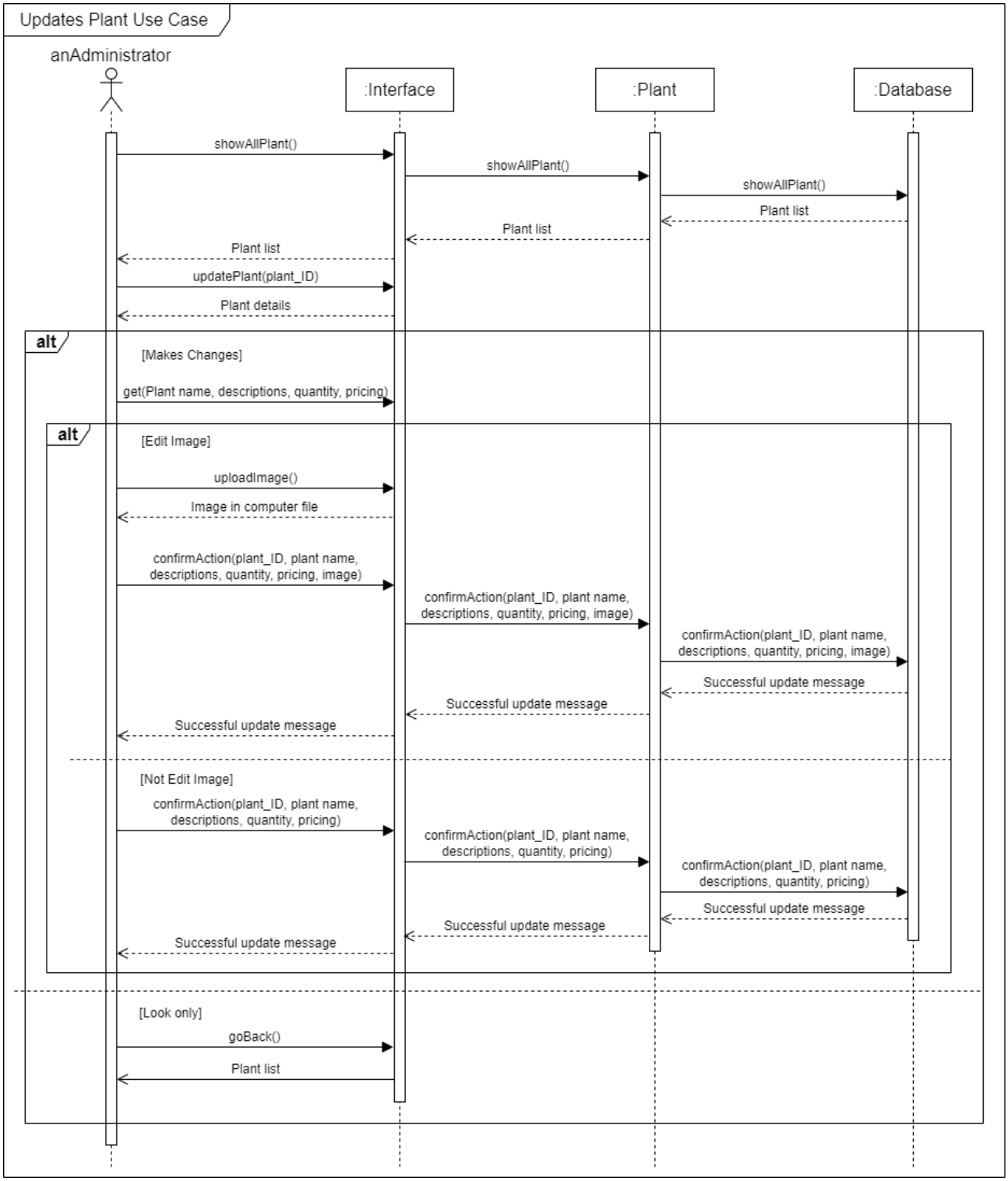



Figure 3.3: Update Plant Sequence Diagram

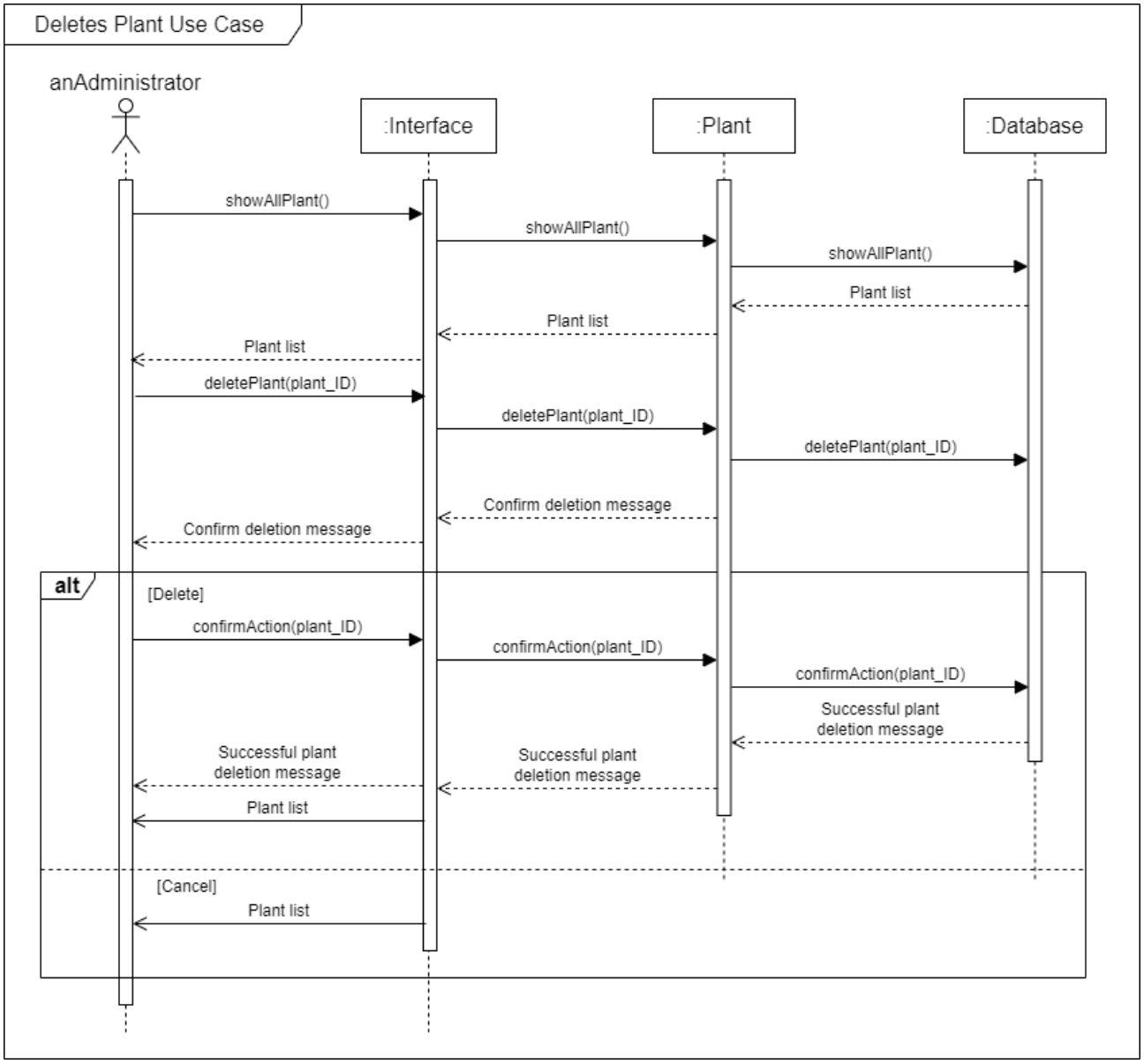



Figure 3.4: Delete Plant Sequence Diagram

### 3.1.2 An administrator searches a plant by plant name.

Table 3.2: An Administrator Searches Plant by Plant Name Use Case Description

| | |
|---|---|
| Use Case Name: | An administrator searches plant by plant name. |
| Description: | The administrator can initiate a search by entering a plant name in the provided search bar. In the event of no matching results, the system will display a message and return the list of existing plants to the administrator. |

| | |
|---|---|
| Primary Actor: | Administrator |
| Precondition | ● The administrator is logged into the system.<br>● There are existing plants in the system. |
| Postcondition | ● The administrator has successfully searched for and viewed information about a plant by its name. |
| Main Success Scenario: | S01: The administrator clicks on the "Plant Management" section.<br>S02: The system displays a list of existing plants.<br>S03: The administrator enters the plant name in the search bar on top.<br>S04: The administrator clicks on the "Search" button.<br>S05: The system scans the plant list for matching plant names and outputs a list of plants with names that match the input. |
| Alternative Scenario: | - |
| Exception Scenario: | A01: @S05 No matching results are found.<br>A01.06: The system displays a message indicating that no plants match the entered name.<br>A01.07: The system displays back the list of existing plants.<br>A01.08: END |

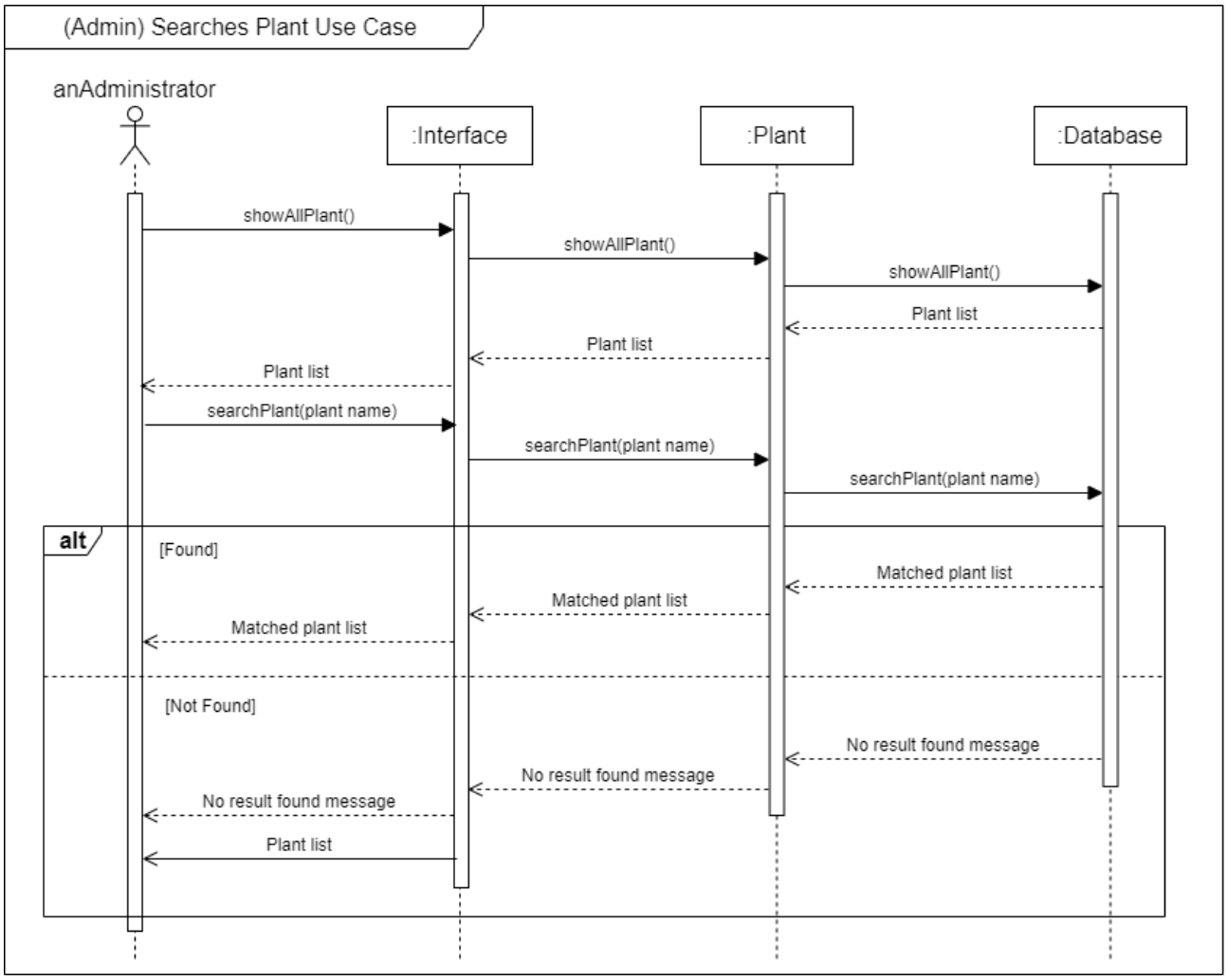



Figure 3.5: Administrator Search Plant Sequence Diagram

### 3.1.3 An administrator views order details and updates order status.

Table 3.3: An Administrator Views Order Details and Updates Order Status Use Case Description

| | |
|---|---|
| Use Case Name: | An administrator views order details and updates the status of ongoing orders for different customers. |
| Description: | Administrators can view and update the status of ongoing orders for different customers. Orders marked as 'Ready' indicate that they are prepared for customer pickup or will be assigned to a delivery man for further processing. Furthermore, the status can be updated to 'Completed' either by the administrator after customer pickup or automatically when the delivery man presses 'Confirm Delivery', signifying the successful conclusion of the order. A search option based on orderID is available, allowing the system to display matching results. |

| | |
|---|---|
| Primary Actor: | Administrator |
| Precondition | ● The administrator is logged into the system. |
| Postcondition | ● The administrator has successfully viewed detailed information about the ongoing order, and the order status has been updated to either 'Ready' or 'Completed', depending on the situation. The system promptly reflects these changes in the database. |
| Main Success Scenario: | S01: The administrator clicks on the "Order Management" section.<br>S02: The system displays a list of ongoing orders with detailed information for each ongoing order, including orderID, customerID, customer details, plant quantity, order date, receive method, total price and order status.<br>S03: The administrator selects "Ready" from the status drop-down menu for a specific order.<br>S04: The administrator clicks on the "Update Status" button to confirm the status change.<br>S05: The system updates the order status to "Ready" and refreshes the ongoing order list with the updated information. |
| Alternative Scenario: | A01: @S02 The administrator enters the orderID in the search bar on top.<br>A01.03: The administrator clicks on the "Search" button.<br>A01.04: The system performs a search based on the entered orderID and displays the matching results of ongoing orders.<br>A01.05: RESUME @S03.<br><br>A02: @S02 The administrator found a specific order that has been picked up by the customer.<br>A02.03: The administrator selects "Completed" from the status drop-down menu for the specific order.<br>A02.04: The administrator clicks on the "Update Status" button to confirm the status change.<br>A02.05: The system updates the order status to "Completed" and refreshes the ongoing order list with the updated information.<br>A02.06: END |
| Exception Scenario: | A03: @S02 No ongoing orders in the system.<br>A03.03: The system displays a message indicating that there are no ongoing orders in the system.<br>A03.04: END<br><br>A04: @A01.04 No matching results are found. |

| | |
|---|---|
| | A04.05: The system displays a message indicating that no ongoing orders match the entered orderID.<br>A04.06: The administrator refines the search by entering a different orderID.<br>A04.07: RESUME @A01.03 |

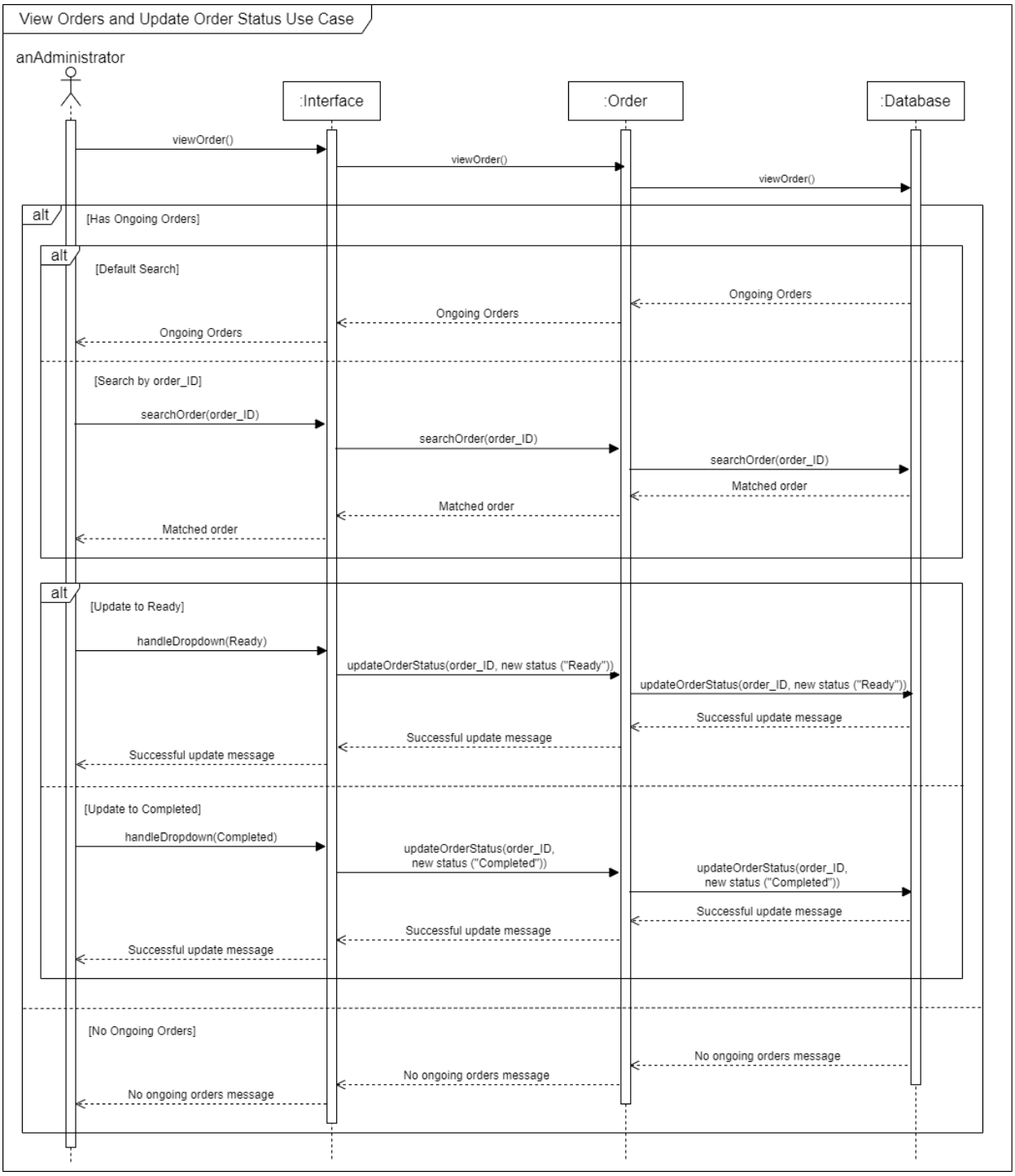



Figure 3.6: View Orders and Update Order Status Sequence Diagram

### 3.1.4 An administrator assigns deliveries to delivery men.

Table 3.4: An Administrator Assigns Deliveries to Delivery Men Use Case Description

| | |
|---|---|
| Use Case Name: | An administrator assigns deliveries to delivery men. |
| Description: | The administrator interacts with the system to select a specific delivery from the list and assigns it to a delivery man. This ensures that deliveries are effectively distributed among delivery personnel for prompt and accurate order delivery. |
| Primary Actor: | Administrator |
| Precondition | ● The administrator is logged into the system.<br>● The administrator has the necessary permissions to assign deliveries.<br>● There are existing deliveries in the system. |
| Postcondition | ● The administrator has successfully assigned the selected delivery to a delivery man. |
| Main Success Scenario: | S01: The administrator clicks on the “Delivery Management” section.<br>S02: The system displays a list of ongoing orders scheduled for delivery, with relevant information such as orderID, customerID, customer details, plant quantity, order date, receive method, total price and “Assign To” status.<br>S03: The administrator selects a delivery man name from the drop-down menu for a specific order.<br>S04: The administrator clicks on the “Assign” button to assign the selected delivery to the delivery man.<br>S05: The system updates the “Assign To” status and returns the updated list of ongoing orders scheduled for delivery. |
| Alternative Scenario: | A01: @S02 The administrator enters the orderID in the search bar on top.<br>A01.03: The administrator clicks on the "Search" button.<br>A01.04: The system performs a search based on the entered orderID and displays the matching results of ongoing orders.<br>A01.05: RESUME @S03. |

| | |
|---|---|
| Exception Scenario: | A02: @S02 No ongoing orders in the system.<br>A02.03: The system displays a message indicating that there are no ongoing delivery orders in the system.<br>A02.04: END<br><br>A03: @A01.04 No matching results are found.<br>A03.05: The system displays a message indicating that no ongoing delivery orders match the entered orderID.<br>A03.06: The administrator refines the search by entering a different orderID.<br>A03.07: RESUME @A01.03 |

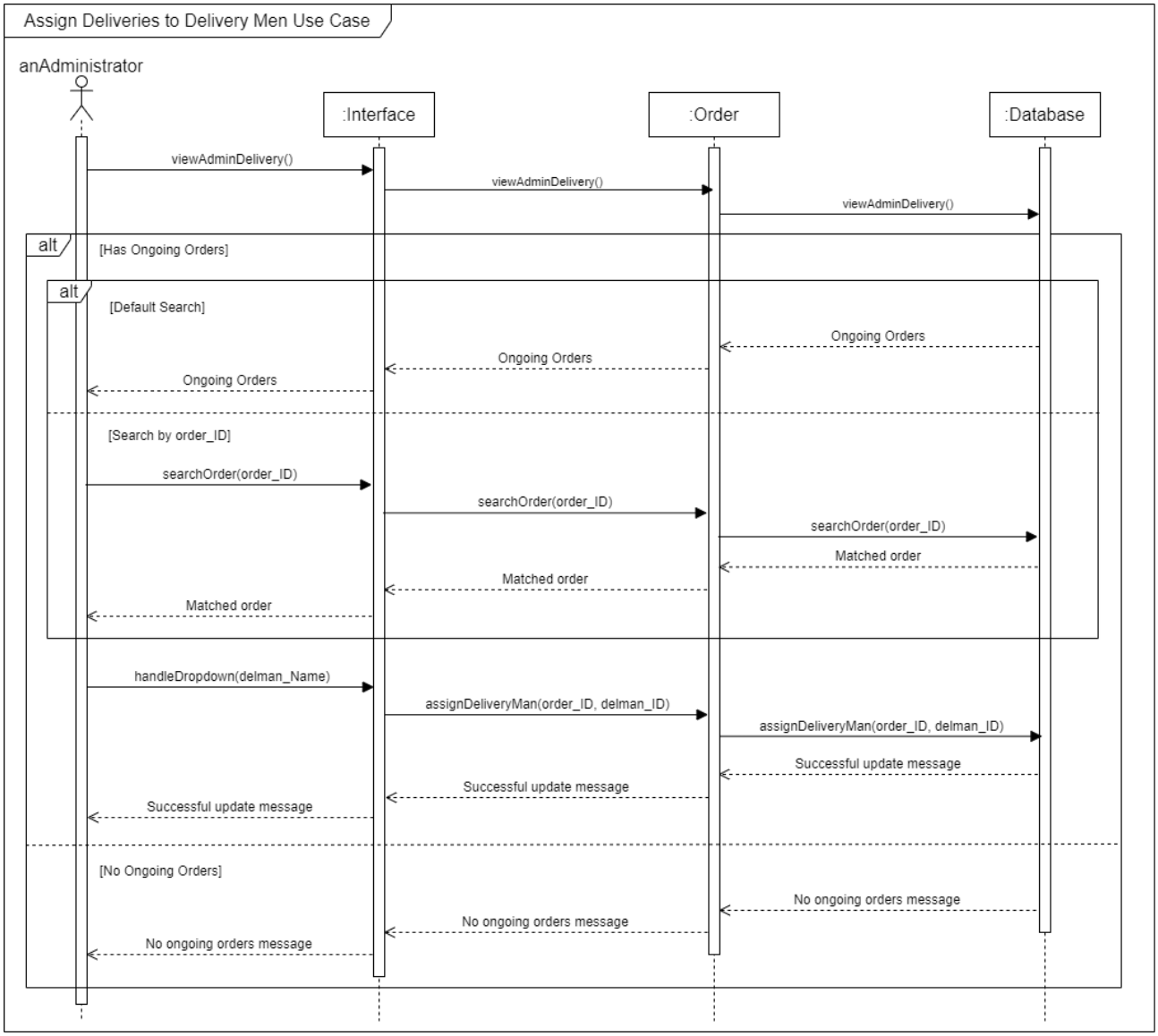



Figure 3.7: Assign Deliveries to Delivery Men Sequence Diagram

## 3.2 Data Dictionary

The User Table Data Dictionary, shown in Table 3.5, is a key component for organizing user information in the Nursery Plant Shopping System. It details essential attributes like user unique identifier(user_ID), which uniquely identifies each user, user email (email), and user password (password) for login credentials, and role-defining booleans (is_customer, is_deliveryman, and is_admin) to distinguish between customers, delivery men, and administrators. The user_ID serves as the primary key, ensuring that each record is unique and easily retrievable.

Table 3.5: User Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| User | user_ID | user ID | int | 1 | 1-9999 | Y | PK | - |
| | email | user email | varchar(100) | eelin@gmail.com | - | Y | - | - |
| | password | user password | varchar(255) | 123456 | - | Y | - | - |
| | is_customer | is customer | boolean | True | - | Y | - | - |
| | is_deliveryman | is deliveryman | boolean | True | - | Y | - | - |
| | is_admin | is administrator | boolean | True | - | Y | - | - |

The Administrator Table Data Dictionary shown as Table 3.6 provides a comprehensive overview of the attributes and specifications for managing administrator records within the database. It includes essential details such as the administrator's unique identifier (admin_ID), and their name (admin_name). These attributes are essential for user authentication and management within the system. The data dictionary also identifies admin_ID as the primary key (PK), serving as a crucial reference point for this table's records.

Table 3.6: Administrator Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| Administrator | admin_ID | administrator ID | int | 90000 | 90000-90005 | Y | PK | - |
| | admin_name | administrator name | varchar(80) | Ee Lin | - | Y | - | - |

The Customer Table Data Dictionary shown as Table 3.7 provides a comprehensive overview of the attributes and specifications for managing customer records within the database. It includes essential details such as the customer's unique identifier (customer_ID), their name (customer_name), address (customer_address), state (customer_state), identification card number (customer_ic), and phone number (customer_phone_number). These attributes are essential for customer account management and interactions within the system. The data dictionary also identifies customer_ID as the primary key (PK) for efficient record referencing within the table.

Table 3.7: Customer Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| Customer | customer_ID | customer ID | int | 91000 | 91000-94999 | Y | PK | - |
| | customer_name | customer name | varchar(80) | Chin Yu Feng | - | Y | - | - |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | customer_address | customer address | varchar(200) | 20, Taman Maju, Section 3/2a, Cheras | - | Y | - | - |
| | customer_state | customer state | varchar(30) | Selangor | - | Y | - | - |
| | customer_ic | customer's identification card number | varchar(30) | 040202016450 | - | Y | - | - |
| | customer_phone_number | customer phone number | bigInt | 60128457865 | - | Y | - | - |

The DeliveryMan Table Data Dictionary shown as Table 3.8 serves as a comprehensive reference guide for managing delivery man records within the database. It contains attributes such as the delivery man's unique identifier (delman_ID), their name (deliveryman_name), address (deliveryman_address), state (deliveryman_state), identification card number (deliveryman_ic), and phone number (deliveryman_phone_number). These attributes are crucial for tracking and managing delivery personnel within the system. The data dictionary also designates delman_ID as the primary key (PK), facilitating efficient record referencing within the table.

Table 3.8: Delivery Man Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| DeliveryMan | delman_ID | delivery man ID | int | 90006 | 90006-90999 | Y | PK | - |
| | deliveryman_name | delivery man name | varchar(80) | Siti Fatimah | - | Y | - | - |
| | deliveryman_address | delivery man address | varchar(200) | 35, Jalan BPU6 Taman Indah, Puchong | - | Y | - | - |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | deliveryman_state | delivery man state | varchar(30) | Selangor | - | Y | - | - |
| | deliveryman_ic | delivery man's identification card number | varchar(30) | 920504018888 | - | Y | - | - |
| | deliveryman_phone_number | delivery man phone number | bigInt | 60147894163 | - | Y | - | - |

The Plant Table Data Dictionary shown as Table 3.9 serves as a comprehensive reference guide for managing plant records within the database. It encompasses key attributes such as the plant's unique identifier (plant_ID), its name (plant_name), image (plant_image), description (plant_description), price (plant_price), and available amount (plant_availability). These attributes are crucial for tracking and managing plant inventory within the system. The "plant_image" attribute stores the file path of the plant image as a string, using varchar(256) and provides an example of the stored value (folder/image1.jpg) for reference. The data dictionary identifies plant_ID as the primary key (PK), facilitating efficient record referencing within the table.

Table 3.9: Plant Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| Plant | plant_ID | plant ID | int | 10000 | 10000-19999 | Y | PK | - |
| | plant_name | plant name | varchar(80) | Aloe Vera | - | Y | - | - |
| | plant_image | plant image | varchar(256) | folder/image1.jpg | - | Y | - | - |
| | plant_description | plant description | varchar(500) | Aloe vera is a succulent plant species of the genus Aloe. The plant is | - | Y | - | - |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | stemless with thick, greenish, fleshy leaves that can fan out from the plant's central stem. The margin of the leaf is serrated with small teeth. | | | | |
| | plant_price | plant price | decimal(5,2) | 10.00 | - | Y | - | - |
| | plant_availability | plant available amount | int | 100 | 1-9999 | Y | - | - |

The Payment Table Data Dictionary shown as Table 3.10 provides an overview for managing payment transactions within the system. It contains attributes such as the payment's unique identifier (payment_ID), the associated customer's identifier (customer_ID), shipping fee (shipping_fee), total cost for the payment (payment_cost), the method for receiving the order (receive_method), the date of order placement (order_date), and the order's total price (order_total_price). These attributes are crucial for tracking and managing payment-related information and order details within the system. The data dictionary designates payment_ID as the primary key (PK), and identifies customer_ID as a foreign key (FK) referencing the "Customer" table, creating a direct link between payments and individual customers.

Table 3.10: Payment Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| Payment | payment_ID | payment ID | int | 60000 | 60000-69999 | Y | PK | - |
| | customer_ID | customer ID | int | 1 | 1-9999 | Y | FK | Customer |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | shipping_fee | shipping fee | decimal(4,2) | 5.00 | 0.00-99.99 | Y | - | - |
| | payment_cost | total cost for payment | decimal(6,2) | 200.00 | 0.00-9999.99 | Y | - | - |
| | receive_method | method to receive order | varchar(10) | Delivery | - | Y | - | - |
| | order_date | date of order placement | date | YYYY-MM-DD | - | Y | - | - |
| | order_total_price | order total price | decimal(6,2) | 200.00 | 0.00-9999.99 | Y | - | - |

The Order Table Data Dictionary shown as Table 3.11 serves as a reference guide for managing customer orders within the system. It encompasses essential attributes such as the order's unique identifier (order_ID), the associated customer's identifier (customer_ID), payment identifier (payment_ID), delivery man identifier (delman_ID), and the order status (order_status). These attributes are fundamental for tracking and managing customer orders and related order details. The data dictionary designates order_ID as the primary key (PK), and identifies customer_ID as a foreign key (FK) referencing the “Customer” table, creating a direct link between orders and individual customers. Similarly, payment_ID and delman_ID are foreign keys (FK) referencing the “Payment” and “DeliveryMan” tables, respectively, establishing connections with payment and delivery information.

Table 3.11: Order Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| Order | order_ID | order ID | int | 70000 | 70000-79999 | Y | PK | - |
| | customer_ID | customer ID | int | 1 | 1-9999 | Y | FK | Customer |
| | payment_ID | payment ID | int | 60000 | 60000-69999 | Y | FK | Payment |

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | delman_ID | delivery man ID | int | 90006 | 90006-90999 | Y | FK | DeliveryMan |
| | order_status | order status | varchar(30) | Ready | - | Y | - | - |

The Order Item Table Data Dictionary shown as Table 3.12 gives an overview for managing individual items within customer orders in the system. It contains attributes such as the order item's unique identifier (order_Item_ID), the associated order's identifier (order_ID), the plant's identifier (plant_ID), the quantity of the plant in the order (plant_quantity), the price of the order item (order_item_price), and whether it needs to be reviewed (to_review). These attributes are used for tracking and managing detailed information about items within customer orders. The data dictionary designates order_Item_ID as the primary key (PK), and identifies order_ID as a foreign key (FK) referencing the “Order” table, establishing a direct link between order items and their respective orders. Similarly, the “plant_ID” attribute is a foreign key (FK) referencing the “Plant” table, connecting each order item to a specific plant in the system.

Table 3.12: Order Item Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| OrderItem | order_item_ID | order Item ID | int | 80000 | 80000-89999 | Y | PK | - |
| | order_ID | order ID | int | 70000 | 70000-79999 | Y | FK | Order |
| | plant_ID | plant ID | int | 10000 | 10000-19999 | Y | FK | Plant |
| | plant_quantity | plant Quantity | int | 5 | 1-50 | Y | - | - |
| | order_item_price | order item price | decimal(5,2) | 100.00 | 0.00-999.99 | Y | - | - |
| | to_review | to review | boolean | True | - | Y | - | - |

The Review Table Data Dictionary shown as Table 3.13 gives an overview for managing customer reviews within the system. It contains attributes such as the review's unique identifier (review_ID), the identifier of the ordered item (order_item_ID), the rating provided by the customer (rating), and any comments or feedback left in the review (comment). These attributes are crucial for tracking and managing customer feedback and their opinions about specific orders and items. The data dictionary designates review_ID as the primary key (PK), and identifies order_Item_ID as a foreign key (FK) referencing the “OrderItem” table, creating a direct connection between reviews and specific order items.

Table 3.13: Review Table Data Dictionary

| Table Name | Attribute Name | Contents | Type | Format | Range | Required | PK/ FK | FK Referenced Table |
|---|---|---|---|---|---|---|---|---|
| Review | review_ID | review ID | int | 91000 | 95000-99999 | Y | PK | - |
| | order_item_ID | order Item ID | int | 80000 | 80000-89999 | Y | FK | OrderItem |
| | rating | plant Rating | int | 5 | 1-5 | Y | - | - |
| | comment | comment | varchar(500) | Receive in good condition. Looks very beautiful | - | Y | - | - |

## 3.3 Data Structures

### 3.3.1 User Array

Table 3.14 below shows User Array Data Structures. It serves as an important element for managing essential user information within the system. The User ID, represented as an integer, serves as a unique identifier for each user, ensuring accurate user tracking and management. The User Email, stored in a varchar(100), captures the email addresses of users, which is crucial for account validation. User Password, which is a varchar(255), responsible for securely storing hashed passwords used for authentication and access control. Additionally, Is Customer, which is a boolean value, indicates whether the user holds the role of a customer. Is Deliveryman, also a boolean value, identifies users who are designated as delivery man. Is Administrator, another boolean attribute, determines if the user has administrative privileges.

Table 3.14: User Array Data Structures

| Attribute | Data Type |
|---|---|
| User ID | int |
| User email | varchar(100) |
| User password | varchar(255) |
| Is customer | boolean |
| Is deliveryman | boolean |
| Is administrator | boolean |

### 3.3.2 Administrator Array

Table 3.15 below shows Administrator Array Data Structures. It serves as a fundamental element for managing administrator information within a system. The Administrator ID, represented as an integer, serves as a primary unique identifier for each administrator, facilitating precise user identification and access control. The Administrator Name, stored as a varchar(80), allows for the storage of administrator names, ensuring user-friendly recognition and presentation.

Table 3.15: Administrator Array Data Structures

| Attribute | Data Type |
|---|---|
| Administrator ID | int |
| Administrator Name | varchar(80) |

### 3.3.3 Customer Array

Table 3.16 below shows Customer Array Data Structures. The Customer array serves as a fundamental data structure for managing crucial customer information within a system. The Customer ID, represented as an integer, acts as a unique identifier for each customer. Customer Name, a varchar(80), stores customer names, enhancing user recognition and presentation. Customer Address and Customer State, both varchars (200) and (30) respectively, record customer contact details and location information for order delivery. The Customer's Identification Card Number, a varchar(30), stores unique identification data, and the bigInt data type is used for the Customer Phone Number, enabling the storage of contact numbers.

Table 3.16: Customer Array Data Structures

| Attribute | Data Type |
|---|---|
| Customer ID | int |
| Customer Name | varchar(80) |
| Customer address | varchar(200) |
| Customer state | varchar(30) |
| Customer’s identification card number | varchar(30) |
| Customer phone number | bigInt |

### 3.3.4 DeliveryMan Array

Table 3.17 presents the Delivery Man Array Data Structures. The Delivery Man array is used for efficient management of essential data regarding delivery personnel within the system. The Delivery Man ID, represented as an integer, assigns a distinctive identifier to each delivery man. Delivery Man Name, as a varchar(80), facilitates easy user identification and improves the presentation of their profiles. Moreover, Delivery Man Address and Delivery Man State, both varchars (200) and (30) respectively, collect contact details and location information, enriching the understanding of delivery personnel's operational context. The Delivery Man’s Identification Card Number, a varchar(30), stores unique identification data crucial for comprehensive identity verification. Additionally, the use of the bigInt data type for the Delivery Man Phone Number accommodates the storage of contact information, ensuring efficient communication within the delivery team.

Table 3.17: Delivery Man Array Data Structures

| Attribute | Data Type |
|---|---|
| Delivery man ID | int |
| Delivery man name | varchar(80) |
| Delivery man address | varchar(200) |

| | |
|---|---|
| Delivery man state | varchar(30) |
| Delivery man's identification card number | varchar(30) |
| Delivery man phone number | bigInt |

### 3.3.5 Plant Array

Table 3.18 presents the Plant Array Data Structures. The Plant array serves as a fundamental data structure for organizing and managing essential information about plants within the system. The Plant ID, represented as an integer, provides a unique identifier for each plant, enabling precise tracking and reference. Plant Name, a varchar(80), offers a recognizable label for each plant, simplifying user identification. Plant Image, stored as a varchar(256), visually represents the plant through images. The Plant Description, with a varchar(500), provides valuable insights into each plant's characteristics, aiding users in making informed decisions. The Plant Price, formatted as a decimal(5,2), denotes the cost of each plant. Plant Available Amount, an integer, tracks the current quantity of each plant in stock, enabling inventory control.

Table 3.18: Plant Array Data Structures

| Attribute | Data Type |
|---|---|
| Plant ID | int |
| Plant name | varchar(80) |
| Plant image | varchar(256) |
| Plant description | varchar(500) |
| Plant price | decimal(5,2) |
| Plant available amount | int |

### 3.3.6 Payment Array

Table 3.19 presents the Payment Array Data Structures. The Payment array serves as a vital data structure for managing payment transactions within the system. Payment ID, an integer, assigns a unique identifier to each payment, ensuring precise tracking and reference. Customer ID, also an integer, links each payment to a specific customer, facilitating accurate association with corresponding customers. Shipping Fee, formatted as a decimal(4,2), captures the cost of shipping associated with the payment. Total Cost for Payment, a decimal(6,2), represents the overall cost encompassing the order and shipping fees. Method to Receive Order, a varchar(10), records the chosen method for receiving the order. Date of Order Placement, in date format, denotes the precise date when the order was placed, aiding in order scheduling. Order Total Price, formatted as a decimal(6,2), represents the total cost of the order, including products and excluding shipping fee.

Table 3.19: Payment Array Data Structures

| Attribute | Data Type |
|---|---|
| Payment ID | int |
| Customer ID | int |
| Shipping fee | decimal(4,2) |
| Total cost for payment | decimal(6,2) |
| Method to receive order | varchar(10) |
| Date of order placement | date |
| Order total price | decimal(6,2) |

### 3.3.7 Order Array

Table 3.20 presents the Order Array Data Structures. The Order array serves as a pivotal data structure for managing customer orders within the system. Order ID, represented as an integer, assigns a unique identifier to each order, facilitating precise tracking and reference. Customer ID, also an integer, links each order to a specific customer, ensuring accurate association with individual users. Payment ID, an integer, connects each order to a corresponding payment. Delivery Man ID, an integer, associates orders with specific delivery personnel. Order Status, a varchar(30), captures the current status of each order, which can be “waiting”, “ready”, “out of delivery” or “completed”.

Table 3.20: Order Array Data Structures

| Attribute | Data Type |
|---|---|
| Order ID | int |
| Customer ID | int |
| Payment ID | int |
| Delivery man ID | int |
| Order status | varchar(30) |

### 3.3.8 OrderItem Array

Table 3.21 presents the Order Item Array Data Structures. The Order Item array serves as a foundational data structure for efficiently managing individual items within customer orders in the system. Order Item ID, represented as an integer, assigns a unique identifier to each order item. Order ID, also an integer, links each order item to a specific order, facilitating accurate association with corresponding orders. Plant ID, an integer, records the unique identifier of the plant included in the order. Plant Quantity, also an integer, captures the quantity of the plant in the order, facilitating inventory management. Order Item Price, formatted as a decimal(5,2), represents the cost of each

order item, enabling accurate pricing calculations. To Review, a boolean, indicates whether the order item requires review, streamlining customer review management within the system.

Table 3.21: Order Item Array Data Structures

| Attribute | Data Type |
|---|---|
| Order Item ID | int |
| Order ID | int |
| Plant ID | int |
| Plant Quantity | int |
| Order item price | decimal(5,2) |
| To review | boolean |

### 3.3.9 Review Array

Table 3.22 presents the Review Item Array Data Structures. The Review array serves as a comprehensive data structure for managing customer feedback and reviews within the system. Review ID, represented as an integer, assigns a unique identifier to each review. Order Item ID, also an integer, associates reviews with specific order items, enabling detailed feedback on purchased products. Plant Rating, an integer ranging from 1 to 5, with 1 indicating the lowest rating and 5 denoting the highest rating, allowing customers to rate the plant's quality. Comment, a varchar(500), allows customers to provide additional feedback about their experience, facilitating in-depth insights for product improvement and customer satisfaction within the system.

Table 3.22: Review Array Data Structures

| Attribute | Data Type |
|---|---|
| Review ID | int |
| Order Item ID | int |
| Plant Rating | int |
| Comment | varchar(500) |

## 3.4 Subsystem Architecture

Figure 3.8 below is the Component-based Architecture Diagram for the Administrator Subsystem. Figure 3.8 illustrates the specific functionalities available to the administrative role within the system. This includes the Order Management component, Plant Management component and Delivery Management component.

In Order Management component, administrator can access two sub-components “View Order Details” and “Update Order Status” associated with it. These functions allow the administrator to view the details of customer orders and update the order statuses as needed.

Moving to the Plant Management component, the administrator is given comprehensive control over the plant inventory with a suite of sub-components. These include “Create New Plant”, which facilitates the addition of new plant to the system; “Search Plant by Plant Name”, offering a targeted search capability; “Update Plant”, allowing for adjustments to existing plant details; and “Delete Plant”, which provides the option to take plants off the list.

Lastly, in Delivery Management component, administrator can access the sub-component “Assign Deliveries” associated with it. This function allows the administrator to assign deliveries to delivery men, optimizing the distribution process.

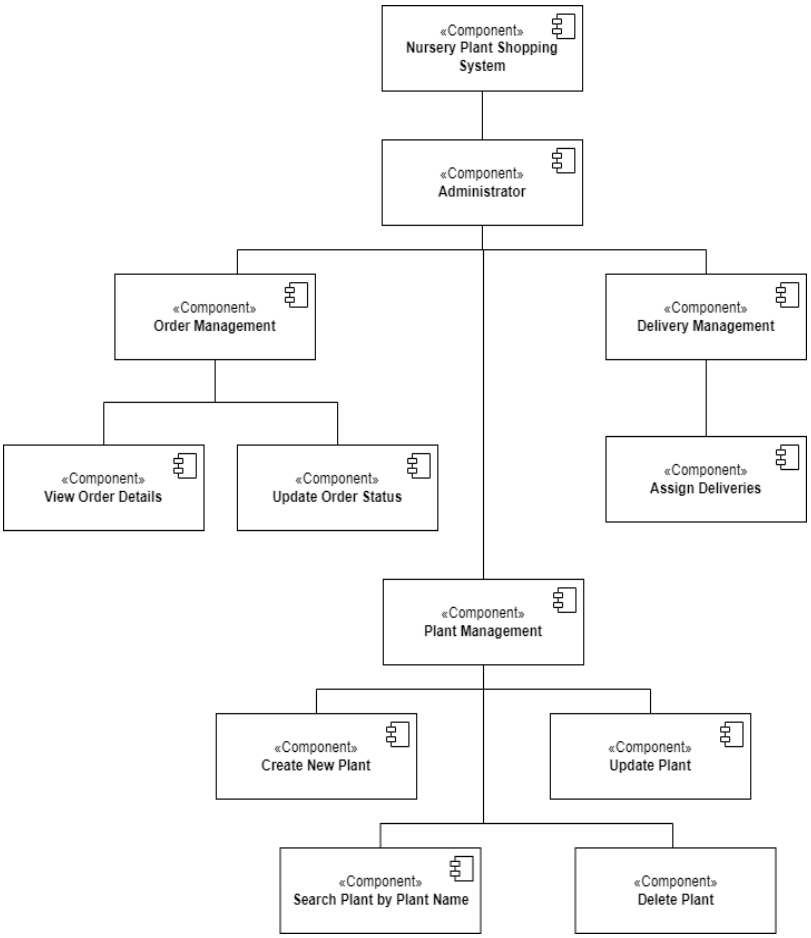



Figure 3.8: Component-based Architecture Diagram for Administrator Subsystem

## 3.5 Subsystem Screens

### 3.5.1 Administrator Home Screen

Figure 3.9 below shows the “Administrator Home Screen”. The “Administrator Home Screen” features a welcoming message at the center of the screen, saying “Welcome to Admin Dashboard”, so administrator know he/she is in the right place. There is a navigation bar at the top of the Administrator Home Screen that neatly organizes essential management functions into tabs. These tabs include “Plant Management” for managing the plants, “Order Management” for tracking and updating the status of customer orders, “Delivery Management” for assigning delivery orders to delivery men, and “Logout” for administrator to safely exit the system and log out of his/her accounts.

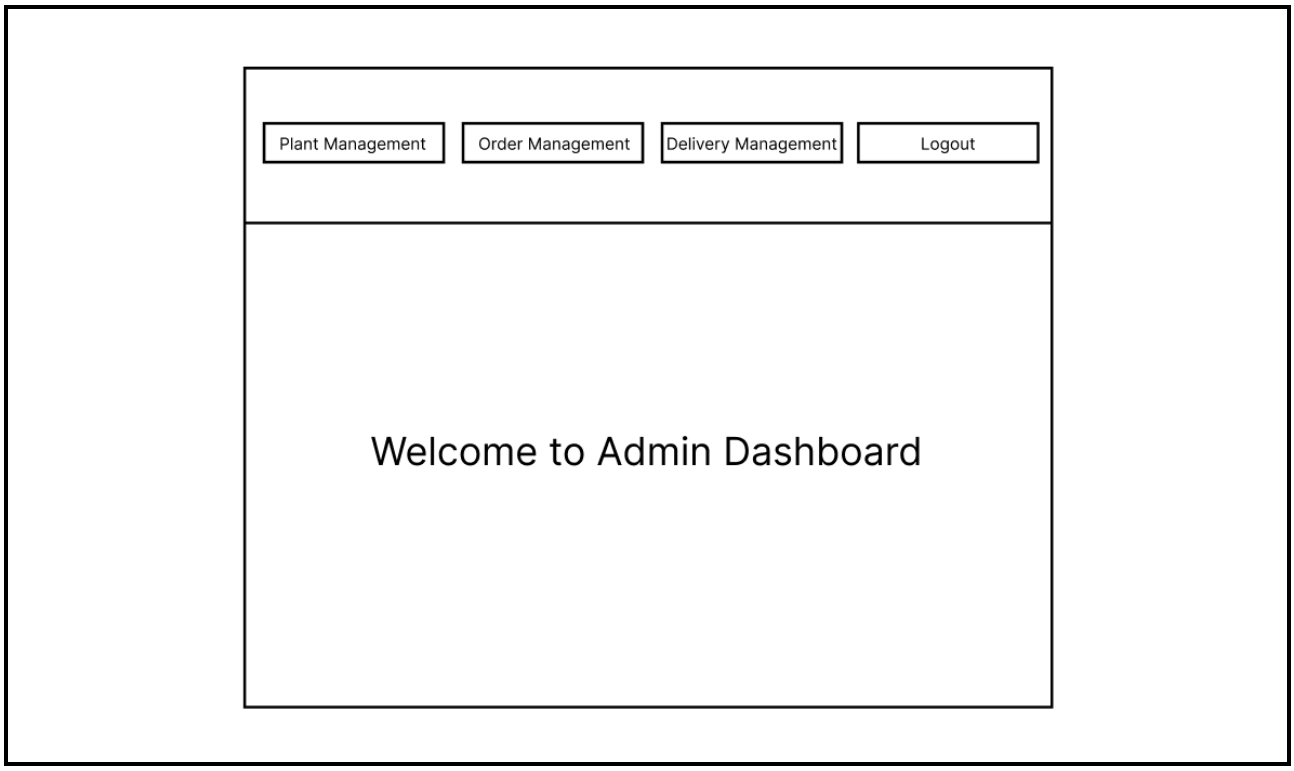



Figure 3.9: Administrator Home Screen

### 3.5.2 Plant Management Screen

Figure 3.10 below shows the “Plant Management Screen”. The “Plant Management Screen” provides an efficient interface for administrator to manage the plants. It shows all the plants with their detailed information. For example, the plant name, plant ID, plant image, descriptive summary, plant price and plant quantity. There is a search bar at the top which allows for quick filtering by plant name. Besides, the “Create New Plant” button allows the administrator to add new plants easily. Under each plant entry, the administrator can also edit the plant information or remove the plant from the plant list. Moreover, customer reviews are visible at the bottom of each plant's information, which aids the administrator to see how satisfied customers are with the plants and how good the plant quality is.

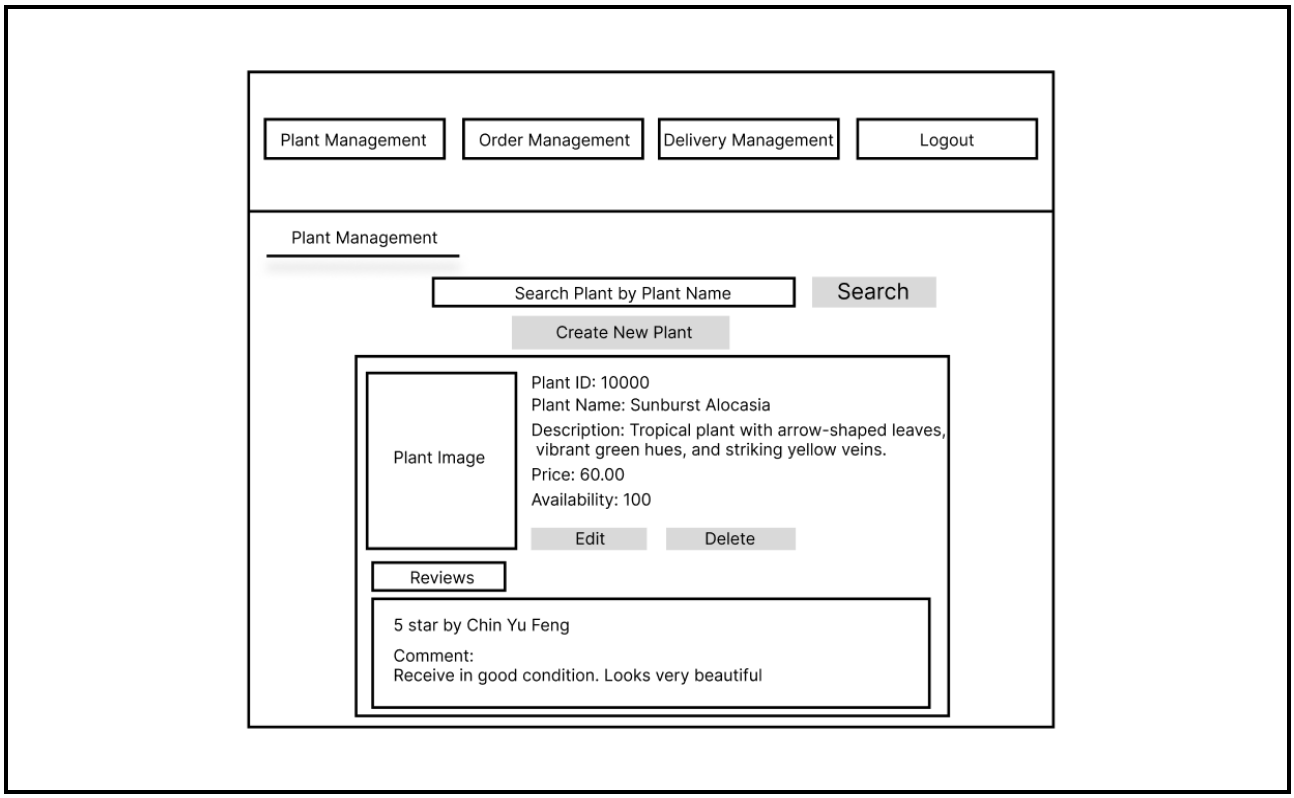



Figure 3.10: Plant Management Screen

### 3.5.3 Create New Plant Screen

Figure 3.11 illustrates the “Create New Plant Screen”, which makes the addition of new plant entries into the plant inventory easy. After clicking the “Create New Plant” button, the administrator is presented with a structured form to input vital information about a new plant, such as its name, detailed descriptions, quantity available for sale, and price. The administrator can also upload a picture of the plant to make the plant information more accurate and eye-catching. Once the details are filled in, the administrator can click the “Confirm” button to submit the form or click the “Back” button to cancel the addition of new plant and return to “Plant Management Screen”.

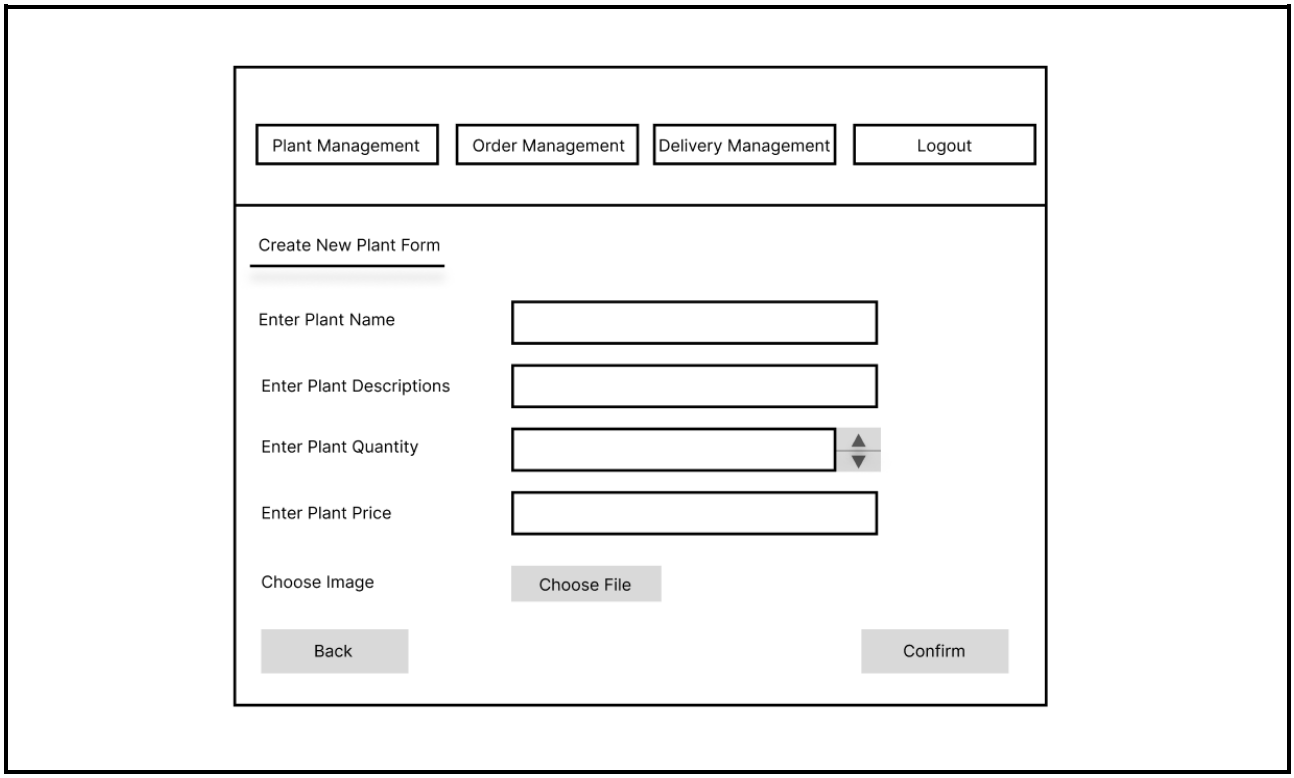



Figure 3.11: Create New Plant Screen

### 3.5.4 Update Plant Screen

Figure 3.12 displays the “Update Plant Screen”, which allows the administrator to modify the details of existing plants. After clicking the “Edit” button of a particular plant, the screen pre-populates with the plant’s current details, including its name, description, quantity, price, and the image of the plant. Administrator can update any piece of information, whether it is the plant’s name, descriptive text, quantity in stock, plant price or the plant image. Upon making the desired changes, the administrator can save the updates by clicking the “Update” button or choose to go back to “Plant Management Screen” without saving changes using the “Back” button.

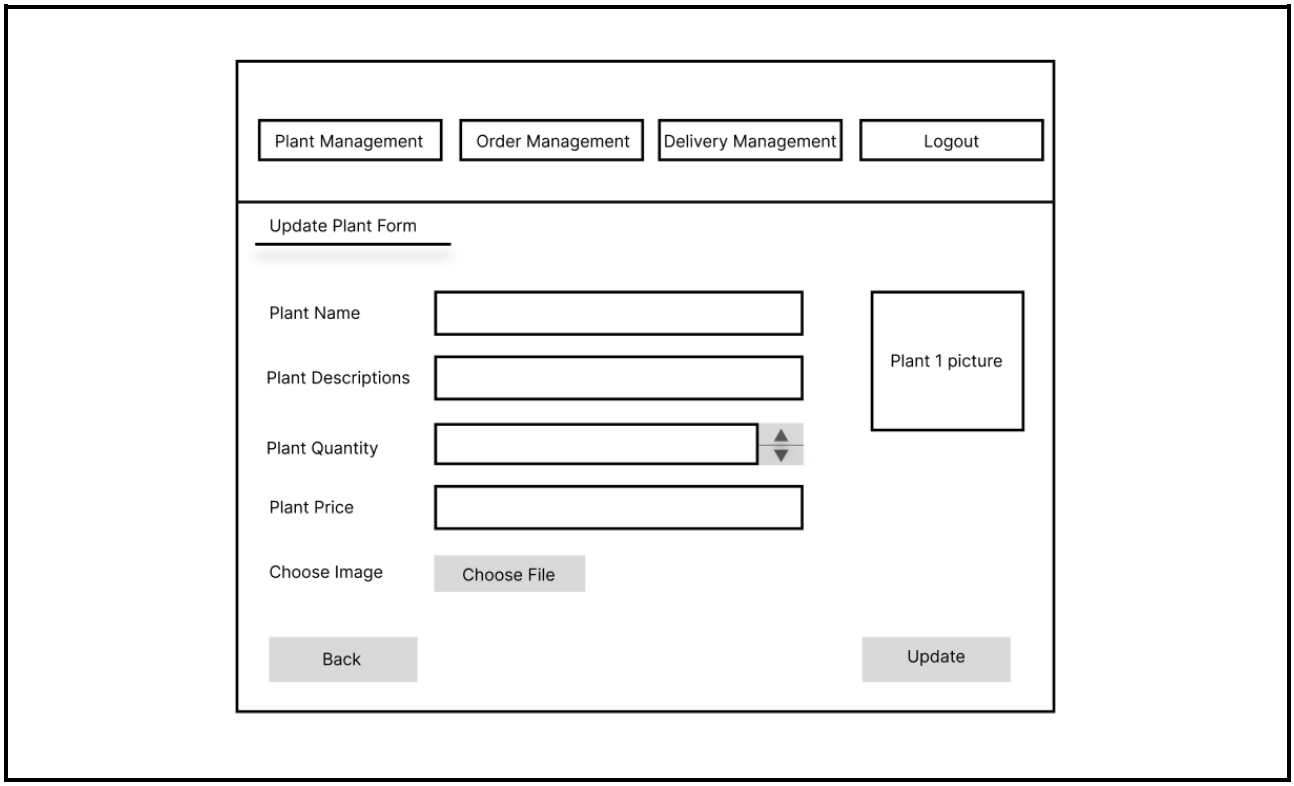



Figure 3.12: Update Plant Screen

### 3.5.5 Order Management Screen

Figure 3.13 shows the “Order Management Screen” which helps the administrator to monitor and process customer orders easily. The screen presents each order separately in boxes with key details like order ID, customer’s full name, email, address, and phone number. The number of plants ordered and the total price are also displayed, along with the order date and receive method. The order status is shown via dropdown menu with options such as “Waiting”, “Ready”, or “Completed”. In this case, administrator can update the order status with the “Update Status” button. Moreover, a search bar at the top allows for quick lookup of orders using the order ID, enhancing the administrator's efficiency in tracking and managing orders.

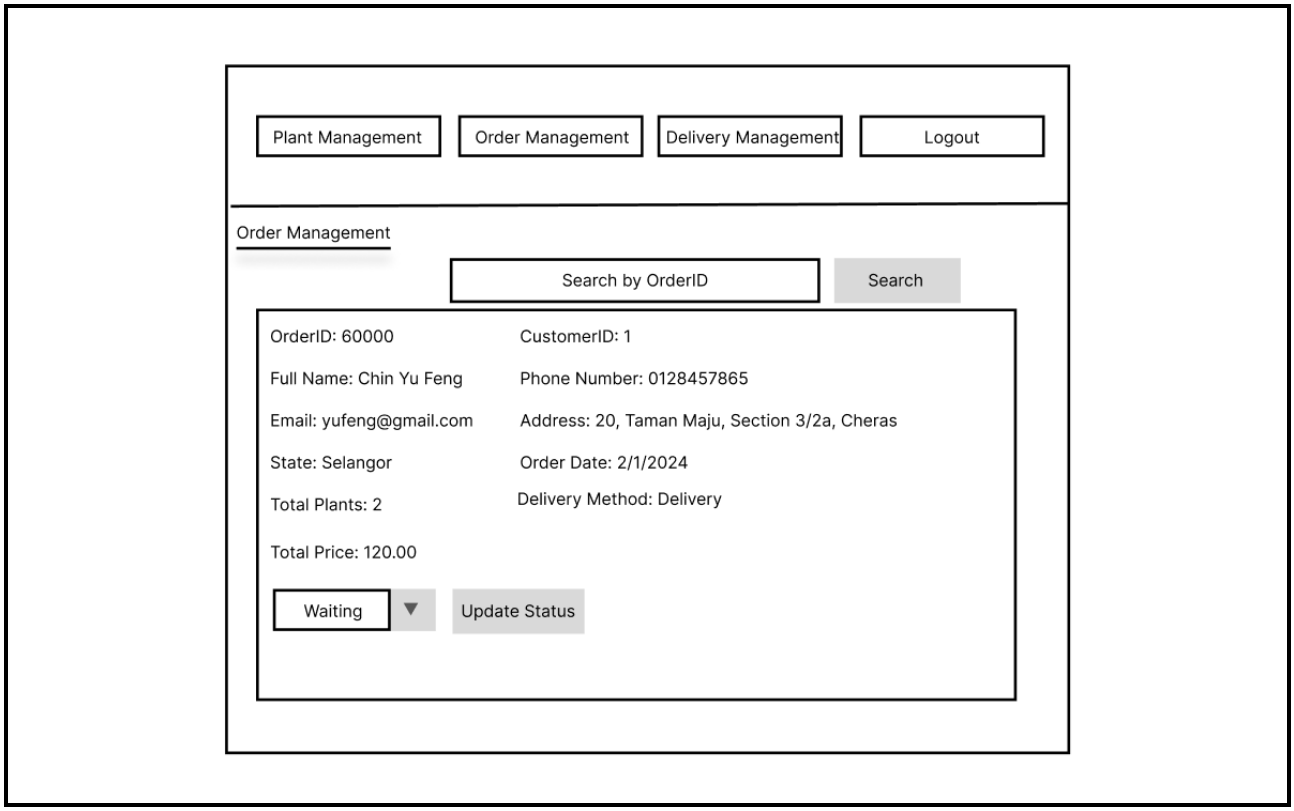



Figure 3.13: Order Management Screen

### 3.5.6 Delivery Management Screen

Figure 3.14 shows the “Delivery Management Screen” which helps the administrator to assign delivery orders easily. The screen presents each delivery order separately in boxes with key details like order ID, customer’s full name, email, address, and phone number. The number of plants ordered and the total price are also displayed, along with the order date and receive method. Each delivery order box includes an “Assign” button, paired with a dropdown menu. The administrator can select a delivery man from the dropdown menu and click on the “Assign” button to assign the delivery order to that particular delivery man. There is also a search bar at the top, which allows for quick lookup of delivery orders using the order ID.

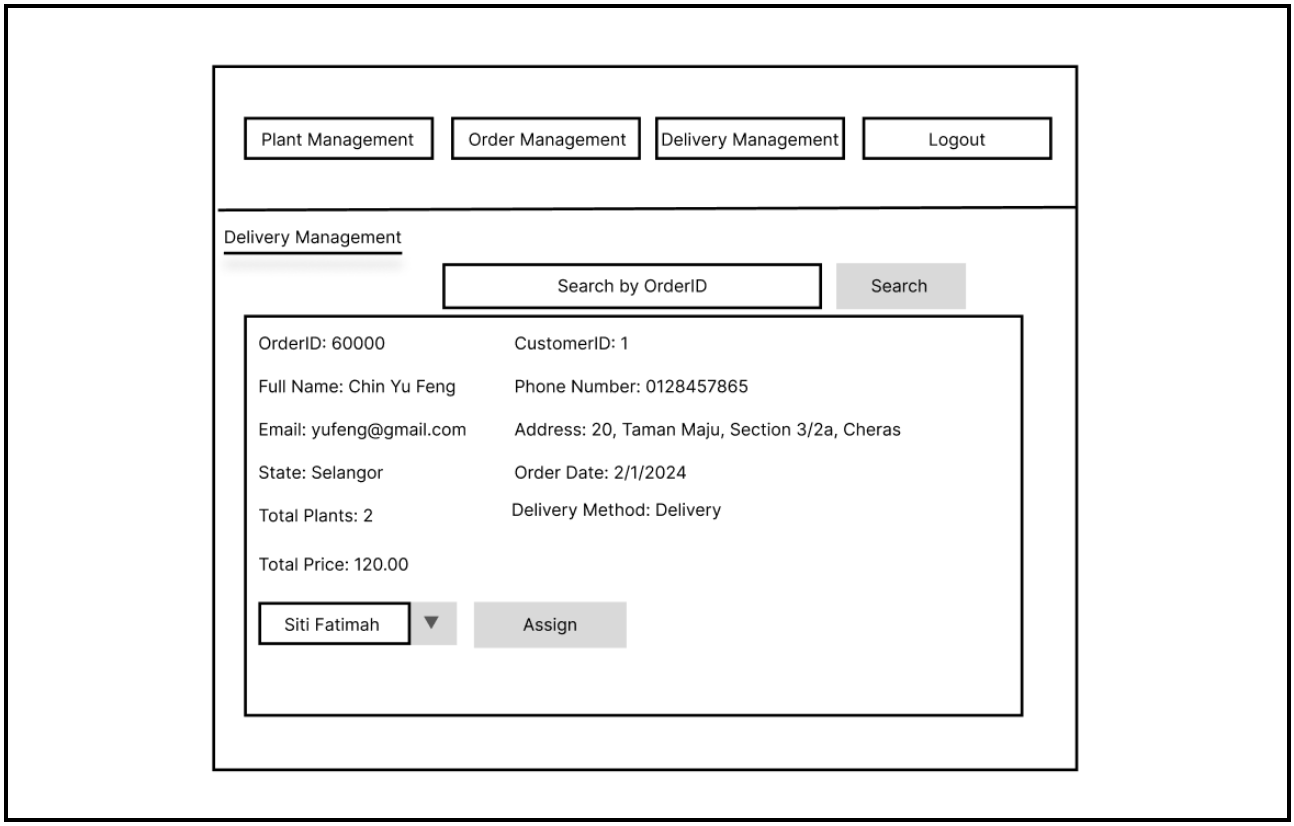



Figure 3.14: Delivery Management Screen

## 3.6 Subsystem Components

### 3.6.1 An administrator manages plants

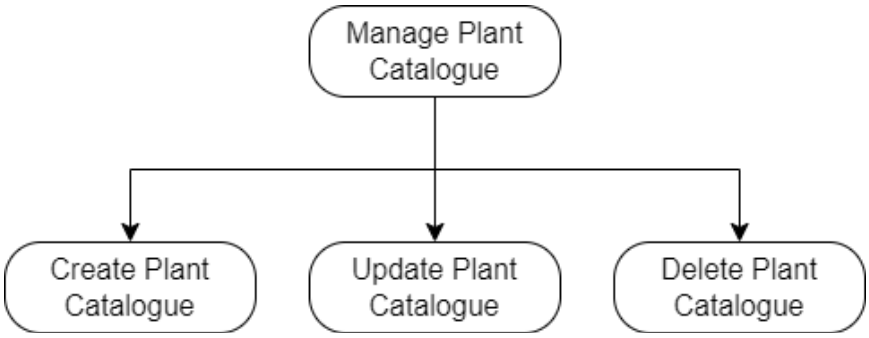



Figure 3.15: Sub-use cases for Manage Plants Use Case

#### 3.6.1.1 An administrator creates a plant

Figure 3.16 shows the Create Plant Activity Diagram. The administrator begins by clicking on the “Plant Management” section, leading the system to display a list of existing plants. If the administrator decides to add a new plant, they click on the “Create New Plant” button, and the system presents a form to input plant details like name, descriptions, quantity, and pricing. After filling in the details and selecting images for upload, the administrator confirms the entry. The system then validates the new plant's name to ensure it does not already exist in the database. If validation passes, the system creates the new plant, assigns it a unique ID, and displays a success message. The list of plants is then updated to include the new addition. Alternatively, if the administrator clicks on the “Back” button at the create new plant form screen, the system will return to the list of existing plants. If the validation fails (due to a duplicate name), then the system will display an error message.

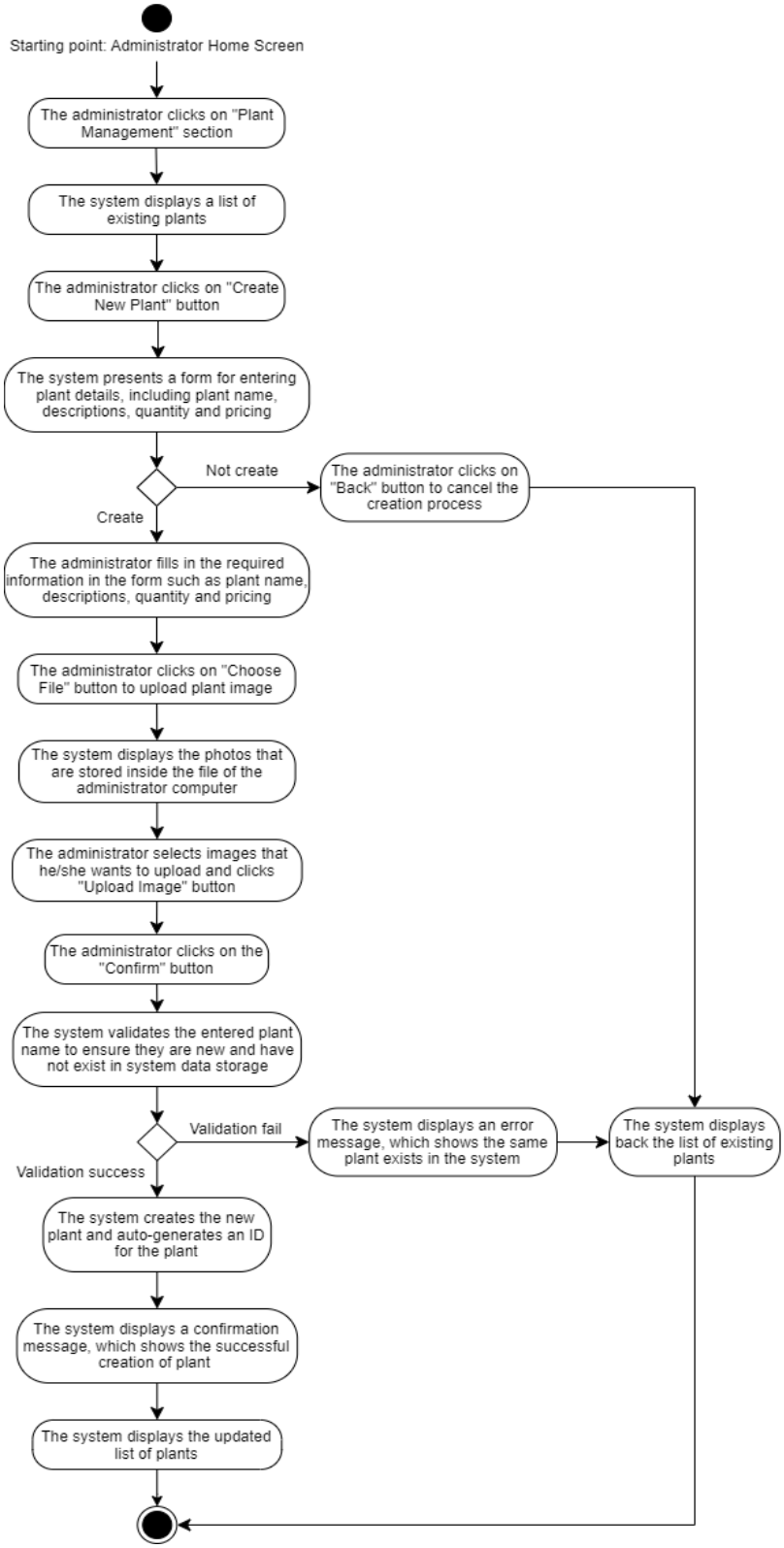



Figure 3.16: Create Plant Activity Diagram

### 3.6.1.2 An administrator updates a plant

Figure 3.17 shows the Update Plant Activity Diagram. In this process, the administrator starts by accessing the “Plant Management” section, prompting the system to display a list of existing plants. To update a plant, the administrator clicks the “Edit” button on a specific plant, leading the system to present the plant's detailed information, such as name, image, descriptions, quantity, and pricing. The administrator then makes the desired changes to the plant information and confirms the update by clicking the “Update” button. Subsequently, the system displays a confirmation message regarding the successful update and returns the revised list of plants. In case the administrator decides not to proceed with the update at the edit stage, clicking the “Back” button will revert the system to the list of existing plants. Additionally, if the administrator wants to update the plant's image, they can click the “Choose File” button and upload a new image before resuming the update process.

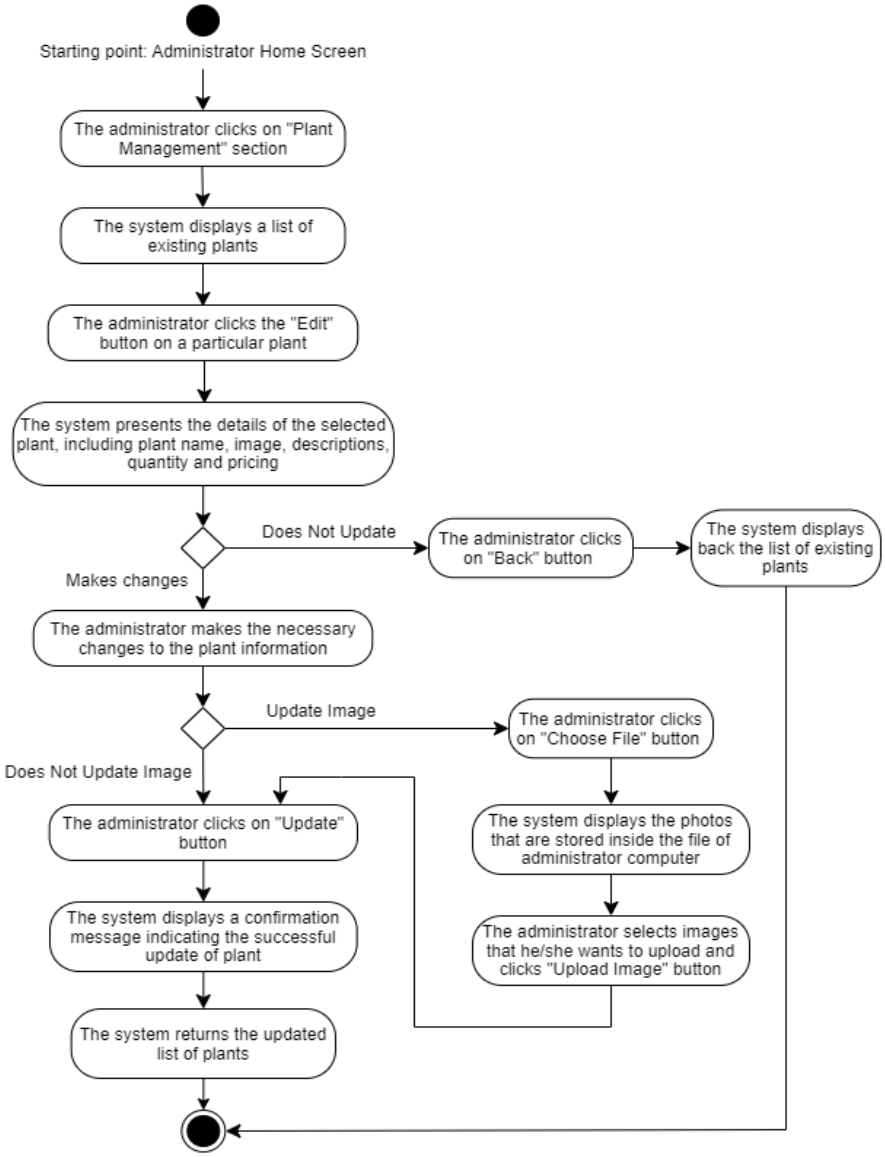



Figure 3.17: Update Plant Activity Diagram

### 3.6.1.3 An administrator deletes a plant

Figure 3.18 shows the Delete Plant Activity Diagram. Starting from the Administrator Home Screen, the admin navigates to the “Plant Management” section, where they are presented with a list of existing plants. Upon selecting the “Delete” button for a specific plant, the system prompts a confirmation message. If the administrator confirms the deletion, the system proceeds to remove the plant from the database and updates the list to reflect this change. The system will then display a confirmation message indicating the successful deletion of the plant. Then, the system will display the updated plant list. Alternatively, if the administrator decides against deletion and clicks “Cancel”, the plant list remains unchanged.

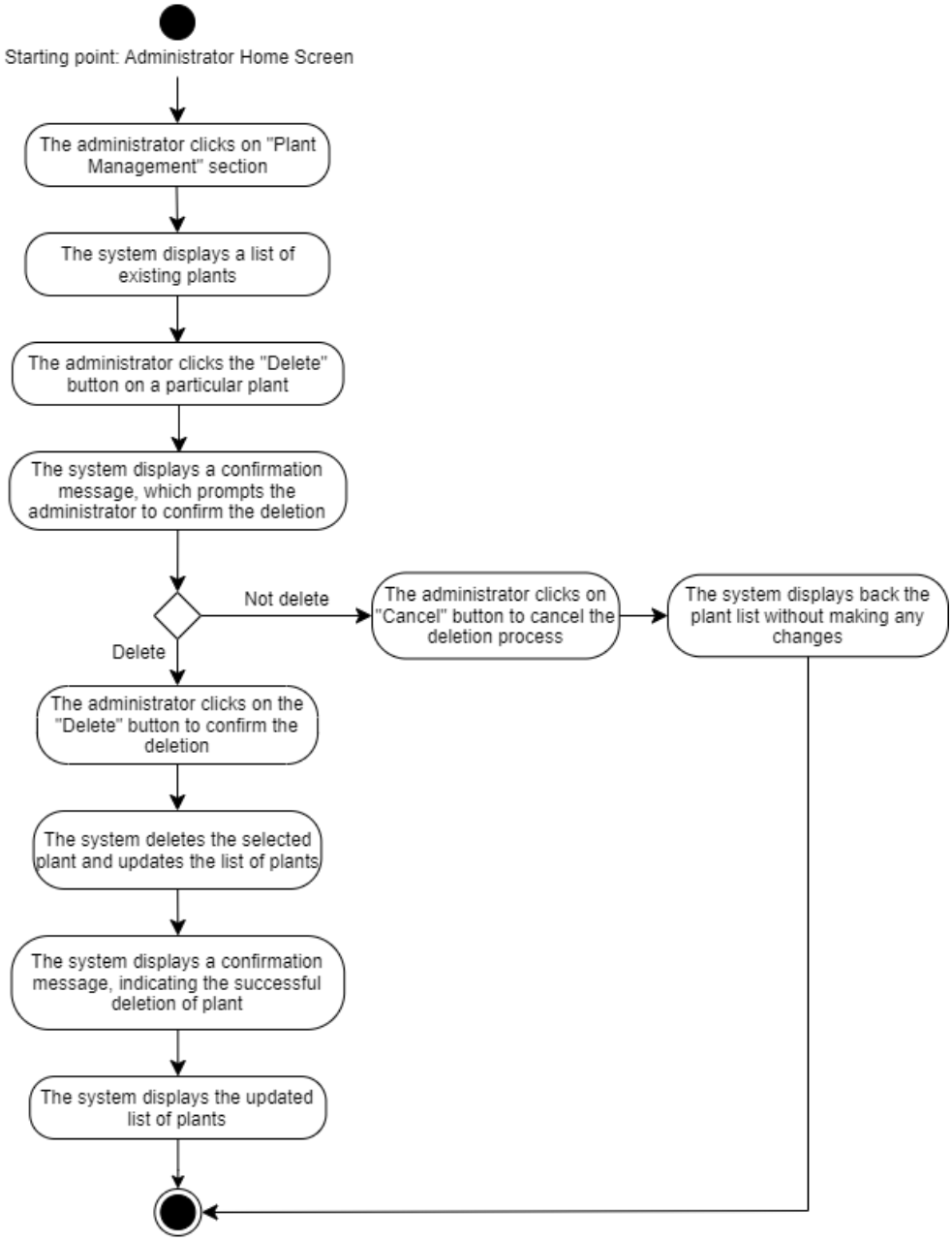



Figure 3.18: Delete Plant Activity Diagram

### 3.6.2 An administrator searches plant

Figure 3.19 presents the Search Plant Activity Diagram, which illustrates the steps an administrator takes to search for a plant within the TLET Nursery Plant Shopping System. The process begins from the Administrator Home Screen, where the administrator selects the “Plant Management” section. The system then displays a list of existing plants. The administrator can search for a specific plant by typing its name into the search bar and clicking the “Search” button. The system searches the plant list and, if found, displays the results matching the search criteria. If no matching plants are found, the system notifies the administrator that there are no plants matching the entered name, allowing them to make necessary adjustments to their search or review the list of existing plants again.

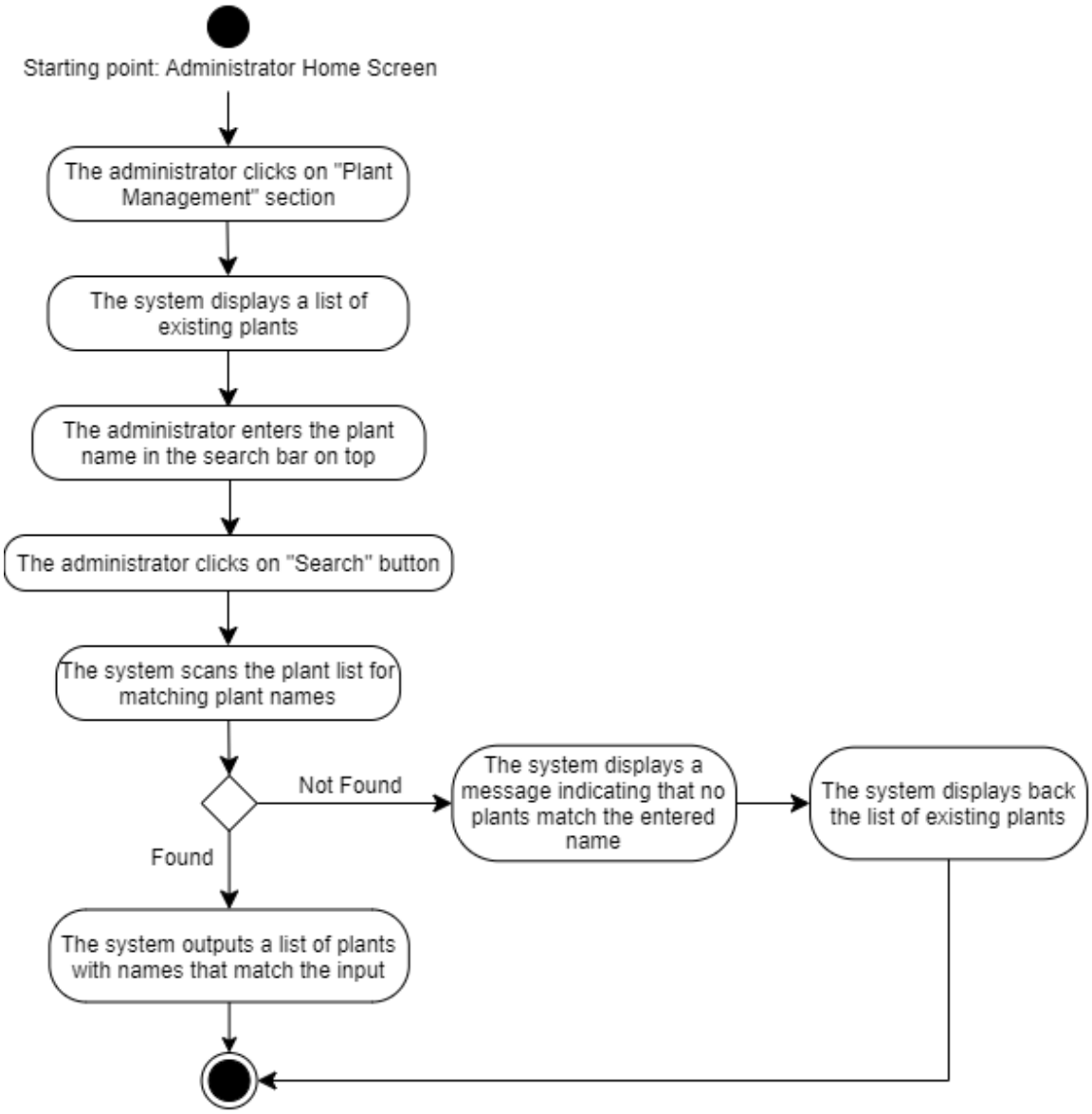



Figure 3.19: Search Plant Activity Diagram

### 3.6.3 An administrator views order details and updates order status

Figure 3.20 shows the View Orders and Update Order Status Activity Diagram. Starting from the Administrator Home Screen, the administrator accesses the “Order Management” section to view a list of ongoing orders. If there are no ongoing orders, the system will display a message stating that there are no ongoing orders in the system. Otherwise, detailed information on the ongoing orders is presented, including customer and order details, which enables the administrator to update the status of an order to either ‘Ready’ or ‘Completed’. Orders marked as ‘Ready’ indicate that they are prepared for customer pickup or will be assigned to a delivery man for further processing, whereas orders marked as ‘Completed’ indicate that customers have received the plants successfully. The system allows searching for orders by order ID, and if no matching orders are found, the administrator can refine the search. Successful updates are immediately reflected in the system's database, ensuring accurate order management.

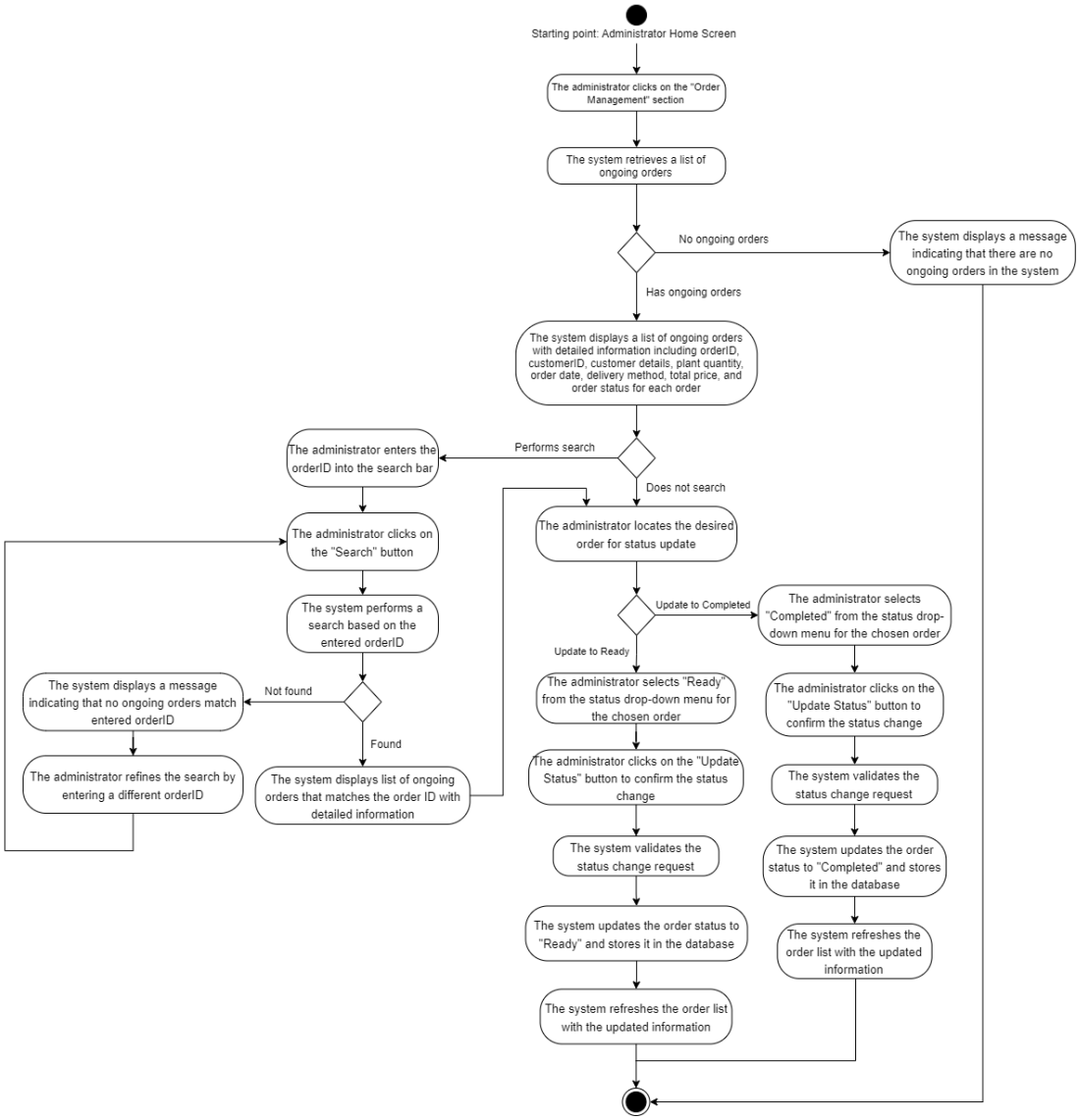



Figure 3.20: View Orders and Update Order Status Activity Diagram

### 3.6.4 An administrator assigns deliveries

Figure 3.21 illustrates the Assign Deliveries to Delivery Men Activity Diagram within the TLET Nursery Plant Shopping System. The process begins with the administrator accessing the “Delivery Management” section from the Administrator Home Screen. The system presents a list of ongoing orders, each with comprehensive details such as order ID, customer information, and delivery particulars. If there are no orders, a message indicates no ongoing orders in the system; otherwise, the administrator can search for specific orders by order ID to update the “Assign To” status. Once a delivery man is selected from the drop-down menu for a particular order, the administrator assigns the delivery, and the system updates the status. The updated list of assigned deliveries is then displayed.

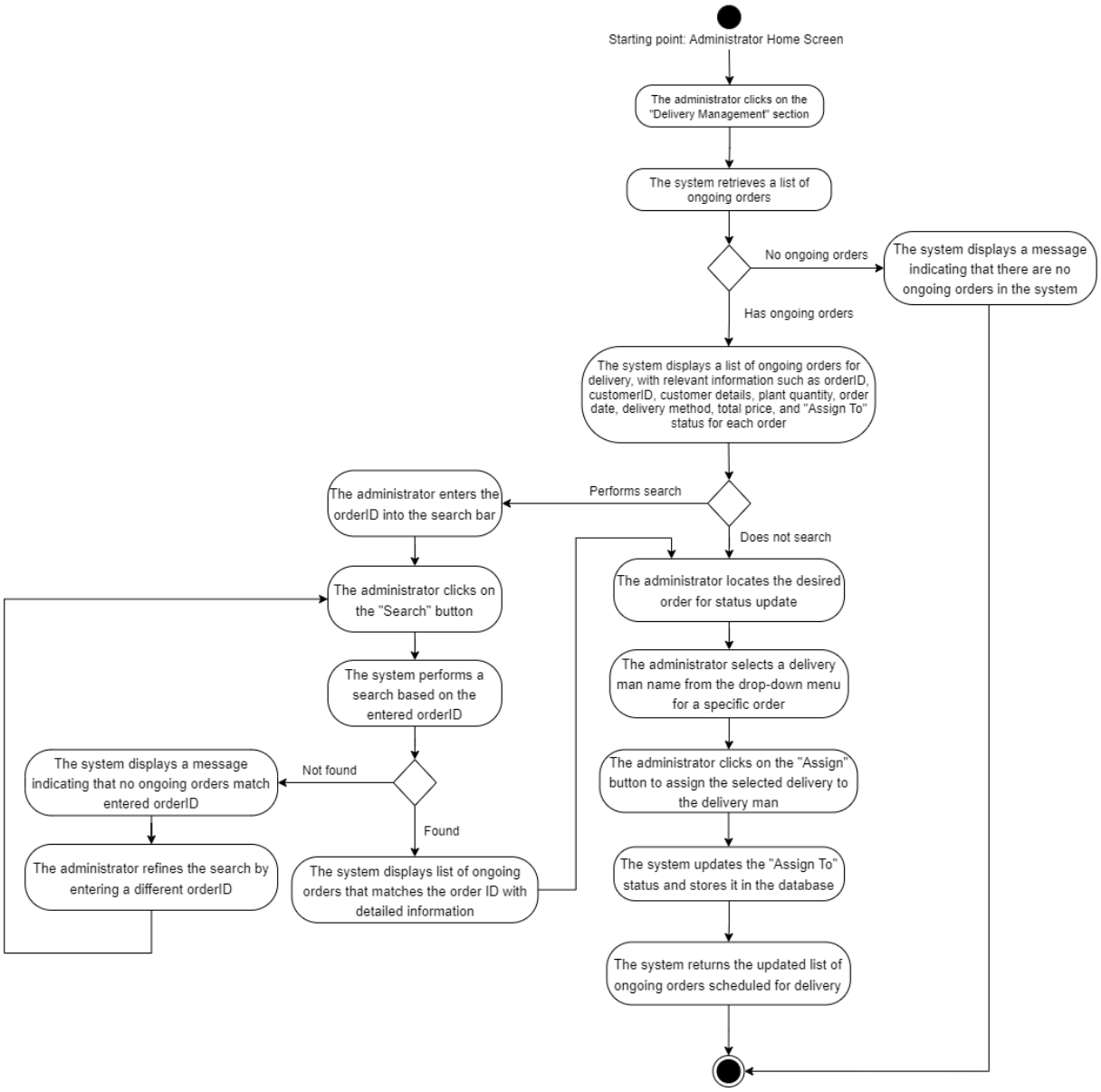



Figure 3.21: Assign Deliveries to Delivery Men Activity Diagram

# 4 Implementation

## 4.1 Development Environment

Figure 4.1 shows the screenshot of the project folder content for administrator. For our nursery plant shopping system, we use Django, which is a high-level Python web framework as the tools for developing the systems. The “app” directory contains several standard Django components. For example, “migrations”, “static” which contains the CSS file, “templates” which contains the HTML templates that are used to render the web pages related to administrator, and some Python files such as “models.py”, “views.py” and “urls.py”. On the other hand, the “TruePltSys” directory contains “media” which is used to stored plant images, and some core Django files like “settings.py”, “urls.py”, “wsgi.py”, and “asgi.py” which configure the overall behaviour of the Django project. Besides, there is “db.sqlite3” file, which is SQLite database file used by the Django to store data. The “manage.py” script is a command-line utility that allows us to interact with this Django project.

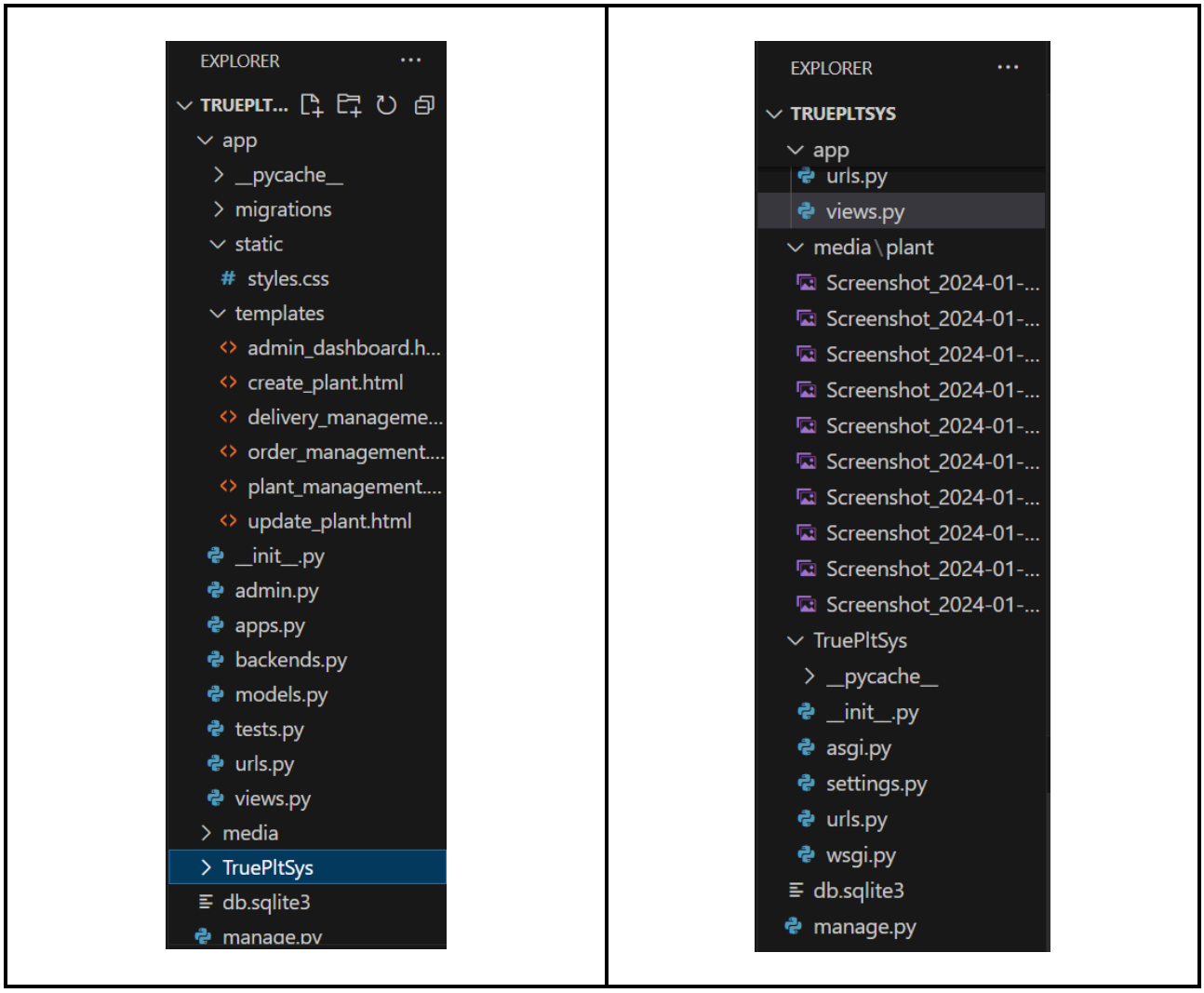



Figure 4.1: Screenshot of the Project Folder Content for Administrator

## 4.2 Main Program Codes

Figure 4.2 below shows the screenshot of admin_dashboard.html file. This is the HTML structure behind the “Administrator Home Screen”. The admin_dashboard.html file includes a welcoming message, personalized greeting for the administrator, and tabs for key functions like Plant Management, Order Management, Delivery Management and Logout.

```html
<!-- by TOH EE LIN -->

{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Admin Dashboard</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{% static 'styles.css' %}">
</head>

<body>
    <!-- Bootstrap Navbar -->
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'adminDashboard' %}">Admin Dashboard</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-item nav-link" href="{% url 'plantManagement' %}">Plant Management</a>
                    <a class="nav-item nav-link" href="{% url 'orderManagement' %}">Order Management</a>
                    <a class="nav-item nav-link" href="{% url 'deliveryManagement' %}">Delivery Management</a>
                    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
                </div>
            </div>
            <div class="ml-auto navbar-text">
                Welcome, Ee Lin
            </div>
        </div>
    </nav>

    <div class="container mt-4 d-flex justify-content-center align-items-center vh-100">
        <h1 class="mb-4 text-center display-4">Welcome to Admin Dashboard</h1>
    </div>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>

</html>
```

Figure 4.2: app/templates/admin_dashboard.html

Figure 4.3 below shows the adminDashboard function from views.py that renders the administrator home screen using the admin_dashboard.html template. The @user_passes_test decorator checks that the requesting user is an administrator before allowing access to this view.

```python
# Administrator
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def adminDashboard(request):
    return render(request, "admin_dashboard.html")
```

Figure 4.3: app/views.py/adminDashboard

Figure 4.4 below shows the screenshot of plant_management.html file. This is the HTML structure behind the “Plant Management Screen”. This plant_management.html file is designed to offer an overview of all plants in the nursery plant shopping system, featuring capabilities for searching plants by name, creating new plant entries, editing existing plants, and deleting plants from the inventory. Each plant is presented in a card layout, displaying its information such as the plant's image, name, ID, description, price, availability, and reviews associated to the plant. Directly from this interface, administrator can also navigate to the edit page for any plant or initiate the deletion process.

```html
<!-- by TOH EE LIN -->

{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Plant Management</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{% static 'styles.css' %}">

    <!--JavaScript to navigate to create form page-->
    <script>
        function redirectToCreatePlantForm() {
            // Redirect to the URL for create a new plant
            window.location.href = "{% url 'createPlant' %}";
        }
    </script>

     <!--JavaScript to navigate to update form page-->
    <script>
        function redirectToUpdatePlantForm(plant_id) {
            // Redirect to the URL for update a plant with the specific ID
            window.location.href = `/adm/plant/update/${plant_id}/`;
        }
    </script>
```

```
    <!-- Include JavaScript for confirmation dialog -->
    <script type="text/javascript">
        function confirmDelete() {
            return confirm('Are you sure you want to delete this plant?');
        }
    </script>
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'adminDashboard' %}">Admin Dashboard</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-item nav-link active" href="{% url 'plantManagement' %}">Plant Management</a>
                    <a class="nav-item nav-link" href="{% url 'orderManagement' %}">Order Management</a>
                    <a class="nav-item nav-link" href="{% url 'deliveryManagement' %}">Delivery Management</a>
                    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
                </div>
            </div>
        </div>
    </nav>

    <div class="container mt-4 main-content">
        <div class="row">
            <div class="col-md-8 mx-auto">
                <div class="text-center mb-0">
                    <!--Display successful plant creation/update message-->
                    {% if messages %}
                        {% for message in messages %}
                            <div class="alert alert-{{ message.tags }}" role="alert">
                                {{ message }}
                            </div>
                        {% endfor %}
                    {% endif %}

                    <form class="form-inline d-inline" action="{% url 'plantManagement' %}" method="get">
                        <input type="text" class="form-control mr-2" name="search_query" placeholder="Search Plant by Plant Name"
                        value="{{ search_query }}" style="width: 80%;">
                        <button type="submit" class="btn btn-primary">Search</button>
                    </form>
                    {% if error_message %}
                        <div class="alert alert-danger mt-2">{{ error_message }} </div>
                    {% endif %}
                </div>
                <div class="text-center">
                    <button class="btn btn-success mt-2" onclick="redirectToCreatePlantForm()">Create New Plant</button>
                </div>
```

```html
                {% for plant in plants %}
                <div class="plant-container card mb-5">
                    <div class="row no-gutters">
                        <div class="col-md-4">
                            <img src="{{plant.plant_image.url}}" class="card-img-top" alt="Plant Image">
                        </div>
                        <div class="col-md-8">
                            <div class="card-body">
                                <h5 class="card-title"><b>{{plant.plant_name}}</b></h5>
                                <p class="card-text"><b>Plant ID:</b> {{plant.id}}</p>
                                <p class="card-text text-justify"><b>Description:</b> <br>{{plant.plant_description}}</br></p>
                                <p class="card-text"><b>Price:</b> {{plant.plant_price}}</p>
                                <p class="card-text"><b>Availability:</b> {{plant.plant_availability}}</p>

                                <div class="info-buttons">
                                    <button class="btn btn-primary btn-sm" onclick="redirectToUpdatePlantForm({{ plant.id }})">
                                        Edit</button>
                                    <form action="{% url 'deletePlant' plant.id %}" method="post" onsubmit="return confirmDelete();"
                                    class="d-inline">
                                        {% csrf_token %}
                                        <input type="submit" value="Delete" class="btn btn-danger btn-sm"/>
                                    </form>
                                </div>
                            </div>
                        </div>
                    </div>

                    <!-- Reviews section -->
                    <div class="card-footer">
                        <h5 class="card-title reviews-title">Reviews</h5>
                        {% for review in plant.reviews %}
                            <div class="list-group">
                                <div class="list-group-item list-group-item-action">
                                    <div class="d-flex w-100 justify-content-between">
                                        <h6 class="mb-1"><b>{{ review.order_item.order.customer.customer_name }}
                                            (Rating: {{ review.rating }} star)</b></h6>
                                    </div>
                                    <p class="mb-1">Comment: {{ review.comment }}</p>
                                </div>
                            </div>
                        {% empty %}
                            <p class="text-muted">No reviews yet.</p>
                        {% endfor %}
                    </div>
                </div>
                {% endfor %}
            </div>
        </div>
    </div>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>

</html>
```

Figure 4.4: app/templates/plant_management.html

### 4.2.1 An administrator creates a plant

Figure 4.5 below shows the screenshot of create_plant.html file. This is the HTML structure behind the “Create New Plant Screen”. The create_plant.html file includes a structured form, which allows the administrator to input vital information about a new plant, such as its name, detailed descriptions, quantity available for sale, price, and a picture of the plant. The navigation bar at the top facilitates easy access back to the main sections of the Admin Dashboard, including Plant Management, Order Management, Delivery Management, and Logout.

```html
<!-- by TOH EE LIN -->

{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Create New Plant</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{% static 'styles.css' %}">
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'adminDashboard' %}">Admin Dashboard</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-item nav-link active" href="{% url 'plantManagement' %}">Plant Management</a>
                    <a class="nav-item nav-link" href="{% url 'orderManagement' %}">Order Management</a>
                    <a class="nav-item nav-link" href="{% url 'deliveryManagement' %}">Delivery Management</a>
                    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
                </div>
            </div>
        </div>
    </nav>

    <div class="container mt-5 form-container">
        <div class="text-center">
            <h2>Create New Plant Form</h2>
        </div>

        {% if error_message %}
            <div class="alert alert-danger" role="alert">
                {{ error_message }}
            </div>
        {% endif %}

        <form action="{% url 'createPlant' %}" method="post" enctype="multipart/form-data" class="mt-3">
            {% csrf_token %}
            <div class="mb-3 form-row">
                <label for="plant-name" class="form-label">Enter Plant Name</label>
                <input type="text" id="plant-name" name="plant_name" class="form-control">
            </div>
            <div class="mb-3 form-row">
                <label for="plant-description" class="form-label">Enter Plant Descriptions</label>
                <textarea id="plant-description" name="plant_description" class="form-control"></textarea>
            </div>
            <div class="mb-3 form-row">
                <label for="plant-quantity" class="form-label">Enter Plant Quantity</label>
                <input type="number" id="plant-quantity" name="plant_availability" class="form-control">
            </div>
```

```html
            <div class="mb-3 form-row">
                <label for="plant-price" class="form-label">Enter Plant Price</label>
                <input type="number" id="plant-price" name="plant_price" class="form-control">
            </div>
            <div class="mb-3 form-row">
                <label for="plant-image" class="form-label">Choose Image</label>
                <input type="file" id="plant-image" name="plant_image" class="form-control">
            </div>
            <div class="form-buttons">
                <a href="{% url 'plantManagement' %}" class="btn btn-secondary">Back</a>
                <input type="submit" value="Confirm" class="btn btn-primary">
            </div>
        </form>
    </div>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>

</html>
```

Figure 4.5: app/templates/create_plant.html

Figure 4.6 shows the createPlant function within the views.py that allows the administrator to add new plants to the plant list. It is decorated to restrict access only to authenticated administrator. The function validates the submitted form data for the new plant, checking details like name, price, and quantity, using regex and numerical checks. On success, a confirmation message is displayed and the administrator is redirected to the “Plant Management Screen”. On failure, errors are handled gracefully and shown to the administrator.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def createPlant(request):
    error_message = None
    if request.method == 'POST':
        plant_name = request.POST.get('plant_name')
        plant_description = request.POST.get('plant_description')
        plant_price = request.POST.get('plant_price')
        plant_availability = request.POST.get('plant_availability')
        plant_image = request.FILES.get('plant_image')

        # Regex for checking if the plant name contains only letters and spaces
        valid_name_pattern = re.compile(r"^[A-Za-z\s]+$")

        if not plant_image:
            error_message = 'You must upload a plant image.'
        elif not valid_name_pattern.match(plant_name):
            error_message = 'Plant name must only contain letters and spaces, and cannot be only numbers or special characters.'
        elif Plant.objects.filter(plant_name__iexact=plant_name).exists():
            error_message = 'A plant with this name already exists in the system.'
        else:
            try:
                plant_price = float(plant_price)
                if plant_price <= 0:
                    raise ValueError('Plant price must be greater than zero.')
```

```python
                plant_availability = int(plant_availability)
                if plant_availability <= 0:
                    raise ValueError('Plant quantity must be greater than zero.')

                Plant.objects.create(
                    plant_name=plant_name,
                    plant_description=plant_description,
                    plant_price=plant_price,
                    plant_availability=plant_availability,
                    plant_image=plant_image
                )
                messages.success(request, 'The plant was successfully created.')
                return redirect('plantManagement')

            except ValueError as e:
                error_message = str(e)

        plants = Plant.objects.all()
        return render(request, "create_plant.html", {'plants': plants, 'error_message': error_message})

    plants = Plant.objects.all()
    return render(request, "create_plant.html", {'plants': plants, 'error_message': error_message})
```

Figure 4.6: app/views.py/createPlant

### 4.2.2 An administrator updates a plant

Figure 4.7 below shows the screenshot of update_plant.html file. This is the HTML structure behind the “Update Plant Screen”. The update_plant.html file includes a form pre-populated with the plant’s current details, including its name, description, quantity, price, and an option to update the plant’s image. The plant image preview is on the right side of the form, which allows the administrator to visually verify the current image before choosing to update it.

```html
<!-- by TOH EE LIN -->

{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Update Plant</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{% static 'styles.css' %}">
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'adminDashboard' %}">Admin Dashboard</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-item nav-link active" href="{% url 'plantManagement' %}">Plant Management</a>
                    <a class="nav-item nav-link" href="{% url 'orderManagement' %}">Order Management</a>
                    <a class="nav-item nav-link" href="{% url 'deliveryManagement' %}">Delivery Management</a>
                    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
                </div>
            </div>
```

```html
        </div>
    </nav>

    <div class="container mt-5 form-container">
        <div class="text-center">
            <h2>Update Plant Form</h2>
        </div>

        {% if error_message %}
            <div class="alert alert-danger" role="alert">
                {{ error_message }}
            </div>
        {% endif %}

        <div class="form-content d-flex justify-content-between">
            <form action="{% url 'updatePlant' plant.id %}" method="post" enctype="multipart/form-data" class="w-100 mr-2">
                {% csrf_token %}
                <div class="form-group">
                    <label for="plant-name">Enter Plant Name</label>
                    <input type="text" id="plant-name" name="plant_name" value="{{ plant.plant_name }}" class="form-control">
                </div>
                <div class="form-group">
                    <label for="plant-description">Enter Plant Descriptions</label>
                    <textarea id="plant-description" name="plant_description" class="form-control">{{ plant.plant_description }}</textarea>
                </div>
                <div class="form-group">
                    <label for="plant-quantity">Enter Plant Quantity</label>
                    <input type="number" id="plant-quantity" name="plant_availability" value="{{ plant.plant_availability }}"
                    class="form-control">
                </div>
                <div class="form-group">
                    <label for="plant-price">Enter Plant Price</label>
                    <input type="number" id="plant-price" name="plant_price" value="{{ plant.plant_price }}" class="form-control">
                </div>
                <div class="form-group">
                    <label for="plant-image">Choose Image</label>
                    <input type="file" id="plant-image" name="plant_image" class="form-control">
                </div>
                <div class="form-buttons">
                    <a href="{% url 'plantManagement' %}" class="btn btn-secondary">Back</a>
                    <input type="submit" value="Update" class="btn btn-primary">
                </div>
            </form>

            <div class="plant-image-preview w-50">
                <img src="{{ plant.plant_image.url }}" alt="Plant image preview" class="img-fluid">
            </div>
        </div>
    </div>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>

</html>
```

Figure 4.7: app/templates/update_plant.html

Figure 4.8 shows the updatePlant function within the views.py that allows the administrator to edit the plant details. This function is secured with the @user_passes_test decorator, ensuring that only administrator can access this feature. It begins by fetching the plant's existing data using its plant ID. If changes are submitted through the POST method, the function validates the new plant name and price against specific criteria, including checking the name for letters and spaces only, ensuring no duplicate names within the system, and confirming that the new price is greater than zero. Upon successful validation, the plant's information is updated in the database.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def updatePlant(request, plant_id):
    plant = get_object_or_404(Plant, pk=plant_id)
    error_message = None

    if request.method == 'POST':
        new_plant_name = request.POST.get('plant_name')
        new_plant_price = request.POST.get('plant_price')

        valid_name_pattern = re.compile(r"^[A-Za-z\s]+$")

        if not valid_name_pattern.match(new_plant_name):
            error_message = 'Plant name must only contain letters and spaces, and cannot be only numbers or special characters.'
        elif Plant.objects.filter(plant_name__iexact=new_plant_name).exclude(pk=plant_id).exists():
            error_message = 'A plant with this name already exists in the system.'
        elif float(new_plant_price) <= 0:
            error_message = 'Plant price must be greater than zero.'
        else:
            plant.plant_name = new_plant_name
            plant.plant_description = request.POST.get('plant_description')
            plant.plant_price = new_plant_price
            plant.plant_availability = request.POST.get('plant_availability')
            plant_image = request.FILES.get('plant_image')
            if plant_image:
                plant.plant_image = plant_image
            plant.save()

            messages.success(request, 'The plant was successfully updated.')
            return redirect('plantManagement')

    return render(request, 'update_plant.html', {'plant': plant, 'error_message': error_message})
```

Figure 4.8: app/views.py/updatePlant

### 4.2.3 An administrator deletes a plant

Figure 4.9 shows the deletePlant function within the views.py that allows the administrator to remove the plant from the plant list. This function is secured with the @user_passes_test decorator, ensuring that only administrator can access this feature. Upon invocation, it first verifies the existence of the specified plant by its plant ID. The “get_object_or_404” method will allow the system to return a 404 error if the plant does not exist. Once the plant is successfully located, it is deleted from the database and the administrator is redirected back to the “Plant Management Screen”. A success message is generated to confirm the removal of the plant.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def deletePlant(request, plant_id):
    plant = get_object_or_404(Plant, id=plant_id)   # This ensures the plant exists, or returns a 404 error
    plant.delete()
    messages.add_message(request, messages.SUCCESS, 'The plant has been deleted.')
    return redirect('plantManagement')
```

Figure 4.9: app/views.py/deletePlant

### 4.2.4 An administrator searches plant

Figure 4.10 shows the plantManagement function within the views.py that allows the administrator to handle searching, filtering, and display of plants. This function is equipped with the @user_passes_test decorator, ensuring that only administrator can execute its operations. The plantManagement function gets the search query and uses it to filter plants by name, showing an

error if no matching plants were found. In the absence of a search query, the function will display all the plants. The function also retrieves related reviews and associates them to each plant object before rendering the final filtered plant view using the plant_management.html template.

```
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def plantManagement(request):
    search_query = request.GET.get('search_query', '')   # Get search query from GET parameters, default to empty string
    error_message = None

    if search_query:
        plants = Plant.objects.filter(plant_name__icontains=search_query)

        if not plants.exists():
            error_message = "No matching plants are found."
    else:
        plants = Plant.objects.all()

    # Iterate over each plant and retrieve associated reviews
    for plant in plants:
        plant.reviews = Review.objects.filter(order_item__plant=plant)

    return render(request, "plant_management.html", {"plants": plants, "error_message": error_message})
```

Figure 4.10: app/views.py/plantManagement

### 4.2.5 An administrator views order details and updates order status

Figure 4.11 below shows the screenshot of order_management.html file. This is the HTML structure behind the “Order Management Screen”. This order_management.html file is designed to allow the administrator to manage the customer orders and update order status as needed. It includes a search functionality that allows for the quick retrieval of orders by order ID. Each order is presented in a card layout, displaying essential information such as order ID, customer details, total quantity of plants ordered, total price, order date, and the receive method. Moreover, the order status is shown via dropdown menu with options such as “Waiting”, “Ready”, or “Completed”. In this case, administrator can update the order status with the “Update Status” button.

```html
<!-- by TOH EE LIN -->

{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Order Management</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{% static 'styles.css' %}">
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'adminDashboard' %}">Admin Dashboard</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-item nav-link" href="{% url 'plantManagement' %}">Plant Management</a>
                    <a class="nav-item nav-link active" href="{% url 'orderManagement' %}">Order Management</a>
                    <a class="nav-item nav-link" href="{% url 'deliveryManagement' %}">Delivery Management</a>
                    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
                </div>
            </div>
        {% for order in orders %}
        <div class="order-container card mb-4">
            <div class="card-body">
                <div class="row">
                    <div class="col-md-7">
                        <p class="card-text"><strong>OrderID:</strong> {{ order.id }}</p>
                        <p class="card-text"><strong>Full Name:</strong> {{ order.customer.customer_name }}</p>
                        <p class="card-text"><strong>Email:</strong> {{ order.customer.user.email }}</p>
                        <p class="card-text"><strong>State:</strong> {{ order.customer.customer_state }}</p>
                        <p class="card-text"><strong>Total Plants:</strong> {{ order.total_plant_quantity }}</p>
                        <p class="card-text"><strong>Total Price:</strong> {{ order.payment.payment_cost }}</p>
                    </div>
                    <div class="col-md-5">
                        <p class="card-text"><strong>UserID:</strong> {{ order.customer.user.id }}</p>
                        <p class="card-text"><strong>Phone Number:</strong> {{ order.customer.customer_phone_number }}</p>
                        <p class="card-text"><strong>Address:</strong> {{ order.customer.customer_address }}</p>
                        <p class="card-text"><strong>Order Date:</strong> {{ order.payment.order_date }}</p>
                        <p class="card-text"><strong>Receive Method:</strong> {{ order.payment.receive_method }}</p>
                    </div>
                </div>
```

```html
                <div class="row mt-4">
                    <div class="col-12">
                        <form action="{% url 'update_order_status' order.id %}" method="post" class="form-inline">
                            {% csrf_token %}
                            <div class="form-group mb-1 mr-2">
                                <label for="status-select-{{ order.id }}" class="sr-only">Order Status:</label>
                                <select id="status-select-{{ order.id }}" class="form-control" name="status">
                                    {% if order.payment.receive_method == 'Delivery' %}
                                        <option value="Waiting" {% if order.order_status == 'Waiting' %}selected{% endif %}>
                                            Waiting</option>
                                        <option value="Ready" {% if order.order_status == 'Ready' %}selected{% endif %}>Ready</option>
                                    {% elif order.payment.receive_method == 'Pickup' %}
                                        <option value="Waiting" {% if order.order_status == 'Waiting' %}selected{% endif %}>
                                            Waiting</option>
                                        <option value="Ready" {% if order.order_status == 'Ready' %}selected{% endif %}>Ready</option>
                                        <option value="Completed" {% if order.order_status == 'Completed' %}selected{% endif %}>
                                            Completed</option>
                                    {% endif %}
                                </select>
                            </div>
                            <button type="submit" class="btn btn-primary mb-2">Update Status</button>
                        </form>
                    </div>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>

</html>
```

Figure 4.11: app/templates/order_management.html

Figure 4.12 shows the orderManagement function within the views.py that allows the administrator to oversee the customer orders. This function is secured with the @user_passes_test decorator, ensuring that only administrator can access this feature. The orderManagement function filters out pickup orders which are of completed status, and delivery orders which are of ready, or out-of-delivery, or completed status in order to focus solely on those still requiring attention. Administrator can search by order ID, with appropriate errors displayed for non-existent or invalid input and an informative message if there are no ongoing orders.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def orderManagement(request):
    search_query = request.GET.get('search_query', '')
    error_message = None
    info_message = None

    orders = (
        Order.objects
        .select_related('customer')
        .prefetch_related('order_items')
        .annotate(total_plant_quantity=Sum('order_items__plant_quantity'))
        .exclude(
            Q(payment__receive_method='Pickup', order_status='Completed') |
            Q(payment__receive_method='Delivery', order_status='Ready') |
            Q(payment__receive_method='Delivery', order_status='Out Of Delivery') |
            Q(payment__receive_method='Delivery', order_status='Completed')
        )
    )

    if search_query:
        try:
            search_query = int(search_query)
            orders = orders.filter(id=search_query)

            if not orders.exists():
                error_message = "No ongoing orders matched the entered order ID."

        except ValueError:
            error_message = "Invalid input. Please enter a numeric order ID."
            orders = Order.objects.none()
    else:
        # Check if there are no ongoing orders
        if not orders.exists():
            info_message = "There are no ongoing orders in the system."

    return render(request, "order_management.html", {"orders": orders, "error_message": error_message, "info_message": info_message})
```

Figure 4.12: app/views.py/orderManagement

Figure 4.13 shows the update_order_status function within the views.py that allows the administrator to update the status of ongoing orders. This function is protected by the @user_passes_test decorator, ensuring that only authenticated administrator can modify the status of orders. Upon accessing an order via its order ID, administrator can update its status based on the current processing stage directly through a POST request. Following the successful update, the function redirects the administrator back to the “Order Management Screen”.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def update_order_status(request, order_id):
    order = get_object_or_404(Order, id=order_id)

    if request.method == 'POST':
        order.order_status = request.POST.get('status')
        order.save()
        return redirect('orderManagement')

    return redirect('orderManagement')
```

Figure 4.13: app/views.py/update_order_status

### 4.2.6 An administrator assigns deliveries

Figure 4.14 below shows the screenshot of delivery_management.html file. This is the HTML structure behind the “Delivery Management Screen”. This delivery_management.html file is designed to allow the administrator to manage the delivery orders and assign the orders to delivery men. It includes a search functionality that allows for the quick retrieval of orders by order ID. Each order is presented in a card layout, displaying essential information such as order ID, customer details, total quantity of plants ordered, total price, order date, and the receive method. Moreover, each delivery order box includes an “Assign” button, paired with a dropdown menu. The administrator can select a delivery man from the dropdown menu and click on the “Assign” button to assign the delivery order to that particular delivery man.

```html
<!-- by TOH EE LIN -->

{% load static %}
<!DOCTYPE html>
<html lang="en">

<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Delivery Management</title>
    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/css/bootstrap.min.css">
    <link rel="stylesheet" href="{% static 'styles.css' %}">
</head>

<body>
    <nav class="navbar navbar-expand-lg navbar-dark bg-dark">
        <div class="container-fluid">
            <a class="navbar-brand" href="{% url 'adminDashboard' %}">Admin Dashboard</a>
            <button class="navbar-toggler" type="button" data-toggle="collapse" data-target="#navbarNavAltMarkup"
            aria-controls="navbarNavAltMarkup" aria-expanded="false" aria-label="Toggle navigation">
                <span class="navbar-toggler-icon"></span>
            </button>
            <div class="collapse navbar-collapse" id="navbarNavAltMarkup">
                <div class="navbar-nav">
                    <a class="nav-item nav-link" href="{% url 'plantManagement' %}">Plant Management</a>
                    <a class="nav-item nav-link" href="{% url 'orderManagement' %}">Order Management</a>
                    <a class="nav-item nav-link active" href="{% url 'deliveryManagement' %}">Delivery Management</a>
                    <a class="nav-item nav-link" href="{% url 'logout' %}">Logout</a>
                </div>
            </div>
        </div>
    </nav>

    <div class="container mt-4 main-content">
        {% if error_message %}
            <div class="alert alert-danger" role="alert">
                {{ error_message }}
            </div>
        {% elif info_message %}
            <div class="alert alert-info" role="alert">
                {{ info_message }}
            </div>
        {% endif %}
        <div class="row">
            <div class="col-md-8 mx-auto">
                <div class="text-center mb-4">
                    <form class="form-inline d-inline" action="{% url 'deliveryManagement' %}" method="get">
                        <input type="text" class="form-control mr-2" name="search_query" placeholder="Search by Order ID"
                        value="{{ search_query }}" style="width: 50%;">
                        <button type="submit" class="btn btn-primary">Search</button>
                    </form>
                </div>
            </div>
        </div>
```

```html
        {% for order in delivery_orders %}
        <div class="delivery-container card mb-4">
            <div class="card-body">
                <div class="row">
                    <div class="col-md-7">
                        <p class="card-text"><strong>OrderID:</strong> {{ order.id }}</p>
                        <p class="card-text"><strong>Full Name:</strong> {{ order.customer.customer_name }}</p>
                        <p class="card-text"><strong>Email:</strong> {{ order.customer.user.email }}</p>
                        <p class="card-text"><strong>State:</strong> {{ order.customer.customer_state }}</p>
                        <p class="card-text"><strong>Total Plants:</strong> {{ order.total_plant_quantity }}</p>
                        <p class="card-text"><strong>Total Price:</strong> {{ order.payment.payment_cost }}</p>
                    </div>
                    <div class="col-md-5">
                        <p class="card-text"><strong>UserID:</strong> {{ order.customer.user.id }}</p>
                        <p class="card-text"><strong>Phone Number:</strong> {{ order.customer.customer_phone_number }}</p>
                        <p class="card-text"><strong>Address:</strong> {{ order.customer.customer_address }}</p>
                        <p class="card-text"><strong>Order Date:</strong> {{ order.payment.order_date }}</p>
                        <p class="card-text"><strong>Receive Method:</strong> {{ order.payment.receive_method }}</p>
                    </div>
                </div>

                <div class="row mt-4">
                    <div class="col-12">
                        <form action="{% url 'assign_delivery' order.id %}" method="post" class="form-inline">
                            {% csrf_token %}
                            <div class="form-group mb-1 mr-2">
                                <label for="assign-select-{{ order.id }}" class="sr-only">Assign To:</label>
                                <select id="assign-select-{{ order.id }}" class="form-control" name="deliveryman">
                                    {% for deliveryman in deliverymen %}
                                        <option value="{{ deliveryman.user.id }}">{{ deliveryman.deliveryman_name }}</option>
                                    {% endfor %}
                                </select>
                            </div>
                            <button type="submit" class="btn btn-primary mb-2">Assign</button>
                        </form>
                    </div>
                </div>
            </div>
        </div>
        {% endfor %}
    </div>

    <!-- jQuery first, then Popper.js, then Bootstrap JS -->
    <script src="https://code.jquery.com/jquery-3.5.1.slim.min.js"></script>
    <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
    <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.5.2/js/bootstrap.min.js"></script>
</body>

</html>
```

Figure 4.14: app/templates/delivery_management.html

Figure 4.15 shows the deliveryManagement function within the views.py that allows the administrator to oversee the delivery orders. This function is protected by the @user_passes_test decorator, ensuring that only authenticated administrator can execute its operation. The function starts by retrieving all delivery orders that are marked “Ready” and are unassigned to any delivery man. It uses “filter”, “select_related”, and “prefetch_related” to efficiently query the required order data from the database. Administrator can search by order ID, with appropriate errors displayed for non-existent or invalid input and an informative message if there are no ongoing delivery orders.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def deliveryManagement(request):
    search_query = request.GET.get('search_query', '')
    error_message = None
    info_message = None

    delivery_orders = (
        Order.objects
        .filter(payment__receive_method='Delivery', delman__isnull=True, order_status='Ready')
        .select_related('customer', 'payment')
        .prefetch_related('order_items')
    )

    deliverymen = DeliveryMan.objects.all()

    if search_query:
        try:
            search_query = int(search_query)
            delivery_orders = delivery_orders.filter(id=search_query)

            if not delivery_orders.exists():
                error_message = "No ongoing delivery orders matched the entered order ID."

        except ValueError:
            error_message = "Invalid input. Please enter a numeric order ID."
            delivery_orders = Order.objects.none()
    else:
        # Check if there are no ongoing delivery orders
        if not delivery_orders.exists():
            info_message = "There are no ongoing delivery orders in the system."

    return render(request, 'delivery_management.html', {'delivery_orders': delivery_orders, 'deliverymen': deliverymen,
                                                         'error_message': error_message, 'info_message': info_message})
```

Figure 4.15: app/views.py/deliveryManagement

Figure 4.16 shows the assign_delivery function within the views.py that allows the administrator to assign deliveries to delivery men. This function is protected by the @user_passes_test decorator, ensuring that only authenticated administrator can assign the delivery orders. Besides, this function identifies the target order and selected deliveryman based on their IDs passed in with the POST request. Utilizing get_object_or_404, the function ensures both the order and the deliveryman exist within the database to prevent any errors related to non-existent entities. After successfully associating the delivery man with the order, the update is saved to the database, and the administrator is redirected back to the “Delivery Management Screen”.

```python
# by TOH EE LIN
@user_passes_test(lambda user: user.is_admin)
def assign_delivery(request, order_id):
    order = get_object_or_404(Order, id=order_id)

    if request.method == 'POST':
        deliveryman_id = request.POST.get('deliveryman')
        deliveryman = get_object_or_404(DeliveryMan, user_id=deliveryman_id)

        order.delman = deliveryman
        order.save()
        return redirect('deliveryManagement')

    return redirect('deliveryManagement')
```

Figure 4.16: app/views.py/assign_delivery

Figure 4.17 shows the CSS code within the styles.css file that defines the visual styling and layout for administrator web pages, including the “Administrator Home Screen”, “Plant Management Screen”, “Create New Plant Screen”, “Update Plant Screen”, “Order Management Screen”, and “Delivery Management Screen”. This CSS code precisely configures various elements including the body, navigation bar, main content area, plant containers, and form components.

```
/* by TOH EE LIN */
body, html {
    margin: 0;
    padding: 0;
    font-family: Arial;
}

/* by TOH EE LIN */
.navbar {
    margin-bottom: 20px;
}

/* by TOH EE LIN */
.main-content {
    padding: 20px;
}

/* by TOH EE LIN */
.plant-container {
    border: 3px solid #333;
    background-color: #ffffff;
    margin-top: 10px;
}

/* by TOH EE LIN */
.plant-container .card-body {
    padding: 15px;
}

/* by TOH EE LIN */
.card-img-top {
    width: 100%;
    height: 100%;
    object-fit: cover;
}

/* by TOH EE LIN */
.card-footer {
    background-color: #f2f2f2;
    padding: 10px;
}

/* by TOH EE LIN */
button, .btn {
    margin: 5px 10px 5px 0px;
    padding: 5px 10px;
}

/* by TOH EE LIN */
.btn-success {
    margin: 30px
}
```

```
/* by TOH EE LIN */
/* Responsive adjustments for smaller screens */
@media (max-width: 768px) {
    .search-bar {
        flex-direction: column;
    }

    .search-container, .create-button {
        width: 100%;
        margin-top: 10px;
    }
}

/* by TOH EE LIN */
.reviews-title {
    font-weight: bold;
    margin: 5px 0px 10px 5px;
}

/* by TOH EE LIN */
/* List group styling for reviews */
.list-group-item {
    padding: 10px;
    border: none;
    margin-bottom: 10px;
}
```

```
/* by TOH EE LIN */
.form-container {
    width: 100%;
    max-width: 650px;
    padding: 30px;
    background-color: #ffffff;
    border: 3px solid #333;
    border-radius: 15px;
}

/* by TOH EE LIN */
.form-container h2 {
   margin-bottom: 20px;
}

/* by TOH EE LIN */
.form-label {
    margin-left: 2px;
}

/* by TOH EE LIN */
.form-row input[type="text"],
.form-row input[type="number"],
.form-row textarea,
.form-row input[type="file"] {
    flex-grow: 1;   /* Grow to use available space */
    padding: 10px;
    border-radius: 5px;
    resize: vertical;
    border: 1px solid #000;
    line-height: 1;
}

/* by TOH EE LIN */
.form-group input[type="text"],
.form-group input[type="number"],
.form-group textarea,
.form-group input[type="file"] {
    flex-grow: 1;
    padding: 10px;
    border-radius: 5px;
    resize: vertical;
    border: 1px solid #000;
    line-height: 1;
}
```

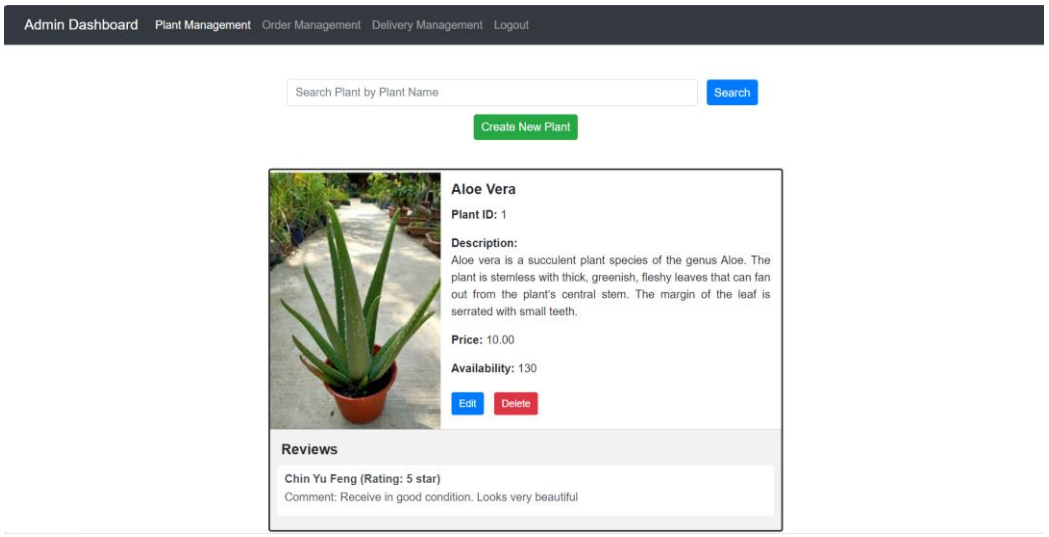
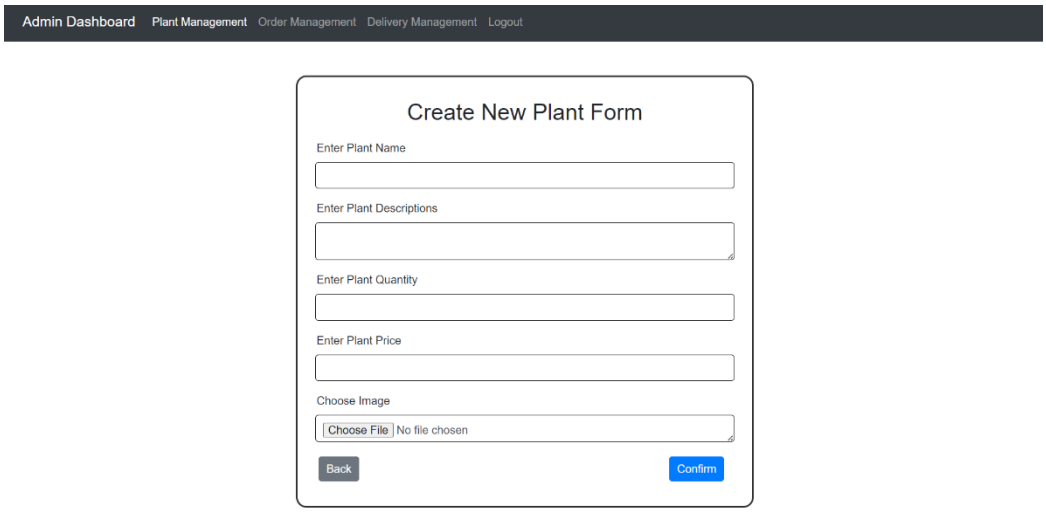
```
/* by TOH EE LIN */
.form-buttons {
    display: flex;
    align-items: center;
}

/* by TOH EE LIN */
.form-buttons .btn-primary {
    margin-left: auto;
}

/* by TOH EE LIN */
.form-buttons input[type="button"]:hover,
.form-buttons input[type="submit"]:hover {
    background-color: #e7e7e7;
}

/* by TOH EE LIN */
.plant-image-preview {
    height: 250px;
    margin: 10px;
    display: flex;
    justify-content: center;
    align-items: center;
}

/* by TOH EE LIN */
.plant-image-preview img {
    margin-left: 20px;
}

/* by TOH EE LIN */
.order-container {
    background-color: #ffffff;
    padding: 0px 20px;
    border: 3px solid #333;
    margin: 10px 80px;
}

/* by TOH EE LIN */
.update-status-button:hover {
    background-color: #e7e7e7;
}

/* by TOH EE LIN */
.delivery-container {
    background-color: #ffffff;
    padding: 0px 20px;
    border: 3px solid #333;
    margin: 10px 80px;
}

/* by TOH EE LIN */
.assign-button:hover {
    background-color: #e7e7e7;
}
```

Figure 4.17: app/static/styles.css

## 4.3 Sample Screens

### 4.3.1 An administrator creates a plant

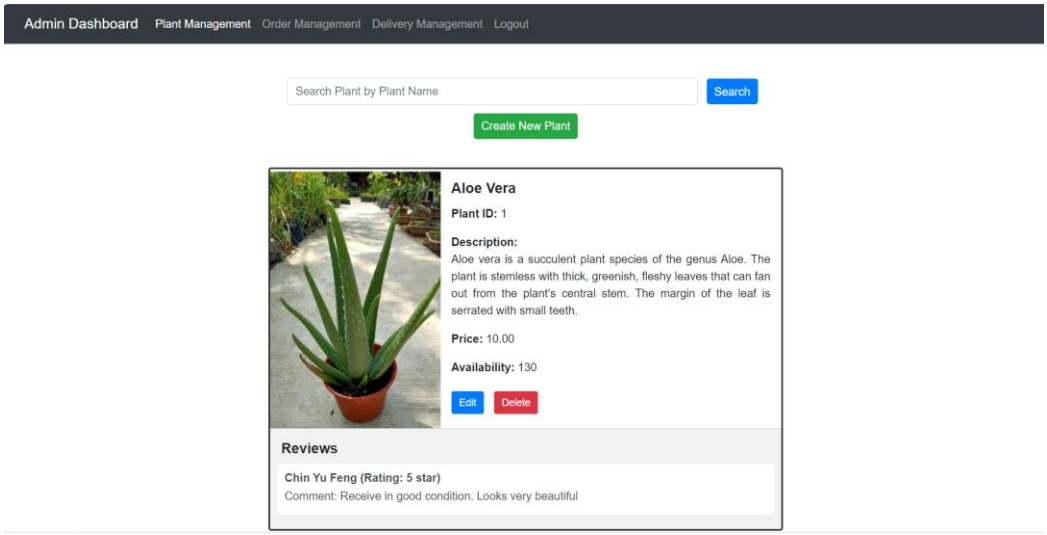
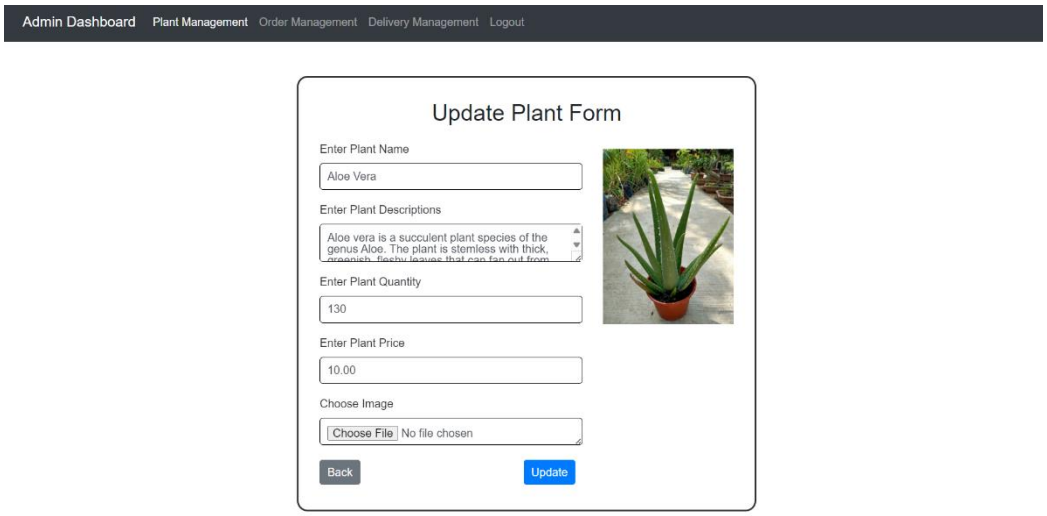
Figure 4.18 below shows the “Plant Management Screen”. After clicking on the “Create New Plant” button, the administrator will be redirected to the “Create New Plant Screen” (Figure 4.19). From here, the administrator can perform addition of new plant into the nursery plant shopping system.

| | |
|---|---|
| Administrator Views Plant Management Screen |  <br>Figure 4.18: Plant Management Screen |
| Administrator Creates New Plant Screen |  <br>Figure 4.19: Create New Plant Screen |

### 4.3.2 An administrator updates a plant

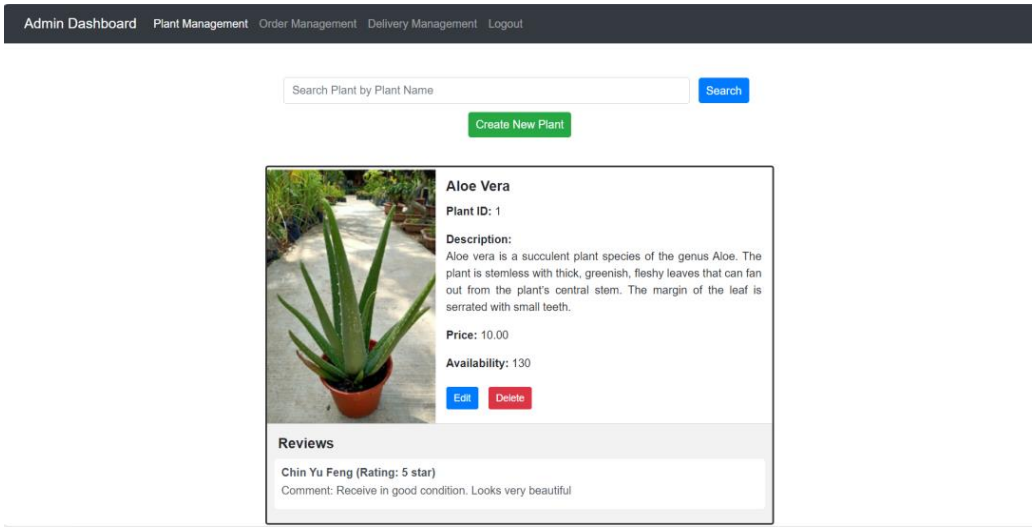
Figure 4.20 below shows the “Plant Management Screen”. After clicking the “Edit” button of a particular plant, the administrator will be redirected to the “Update Plant Screen” (Figure 4.21), which

pre-populates with that plant's current details. From here, the administrator can update any piece of information about that plant.

| | |
|---|---|
| Administrator Views Plant Management Screen |  <br>Figure 4.20: Plant Management Screen |
| Administrator Updates Plant Screen |  <br>Figure 4.21: Update Plant Screen |

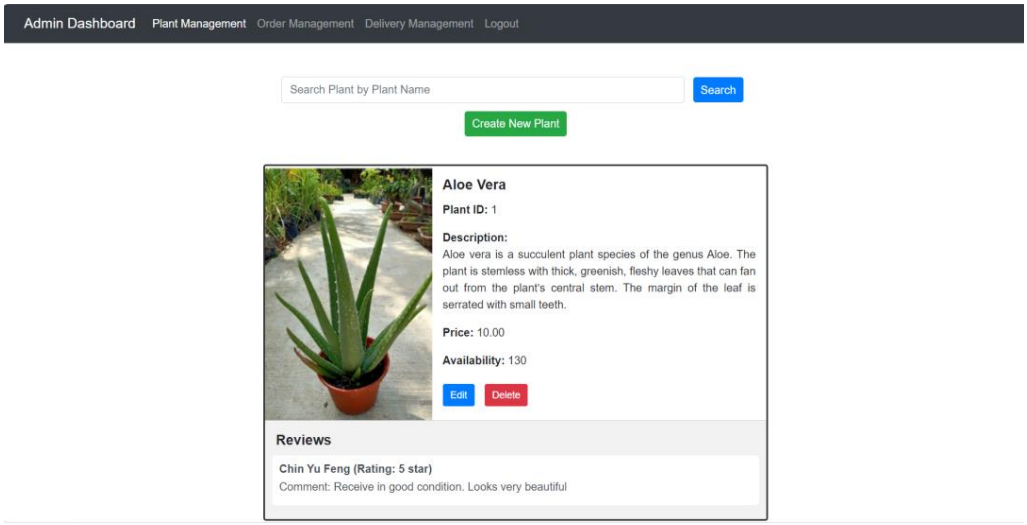
### 4.3.3 An administrator deletes a plant

Figure 4.22 below shows the "Plant Management Screen". From here, the administrator can click on the "Delete" button of a particular plant to delete the plant.

| | |
|---|---|
| Administrator Views Plant Management Screen |  <br>Figure 4.22: Plant Management Screen |

### 4.3.4 An administrator searches plant

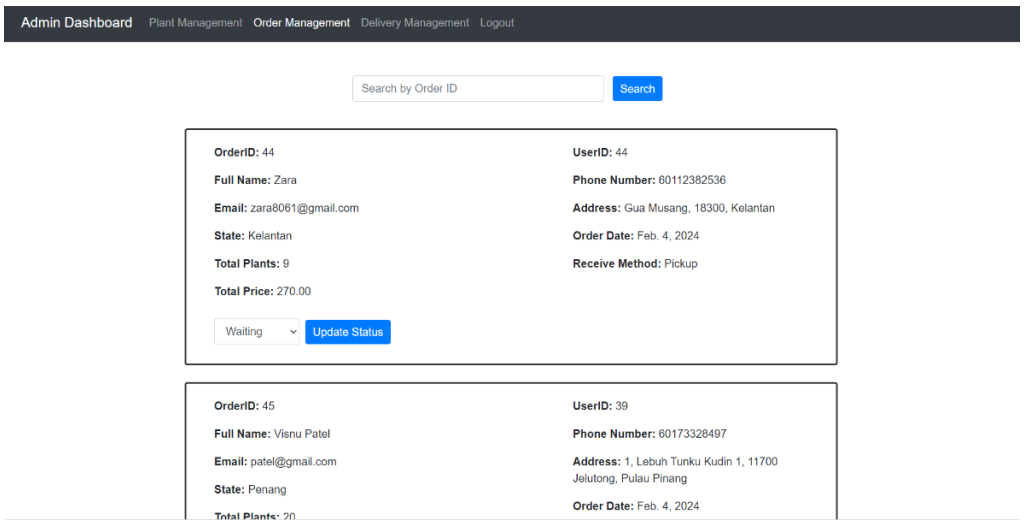
Figure 4.23 below shows the “Plant Management Screen”. From here, the administrator can enter a plant name in the search bar and click on the “Search” button to search for matching plant.

| | |
|---|---|
| Administrator Views Plant Management Screen |  <br>Figure 4.23: Plant Management Screen |

### 4.3.5 An administrator views order details and updates order status

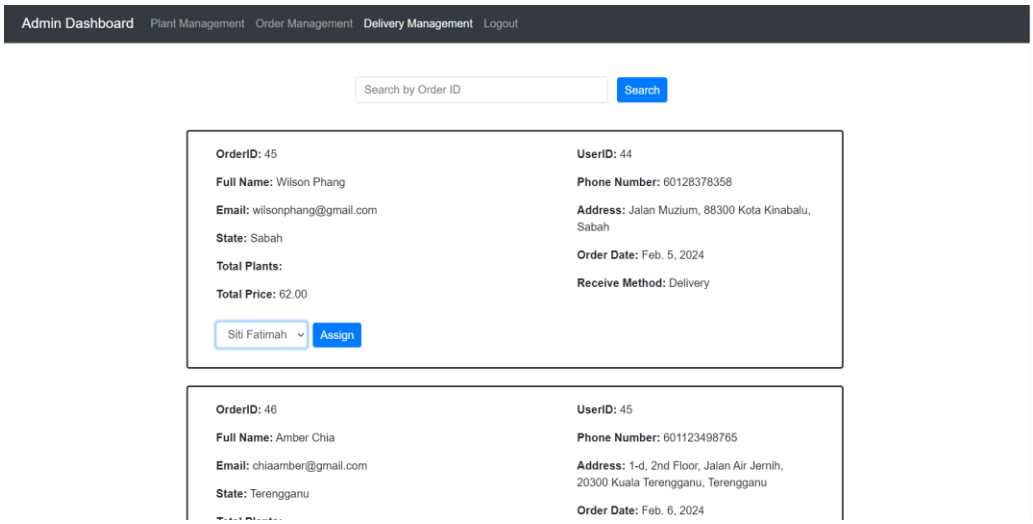
Figure 4.24 below shows the “Order Management Screen”. From here, the administrator can enter an order ID in the search bar and click on the “Search” button to search for the matching order. The

administrator also can select the appropriate order status from the dropdown menu and click on the "Update Status" button to update the selected status for the order.

| | |
|---|---|
| Administrator Views Order Management Screen |  <br>Figure 4.24: Order Management Screen |

### 4.3.6 An administrator assigns deliveries

Figure 4.25 below shows the "Delivery Management Screen". From here, the administrator can enter an order ID in the search bar and click on the "Search" button to search for the matching delivery order. The administrator also can select the delivery man name from the dropdown menu and click on the "Assign" button to assign the delivery order to the selected delivery man.

| | |
|---|---|
| Administrator Views Delivery Management Screen |  <br>Figure 4.25: Delivery Management Screen |

# 5 Testing

## 5.1 Test Data

Table 5.1 below shows the Administrator Credentials, including the administrator's email, first name, last name, and password. This table contains the necessary credentials for logging in as an administrator.

Table 5.1: Administrator Credentials

| Administrator Email | Administrator First Name | Administrator Last Name | Administrator Password |
|---|---|---|---|
| eelin@gmail.com | Toh | Ee Lin | 123456 |

Table 5.2 below shows the Create Plant Test Data. This table provides test data for creating plant records, encompassing both positive outcomes and scenarios that trigger error messages. It includes details such as the plant name, plant description, plant quantity, plant price, and plant image for various plant entries.

Table 5.2: Create Plant Test Data

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Money Tree | Characterised by its unique tree-like plant with a stout trunk. It has been said to bring good fortune and success. | 80 | 35.00 | |
| Cactus | Cactus is a diverse group of succulent plants known for their unique and often unusual appearance. It typically has fleshy, water-storing stems that are adapted to reduce water loss. | 15 | 55.00 | |
| ??12 | Characterised by its unique tree-like plant with a stout trunk. It has been said to bring good fortune and success. | 80 | 35.00 | |
| Jade Plant | Rounded green fleshy leaves with red lining believed to bring wealth and good fortune | 0 | 29.00 | |

| | | | | |
|---|---|---|---|---|
| Jade Plant | Rounded green fleshy leaves with red lining believed to bring wealth and good fortune | 50 | 0.00 | |
| Jade Plant | Rounded green fleshy leaves with red lining believed to bring wealth and good fortune. | 50 | 29.00 | (blank) |

Table 5.3 below shows the Update Plant Test Data. This table showcases test data for updating plant records, encompassing various scenarios that may yield either successful updates or trigger error messages. It includes details such as the plant name, plant description, plant quantity, plant price, and plant image.

Table 5.3: Update Plant Test Data

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| aloe vera | Aloe vera is a succulent plant species of the genus Aloe. The plant is stemless with thick, greenish, fleshy leaves that can fan out from the plant's central stem. The margin of the leaf is serrated with small teeth. | 50 | 20.00 | |
| Cactus | Cactus is a diverse group of succulent plants known for their unique and often unusual appearance. It typically has fleshy, water-storing stems that are adapted to reduce water loss. | 15 | 55.00 | |
| 120 | Aloe vera is a succulent plant species of the genus Aloe. The plant is stemless with thick, greenish, fleshy leaves that can fan out from the plant's central stem. The margin of the leaf is serrated with small teeth. | 50 | 20.00 | |
| Aloe Vera | Aloe vera is a succulent plant species of the genus Aloe. The plant is stemless with thick, greenish, fleshy leaves that can fan out from the plant's central stem. The margin of the leaf is serrated with small teeth. | 50 | 0.00 | |

Table 5.4 below shows the Delete Plant Test Data. In this table, the entry "Money Tree" represents the specific plant test data that is intended for deletion. Upon clicking the "Delete" button associated with this plant, it serves as a verification step to check whether the "Money Tree" plant record remains within the system or has been effectively removed. This serves as a test scenario for evaluating the deletion functionality.

Table 5.4: Delete Plant Test Data

| Plant Name |
|---|
| Money Tree |

Table 5.5 below shows the Search Plant Test Data. In this table, a variety of plant names are presented to be entered into the search bar. These plant names are the test data for searching plant, which encompassing both valid inputs that yield search results and invalid inputs that trigger error messages.

Table 5.5: Search Plant Test Data

| Plant Name |
|---|
| sunflower |
| h |
| P |
| na |
| (blank) |
| / |
| 20 |

Table 5.6 below shows the Search Order Test Data. In this table, various order IDs are provided to be entered into the search bar. These order IDs are the test data for searching orders, which encompassing both valid inputs that yield successful searches and invalid inputs that trigger error messages.

Table 5.6: Search Order Test Data

| Order ID |
|---|
| 43 |
| 20 |
| (blank) |
| a |
| 43w |
| / |

Table 5.7 below shows the Update Order Status Test Data. This table presents test data for updating order statuses, specifically testing the functionality of the nursery plant shopping system's order status update feature. It includes order IDs and the corresponding statuses to which they should be updated.

Table 5.7: Update Order Status Test Data

| Order ID | Order Status (Update to) |
|---|---|
| 43 | Ready |
| 44 | Ready |
| 44 | Completed |
| 45 | Ready |
| 46 | Ready |

Table 5.8 below shows the Search Delivery Order Test Data. In this table, various order IDs are provided to be entered into the search bar. These order IDs are the test data for searching delivery orders, which encompassing both valid inputs that yield successful searches and invalid inputs that trigger error messages.

Table 5.8: Search Delivery Order Test Data

| Order ID |
|---|
| 45 |
| 80 |
| (blank) |
| zz |
| 45a |
| @ |

Table 5.9 below presents the Assign Delivery Order Test Data. This table includes test data for assigning delivery orders to specific delivery man. It provides order IDs and the corresponding names of delivery man to whom the delivery orders should be assigned. These entries serve as test cases to evaluate the functionality of the nursery plant shopping system in assigning delivery orders to delivery man.

Table 5.9: Assign Delivery Order Test Data

| Order ID | Delivery Man Name (Assign to) |
|---|---|
| 45 | Siti Fatimah |
| 46 | Varna |

## 5.2 Acceptance Test

Table 5.10 shows the administrator acceptance test with each of its use cases.

Module: *Administrator*

Developer: *Toh Ee Lin*

Table 5.10: Administrator Acceptance Test

| Criteria | Fulfilled? | Remarks |
|---|---|---|
| Create Plant | | |
| Update Plant | | |
| Delete Plant | | |
| Search Plant by Plant Name | | |
| View Order Details and Update Order Status | | |
| Assign Deliveries to Delivery Men | | |

Date tested:

% Completed:

Tested by:

Verified by:

## 5.3 Test Results

### Create Plant Test Results:

Figure 5.1 shows the insertion of new plant named “Money Tree” and its details into the “Create New Plant Screen”. After clicking the “Confirm” button, the administrator is redirected back to the “Plant Management Screen” and a successful plant creation message is shown, as in Figure 5.2. The new plant record named “Money Tree” and its details can be seen at the bottom of the screen, as in Figure 5.3.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Money Tree | Characterised by its unique tree-like plant with a stout trunk. It has been said to bring good fortune and success. | 80 | 35.00 | |

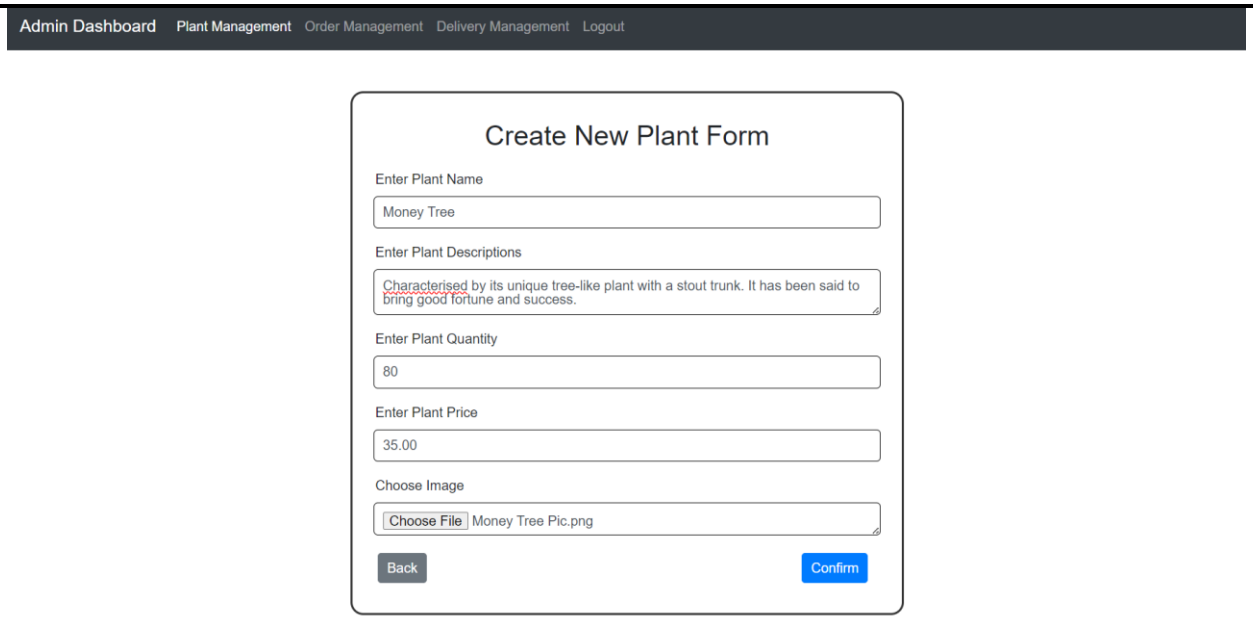



Figure 5.1: Inserting New Plant Details

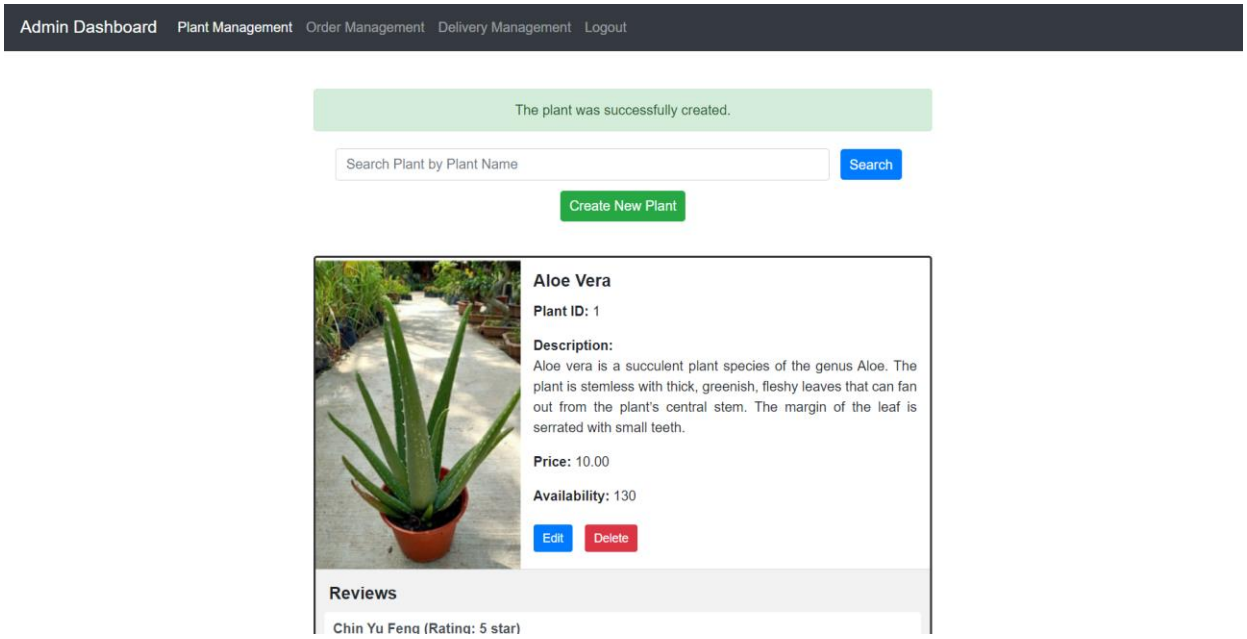



Figure 5.2: Successful Plant Creation

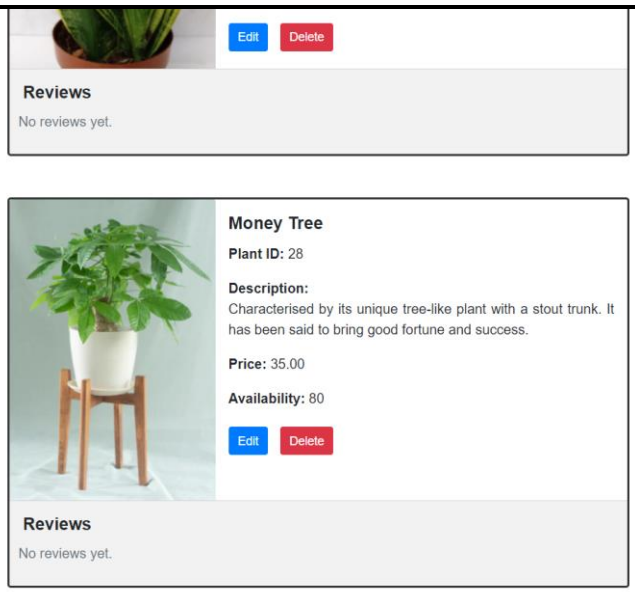



Figure 5.3: New Plant Record

Figure 5.4 shows the attempt to insert a plant named “Cactus” and its details into the “Create New Plant Screen”. Since “Cactus” already exists in the system, an error message is displayed, as shown in Figure 5.5.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Cactus | Cactus is a diverse group of succulent plants known for their unique and often unusual appearance. It typically has fleshy, water-storing stems that are adapted to reduce water loss. | 15 | 55.00 | |

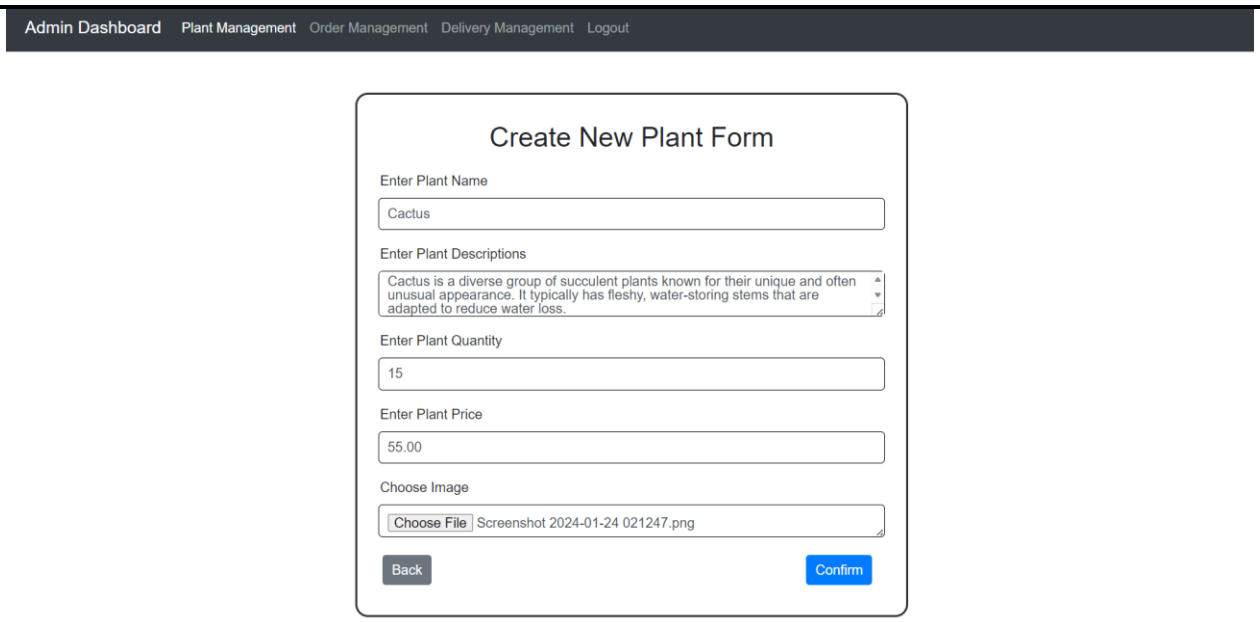



Figure 5.4: Inserting Existed Plant Details

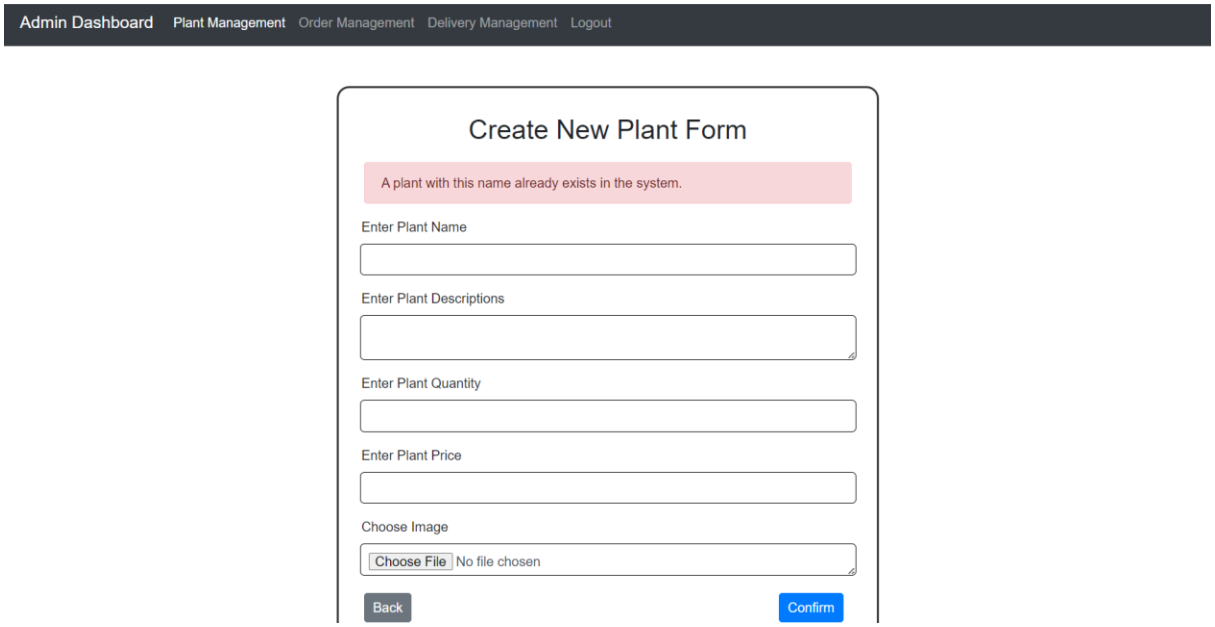



Figure 5.5: Plant Already Exist Error Message

In Figure 5.6, an invalid plant name is inserted into the “Create New Plant Screen”. An error message is displayed, as depicted in Figure 5.7, stating that plant name must only contain letters and spaces, and cannot be only numbers or special characters.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| ??12 | Characterised by its unique tree-like plant with a stout trunk. It has been said to bring good fortune and success. | 80 | 35.00 | |

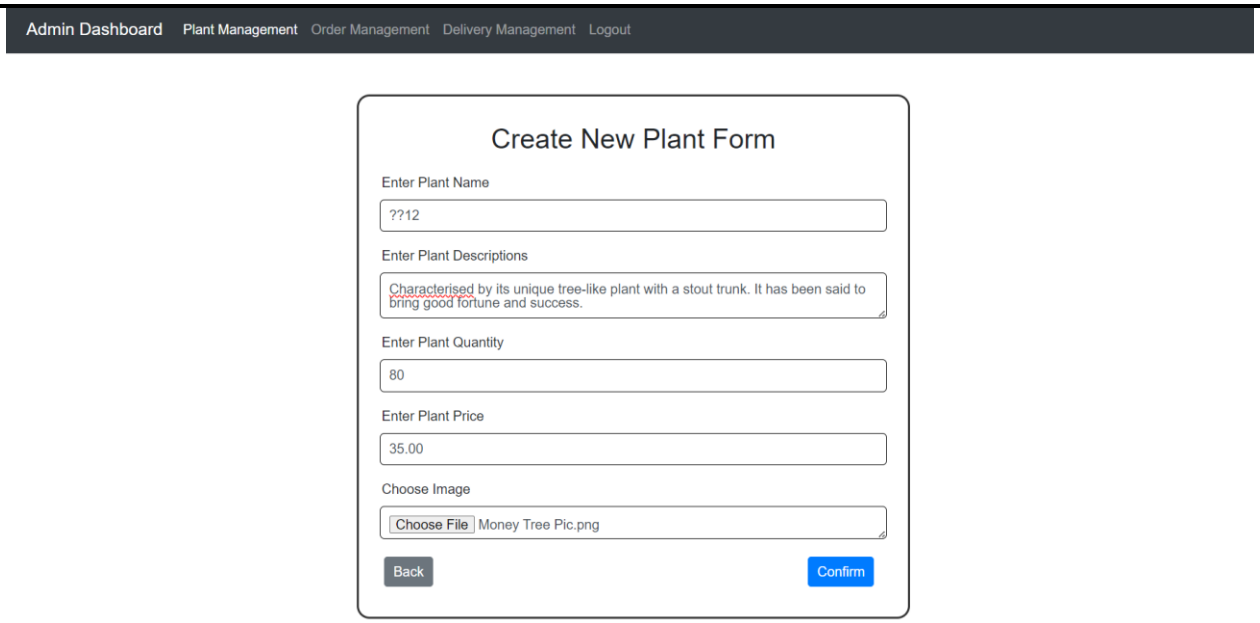



Figure 5.6: Inserting Invalid Name

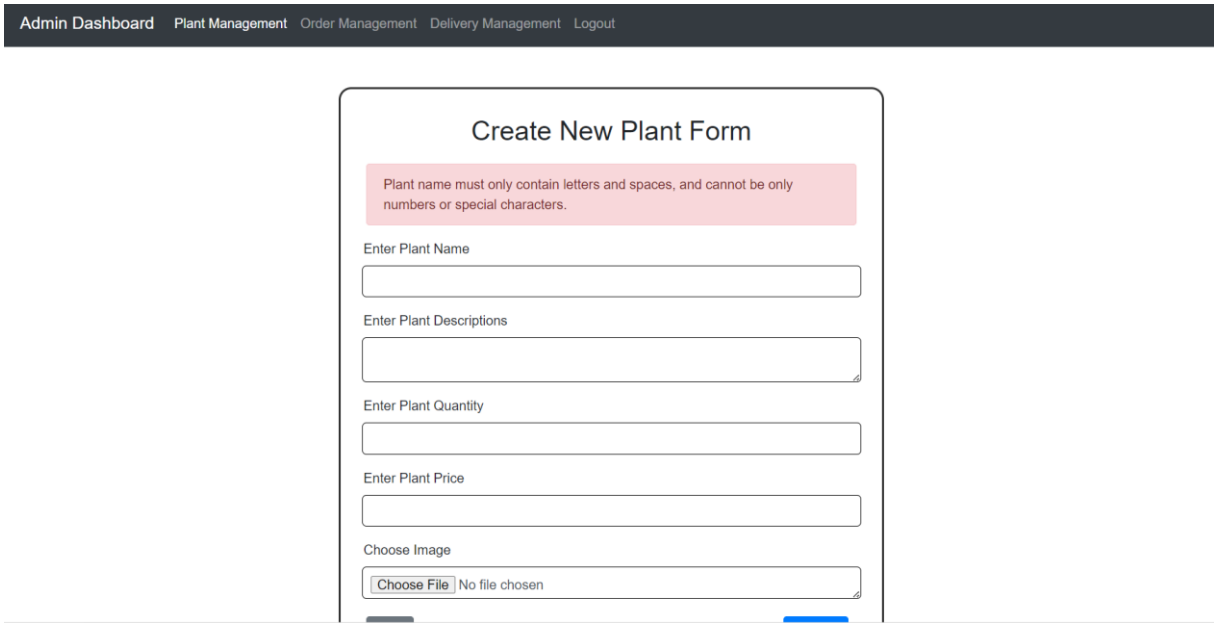



Figure 5.7: Invalid Plant Name Error Message

Figure 5.8 illustrates an attempt to insert an invalid plant quantity into the “Create New Plant Screen”. An error message is displayed, as shown in Figure 5.9, indicating that the plant quantity must be greater than zero.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Jade Plant | Rounded green fleshy leaves with red lining believed to bring wealth and good fortune | 0 | 29.00 | |

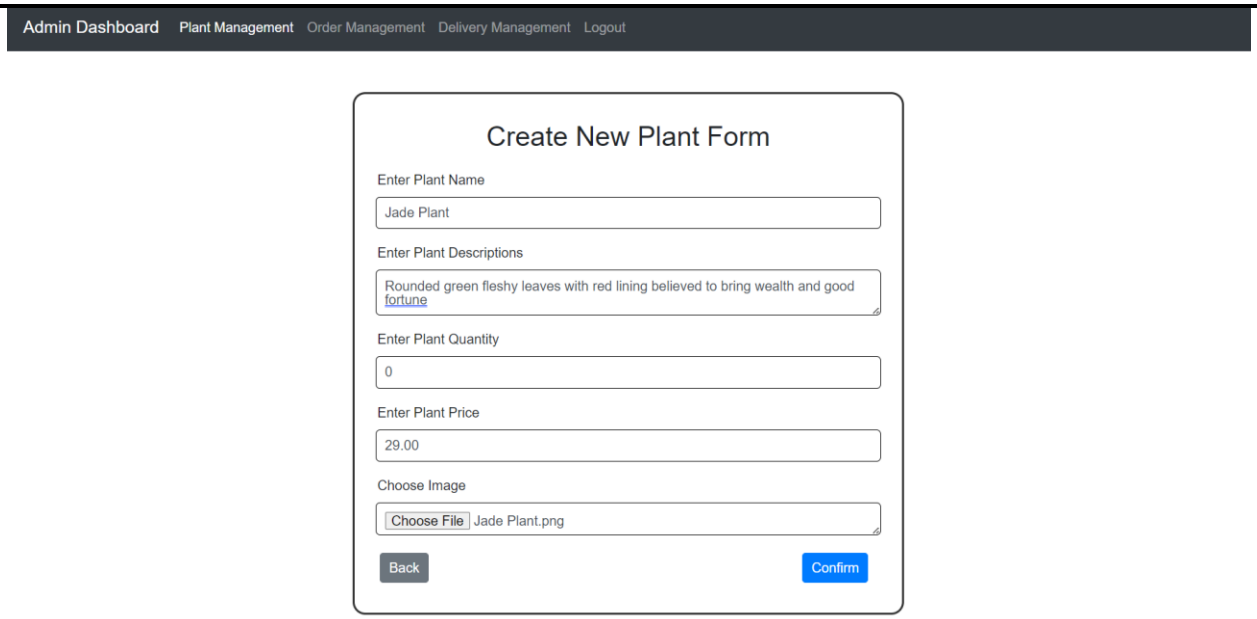



Figure 5.8: Inserting Invalid Quantity

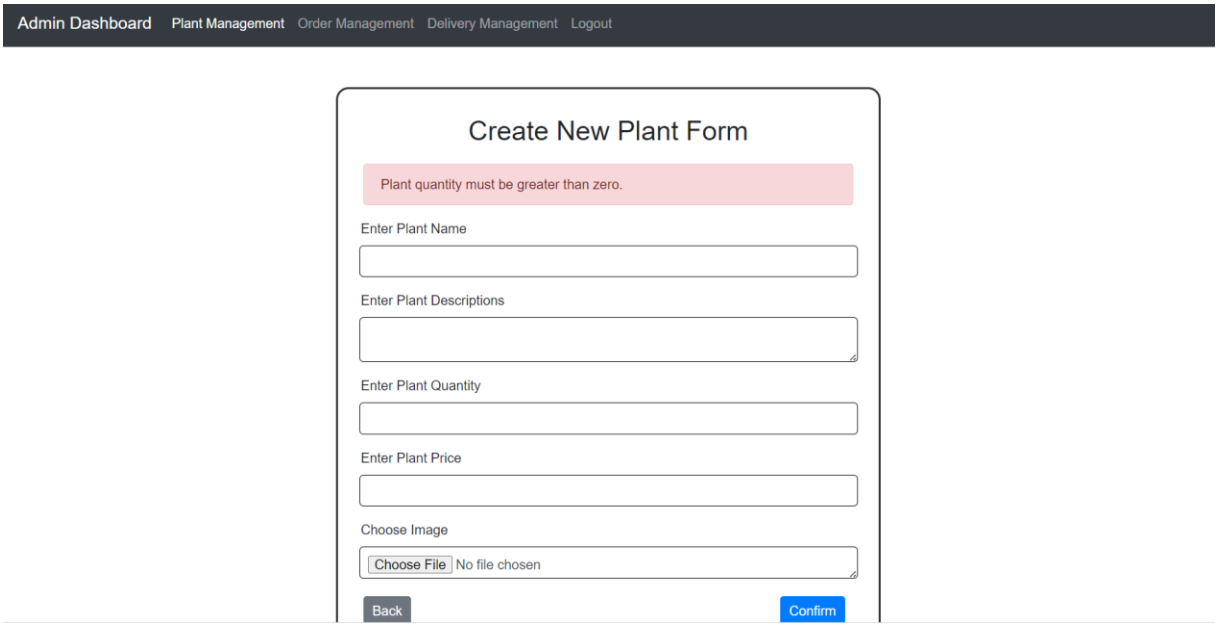



Figure 5.9: Invalid Plant Quantity Error Message

Figure 5.10 illustrates an attempt to insert an invalid plant price into the “Create New Plant Screen”. An error message is displayed, as shown in Figure 5.11, indicating that the plant price must be greater than zero.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Jade Plant | Rounded green fleshy leaves with red lining believed to bring wealth and good fortune | 50 | 0.00 | |

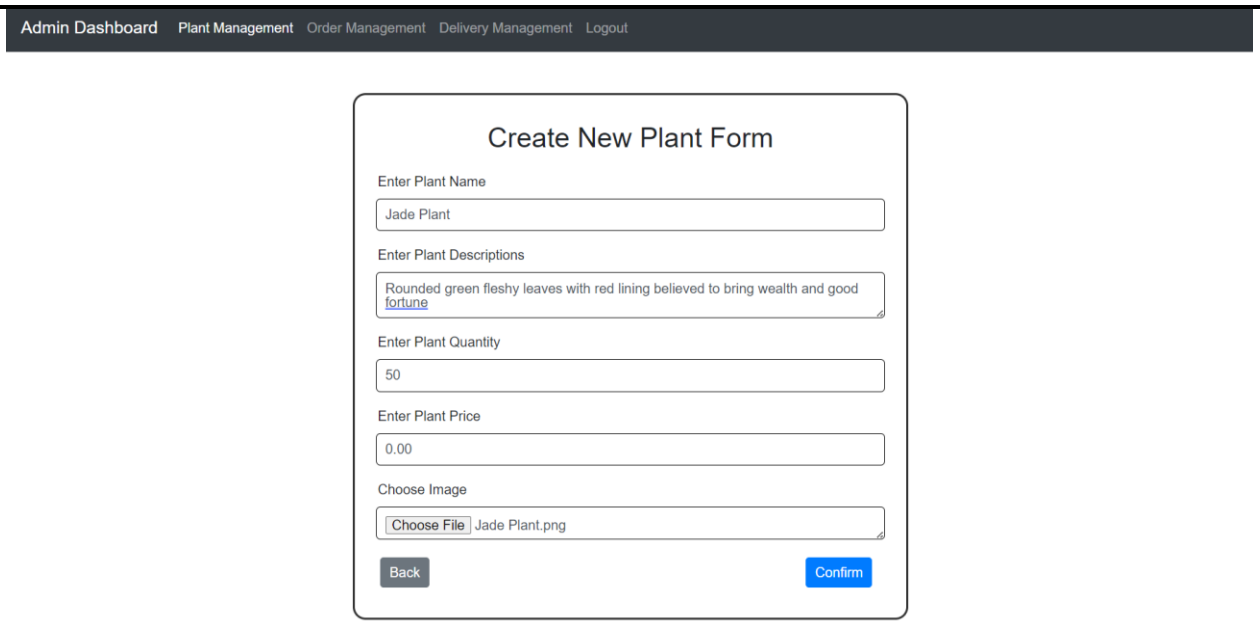



Figure 5.10: Inserting Invalid Price

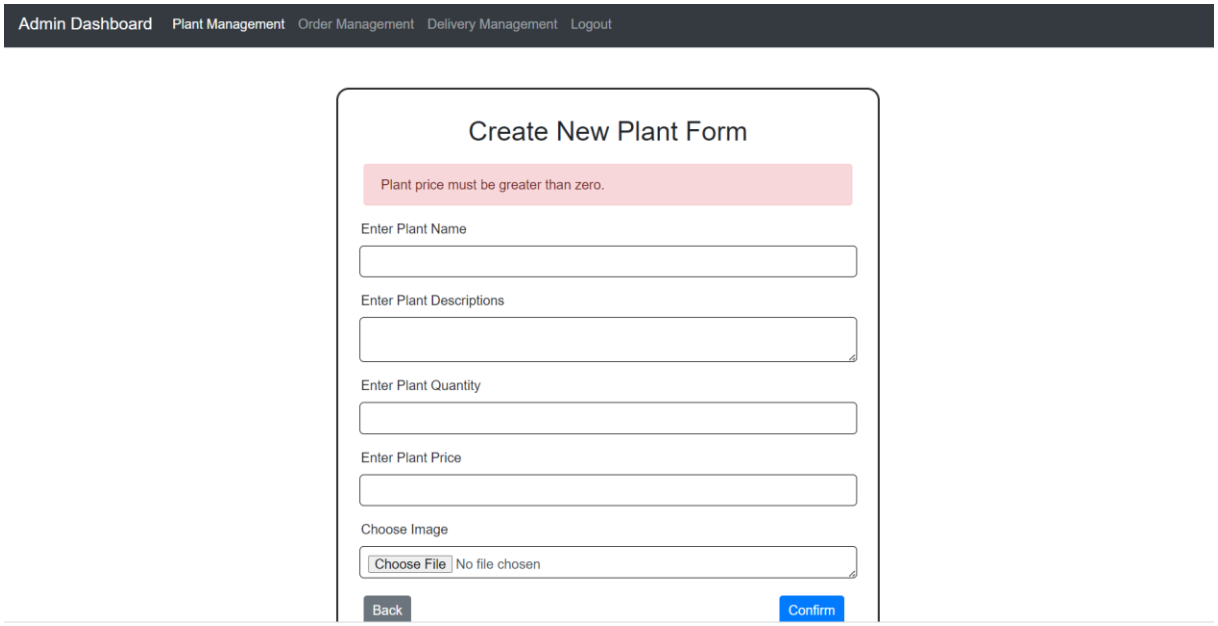



Figure 5.11: Invalid Plant Price Error Message

In Figure 5.12, there is no insertion of a plant image on the “Create New Plant Screen”. An error message is displayed, as shown in Figure 5.13, which states that a plant image must be uploaded.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Jade Plant | Rounded green fleshy leaves with red lining believed to bring wealth and good fortune. | 50 | 29.00 | |

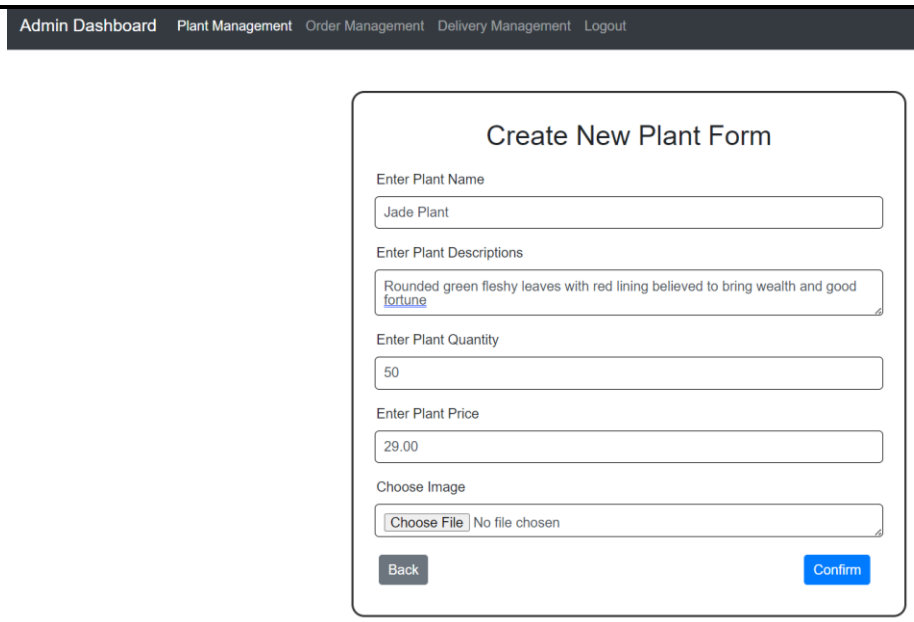



Figure 5.12: No Plant Image Insertion

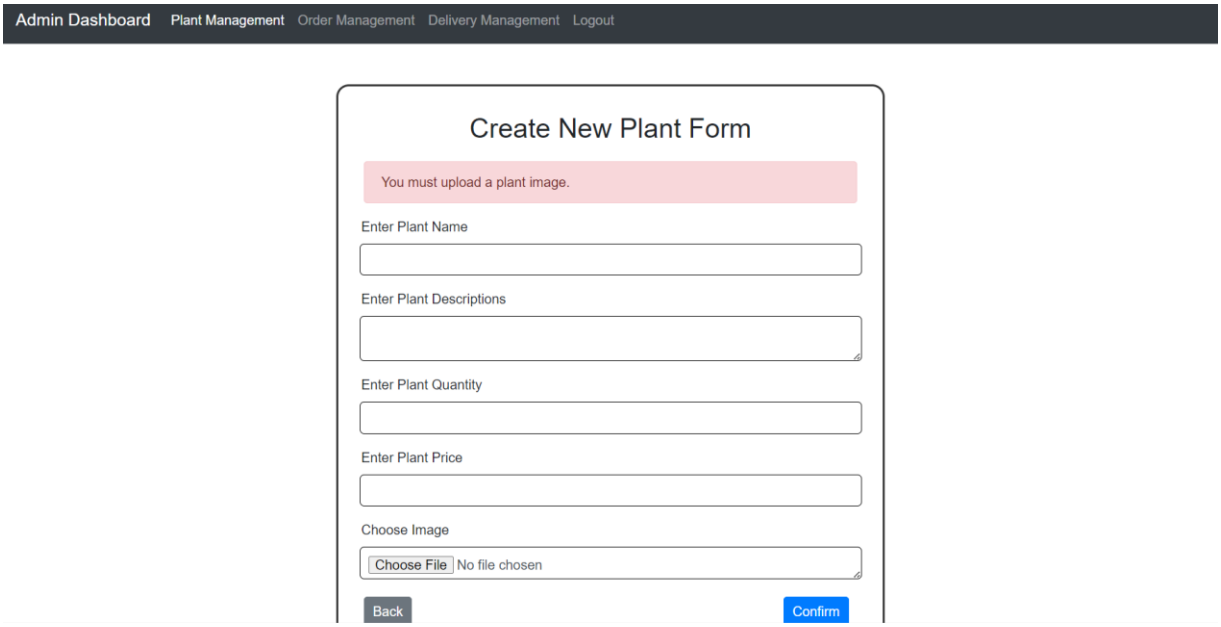



Figure 5.13: Empty Plant Image Error Message

## Update Plant Test Results:

Figure 5.14 shows the initial plant details for “Aloe Vera” in the “Update Plant Screen”. Figure 5.15 displays the insertion of new plant details into the “Update Plant Screen”. After clicking the “Update” button, the administrator is redirected back to the “Plant Management Screen” where a successful plant update message is shown, as illustrated in Figure 5.16. From there, the administrator can view the updated details for “Aloe Vera”.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| aloe vera | Aloe vera is a succulent plant species of the genus Aloe. The plant is stemless with thick, greenish, fleshy leaves that can fan out from the plant's central stem. The margin of the leaf is serrated with small teeth. | 50 | 25.00 | |

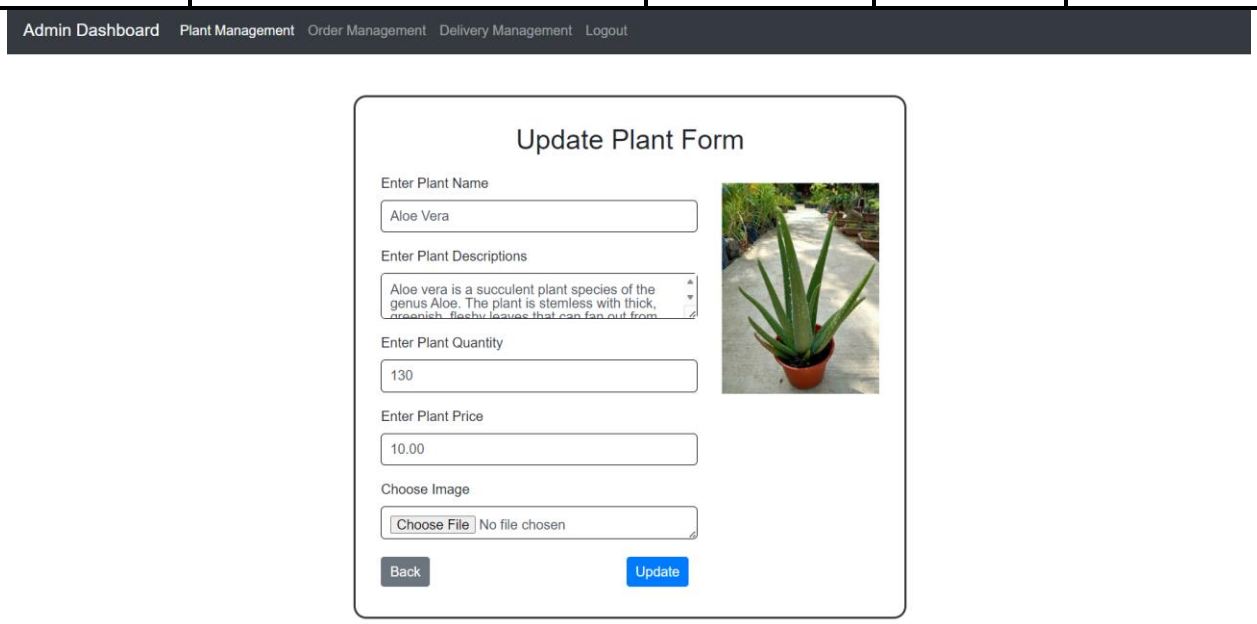



Figure 5.14: Initial Plant Details

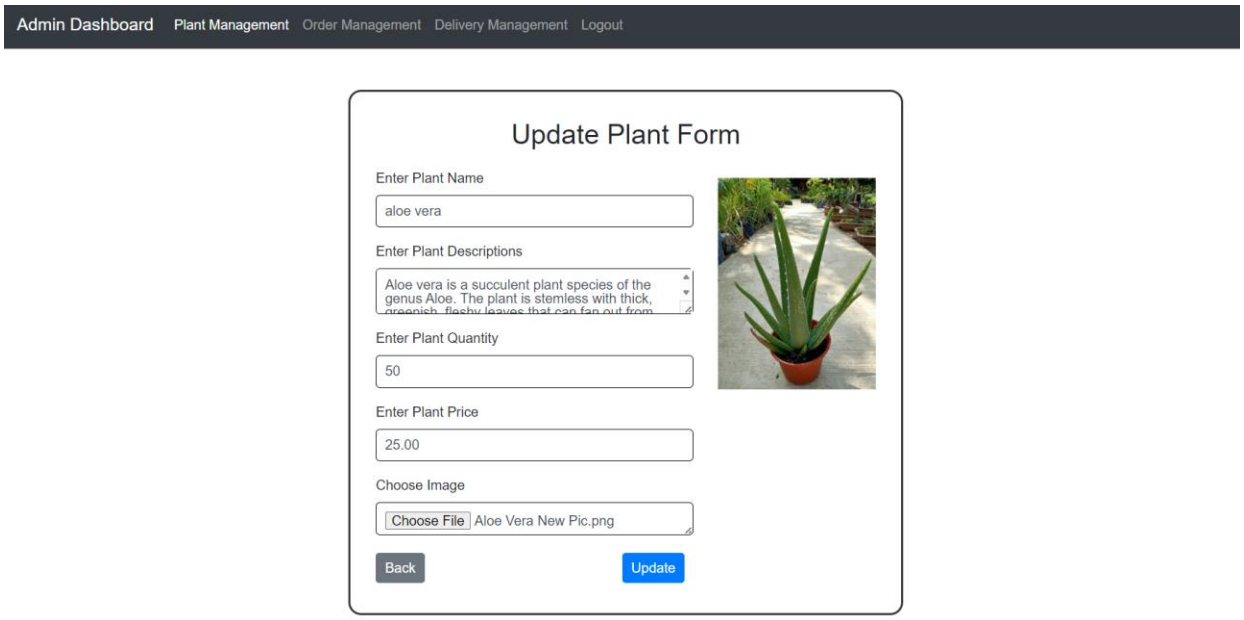



Figure 5.15: Inserting New Plant Details

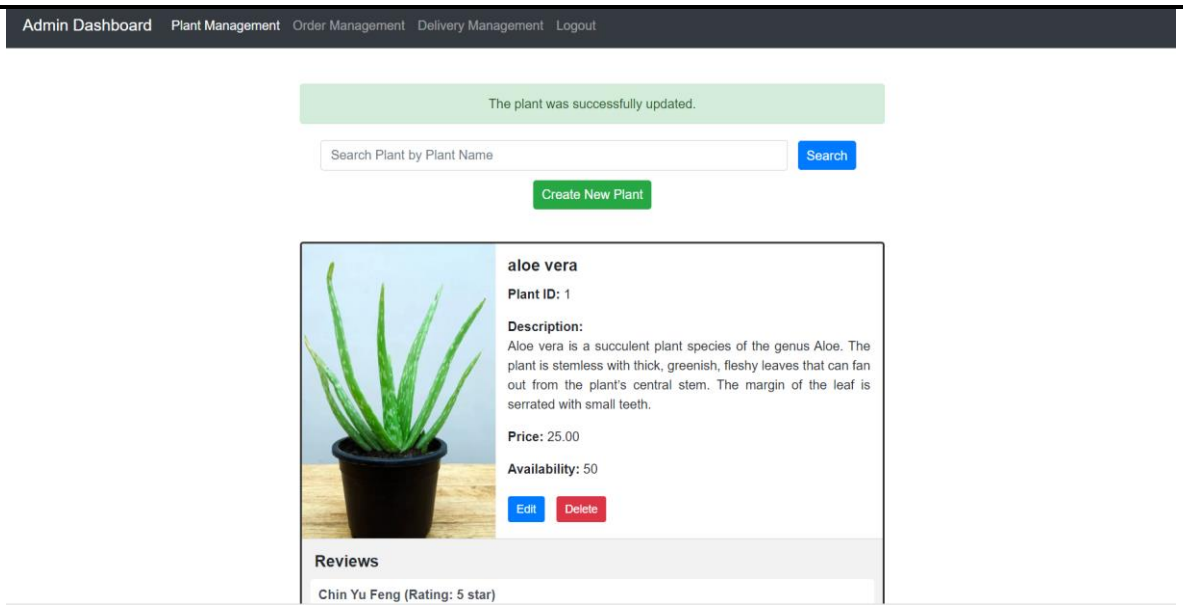



Figure 5.16: Updated Plant Details

Figure 5.17 illustrates the initial plant details for "Sunflower Plant" in the "Update Plant Screen". Figure 5.18 displays the attempt to insert a plant named "Cactus" and its details into the "Update Plant Screen". However, since "Cactus" already exists in the system, an error message is shown, as depicted in Figure 5.19.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Cactus | Cactus is a diverse group of succulent plants known for their unique and often unusual appearance. It typically has fleshy, water-storing stems that are adapted to reduce water loss. | 15 | 55.00 | |

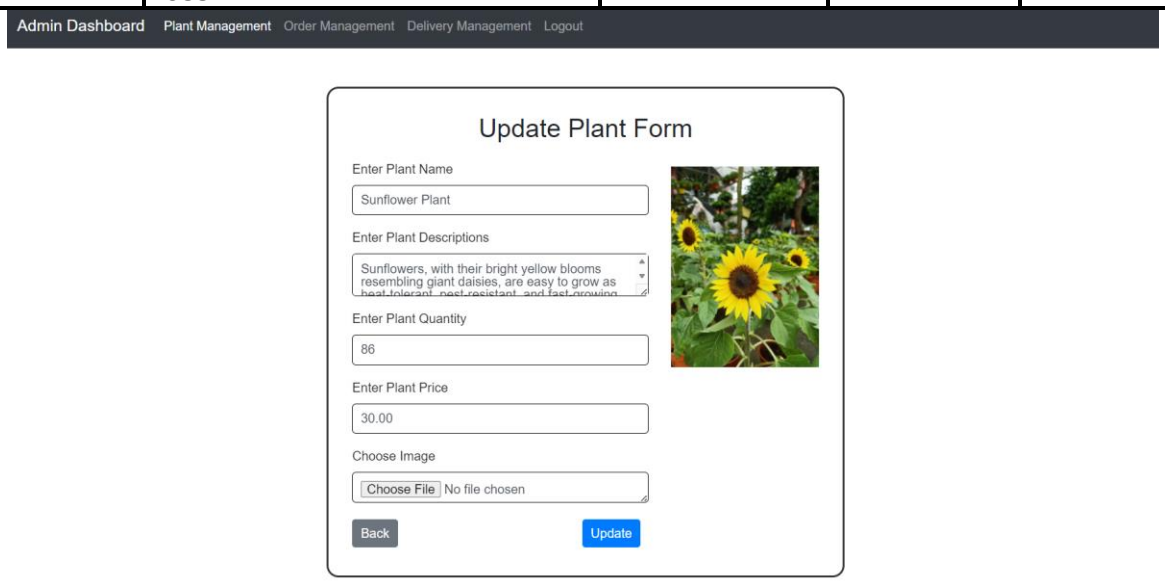



Figure 5.17: Initial Plant Details

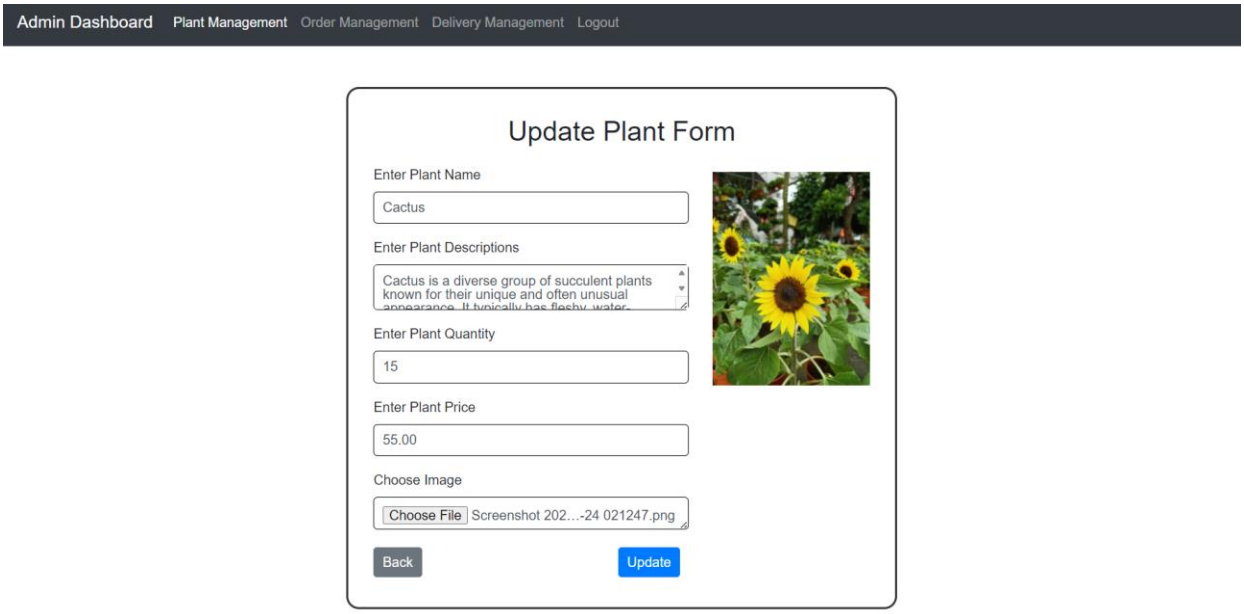



Figure 5.18: Inserting Existed Plant Details

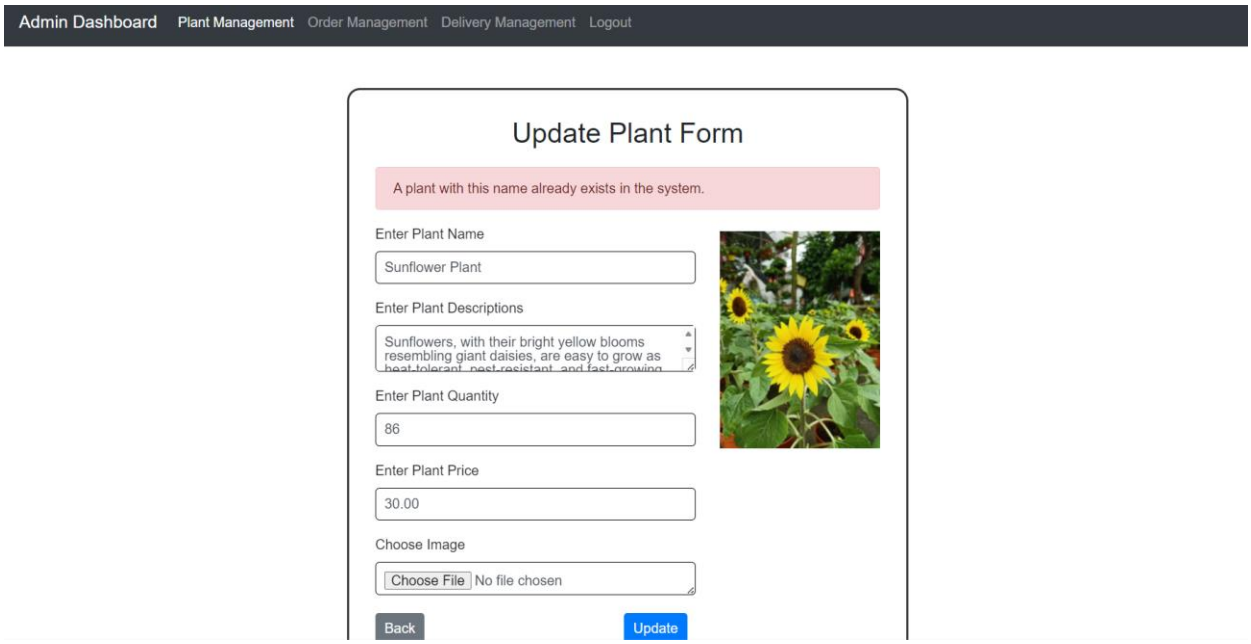



Figure 5.19: Plant Already Exist Error Message

Figure 5.20 illustrates an attempt to insert an invalid plant name into the “Update Plant Screen”. An error message is displayed, as shown in Figure 5.21, stating that plant name must only contain letters and spaces, and cannot be only numbers or special characters.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| 120 | Aloe vera is a succulent plant species of the genus Aloe. The plant is stemless with thick, greenish, fleshy leaves that can fan out from the plant's central stem. The margin of the leaf is serrated with small teeth. | 50 | 20.00 | |

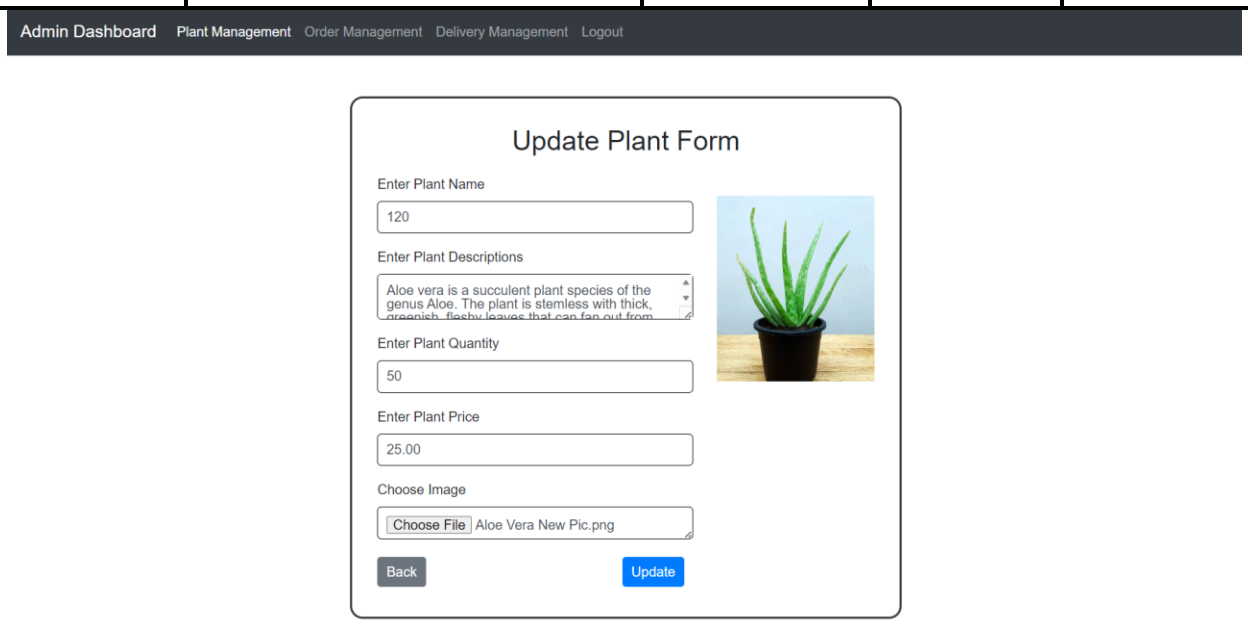



Figure 5.20: Inserting Invalid Plant Name

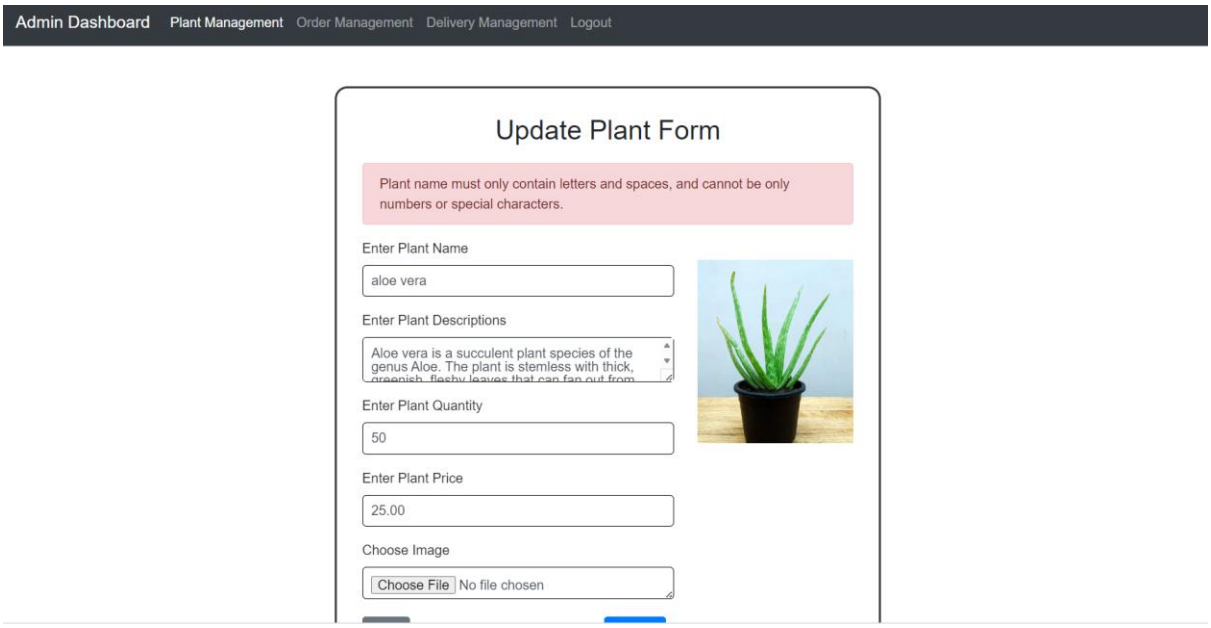



Figure 5.21: Invalid Plant Name Error Message

Figure 5.22 illustrates an attempt to insert an invalid plant price into the “Update Plant Screen”. An error message is displayed, as shown in Figure 5.23, indicating that the plant price must be greater than zero.

| Plant Name | Plant Description | Plant Quantity | Plant Price | Plant Image |
|---|---|---|---|---|
| Aloe Vera | Aloe vera is a succulent plant species of the genus Aloe. The plant is stemless with thick, greenish, fleshy leaves that can fan out from the plant’s central stem. The margin of the leaf is serrated with small teeth. | 50 | 0.00 | |

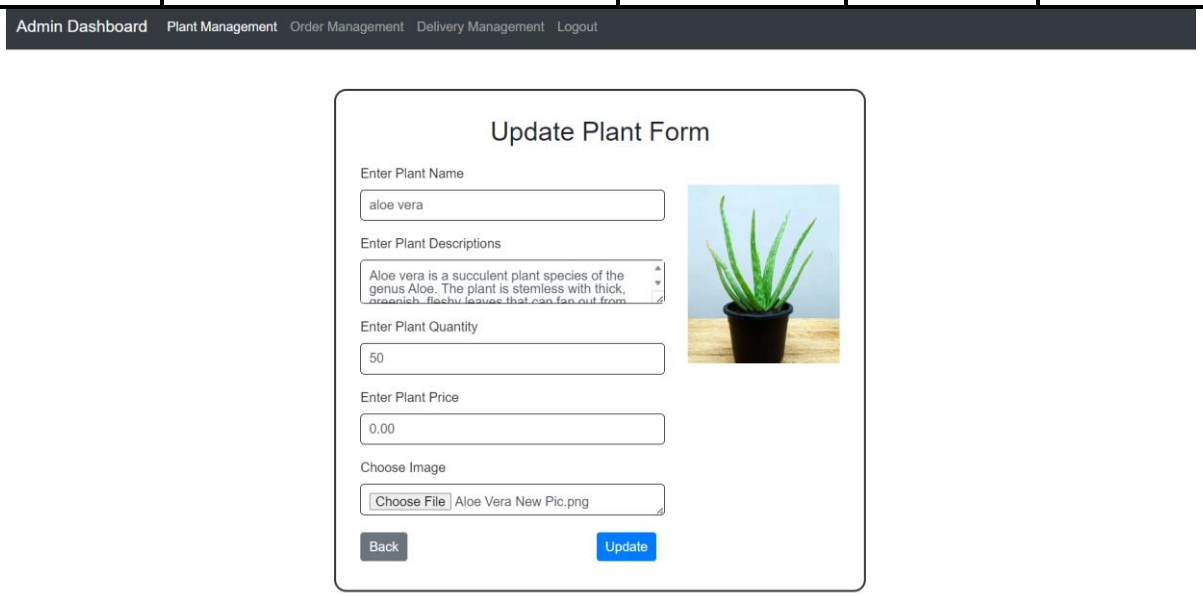



Figure 5.22: Inserting Invalid Plant Price

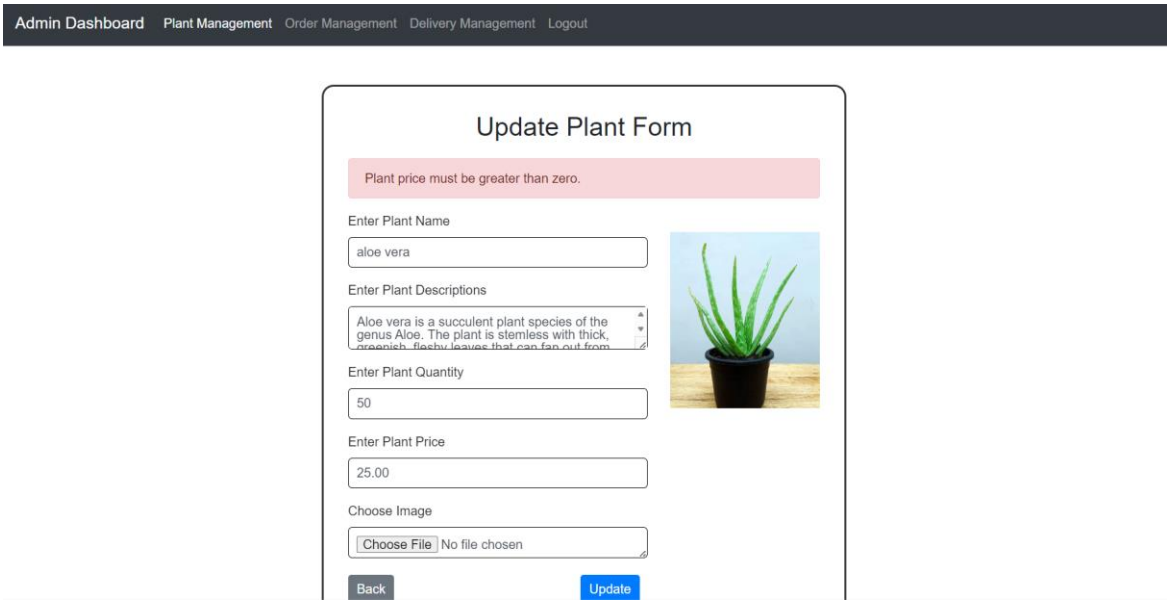



Figure 5.23: Invalid Plant Price Error Message

## Delete Plant Test Results:

When the “Delete” button of the “Money Tree” plant record is clicked, a plant deletion confirmation message appears on the screen, as depicted in Figure 5.24. Upon clicking the “OK” button, a successful plant deletion message is displayed, as shown in Figure 5.25. Subsequently, the “Money Tree” plant record is removed from the screen, as illustrated in Figure 5.26.

| **Plant Name** |
| --- |
| Money Tree |

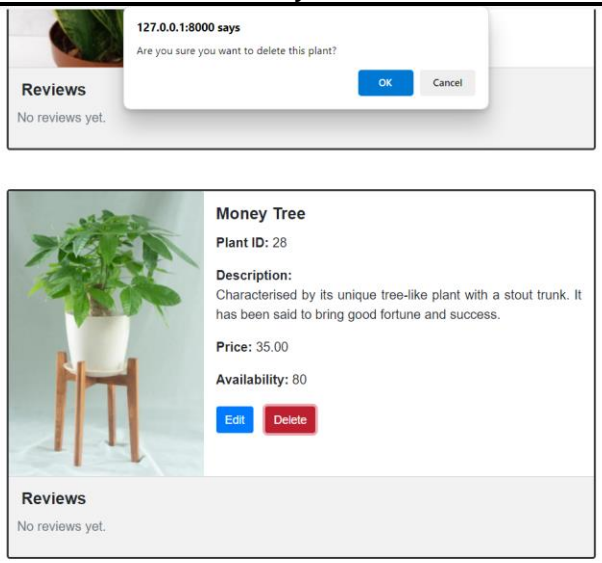



Figure 5.24: Plant Deletion Confirmation Message

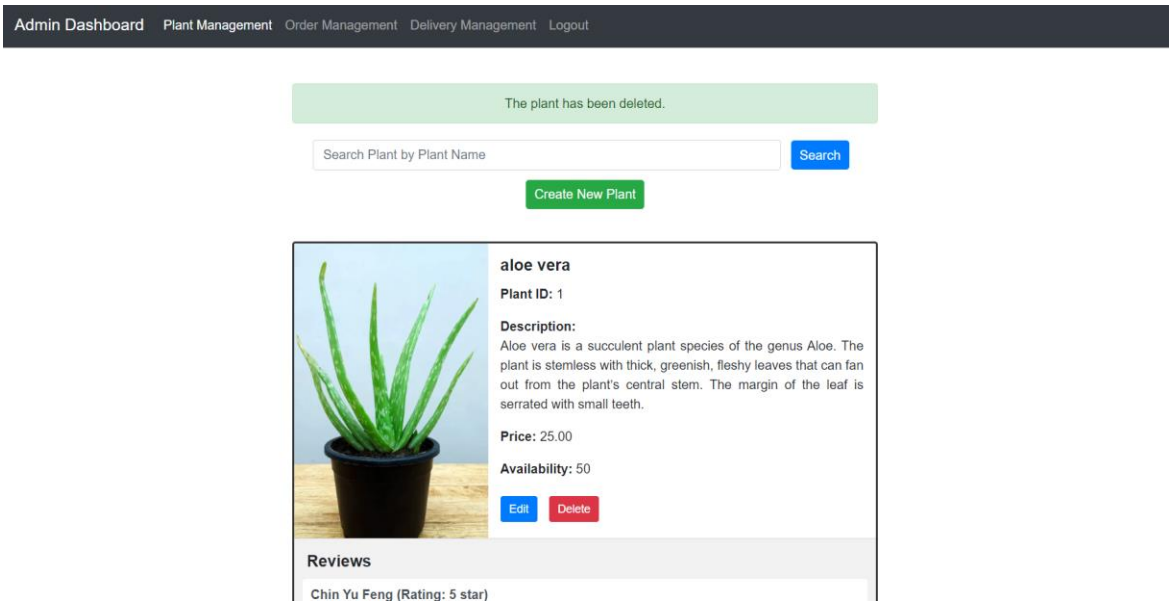



Figure 5.25: Successful Plant Deletion Message

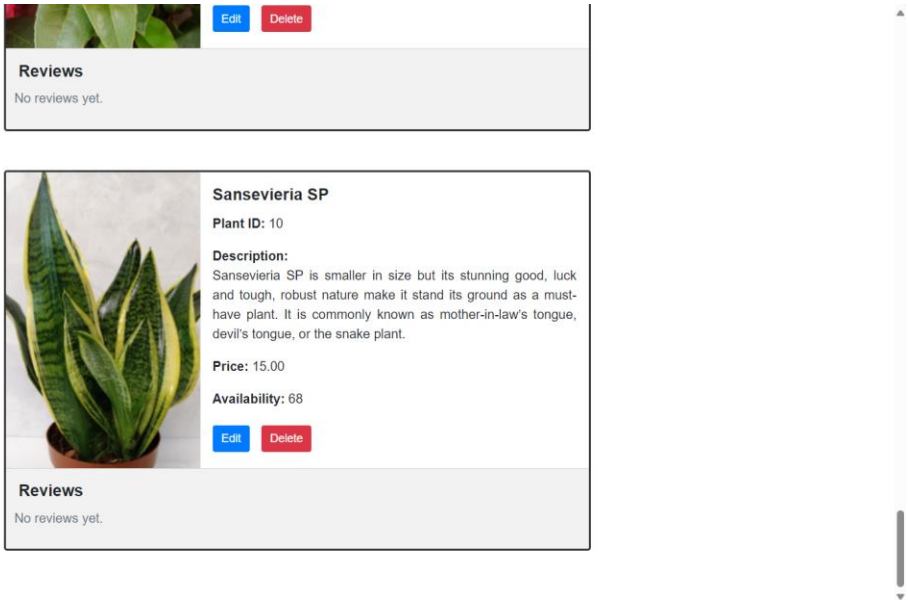



Figure 5.26: Plant Record Removed

## Search Plant Test Results:

In Figure 5.27, the input "s" is entered into the search bar. After clicking on the "Search" button, all plant records with plant names containing "s" are displayed on the screen, as depicted in Figure 5.28.

| Plant Name |
| --- |
| s |

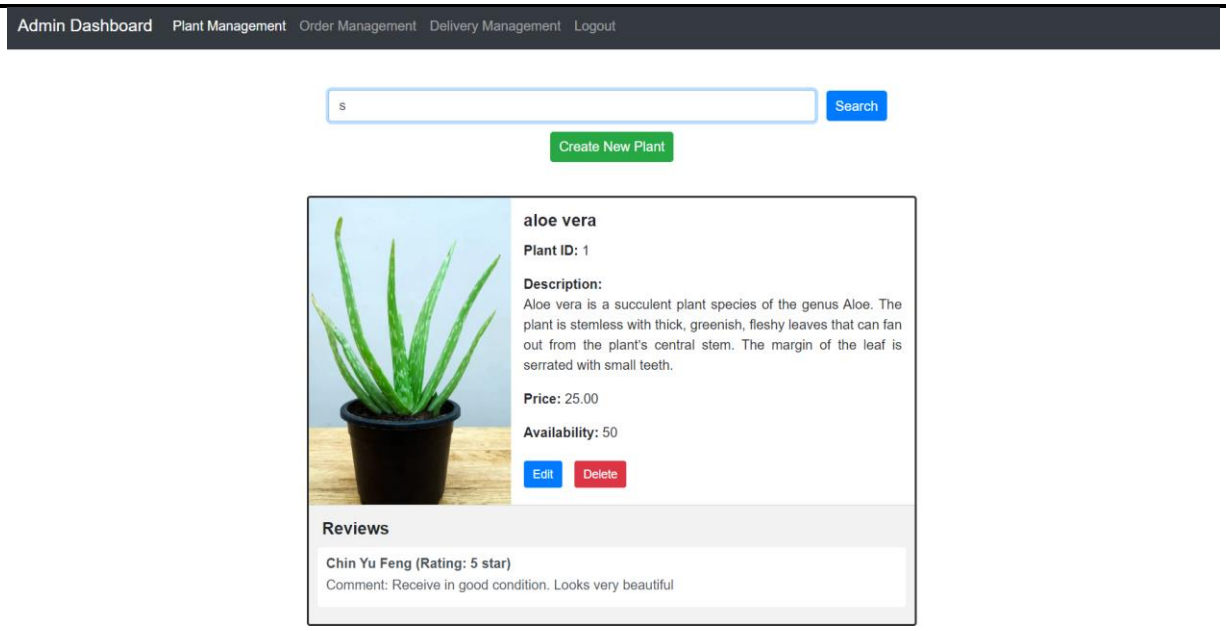



Figure 5.27: Search Plant Using "s"

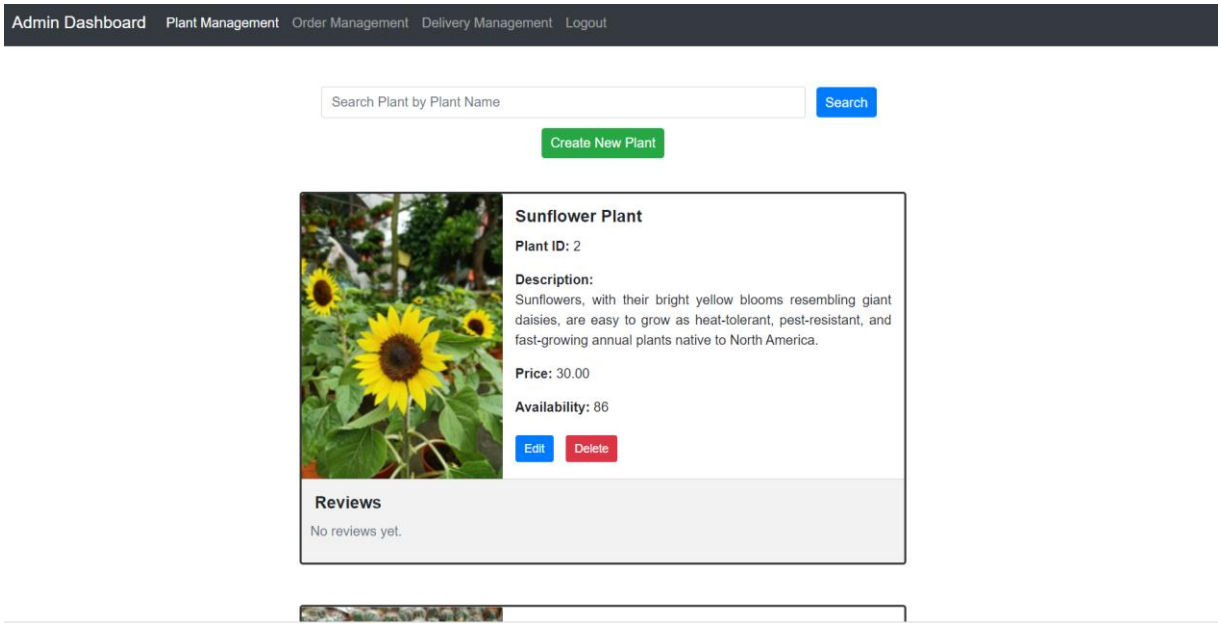



Figure 5.28: Plant Records With “s”

In Figure 5.29, the input “na” is entered into the search bar. Upon clicking the “Search” button, an error message is displayed on the screen, as depicted in Figure 5.30, indicating that no matching plants were found.

| **Plant Name** |
| --- |
| na |

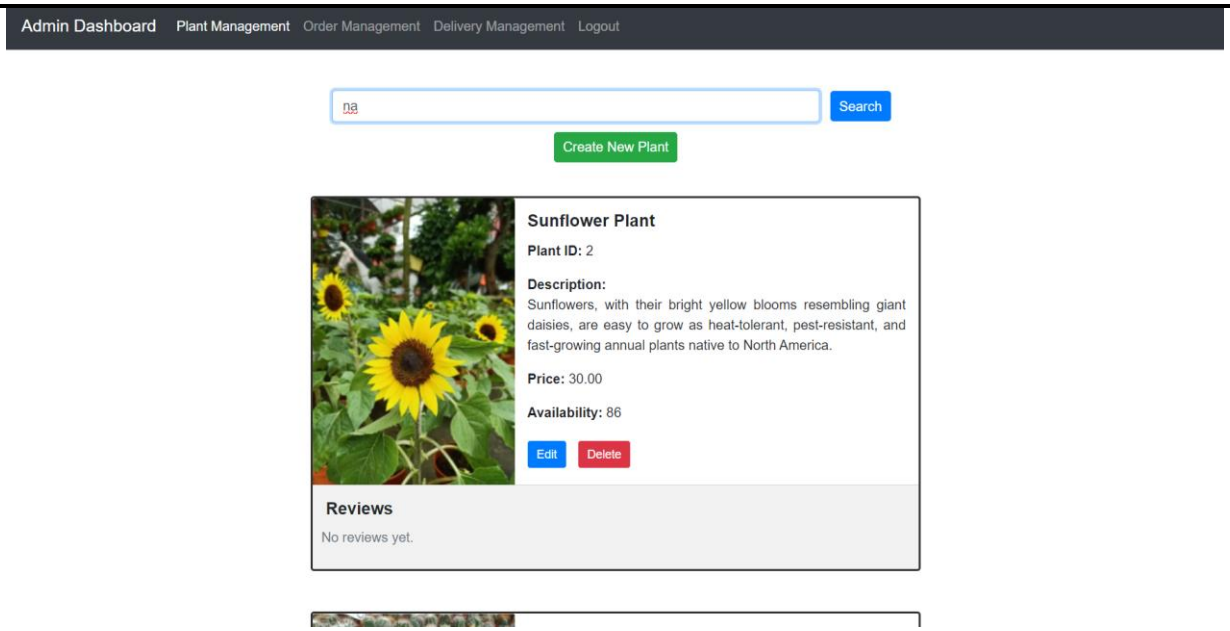



Figure 5.29: Search Plant Using “na”

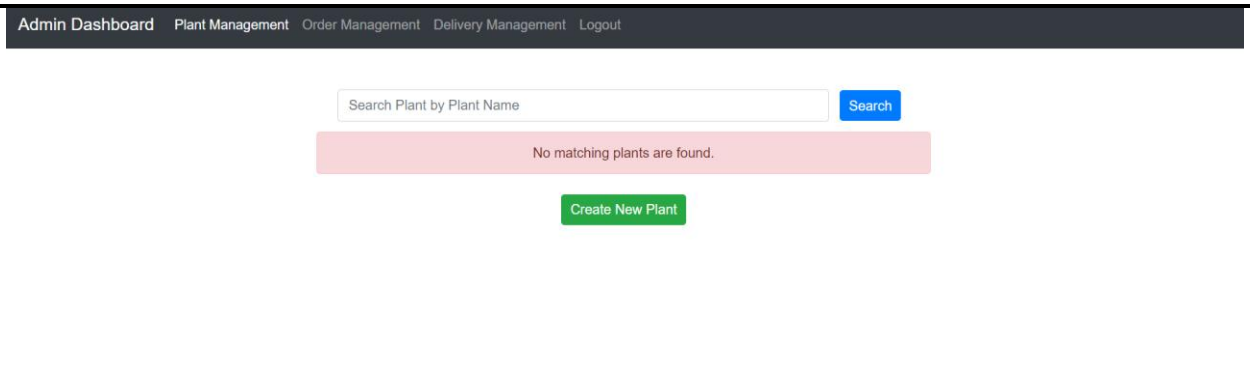



Figure 5.30: No Matching Plants Error Message

After clicking on the “Search” button without entering any input into the search bar, all the plants and their details will be displayed on the screen, as depicted in Figure 5.31.

| **Plant Name** |
| --- |
| (blank) |

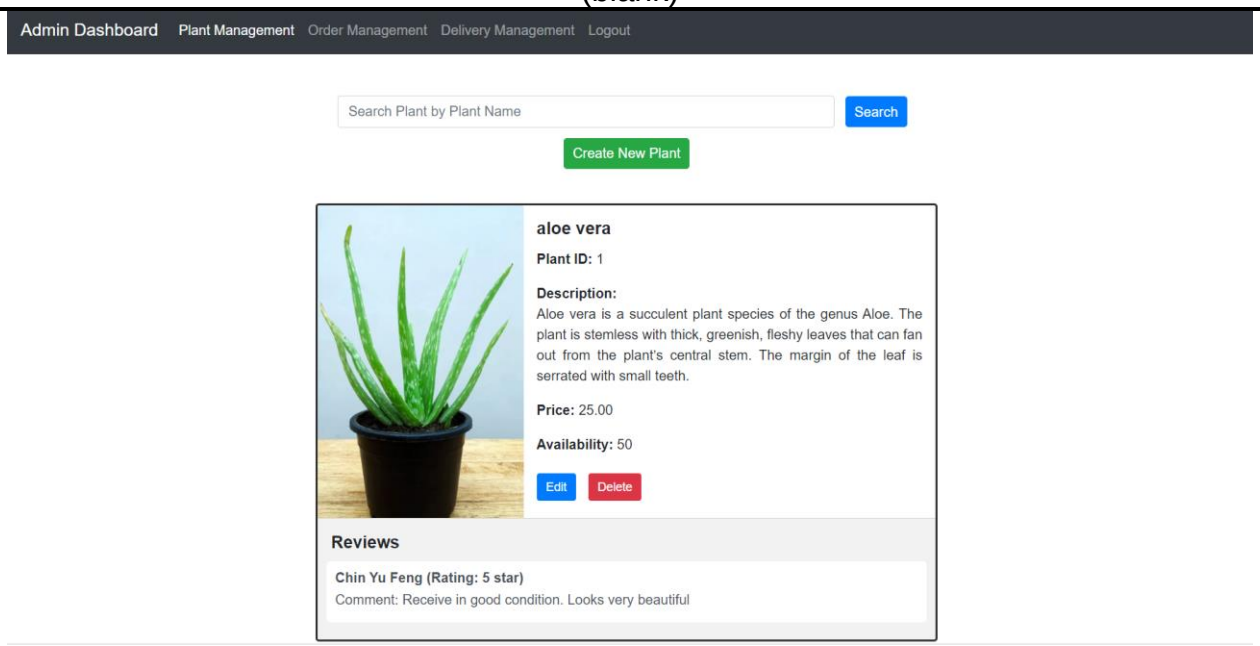



Figure 5.31: Search Plant Without Input

In Figure 5.32, the input “/” is entered into the search bar. Upon clicking the “Search” button, an error message is displayed on the screen, as depicted in Figure 5.33, indicating that no matching plants were found.

| **Plant Name** |
| --- |
| / |

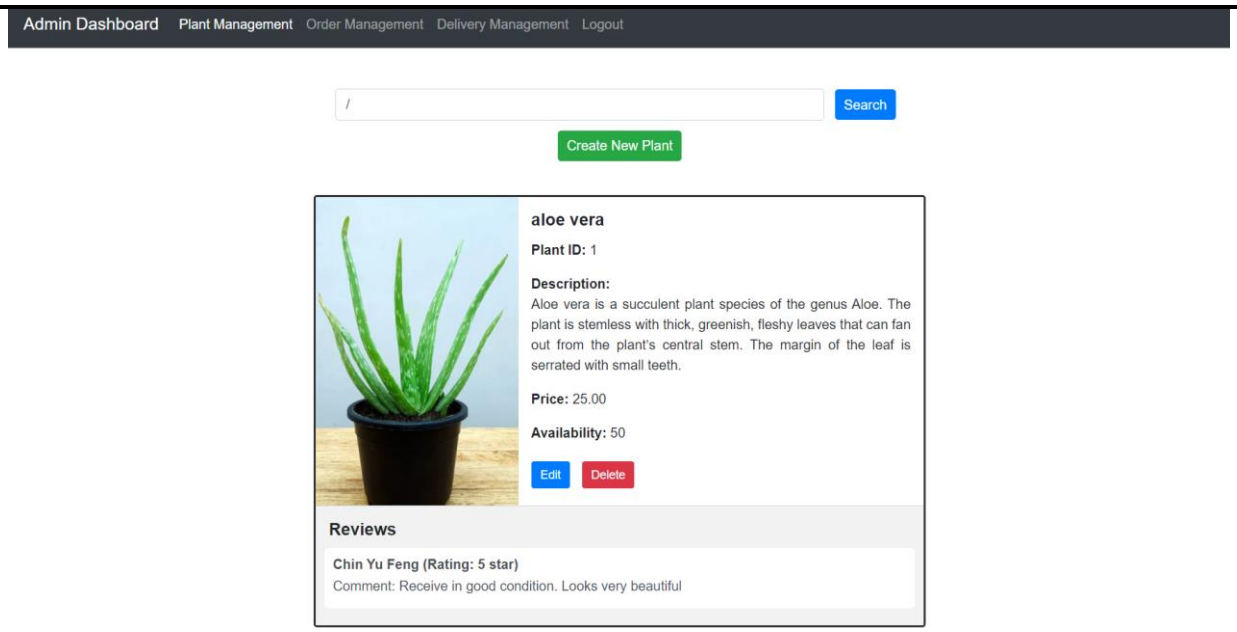



Figure 5.32: Search Plant Using "/"

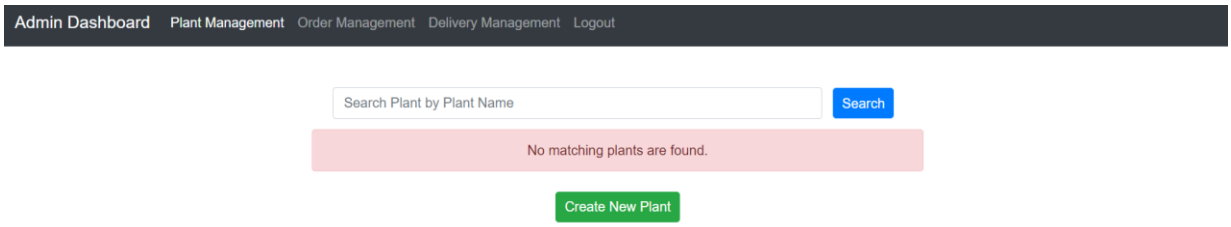



Figure 5.33: No Matching Plants Error Message

In Figure 5.34, the input "20" is entered into the search bar. Upon clicking the "Search" button, an error message is displayed on the screen, as depicted in Figure 5.35, indicating that no matching plants were found.

| **Plant Name** |
| --- |
| 20 |

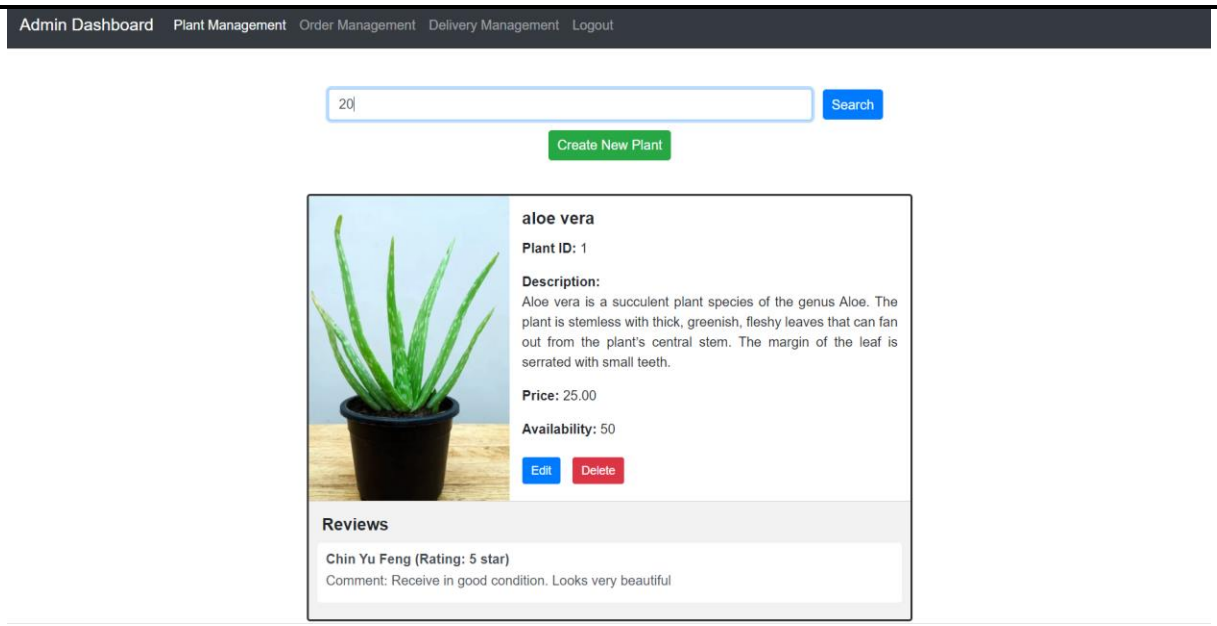



Figure 5.34: Search Plant Using "20"

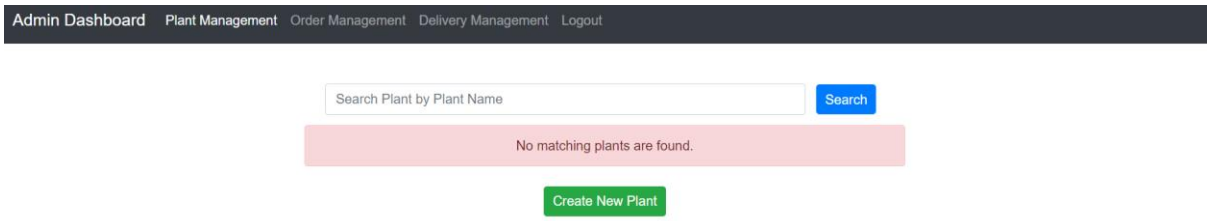



Figure 5.35: No Matching Plants Error Message

## Search Order Test Results:

| In Figure 5.36, the input "43" is entered into the search bar. When the "Search" button is clicked, the order with ID 43 is displayed on the screen, as shown in Figure 5.37. |
|---|
| **Order ID** |
| 43 |

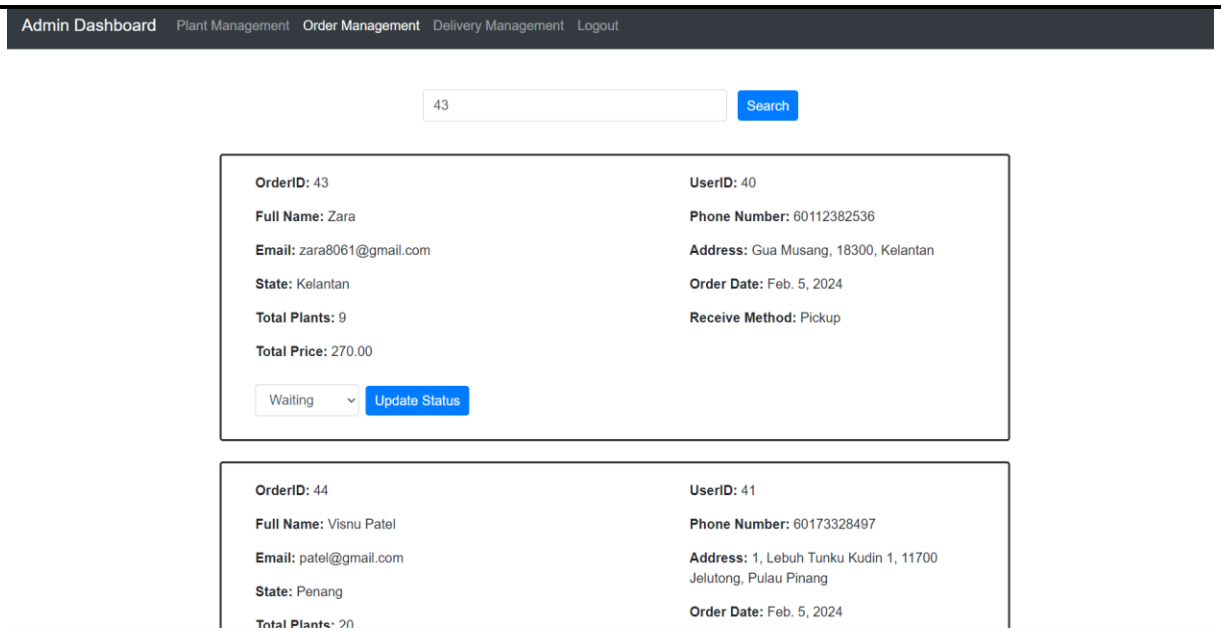



Figure 5.36: Search Order Using "43"

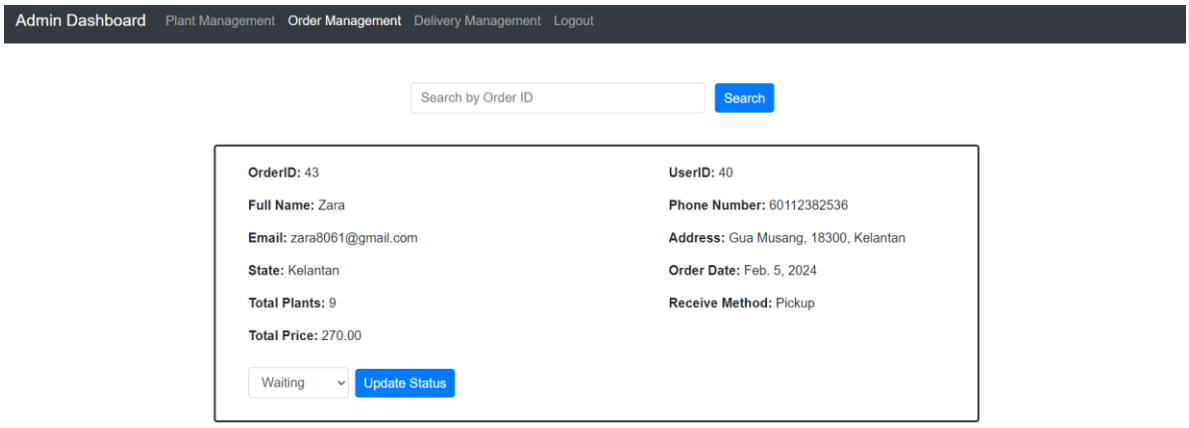



Figure 5.37: Order with ID 43 Shown

In Figure 5.38, the input "20" is entered into the search bar. Upon clicking the "Search" button, an error message is displayed on the screen, as shown in Figure 5.39, indicating that no ongoing orders match the entered order ID.

| **Order ID** |
|---|
| 20 |

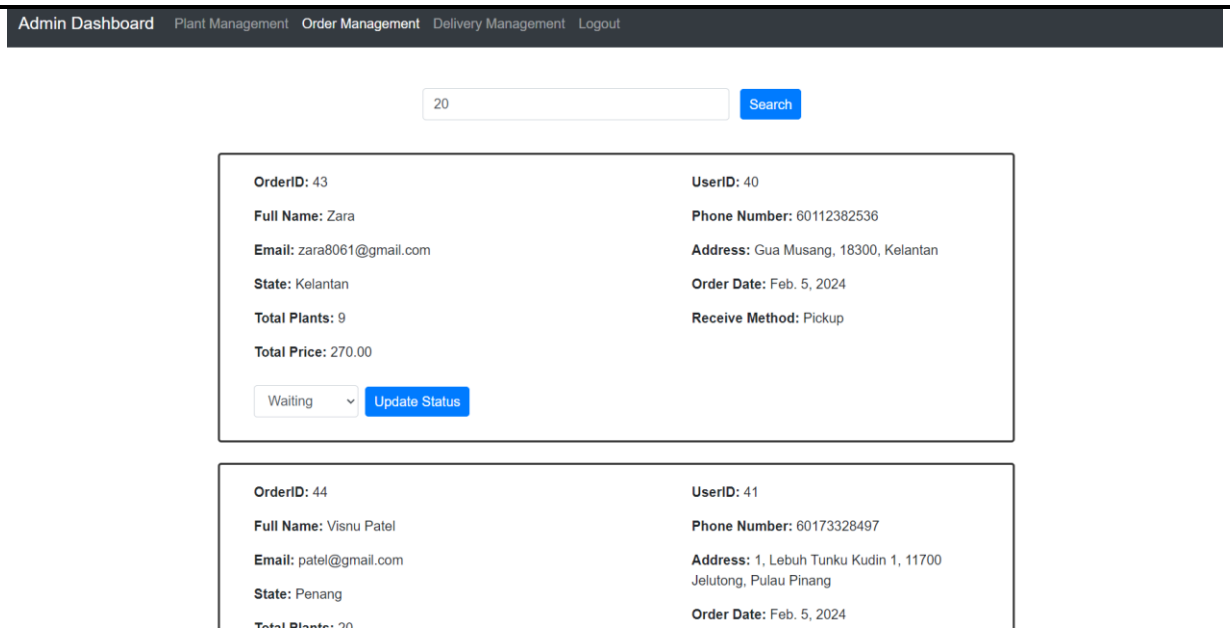



Figure 5.38: Search Order Using "20"

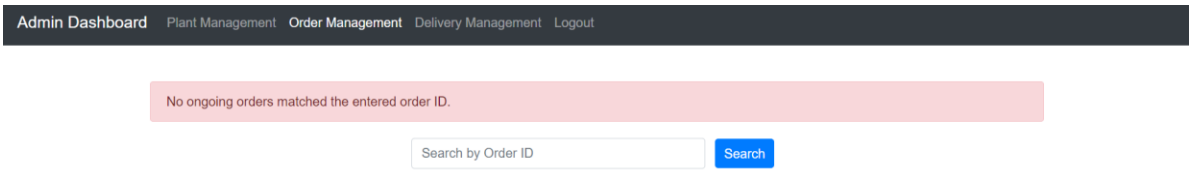



Figure 5.39: No Matching Orders Error Message

After clicking on the "Search" button without entering any input into the search bar, all the orders and their details will be displayed on the screen, as depicted in Figure 5.40.

| **Order ID** |
| --- |
| (blank) |

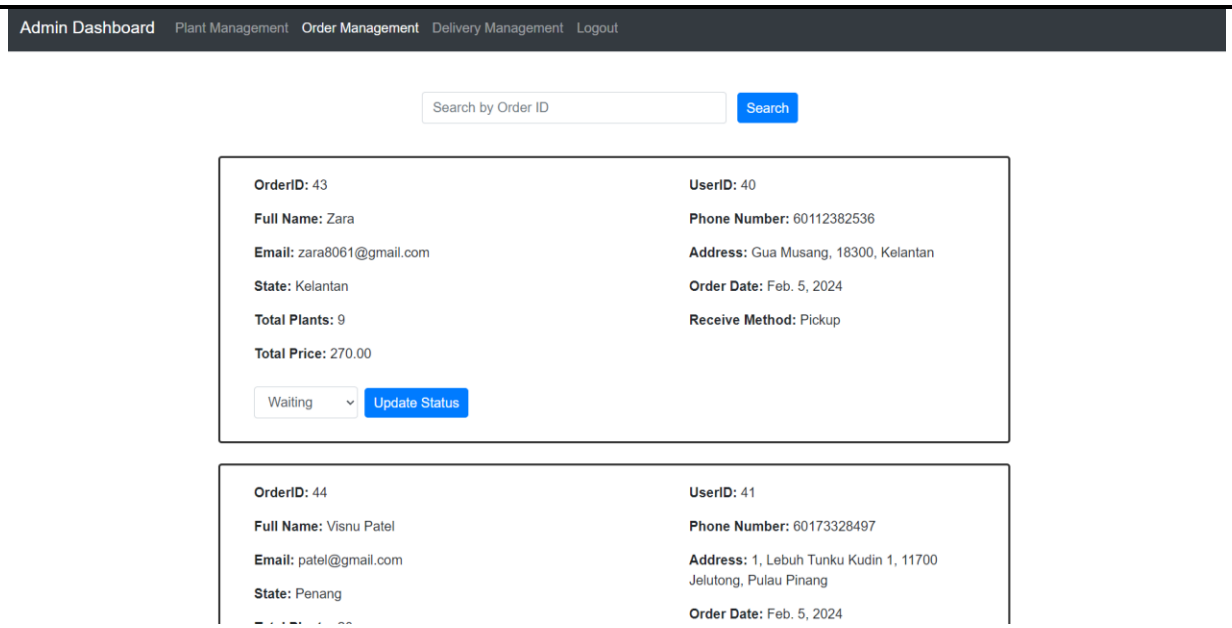



Figure 5.40: Search Order Without Input

In Figure 5.41, the input “a” is entered into the search bar. When the “Search” button is clicked, an error message is displayed on the screen, as shown in Figure 5.42. The message states that the input is invalid and requests the re-entry of a numeric order ID.

| **Order ID** |
| --- |
| a |

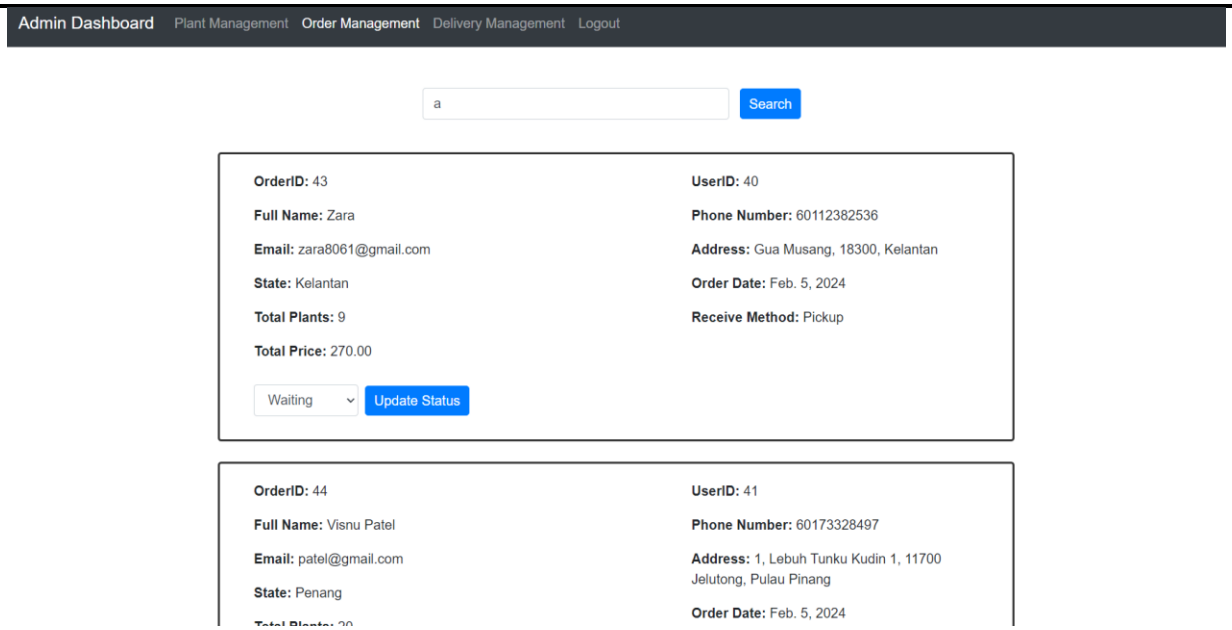



Figure 5.41: Search Order Using “a”

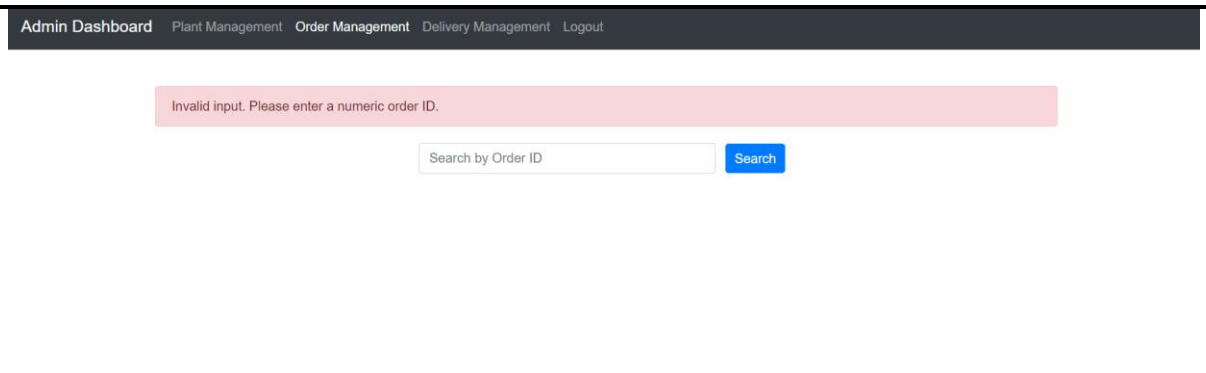



Figure 5.42: Invalid Input Error Message

In Figure 5.43, the input “43w” is entered into the search bar. When the “Search” button is clicked, an error message is displayed on the screen, as shown in Figure 5.44. The message states that the input is invalid and requests the re-entry of a numeric order ID.

| **Order ID** |
|---|
| 43w |

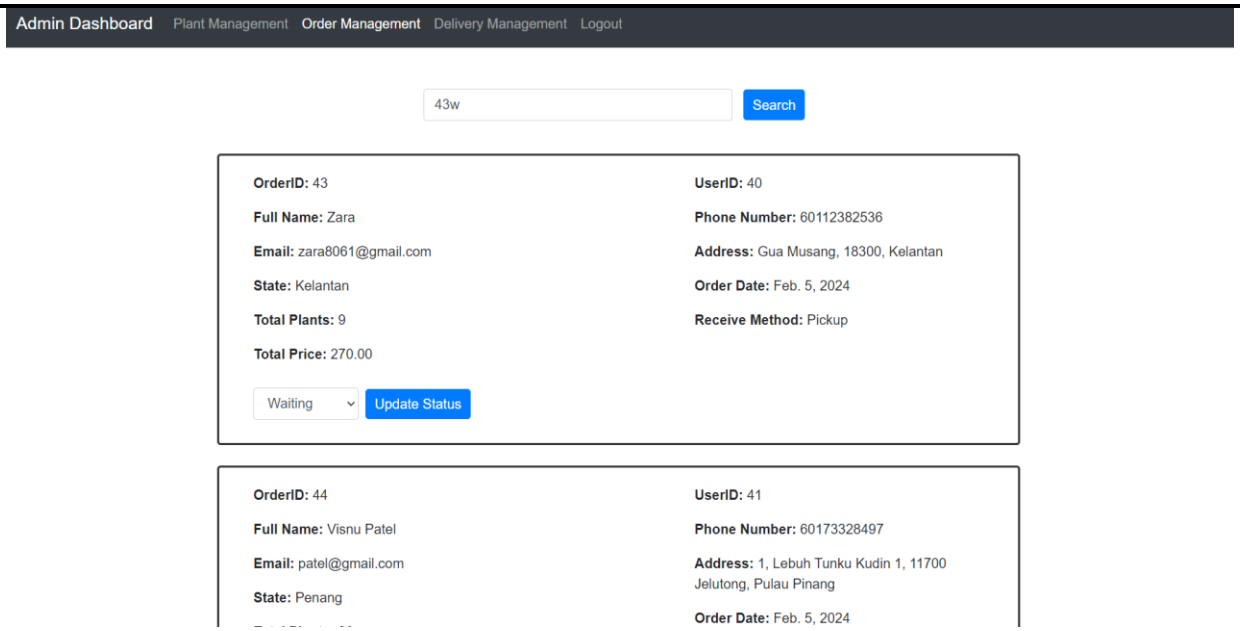



Figure 5.43: Search Order Using “43w”

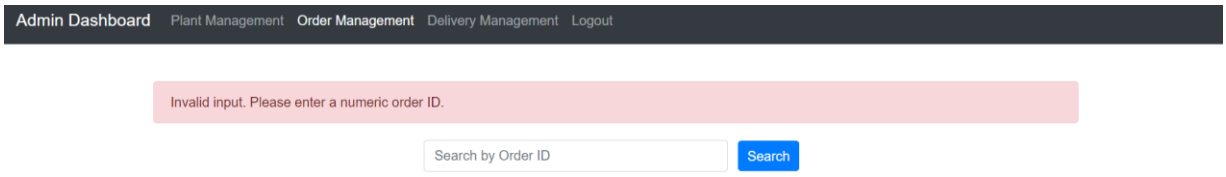



Figure 5.44: Invalid Input Error Message

In Figure 5.45, the input "/" is entered into the search bar. When the "Search" button is clicked, an error message is displayed on the screen, as shown in Figure 5.46. The message states that the input is invalid and requests the re-entry of a numeric order ID.

| Order ID |
|---|
| / |

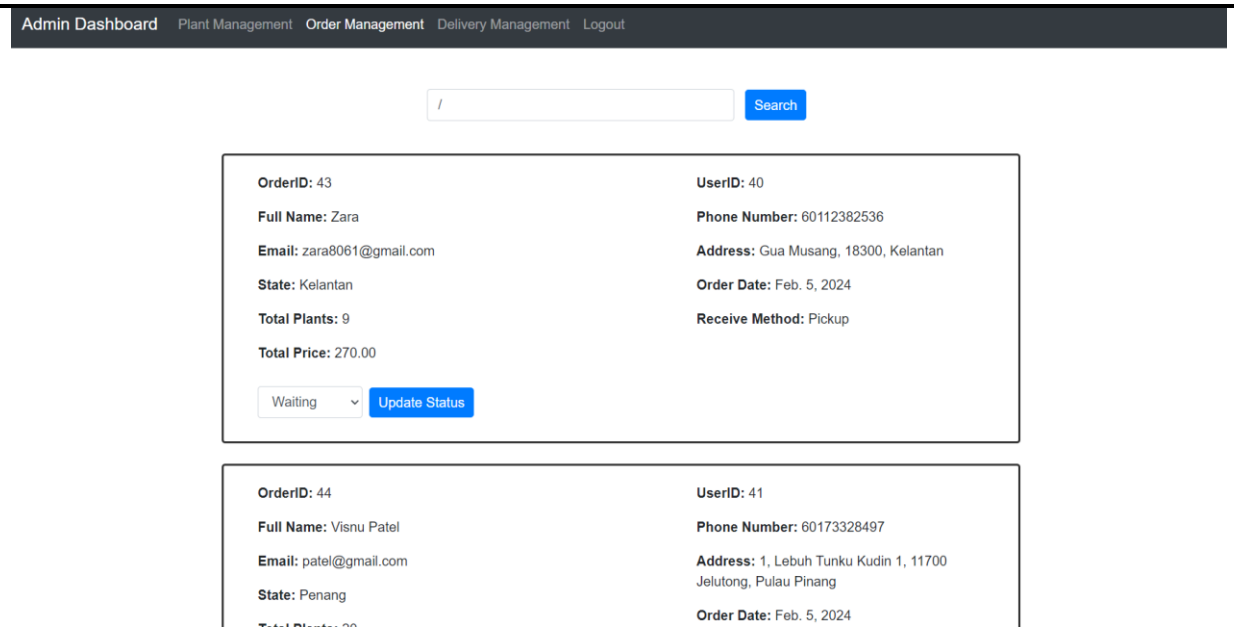



Figure 5.45: Search Order Using "/"

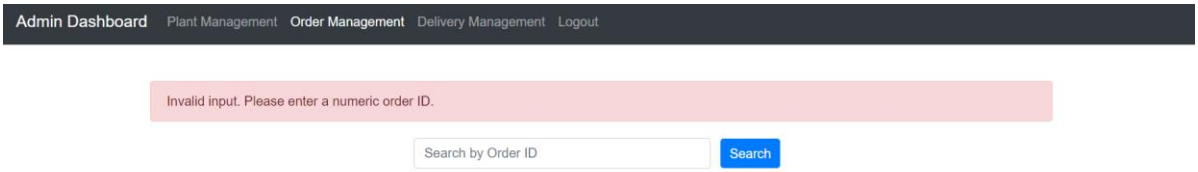



Figure 5.46: Invalid Input Error Message

## Update Order Status Test Results:

In Figure 5.47, the status "Ready" is selected for the pickup order with ID 43. After clicking the "Update Status" button, the "Order Management Screen" is refreshed, and now the pickup order with ID 43 has the status "Ready", as shown in Figure 5.48.

| Order ID | Order Status (Update to) |
|---|---|
| 43 | Ready |

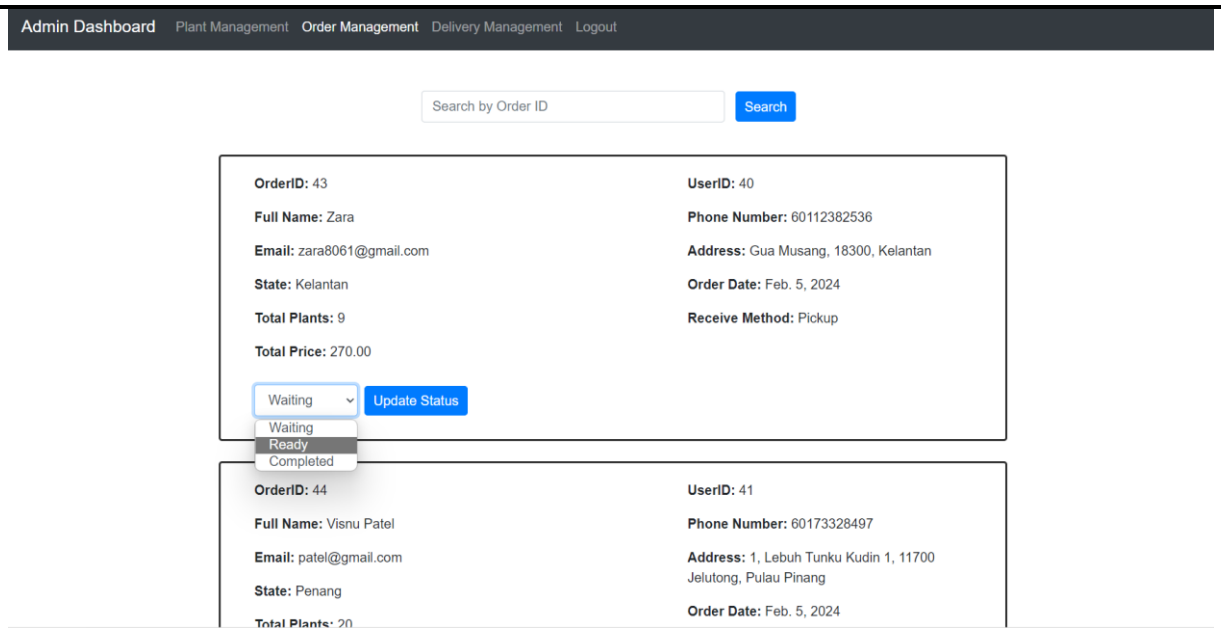



Figure 5.47: Status "Ready" Selected For Pickup Order With ID 43

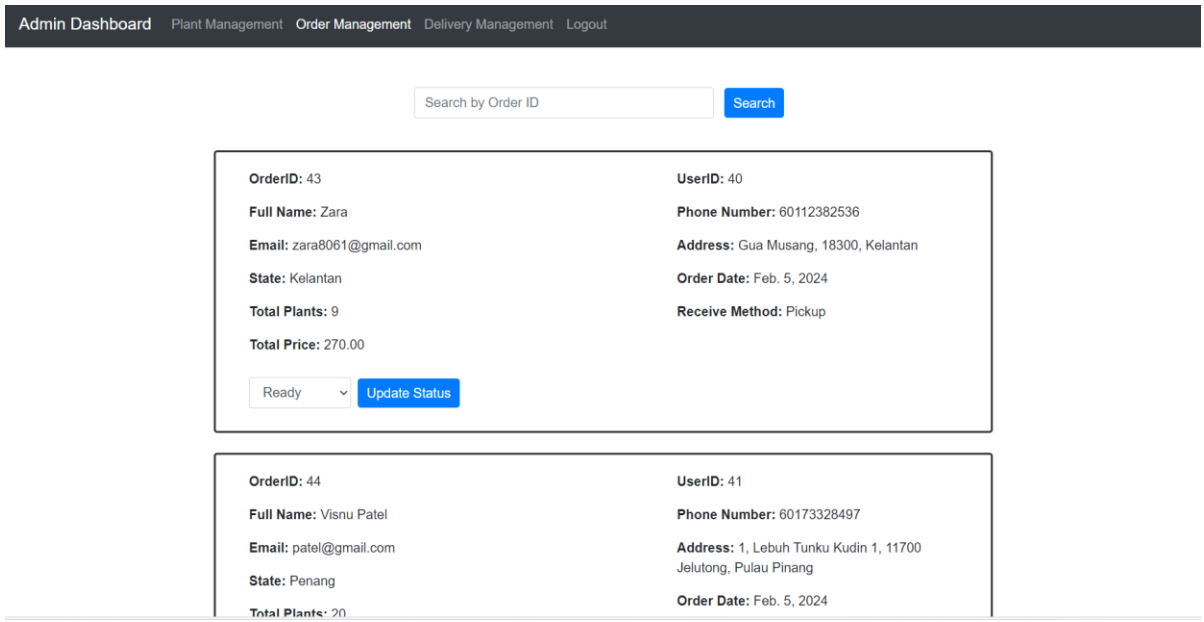



Figure 5.48: Pickup Order With ID 43 Status Updated To Ready

In Figure 5.49, the status "Completed" is selected for the pickup order with ID 43. After clicking the "Update Status" button, the "Order Management Screen" is refreshed, and the order with ID 43 is removed from the screen as it has already been completed, as shown in Figure 5.50.

| Order ID | Order Status (Update to) |
|---|---|
| 43 | Completed |

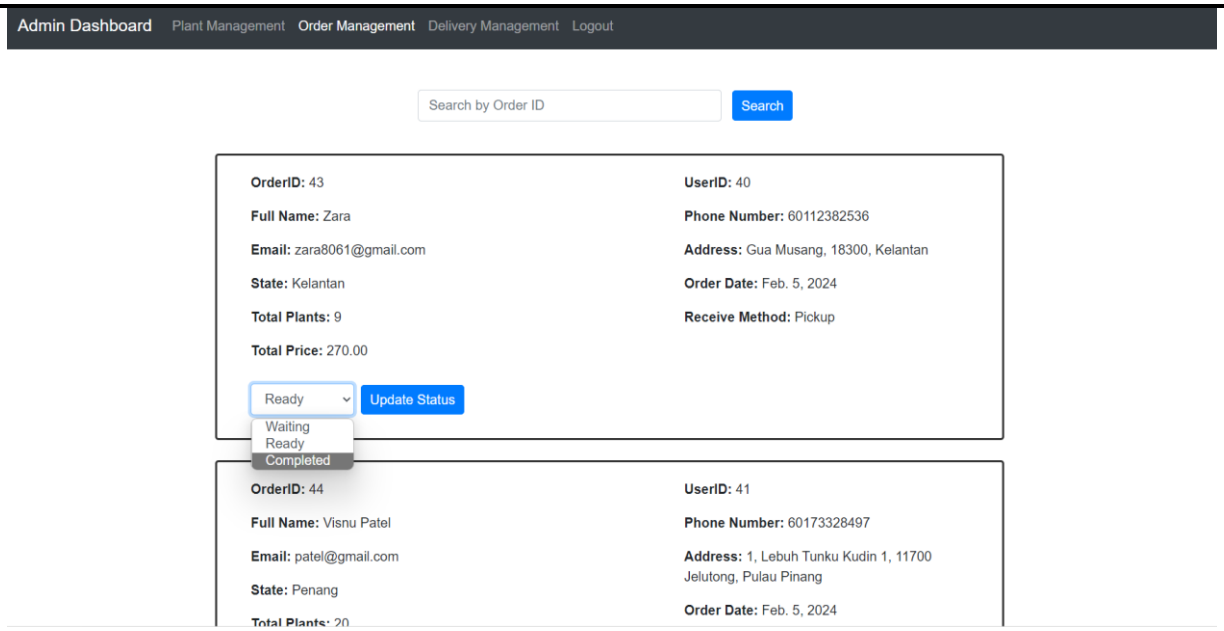



Figure 5.49: Status "Completed" Selected For Pickup Order With ID 43

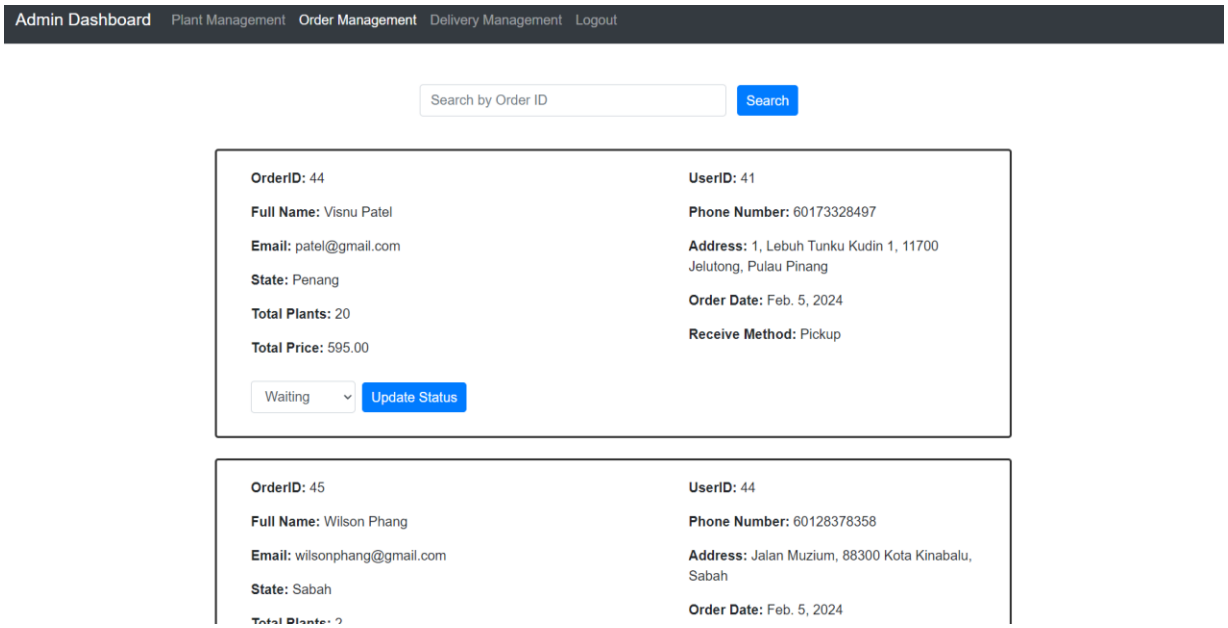



Figure 5.50: Pickup Order With ID 43 Removed From Screen

In Figure 5.51, the status "Ready" is selected for the delivery order with ID 45. After clicking the "Update Status" button, the "Order Management Screen" is refreshed, and the delivery order with ID 45 is removed from the screen, as depicted in Figure 5.52. It will later appear on the "Delivery Management Screen" for assignment to a delivery man.

| Order ID | Order Status (Update to) |
|---|---|
| 45 | Ready |

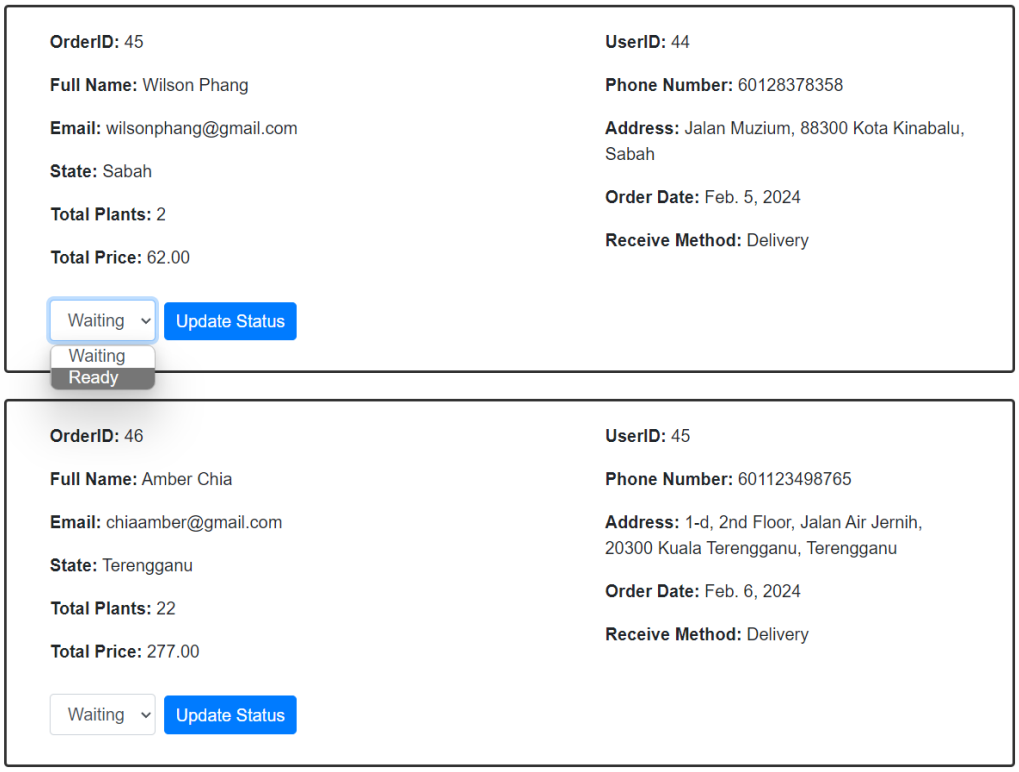



Figure 5.51: Status "Ready" Selected For Delivery Order With ID 45

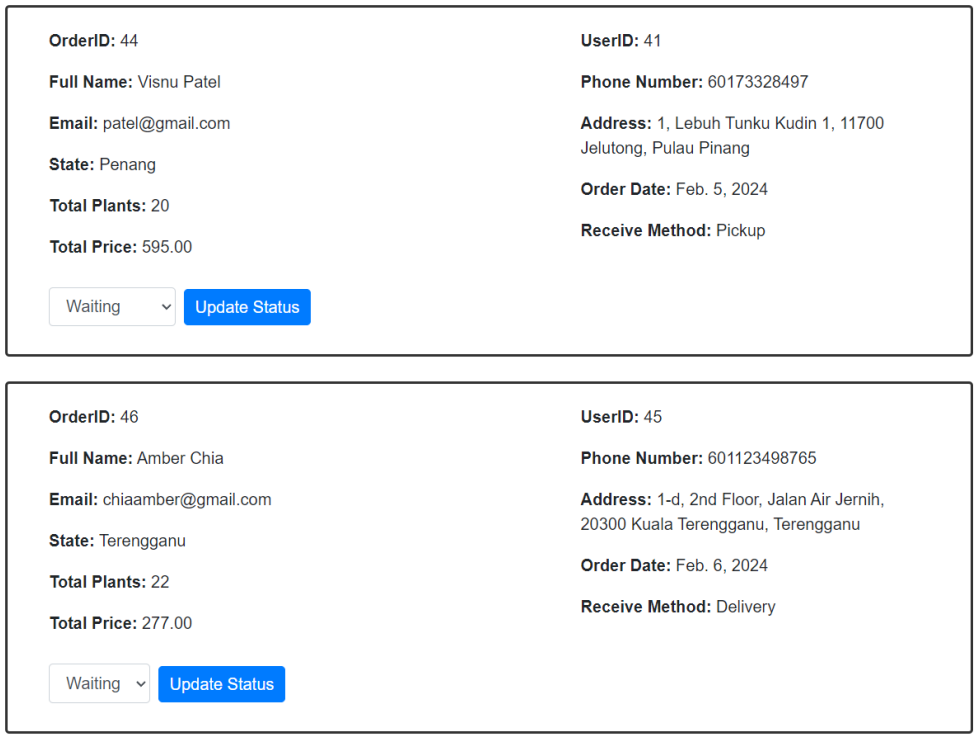



Figure 5.52: Delivery Order With ID 45 Removed From Screen

In Figure 5.53, the status “Ready” is selected for the delivery order with ID 46. After clicking the “Update Status” button, the “Order Management Screen” is refreshed, and the delivery order with ID 46 is removed from the screen, as depicted in Figure 5.54. It will later appear on the “Delivery Management Screen” for assignment to a delivery man.

| Order ID | Order Status (Update to) |
|---|---|
| 46 | Ready |

Figure 5.53: Status “Ready” Selected For Delivery Order With ID 46

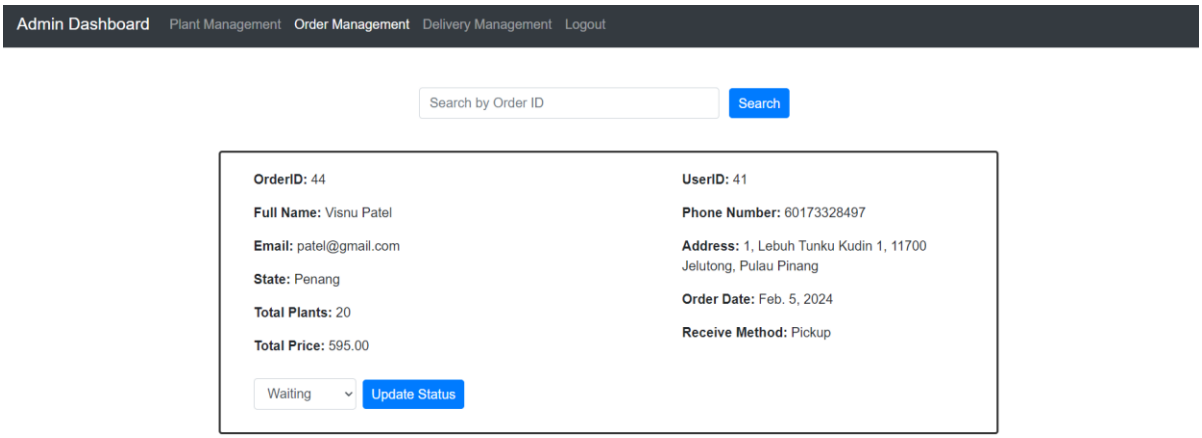



Figure 5.54: Delivery Order With ID 46 Removed From Screen

## Search Delivery Order Test Results:

In Figure 5.55, the input “45” is entered into the search bar. When the “Search” button is clicked, the delivery order with ID 45 is displayed on the screen, as shown in Figure 5.56.

| Order ID |
| --- |
| 45 |

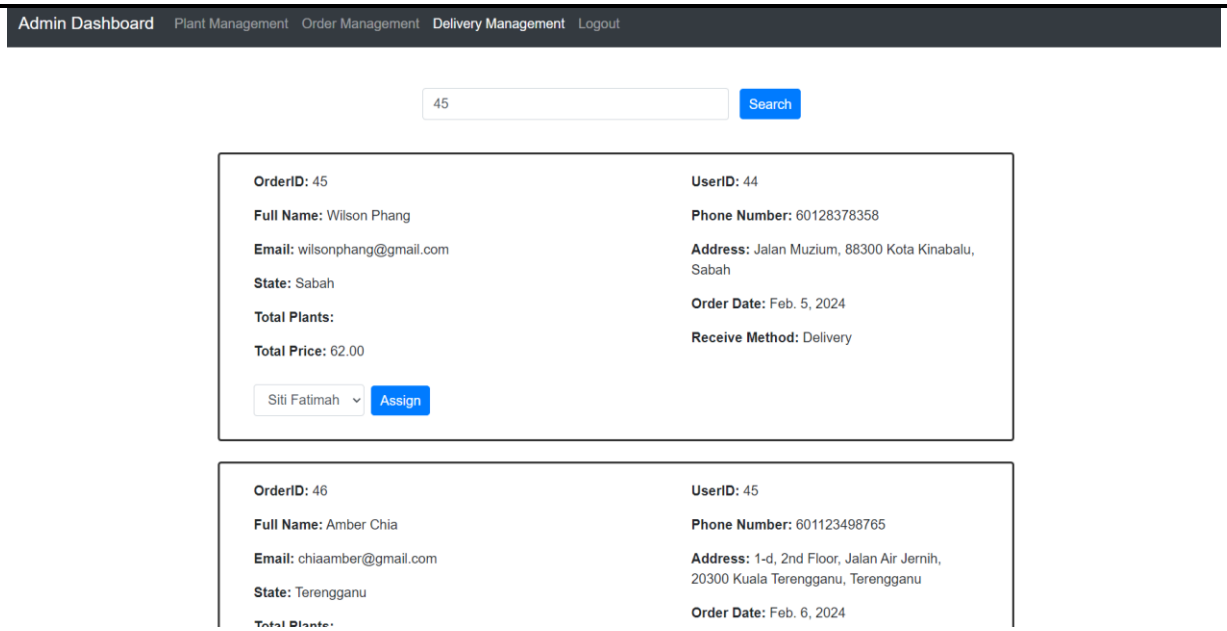



Figure 5.55: Search Delivery Order Using “45”

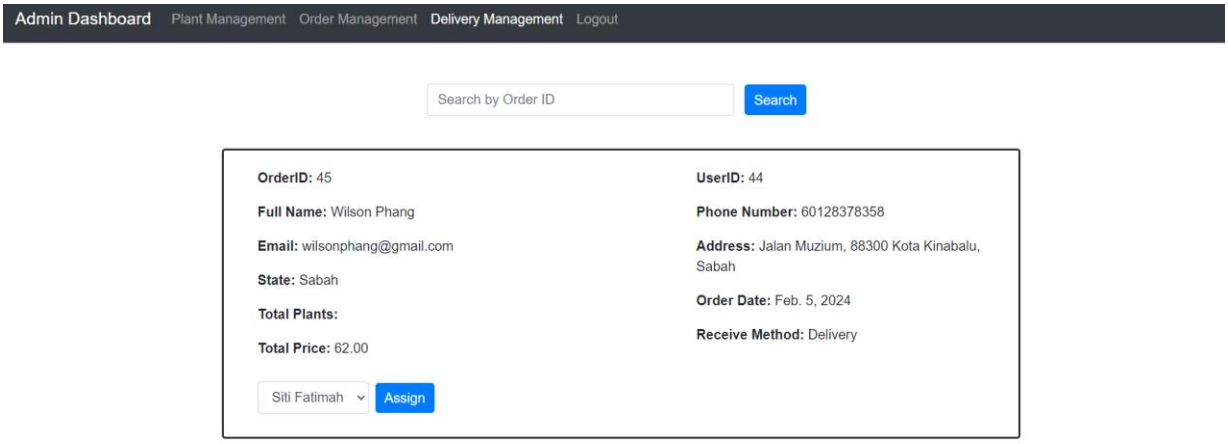



Figure 5.56: Delivery Order with ID 45 Shown

In Figure 5.57, the input “80” is entered into the search bar. Upon clicking the “Search” button, an error message is displayed on the screen, as shown in Figure 5.58, indicating that no ongoing delivery orders match the entered order ID.

| **Order ID** |
|---|
| 80 |

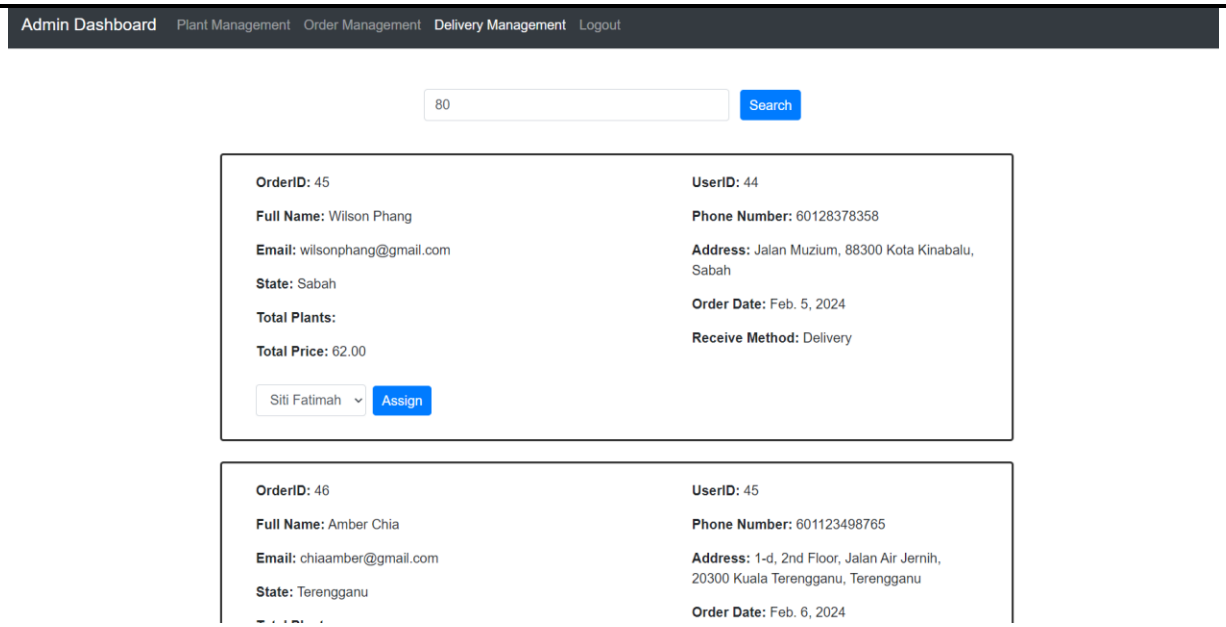



Figure 5.57: Search Delivery Order Using "80"

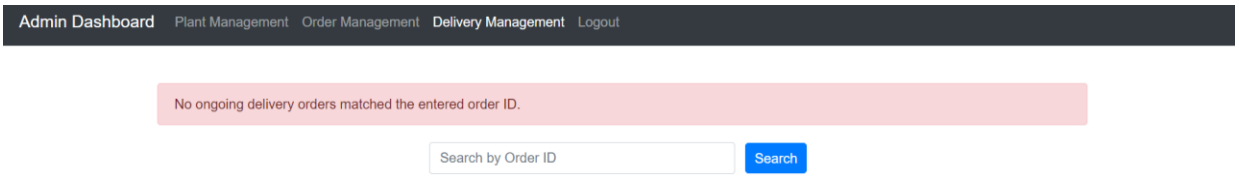



Figure 5.58: No Matching Delivery Orders Error Message

After clicking on the "Search" button without entering any input into the search bar, all the delivery orders and their details will be displayed on the screen, as depicted in Figure 5.59.

| **Order ID** |
|---|
| (blank) |

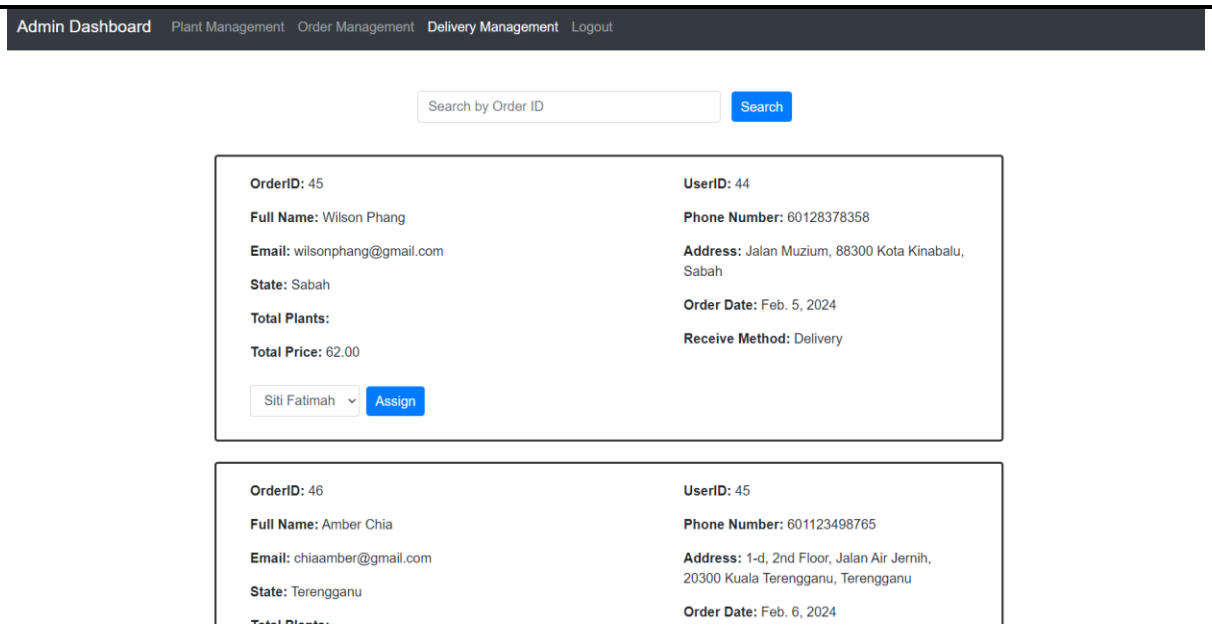



Figure 5.59: Search Delivery Order Without Input

In Figure 5.60, the input “zz” is entered into the search bar. When the “Search” button is clicked, an error message is displayed on the screen, as shown in Figure 5.61. The message states that the input is invalid and requests the re-entry of a numeric order ID.

**Order ID**

zz

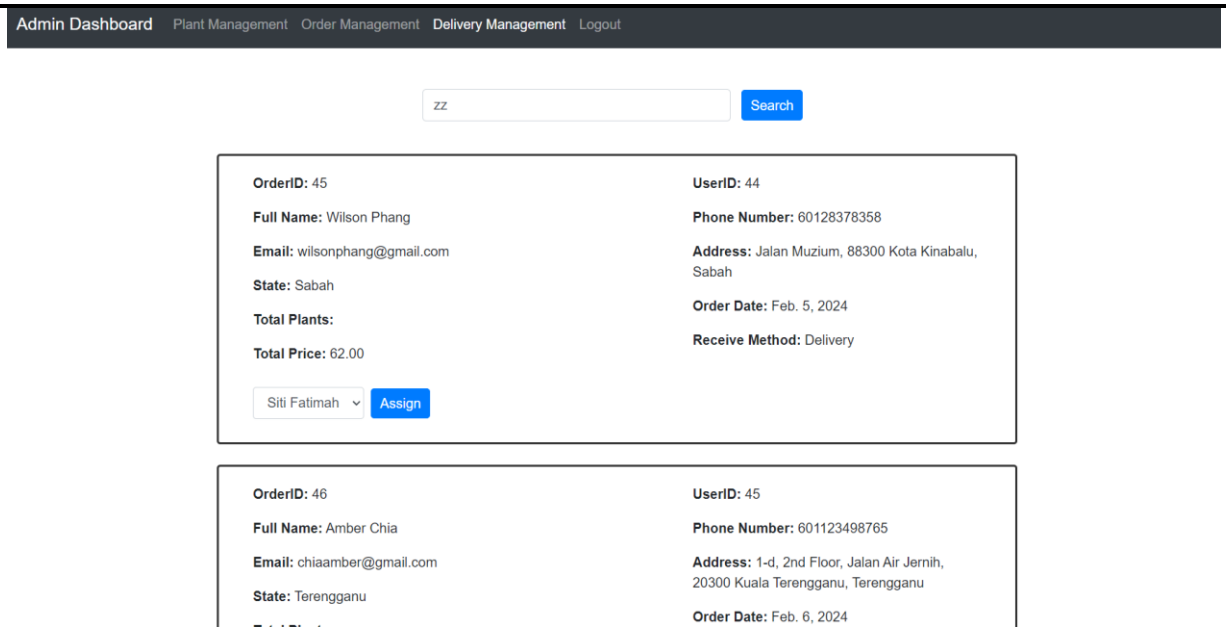



Figure 5.60: Search Delivery Order Using “zz”

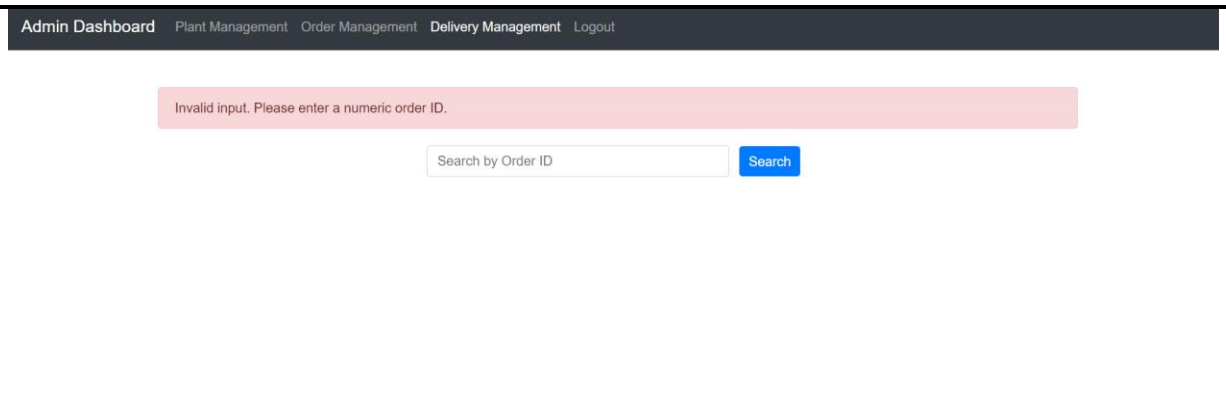



Figure 5.61: Invalid Input Error Message

In Figure 5.62, the input “45a” is entered into the search bar. When the “Search” button is clicked, an error message is displayed on the screen, as shown in Figure 5.63. The message states that the input is invalid and requests the re-entry of a numeric order ID.

| **Order ID** |
|---|
| 45a |

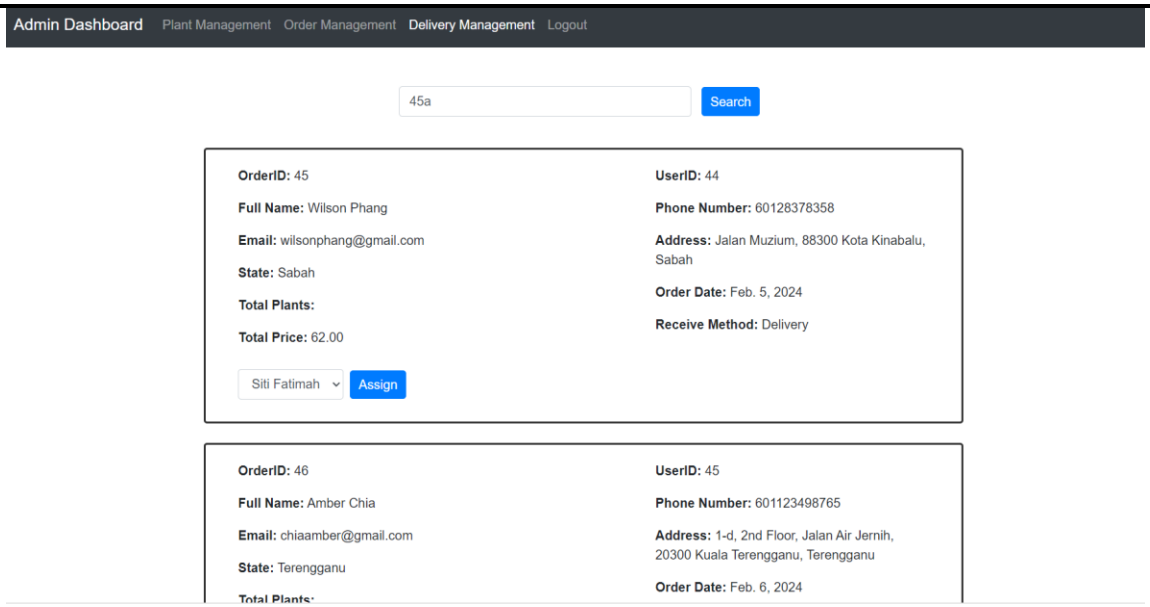



Figure 5.62: Search Delivery Order Using “45a”

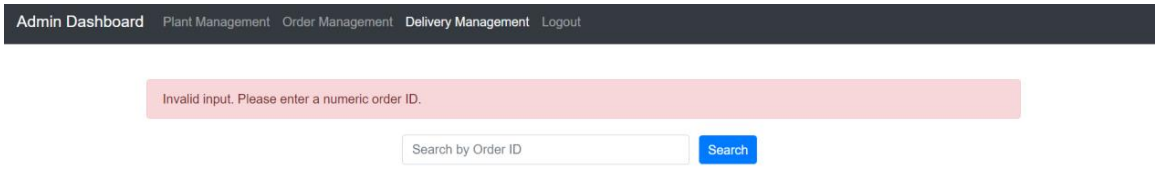



Figure 5.63: Invalid Input Error Message

In Figure 5.64, the input "@" is entered into the search bar. When the "Search" button is clicked, an error message is displayed on the screen, as shown in Figure 5.65. The message states that the input is invalid and requests the re-entry of a numeric order ID.

| **Order ID** |
|---|
| @ |

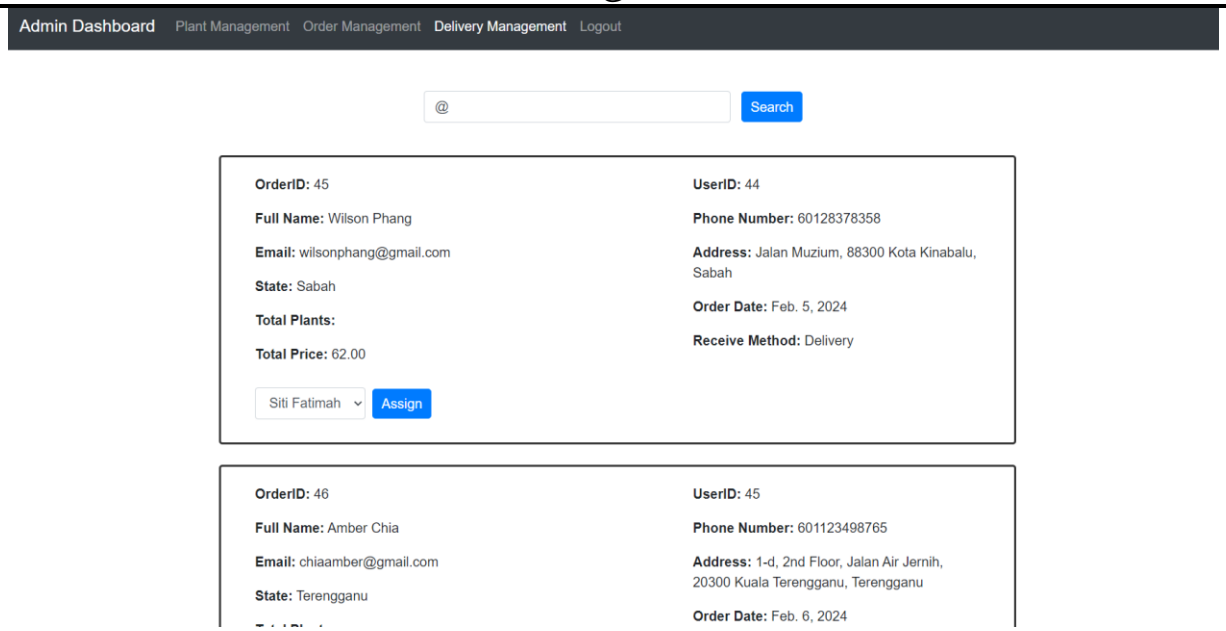



Figure 5.64: Search Delivery Order Using "@"

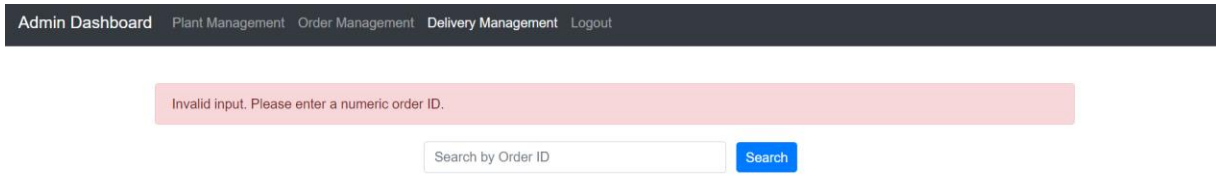



Figure 5.65: Invalid Input Error Message

## Assign Delivery Order Test Results:

In Figure 5.66, the delivery man named "Siti Fatimah" is selected for the delivery order with ID 45. After clicking the "Assign" button, the "Delivery Management Screen" is refreshed, and the delivery order with ID 45 is removed from the screen as it has already been assigned, as shown in Figure 5.67.

| Order ID | Delivery Man Name (Assign to) |
|---|---|
| 45 | Siti Fatimah |

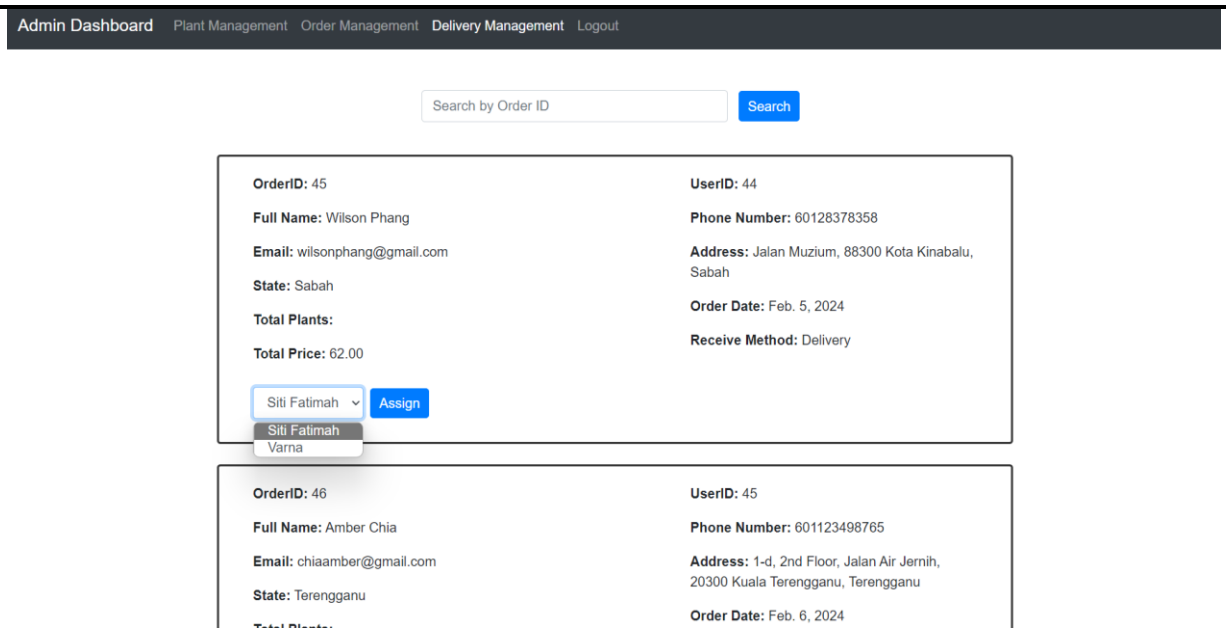



Figure 5.66: Delivery Man Named "Siti Fatimah" Selected

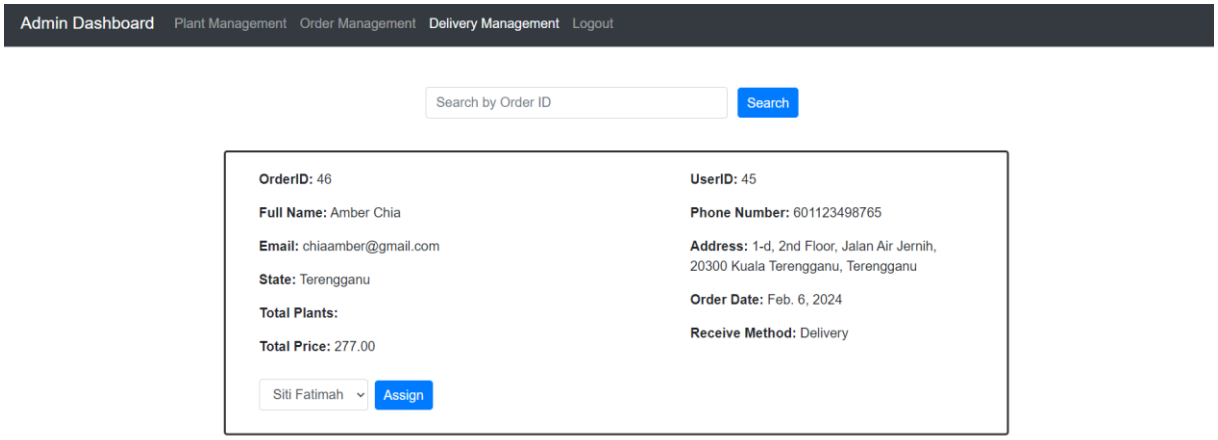



Figure 5.67: Delivery Order With ID 45 Assigned And Removed From Screen

In Figure 5.68, the delivery man named "Varna" is selected for the delivery order with ID 46. After clicking the "Assign" button, the "Delivery Management Screen" is refreshed, and the delivery order with ID 46 is removed from the screen as it has already been assigned, as shown in Figure 5.69.

| Order ID | Delivery Man Name (Assign to) |
|---|---|
| 46 | Varna |

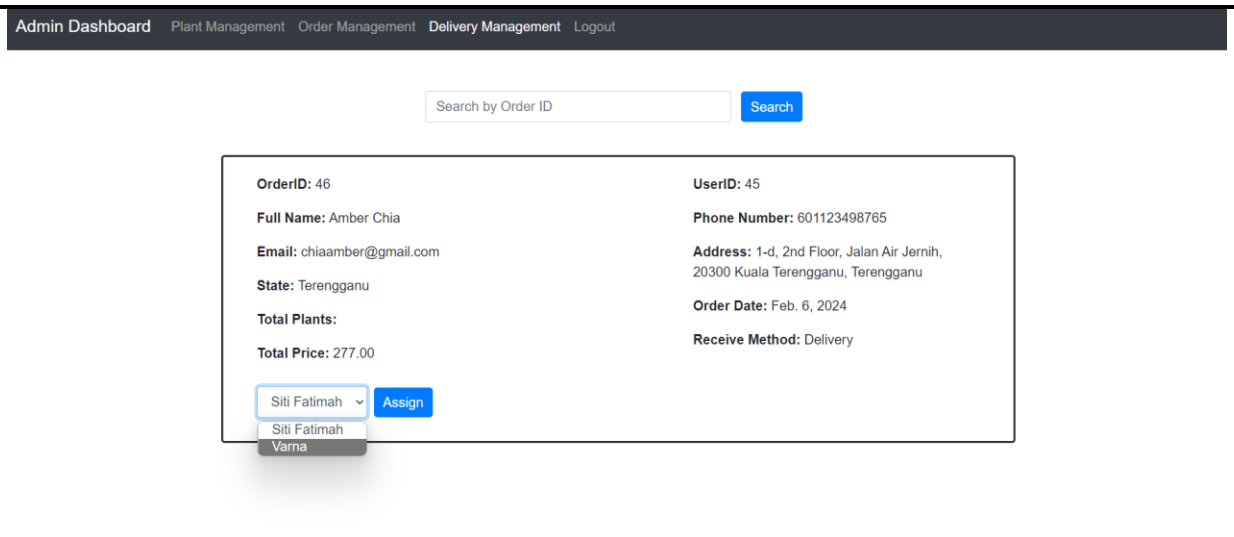



Figure 5.68: Delivery Man Named "Varna" Selected

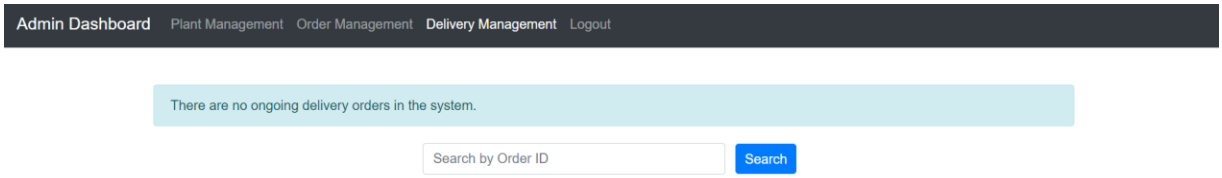



Figure 5.69: Delivery Order With ID 46 Assigned And Removed From Screen

# 6 Conclusion

## 6.1 Project Achievements

This nursery plant shopping system has been well-integrated and developed to enable administrators, customers, guests, and delivery men to perform their respective functions. We managed to successfully complete and deliver the project on time, following a thorough verification and validation process.

One of my key achievements in this project was the development of administrator modules that meet all the essential requirements for plant management. These modules include the ability to create, update, and delete plants, search plants by name, view order details, update order statuses, and assign deliveries to delivery men.

Additionally, throughout the development of this system, I acquired basic skills in utilizing the Django framework and learned the concepts and syntax of Python. This achievement has equipped me to effectively handle future projects involving the Django framework and Python, eventually reducing development time and enhancing productivity.

## 6.2 Quality Assurance

To ensure high software quality, our team starts by thoroughly understanding the project requirements and identifying any challenges we need to overcome. We then focus on resolving any issues during the design phase to improve the overall quality of our software.

At the beginning of the project, we create a detailed plan that outlines how we will manage quality and handle any changes that may arise. Throughout the development process, we conduct multiple rounds of testing and review, using at least three different sets of test data. This ensures that our software fulfils all the requirements established at the beginning of the project.

Additionally, we make sure to carefully document any deviations or discrepancies we encounter while working on the software. This allows us to address these issues promptly and ensures that our final product aligns with what was originally planned.

Furthermore, we maintain open lines of communication within our team, regularly sharing updates and reporting any issues or non-compliance instances we come across. This collaborative approach helps us stay on top of any challenges and ensures that we can address them effectively.

## 6.3 Problems Encountered

Throughout the project development process, we encountered several significant challenges, particularly in the coding aspects, which required extensive research and study. Despite the comprehensive documentation provided by the Django framework, we found that the guidelines were lacking, making it necessary for us to conduct testing to fully understand the syntax and functionality.

Since it was our first time using Django, Python, HTML, CSS and Bootstrap, the lack of prior experience posed a big challenge to us. We found ourselves needing to invest additional time and effort into studying and familiarizing ourselves with these languages and frameworks, which inevitably slowed down our progress.

In the project development process, we encountered two main challenges. Firstly, we faced time constraints in implementing a feature to display only the latest review under each plant details box, with the option to view earlier reviews upon clicking. Secondly, despite our best efforts, we encountered obstacles in integrating a third-party payment gateway into our system. As a result, the current state of our system only allows for the display of a success message when customers select a payment method, without the actual processing of payments.

## 6.4 Remarks/Comments

This assignment proved challenging as it was our first experience completing a Python project. We lacked familiarity with the language's basics, and adjusting to its syntax required some time. We had to do a lot of research on Python, Django, HTML, CSS, and Bootstrap to understand how to use them.

Despite these challenges, this project provided valuable exposure to web application development, offering an experience that will benefit us in the future. Through this process, we learned that focusing too much on a particular issue can mess up the project schedule we originally set. Hence, it is essential to address issues promptly and maintain a flexible approach.

Additionally, we recognized the importance of teamwork in overcoming challenges efficiently. By collaborating, we were able to benefit from a variety of perspectives and ideas, ultimately saving time and enhancing problem-solving capabilities.