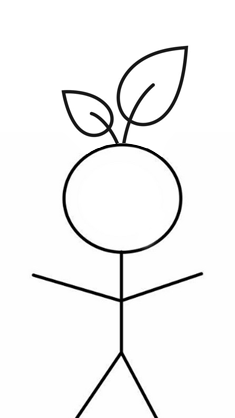
**Growing Pains – An Online Plant Store System**

**(OPSS)**

**Object Oriented Software Development - Project**

****

**By:** Mark Lambert (C00192497)

A black and grey logo

Description automatically generated**Course:** Software Development (CW\_KCSOF\_B)

**Submission Date:** 11th April 2025

**Table of Contents Page**

[**Table of Figures** 1](#_Toc194871181)

[**Summary** 2](#_Toc194871182)

[**Requirements** 3](#_Toc194871183)

[**Functional** 3](#_Toc194871184)

[**Non-Functional** 5](#_Toc194871185)

[**Database Tables** 7](#_Toc194871186)

[**Customer Table** 7](#_Toc194871187)

[**Product Table** 8](#_Toc194871188)

[**Orders** 9](#_Toc194871189)

[**ER Diagram** 10](#_Toc194871190)

[**Interesting Source Code Snippets** 11](#_Toc194871191)

[**Retrieval by List** 11](#_Toc194871192)

[**Displaying a List of Objects** 12](#_Toc194871193)

[**Event Handling and View/Control Interaction** 13](#_Toc194871194)

[**Filtering a Catalogue of Products** 14](#_Toc194871195)

[**Hashing Passwords** 15](#_Toc194871196)

[**Writing to an Error File** 16](#_Toc194871197)

[**Method to Store Images** 17](#_Toc194871198)

[**CardLayout as a Central Navigation Tool** 18](#_Toc194871199)

[**Driver and main() method** 19](#_Toc194871200)

[**Test Cases** 20](#_Toc194871201)

# **Table of Figures**

[Figure 1: Structure of Customer Table 7](#_Toc194870508)

[Figure 2: Sample Data of Customer Table 7](#_Toc194870509)

[Figure 3: Structure of Product table 8](#_Toc194870510)

[Figure 4: Sample Data for Product table 8](#_Toc194870511)

[Figure 5: Structure of Orders Table 9](#_Toc194870512)

[Figure 6: Sample Data for Orders Table 9](#_Toc194870513)

[Figure 7: ER Diagram for GrowingPains 10](#_Toc194870514)

[Figure 8: getAllProducts() - ProductCrud 11](#_Toc194870515)

[Figure 9: getProducts - BrowsePanel 12](#_Toc194870516)

[Figure 10: getProducts - BrowsePanel 13](#_Toc194870517)

[Figure 11: handleFilter() - BrowsePanel 14](#_Toc194870518)

[Figure 12: Hashing 15](#_Toc194870519)

[Figure 13: Error Writing 16](#_Toc194870520)

[Figure 14: image\_path in Product table 17](#_Toc194870521)

[Figure 15: CardLayout 18](#_Toc194870522)

[Figure 16: Driver class 19](#_Toc194870523)

# **Summary**

A rapidly growing houseplant store wants to expand its business to build an Online Plant Store System (OPSS) to manage its expanding business and improve customer engagement. The system should aim to meet the following requirements:–

1. Facilitate the buying of a diverse range of plants and plant accessories
2. A comprehensive marketplace experience, making the interface accessible to the user
3. A responsive, dynamic application that responds to user inputs and updates the backend database as the user interacts with the application
4. The store catalogue must have a filter feature to enhance user experience, allowing users to sort items and accessories by price, type etc
5. The user must be able to edit their account information as well as view and cancel any orders made
6. Users must also be able to set personal reminders, notifying them of when to water their plants

The system should also feature a user-interface which keeps the design aesthetics of the houseplant store in mind.

The system should allow users to browse the store catalogue, which will include filtering options such as plant species, type, price and accessory. Each plant listing will include brief descriptions, care instructions and pricing information, which can be visible when a customer selects a plant.

Once the user selects a plant, they can add it to their cart to proceed with the checkout process. Users may update their cart or remove items. When the user initiates the checkout process, they must enter in payment details before finally placing the order.

Another integrated feature should allow users to set a reminder by selecting a date. The reminder will take input on the plant type and species (e.g., succulent, tropical, houseplant) and notify the user of when to next water their plant.

# **Requirements**

The following document outlines the requirements for the Online Plant Store System (OPSS). To ensure that all corners of the requirements finding process were covered, the **FURPS+** model to assess functional and on-functional requirements was considered.

## **Functional**

|  |  |
| --- | --- |
| **Requirement ID :** | **FR001:** Select Item |
| **Definition:** | The system shall display detailed information when a user selects an item from the catalogue. |
| **Specification:** | - On click of a product in the BrowsePanel the system should:  - Form a SELECT query on the Product table  - Display a new ProductPanel with the following using JLabel’s: 200x200 image of the product, product name, price and description |

|  |  |
| --- | --- |
| **Requirement ID :** | **FR002:** Cart Management |
| **Definition:** | Registered users may add products to their cart, from which they may alter the product quantity via a JSpinner |
| **Specification:** | - On click of the “Add to Cart” button the system must:  - Validate the user is first logged in  - Update the users cart object to display the product quantities and total price |

|  |  |
| --- | --- |
| **Requirement ID :** | **FR003:** Checkout Process |
| **Definition:** | Users complete the Order by initiating a checkout process, validated by inputting payment details |
| **Specification:** | - On click of the “Checkout” button the system must:  - Build a form to input: Card Number, Card Holder, Address, CVV and Expiration Date (via JComboBox’s)  - On submit, the system will generate an INSERT query into the Orders table |

|  |  |
| --- | --- |
| **Requirement ID :** | **FR004:** Browse Catalogue |
| **Definition:** | The system shall display a populated catalogue of items with a scrollable UI |
| **Specification:** | - A JPanel displaying a series of product item containers which hold information about each product in the Product table.  - Products are retrieved via a SELECT query in the Products table |

|  |  |
| --- | --- |
| **Requirement ID :** | **FR005:** Order History |
| **Definition:** | Users must be able to view past orders with the aim of cancelling orders should they wish |
| **Specification:** | - A JTable displaying a history of all orders made by the logged in user.  - The table is populated via a SELECT query on the Customer table which INNER JOINS with the Orders table  - When an order is selected, the user may cancel the order by clicking the “Cancel Order” button.  - Onclick, a DELETE query in the Orders table is generated |

## **Non-Functional**

|  |  |
| --- | --- |
| **Requirement ID :** | **NFR001:** Usability |
| **Definition:** | The system must be both learnable and accessible for new users |
| **Specification:** | - Learnability: Users must be able to comfortably adapt to the systems GUI, enabling them to purchase products quickly  - Accessible: The system must be designed bearing in mind users who may have vision impairments, such as font sizes, colours etc.  - Logging errors to a standard text file is a must, ensuring users can learn in more detail what errors may have occurred |

|  |  |
| --- | --- |
| **Requirement ID :** | **NFR002:** Reliability |
| **Definition:** | The system must reliably deal with invalid data input from the user |
| **Specification:** | - Data input must be handled appropriately according to what may constitute as “bad data” or malicious data.  - Preventative measures against SQL Injection by using prepared statements |

|  |  |
| --- | --- |
| **Requirement ID :** | **NFR003:** Performance |
| **Definition:** | The system must respond quickly and appropriately to user input |
| **Specification:** | - Interaction between the system and database must be seamless, ensuring the customer is met with a responsive application  - Any image scaling must be handled appropriately and with care, to ensure the performance drawback is not apparent to the user |

|  |  |
| --- | --- |
| **Requirement ID :** | **NFR004:** Supportability |
| **Definition:** | The system must be maintainable for future iterations and expansion |
| **Specification:** | - Code must be well documented and conform to standard Object-Oriented principles  - System architecture must be well organised and make use of a MVC structure  - Extensive version history must be available on a version control platform |

|  |  |
| --- | --- |
| **Requirement ID :** | **NFR005:** Security |
| **Definition:** | The system must be secure for the user to use |
| **Specification:** | - Any sensitive or precious data shall be handled with care  - Passwords must be securely stored in the database by first hashing the input. A standard SHA-256 algorithm may be deployed. |

# **Database Tables**

## **Customer Table**

The customer table represents information relating to a user who has registered to make anaccount. The fields in Figure 1 represent the information that captures a customer, which gets encapsulated into three separate objects, Address, Account and Customer.

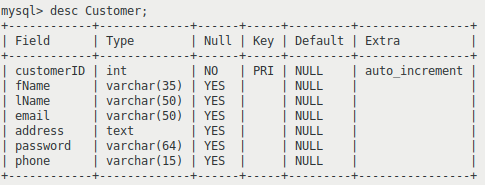
****

Figure : Structure of Customer Table

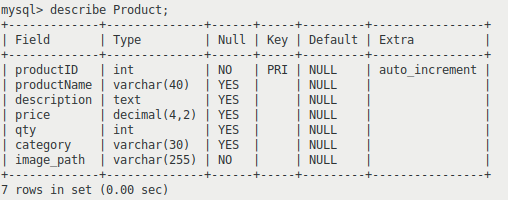


Figure : Sample Data of Customer Table

[[1]](#footnote-1)

## **Product Table**

The product table contains fields which uniquely describe a product, which as of the current release, may have two categories: Plant and Accessory.



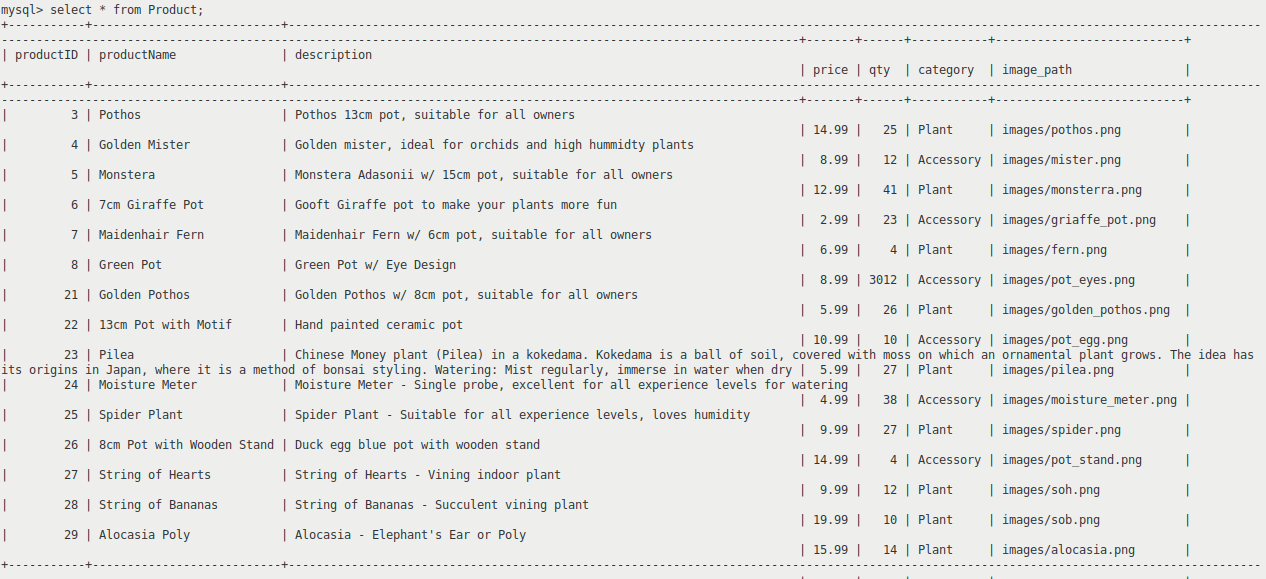


Figure : Sample Data for Product table

Figure : Structure of Product table

[[2]](#footnote-2)

## **Orders**

The orders table captures data relating to each order a customer has made. A record gets added to this table once the user successfully completes the checkout process. A key area for future development is to simplify the many-to-many relationship that exists between the Orders and Product table.

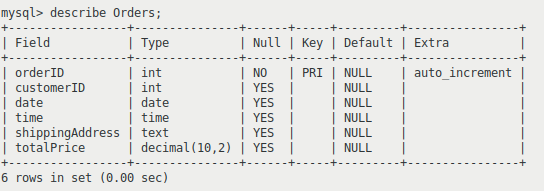


Figure : Structure of Orders Table

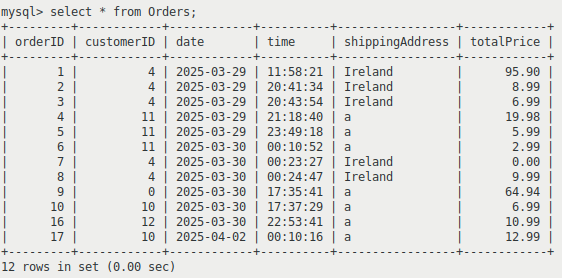


Figure : Sample Data for Orders Table

# **ER Diagram**

**A screenshot of a diagram

AI-generated content may be incorrect.**

Figure : ER Diagram for GrowingPains

[[3]](#footnote-3)

# **Interesting Source Code Snippets**

## A screenshot of a computer code AI-generated content may be incorrect.**Retrieval by List**

Figure : getAllProducts() - ProductCrud

Lists are commonly used throughout the system, particularly when performing retrieval operations on the Product table. As seen in Figure 8- it highlights this retrieval process, where:

* The result of the select query is encapsulated within a DisplayItem object.
* Each DisplayItem gets added to a List of DisplayItem’s, where each entry in the database is a DisplayItem, and each field is encapsulated within said item.
* The method returns the full list of products.

## 

## **Displaying a List of Objects**

Figure : getProducts - BrowsePanel

Figure 9 demonstrates how to display the List of DisplayItem’s retrieved from Figure 8

* Firstly, a **for each** loop is utilised. In this case, for each Item in the DisplayItem list (DisplayItem extends the abstract class Item), extract the information from the DisplayItem
* Create a new JPanel container for each DisplayItem which holds the extracted information.
* The container makes use of a **BorderLayout,** allowing the individual pieces of product information to be displayed nicely within the container.
* Each container is added to the main JPanel **gridPanel**, which is the container for the **BrowsePanel** content area.

## A screen shot of a computer code AI-generated content may be incorrect. **Event Handling and View/Control Interaction**

Figure : getProducts - BrowsePanel

Figure 10 displays how an event is handled between the user’s **mouse** and the **imgLabel** associated with a Product in the **BrowsePanel.**

The method mouseClicked gets triggered when the users clicks the imgLabel. On click the **ProductPanel** p – representing **the selected items detailed panel,** gets assigned all the GUI elements associated with the selected product.

The final piece of code displays the relationship between a view packaged class, which handles any GUI based logic, and a controller packaged class, which handless the more business end logic concerned with the GUI interface. In this instance **CONTROL** represents the final class level variable of type **BrowseControl**. BrowseControl is a custom controller class which contains the necessary functions to operate on the Browse section of the system.

The purpose of diversifying the two packages is to provide an easier platform for expansion in the future, as the abstraction makes it much easier to change information and keeps the system modular.

## **Filtering a Catalogue of Products**

Figure : handleFilter() - BrowsePanel

Figure 11 details the process of filtering the catalogue of items:

* A **JComboBox** is used to store the filters, which in this case are “Plant” and “Accessory” which are represented as “category” in the Product table.
* When the user selects a filter from the list and clicks the JButton **filterBtn,** a ActionEvent is triggered, calling **removeAll()** method on the gridPanel, removing all components from the container.
* **getProducts()** is then called with the argument **CONTROL.filterCatalogue(….)**
* This returns a list of items retrieved by querying the Product table based on the “category=?” where ? = the selected item from the JComboBox filterList.
* Finally, to dynamically display this new catalogue to the user, the revalidate() and repaint() methods are called.

## **Hashing Passwords**

Figure : Hashing

**MessageDigest** allows us to make use of the functionality of the SHA-256 algorithm. **Hashing** then follows the following steps:

* Convert the password string into an array of bytes
* The output of that array of bytes gets supplied to the argument **digest,** which finalises the hashing process by performing operations such as padding.
* The result of that function, is then ran through a for each loop, for each byte in the result, convert it back to a string, formatting each byte as a hexadecimal number

## A computer screen shot of a computer error AI-generated content may be incorrect.**Writing to an Error File**

Figure : Error Writing

Figure 13shows a sample throw of a custom Exception **ValidationException**, which is thrown when a field is left blank in the account creation process. When the exception gets caught:

* The handleError method is invoked, with the Exception as an argument and a string describing the type of error.
* **handleError** creates a popup to alert the user of the error
* A static instance of an **ErrorWriter** object located in the GrowingPains class is then called with the logError() method to write to a file which is opened on system load.

## **Method to Store Images**

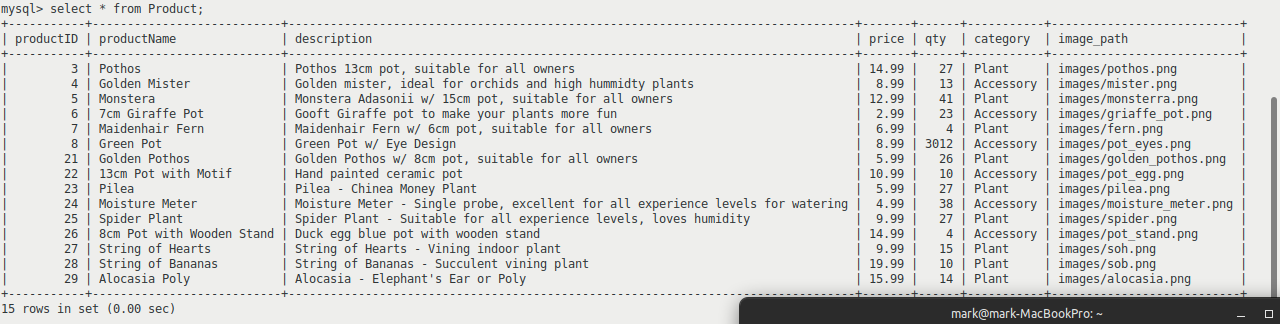
****

Figure : image\_path in Product table

Observing Figure 14, noting the image\_path, each product contains the relative path to a .png image, which relates to the plant. The images are located within a folder named **images** within the **view** folder.

## A screenshot of a computer code AI-generated content may be incorrect.**CardLayout as a Central Navigation Tool**

Figure : CardLayout

Figure 15outlines the relationship of a CardLayout manager being used to switch to a new JPanel.

* The CardLayout works exactly like a deck of cards, where you can call the add() method to add a card to the deck, in this case the card being a JPanel container.
* A second argument is supplied to add(), a String indexing the added JPanel enabling you to easily reference the “card” when calling the show() method.
* The show() method brings the JPanel with the matching index to the foreground.

## A screenshot of a computer program AI-generated content may be incorrect.**Driver and main() method**

Figure : Driver class

Figure 16 shows the benefits of the Object-Oriented approach to Software Development. The driver class for OPSS contains roughly 10 lines of code excluding comments, allowing for a much more manageable and extensible codebase for future iterations.

# **Test Cases**

|  |  |
| --- | --- |
| **Name** | **TC-001:** Select Item |
| **Requirement** | Verify that the system successfully updates to display details of a selected item from a database of items. |
| **Preconditions** | The system is displaying the full catalogue |
| **Steps** | 1. Click on the **image** of the third item. 2. Return to previous page 3. Click on the **name** of the first item. 4. Return to previous page |
| **Expected Results** | 1. The item details window appears, with a larger image and more detailed information 2. Verify that items can be selected by clicking icon **or** thumbnail 3. Return to browsing catalogue |

|  |  |
| --- | --- |
| **Name** | **TC-002:** Add to Cart |
| **Requirement** | Verify that the system successfully allows a user to enter item(s) to cart. |
| **Preconditions** | The user is viewing the catalogue |
| **Steps** | 1. Click on the **second** product 2. Click the “Add to Cart” button 3. Return to catalogue 4. Click on the **fourth** product 5. Change **quantity** to 2, add to cart |
| **Expected Results** | 1. The item details window appears, with a larger image and more detailed information 2. The system alerts the user to the fact that the item has been added successfully 3. Return to browsing catalogue |

|  |  |
| --- | --- |
| **Name** | **TC-003:** Checkout |
| **Requirement** | Verify that the store system successfully allows the user to checkout their items |
| **Preconditions** | The customer has items in their cart |
| **Steps** | 1. Click on the **View Cart** button 2. Click on the **Proceed to Checkout** button 3. Enter your **login details** 4. Input **personal information** 5. Click on the **Confirm Order** button |
| **Expected Results** | 1. The system displays the users Cart 2. The system begins the Checkout process 3. System prompts user for login details 4. System prompts user for shipping & billing information 5. System updates to confirm to the user that their order has been successfully placed |

|  |  |
| --- | --- |
| **Name** | **TC-004:** Filter Catalogue |
| **Requirement** | Verify that the system allows the user to apply filter(s) to the Catalogue of Items |
| **Preconditions** | The system is displaying the full catalogue |
| **Steps** | 1. Click on the **Filter** button 2. Select **one** filter 3. Click the **Apply Filter** button 4. Click on the **Filter** button 5. Select **another** filter 6. Click on the **Apply Filter** button |
| **Expected Results** | 1. The system updates to show a list of filters to choose from 2. When applied, the system displays just items matching the filter tag 3. Applying another filter will result in a more specific list of items |

|  |  |
| --- | --- |
| **Name** | **TC-005:** Schedule Reminder |
| **Requirement** | Verify that the system successfully sets and alerts the user when a Reminder is Scheduled |
| **Preconditions** | The user is logged in |
| **Steps** | 1. Click on the **Schedule Reminder** button 2. Input **today’s date** 3. Click on the **Set Reminder** button to confirm |
| **Expected Results** | 1. The system will display the Schedule Reminder page 2. The system will update to display the reminder the user has just input |

|  |  |
| --- | --- |
| **Name** | **TC-006:** Browse Catalogue |
| **Requirement** | Verify that the system allows the user to browse the catalogue of items |
| **Preconditions** | The user is on the page displaying the catalogue |
| **Steps** | 1. Click on the **Home** page 2. Scroll to browse the catalogue |
| **Expected Results** | 1. The system will update in real time to display the catalogue – containing items - for the user |

1. Note, password field has been cropped for readability reasons, as the hashed password stretches the screenshot aspect ratio out, making text too small [↑](#footnote-ref-1)
2. Note, the description for productID: 23 is intentionally long-winded to demonstrate usage of JTextArea elements. [↑](#footnote-ref-2)
3. The Many to Many relationship between the Orders and Product tables in the current iteration of the OPSS has not been simplified to include an Order/Product table.

   Future expansion during the Summer of ’25 will ensure this implementation is appropriately handled. [↑](#footnote-ref-3)