Soil Enthusiast

Software Requirements Specification – SRS

1. Introduction:

The Soil Enthusiast application aims to provide users with a comprehensive platform for managing eco-friendly products, user registrations, login authentication, compost calculation, and payment processing. This Software Requirements Specification (SRS) document outlines the functional and non-functional requirements of the application.

2. Purpose

The purpose of the Soil Enthusiast application is to offer users an intuitive and efficient interface for interacting with various functionalities related to eco-friendly products, user management, and payment processing. The application targets environmentally conscious individuals and organizations seeking sustainable solutions.

3. Scope The scope of the Soil Enthusiast application includes the following modules:

1. User Registration and Login
2. Product Management and User Dashboard
3. Compost Calculator
4. Billing and Payment Processing

4. Functional Requirements

4.1 **User Registration and Login Module**

- FR1: The system shall provide a user registration form (UserRegistrationForm.java) for capturing user details.
- FR2: The system shall validate user input for registration, including email format validation.
- FR3: The system shall store user registration data in a MySQL database.
- FR4: The system shall provide a login form (LoginFrame.java) for user authentication.
- FR5: The system shall validate user credentials during login and grant access to the user dashboard upon successful authentication.

4.2 **Product Management and User Dashboard Module**

- FR6: The system shall display product categories and details dynamically (UserDashboard.java).
- FR7: The system shall allow users to add, remove, and update items in the cart within the user dashboard.
- FR8: The system shall calculate the total amount in the cart based on user interactions.
- FR9: The system shall provide a compost calculator (CompostCalculatorFrame.java) for users to calculate compost needed based on project dimensions.
- FR10: The system shall save compost calculation data to the database.

4.3 **Billing and Payment Processing Module**

- FR11: The system shall display a billing dashboard (BillingDashboard.java) summarizing cart items and total amount.
- FR12: The system shall provide payment options (PaymentOptionsFrame.java) including radio buttons for payment modes and an address input field.
- FR13: The system shall validate user inputs during payment and save order details to the database.
- FR14: The system shall generate a receipt upon successful payment and open the home screen.

5. Non-Functional Requirements

5.1 **Usability**

- NFR1: The application shall have an intuitive and user-friendly interface for ease of navigation.
- NFR2: The application shall provide clear error messages for invalid user inputs or failed operations.

5.2 **Performance**

- NFR3: The application shall handle concurrent user sessions efficiently without significant performance degradation.
- NFR4: The database operations shall be optimized for quick data retrieval and storage.

5.3 **Reliability**

- NFR5: The application shall ensure data integrity and consistency in database transactions.
- NFR6: The application shall have backup and recovery mechanisms in place to handle data loss situations.

5.4 **Security**

- NFR7: The application shall implement secure authentication mechanisms to protect user credentials.
- NFR8: The application shall encrypt sensitive data such as passwords and payment information.

6. Constraints

- The application requires a MySQL database for storing user data, product information, and order details.
- The application relies on Java Swing for creating graphical user interfaces (GUIs) and handling user interactions.

7. Flow of the Program

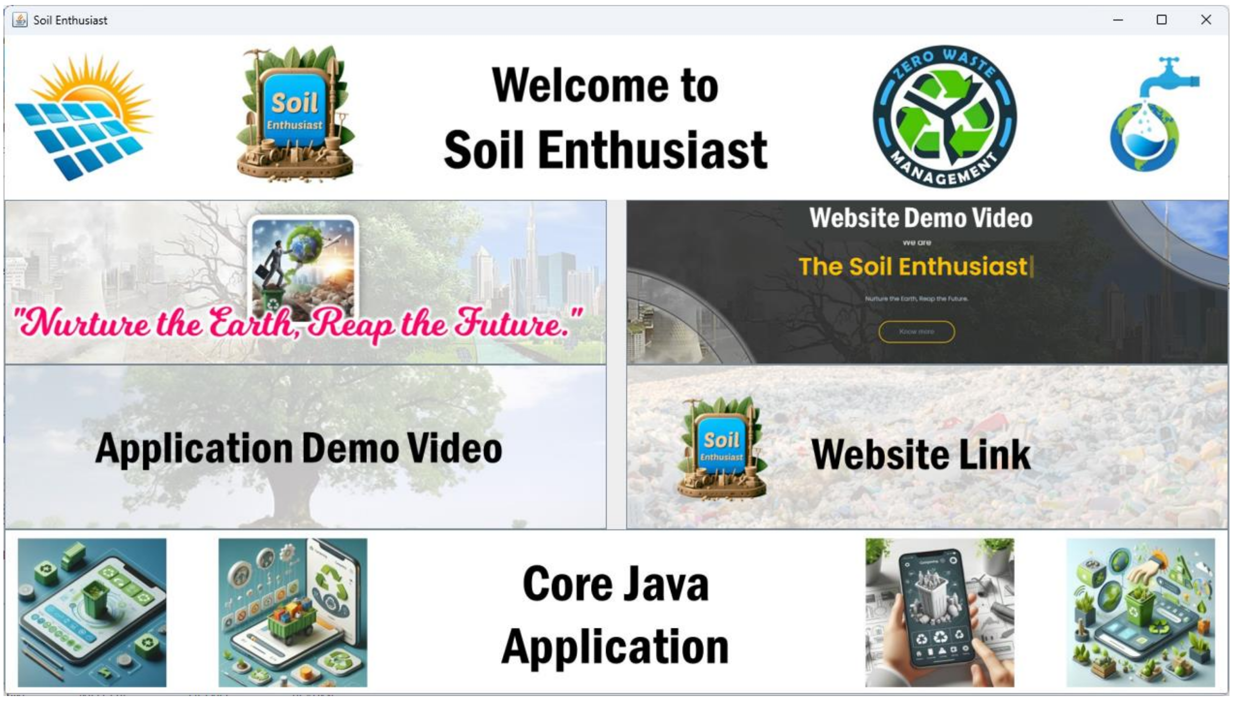
1. **Soil_Enthusiast_Home.java**:

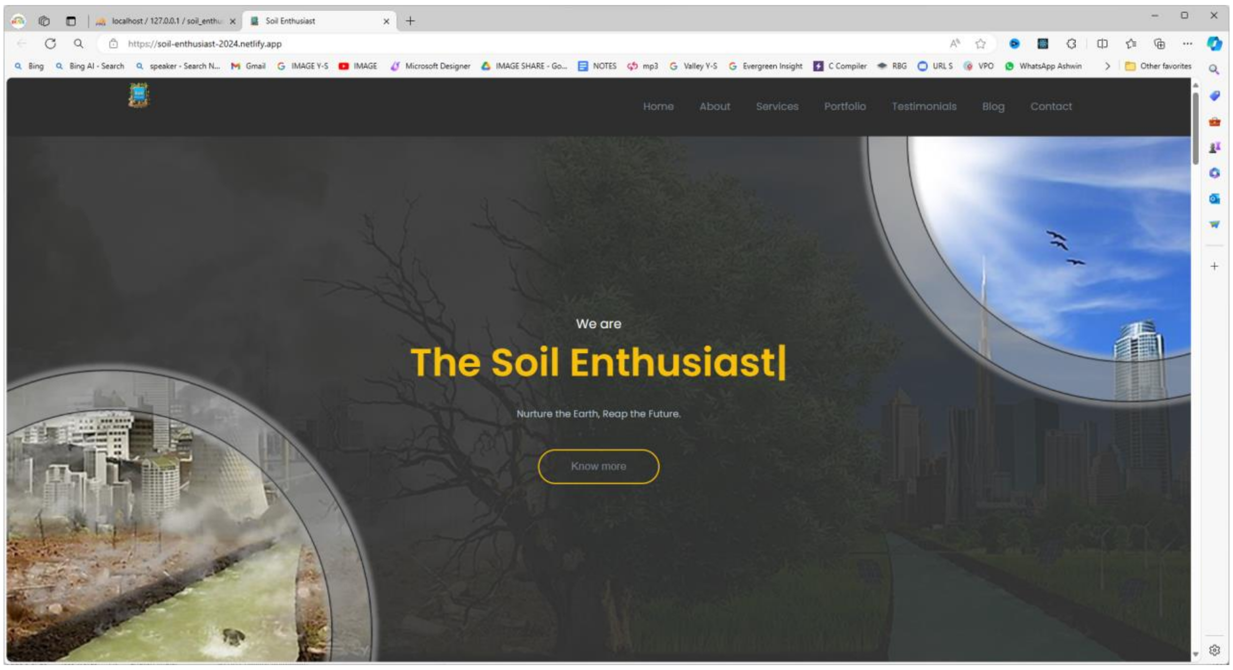
- Extends **JFrame** to create the main GUI window for the Soil Enthusiast application.
- Sets up the layout, title, and default close operation of the JFrame.
- Adds components such as an image label, buttons for website demo and video, buttons for app demo and video, and a login button.
- Implements actions for button clicks to open webpages or videos and login forms.

2. **SoilEnthusiast.java**:
    - Defines a class to execute commands related to database setup and user registration.
    - Executes commands to compile and run Java files for database setup and user registration.
3. **DatabaseSetup.java**:
    - Establishes a connection to a MySQL database and creates the required database and tables if they don't exist.
4. **TableSetup.java**:
    - Establishes a connection to a MySQL database and creates necessary tables if they don't exist.
5. **UserRegistrationForm.java**:
    - Creates a Swing-based user registration form with input fields for user details.
    - Validates user input and stores registration data in the database.
6. **LoginFrame.java**:
    - Creates a Swing-based login form for user authentication.
    - Connects to the database to validate user credentials and opens a user dashboard upon successful login.
7. **UserDashboard.java**:
    - Implements the user dashboard interface with product categories, cart functionality, checkout, and compost calculator.
    - Displays dynamic product information based on user interactions and updates cart and total amount accordingly.
8. **CompostCalculatorFrame.java**:
    - Creates a Swing-based frame for compost calculation with input fields for project type, dimensions, and a calculate button.
    - Calculates compost needed based on user inputs and saves the calculation to the database.
9. **BillingDashboard.java**:
    - Creates a Swing-based billing dashboard for cart summary and payment options.
    - Proceeds to payment options upon user action, interacting with the PaymentOptionsFrame.
10. **PaymentOptionsFrame.java**:
    - Creates a Swing-based frame for payment options with radio buttons for payment modes, an address input field, and a proceed button.
    - Validates user inputs, saves order details to the database, generates a receipt, and opens the home screen upon payment completion.

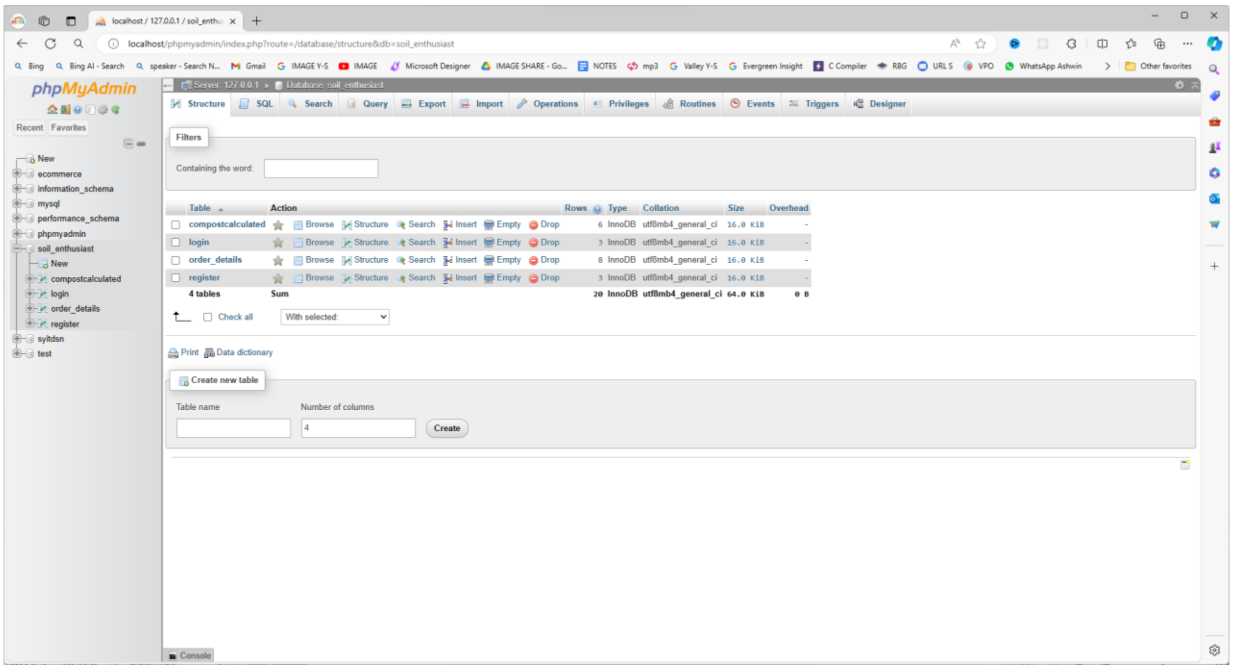
Data Flow:

The data flow within the Soil Enthusiast application begins with the Soil_Enthusiast_Home.java class, which creates the main graphical user interface (GUI) window displaying image labels and buttons for website demos, videos, and user login.





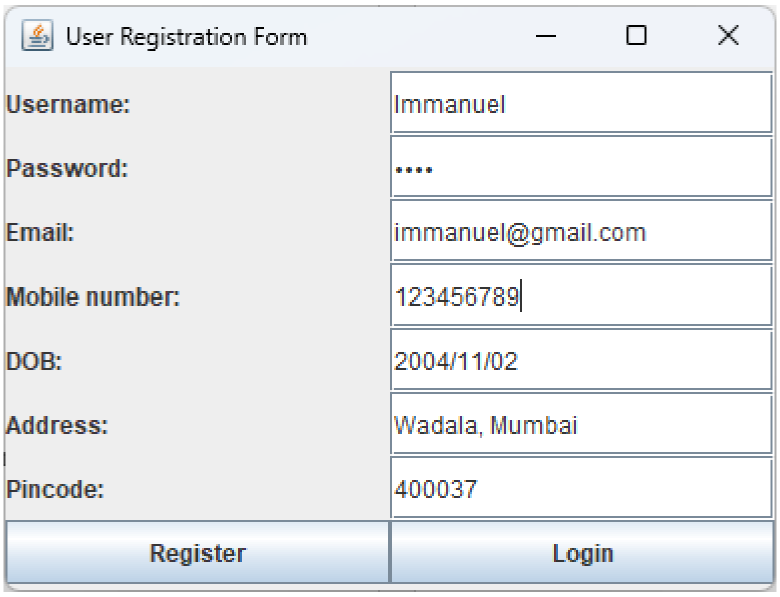
Upon user interaction, the SoilEnthusiast.java class executes commands related to database setup and user registration, establishing connections with a MySQL database through DatabaseSetup.java and TableSetup.java to create required databases and tables.


localhost/phpmyadmin/index.php?route=/database/structure&db=soil_enthusiast

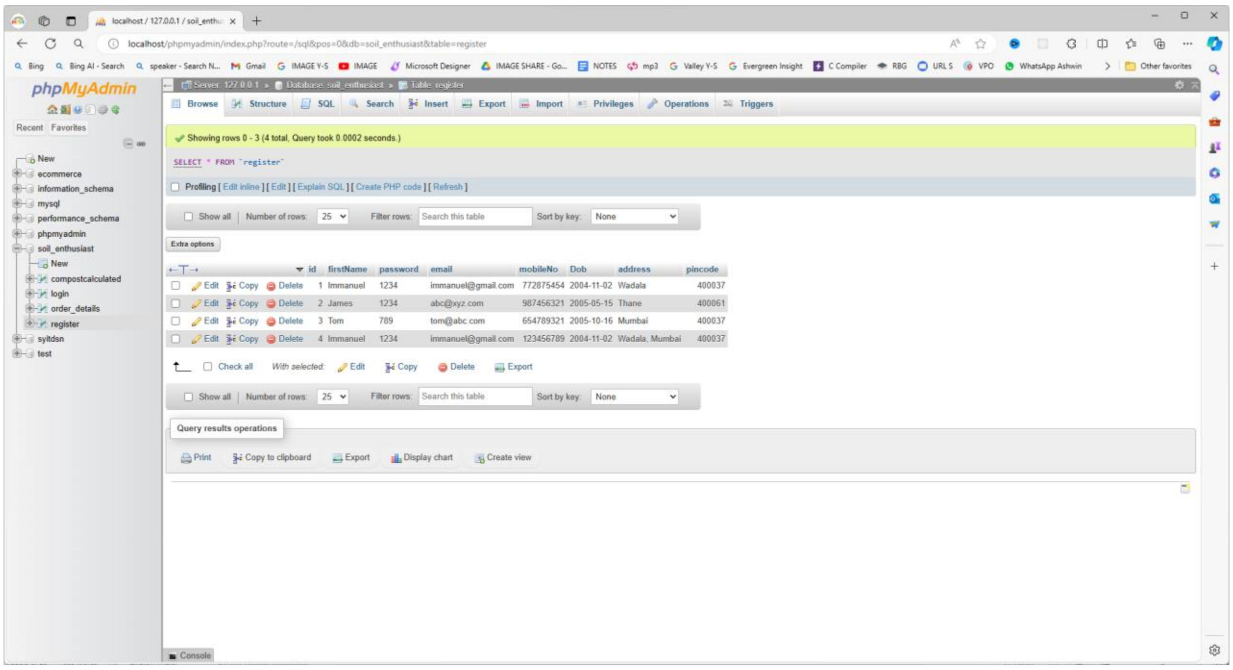
Server: 127.0.0.1 » Database: soil_enthusiast

Table	Action	Rows	Type	Collation	Size	Overhead
compostcalculated	Browse Structure Search Insert Empty Drop	6	InnoDB	utf8mb4_general_ci	16.0 KiB	-
login	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-
order_details	Browse Structure Search Insert Empty Drop	8	InnoDB	utf8mb4_general_ci	16.0 KiB	-
register	Browse Structure Search Insert Empty Drop	3	InnoDB	utf8mb4_general_ci	16.0 KiB	-
4 tables	Sum	20	InnoDB	utf8mb4_general_ci	64.0 KiB	0 B

The UserRegistrationForm.java class then generates a user registration form with input fields and validation, allowing users to register their details.



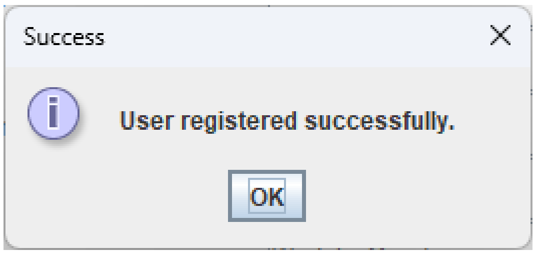
Username:	Immanuel
Password:	••••
Email:	immanuel@gmail.com
Mobile number:	123456789
DOB:	2004/11/02
Address:	Wadala, Mumbai
Pincode:	400037


localhost/phpmyadmin/index.php?route=/sql&pos=0&db=soil_enthusiast&table=register
Server: 127.0.0.1 » Database: soil_enthusiast » Table: register
Browse
Structure
SQL
Search
Insert
Export
Import
Privileges
Operations
Triggers

Showing rows 0 - 3 (4 total, Query took 0.0002 seconds.)

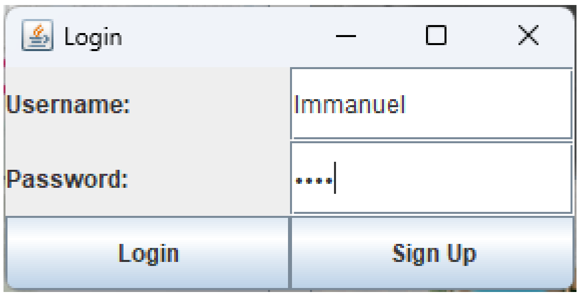
```
SELECT * FROM `register`
```

id	firstName	password	email	mobileNo	Dob	address	pincode
1	Immanuel	1234	immanuel@gmail.com	772875454	2004-11-02	Wadala	400037
2	James	1234	abc@xyz.com	987456321	2005-05-15	Thane	400061
3	Tom	789	tom@abc.com	654789321	2005-10-16	Mumbai	400037
4	Immanuel	1234	immanuel@gmail.com	123456789	2004-11-02	Wadala, Mumbai	400037



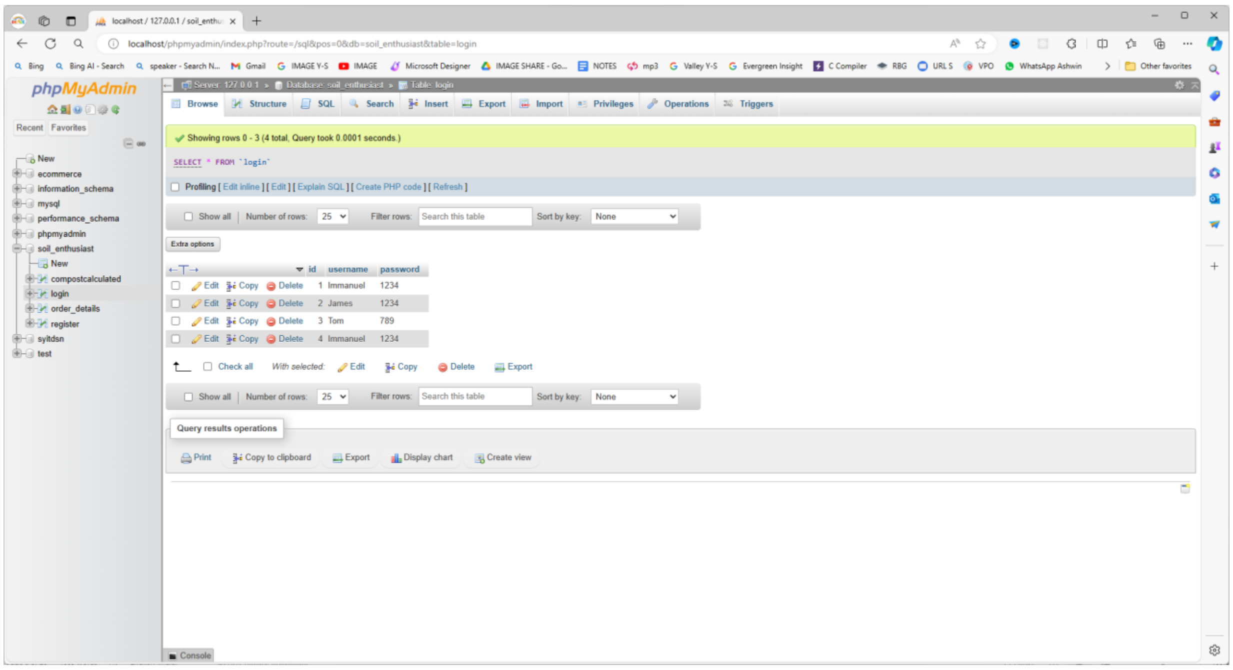
User registered successfully.

The LoginFrame.java class handles user authentication through a login form, granting access to the UserDashboard.java interface upon successful login.



Login

Username:	Immanuel
Password:	••••



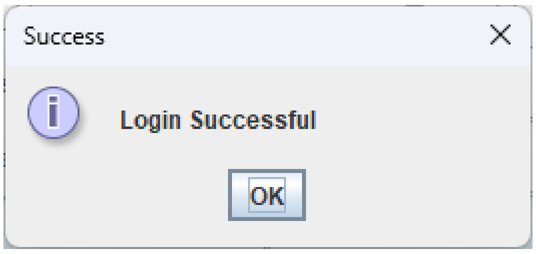
localhost/phpmyadmin/index.php?route=/sql&pos=0&db=soil_enthusiast&table=login

Server: 127.0.0.1 » Database: soil_enthusiast » Table: login

Showing rows 0 - 3 (4 total, Query took 0.0001 seconds.)

```
SELECT * FROM `login`
```

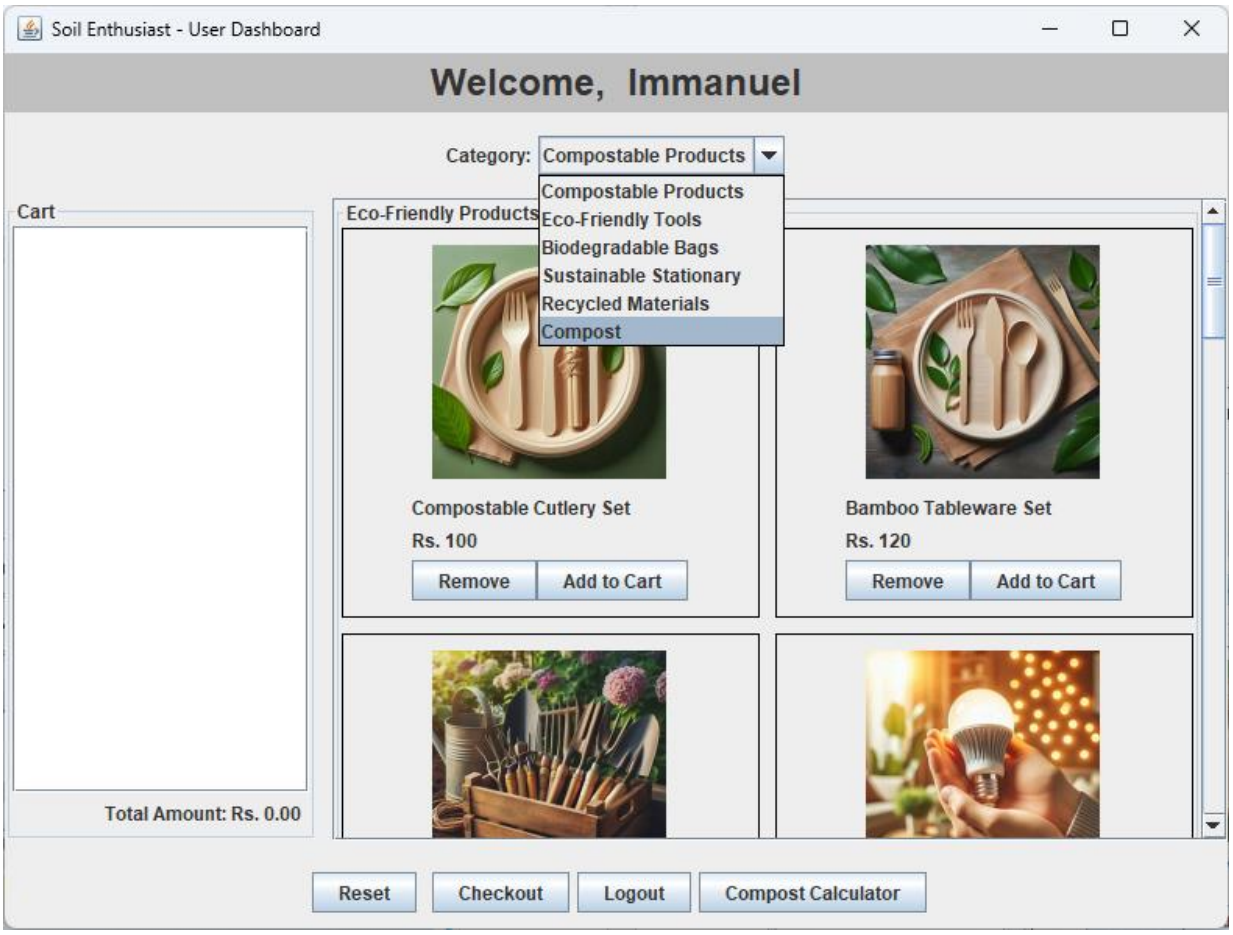
id	username	password
1	Immanuel	1234
2	James	1234
3	Tom	789
4	Immanuel	1234



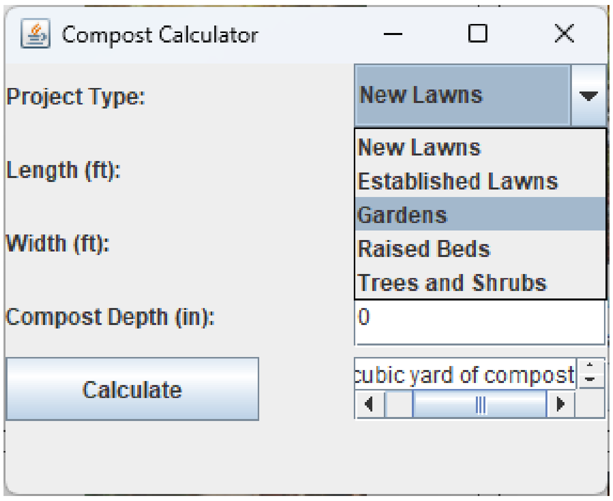
Success

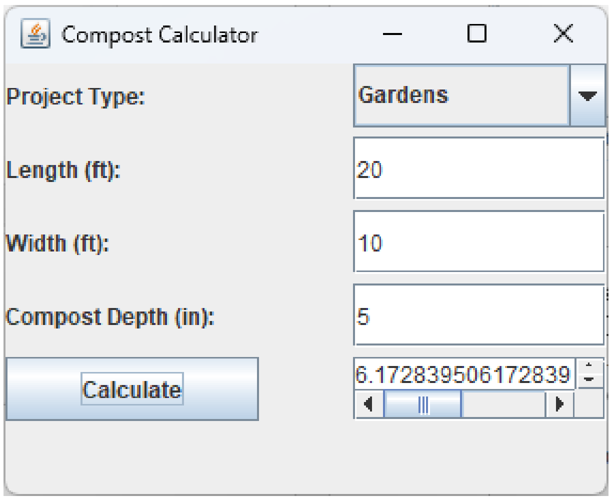
Login Successful

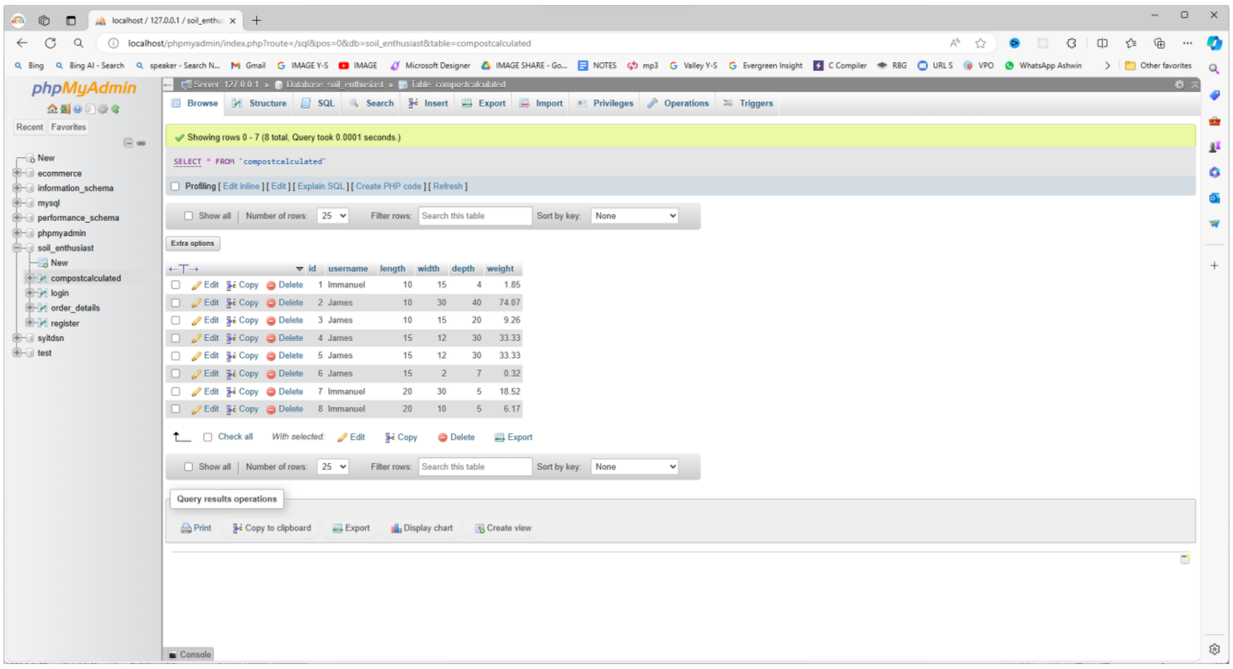
Within the UserDashboard, users can interact with product categories, manage cart functionalities, perform checkout processes.



The users can utilize the compost calculator feature provided by CompostCalculatorFrame.java.



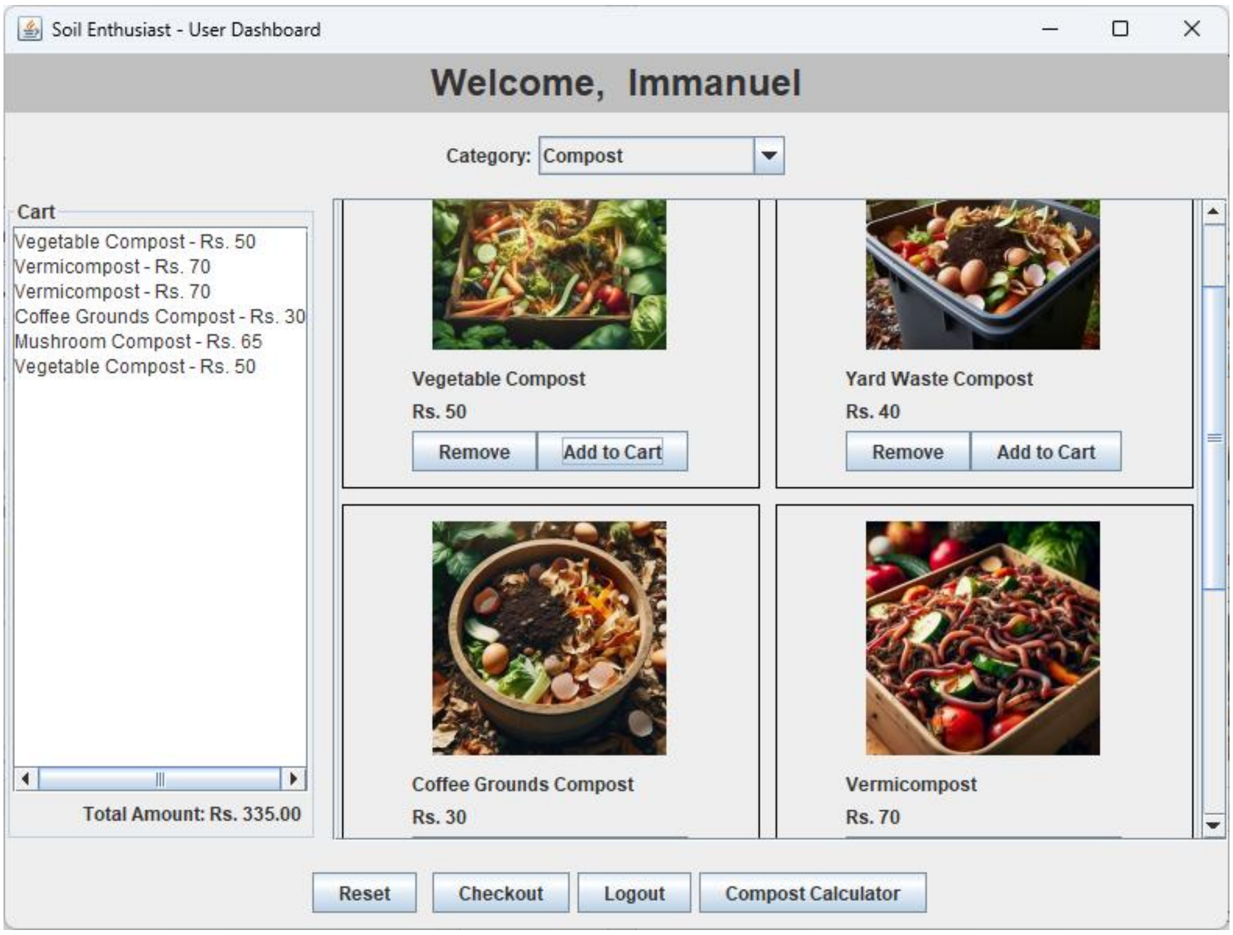



Server: 127.0.0.1 » Database: soil_enthusiast » Table: compostcalculated

Showing rows 0 - 7 (8 total, Query took 0.0001 seconds.)

```
SELECT * FROM `compostcalculated`
```

id	username	length	width	depth	weight
1	Immanuel	10	15	4	1.85
2	James	10	30	40	74.07
3	James	10	15	20	9.26
4	James	15	12	30	33.33
5	James	15	12	30	33.33
6	James	15	2	7	0.32
7	Immanuel	20	30	5	18.52
8	Immanuel	20	10	5	6.17


Soil Enthusiast - User Dashboard

Welcome, Immanuel

Category: Compost

Cart

Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50

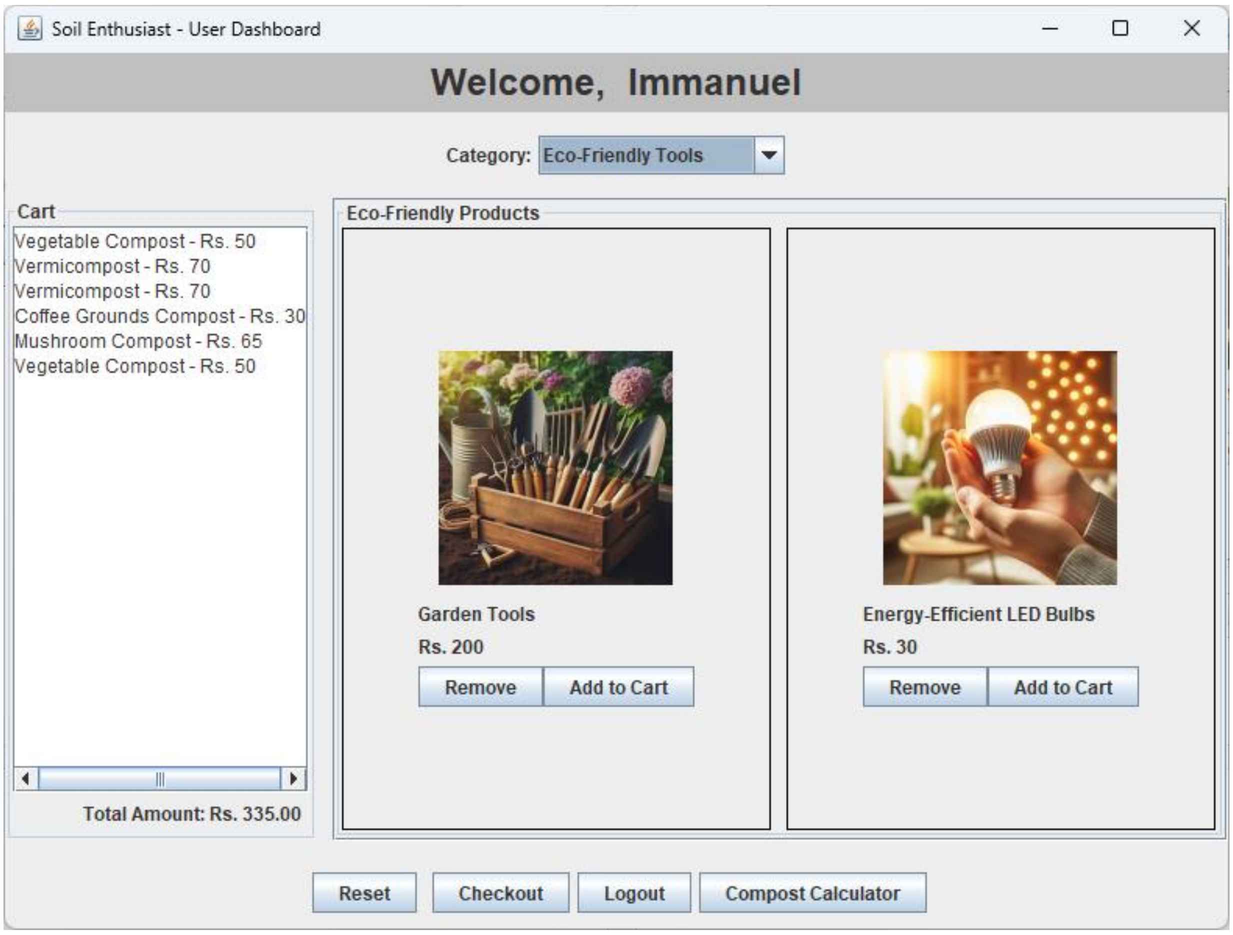
Total Amount: Rs. 335.00

Vegetable Compost
Rs. 50

Yard Waste Compost
Rs. 40

Coffee Grounds Compost
Rs. 30

Vermicompost
Rs. 70


Soil Enthusiast - User Dashboard

Welcome, Immanuel

Category: Eco-Friendly Tools

Cart

Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50

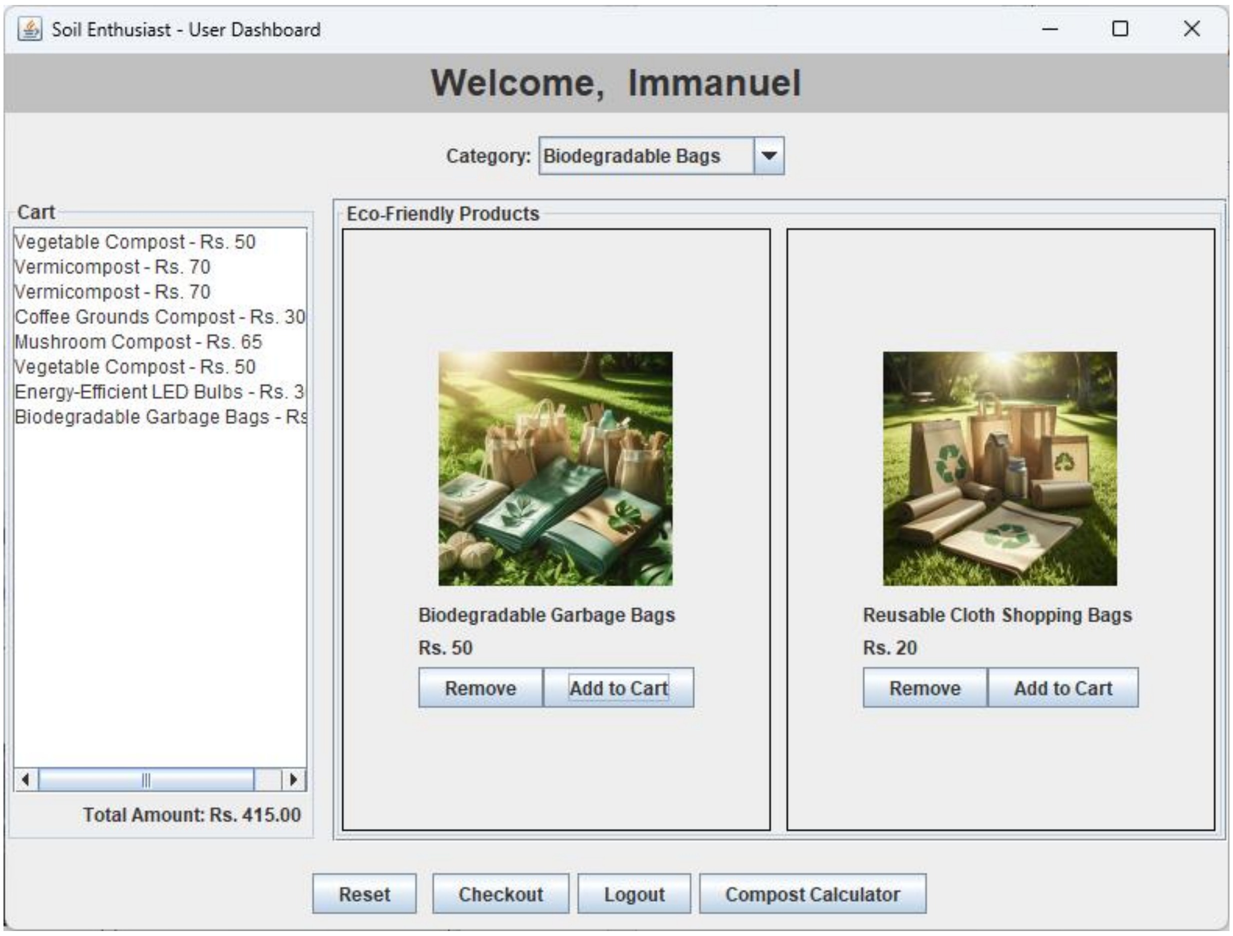
Total Amount: Rs. 335.00

Eco-Friendly Products

Garden Tools
Rs. 200
Remove | Add to Cart

Energy-Efficient LED Bulbs
Rs. 30
Remove | Add to Cart

Reset | Checkout | Logout | Compost Calculator


Soil Enthusiast - User Dashboard

Welcome, Immanuel

Category: Biodegradable Bags

Cart

Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50
Energy-Efficient LED Bulbs - Rs. 3
Biodegradable Garbage Bags - Rs

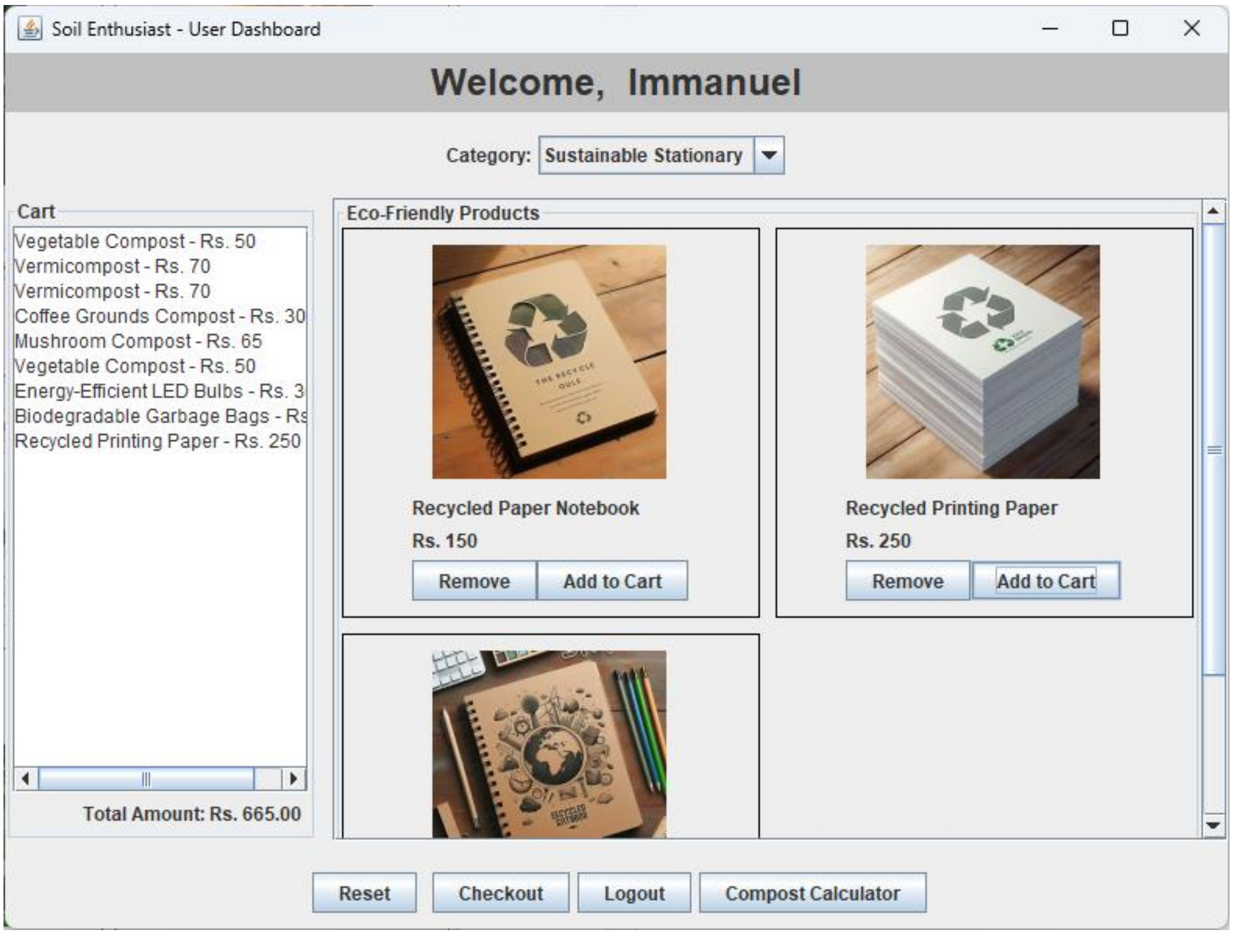
Total Amount: Rs. 415.00

Eco-Friendly Products

Biodegradable Garbage Bags
Rs. 50
Remove | Add to Cart

Reusable Cloth Shopping Bags
Rs. 20
Remove | Add to Cart

Reset | Checkout | Logout | Compost Calculator


Soil Enthusiast - User Dashboard

Welcome, Immanuel

Category: Sustainable Stationary

Cart

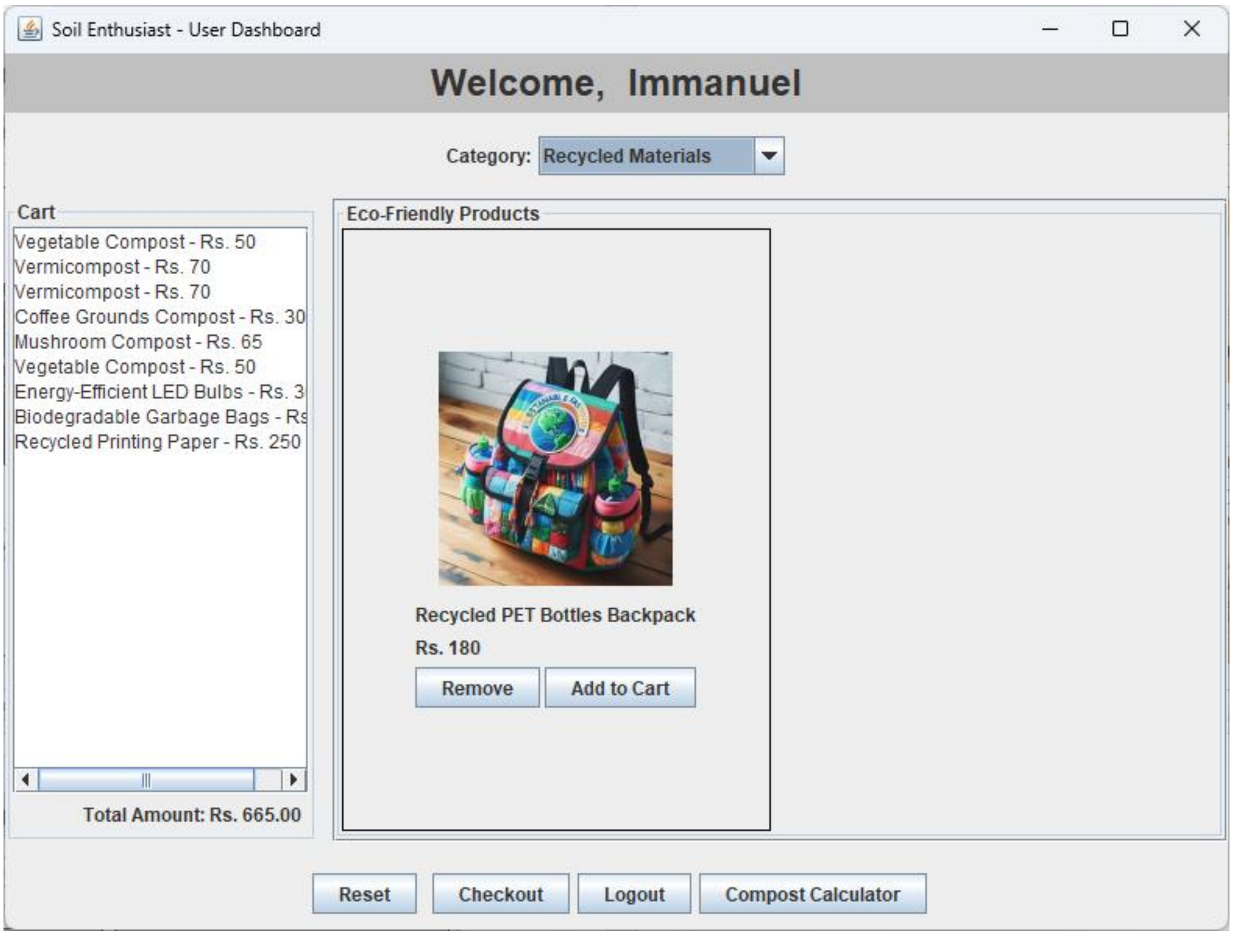
Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50
Energy-Efficient LED Bulbs - Rs. 3
Biodegradable Garbage Bags - Rs
Recycled Printing Paper - Rs. 250

Total Amount: Rs. 665.00

Eco-Friendly Products

Recycled Paper Notebook
Rs. 150

Recycled Printing Paper
Rs. 250


Soil Enthusiast - User Dashboard

Welcome, Immanuel

Category: Recycled Materials

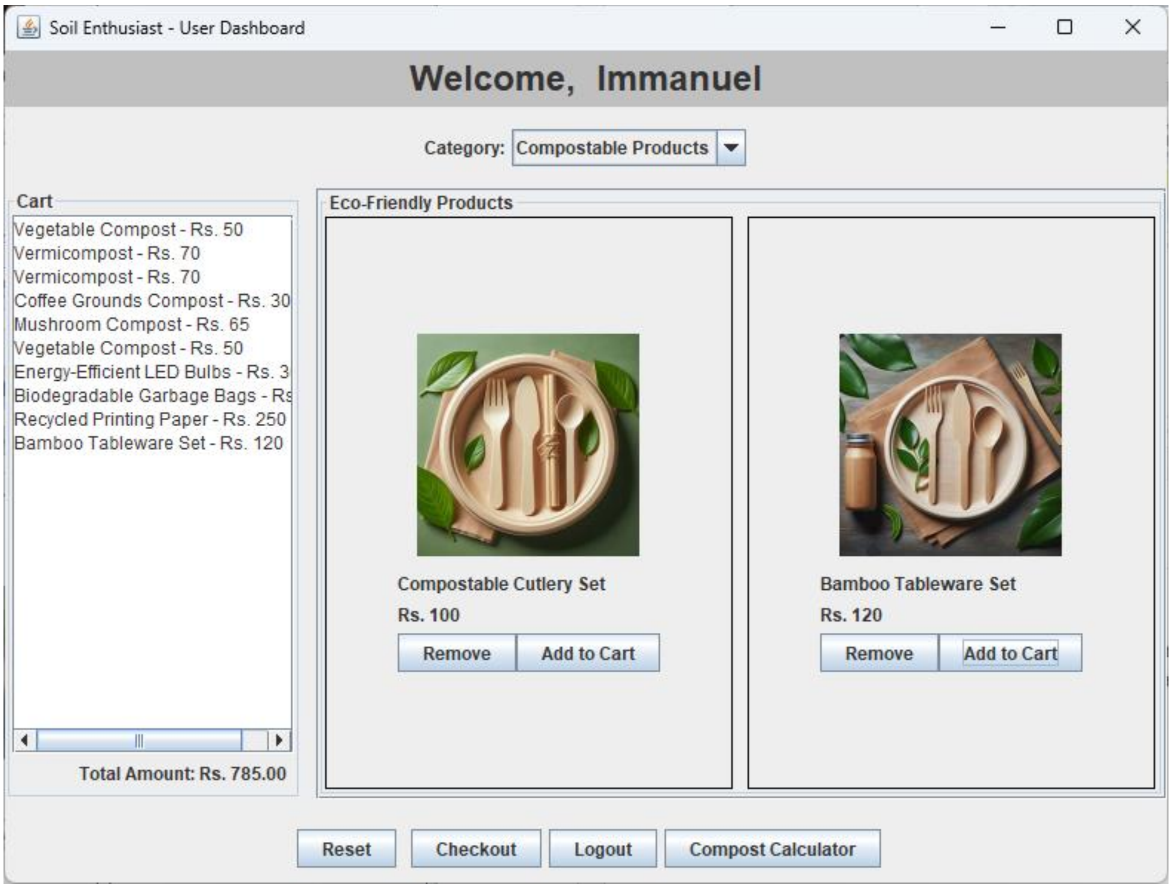
Cart

Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50
Energy-Efficient LED Bulbs - Rs. 3
Biodegradable Garbage Bags - Rs
Recycled Printing Paper - Rs. 250

Total Amount: Rs. 665.00

Eco-Friendly Products

Recycled PET Bottles Backpack
Rs. 180


Soil Enthusiast - User Dashboard

Welcome, Immanuel

Category: Compostable Products

Cart

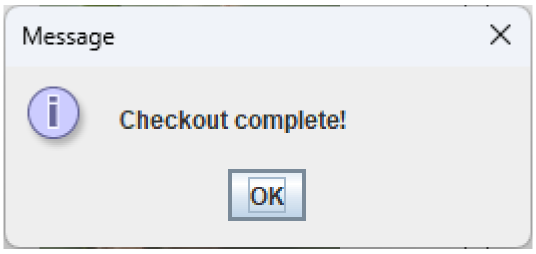
```
Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50
Energy-Efficient LED Bulbs - Rs. 3
Biodegradable Garbage Bags - Rs
Recycled Printing Paper - Rs. 250
Bamboo Tableware Set - Rs. 120
```

Total Amount: Rs. 785.00

Eco-Friendly Products

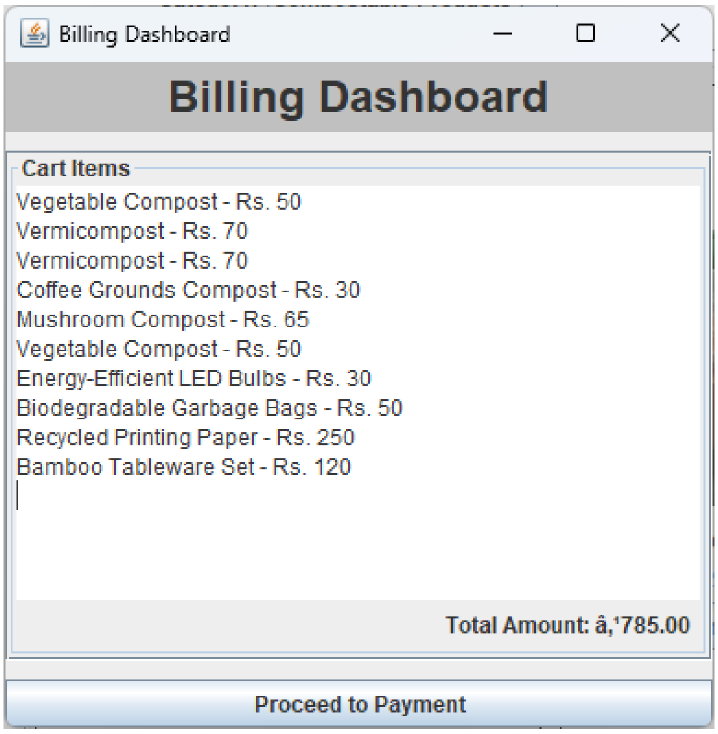
Compostable Cutlery Set
Rs. 100

Bamboo Tableware Set
Rs. 120


Message

Checkout complete!

The BillingDashboard.java class manages cart summary and payment options.

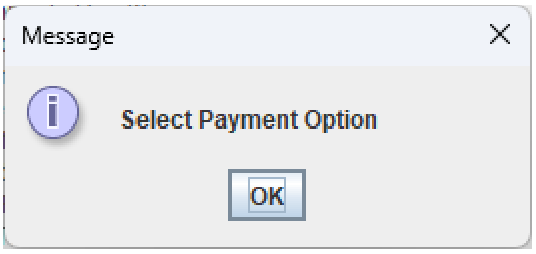

Billing Dashboard

Billing Dashboard

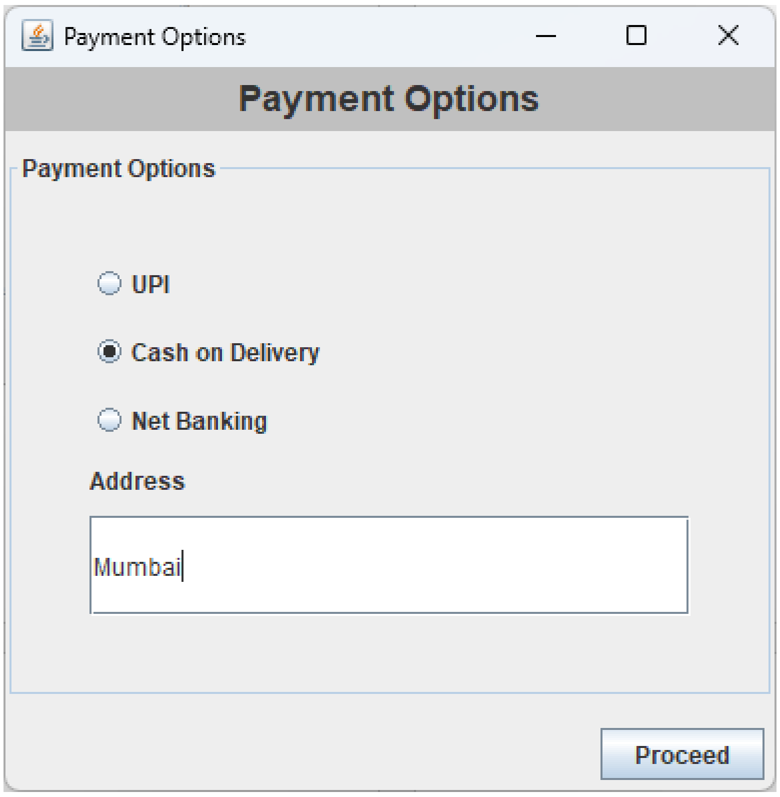
Cart Items

```
Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50
Energy-Efficient LED Bulbs - Rs. 30
Biodegradable Garbage Bags - Rs. 50
Recycled Printing Paper - Rs. 250
Bamboo Tableware Set - Rs. 120
```

Total Amount: â,¹785.00


Message

Select Payment Option

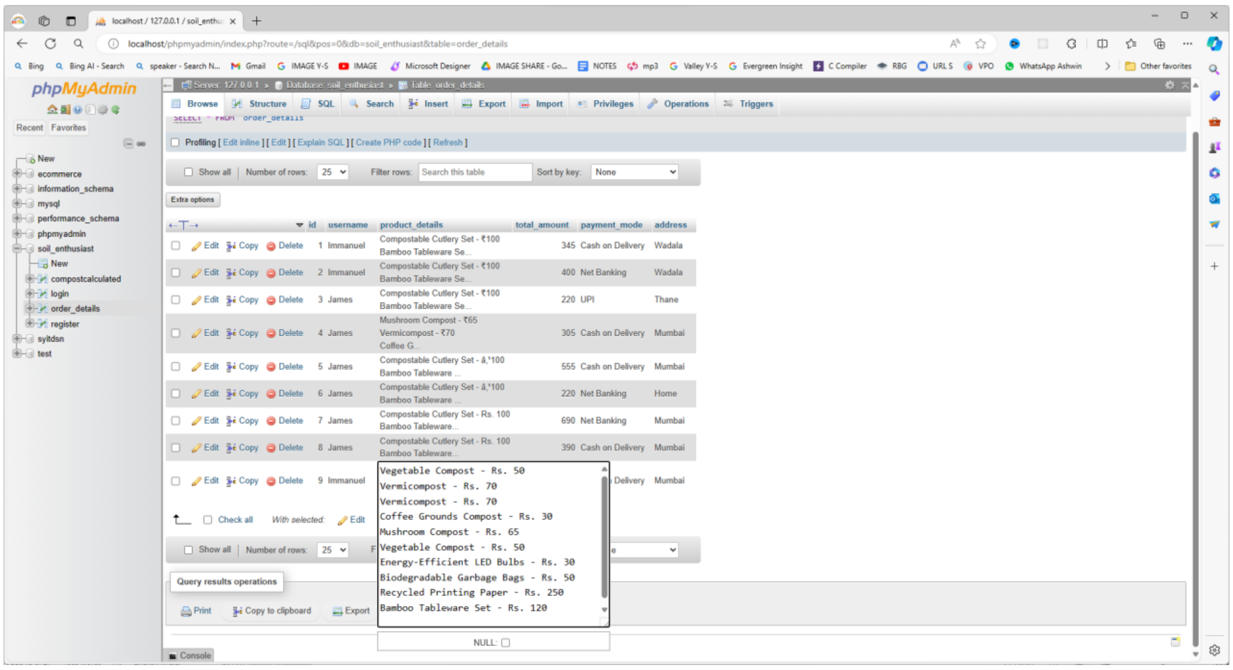

Payment Options

Payment Options

Payment Options

UPI [✓ Cash on Delivery] Net Banking

Address: Mumbai

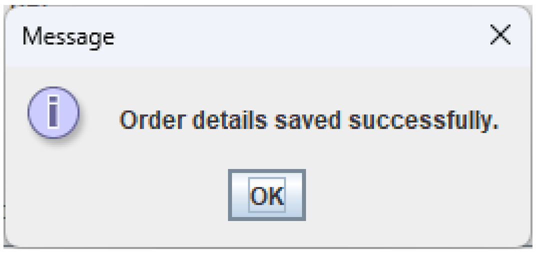

localhost/phpmyadmin/index.php?route=/sql&pos=0&db=soil_enthusiast&table=order_details

Server: 127.0.0.1 » Database: soil_enthusiast » Table: order_details

```
SELECT * FROM order_details
```

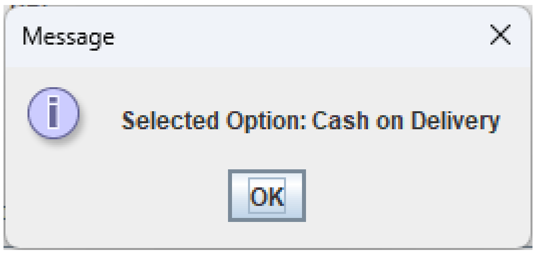
Show all | Number of rows: 25 | Filter rows: Search this table | Sort by key: None

id	username	product_details	total_amount	payment_mode	address
1	Immanuel	Compostable Cutlery Set - ₹100 Bamboo Tableware Se...	345	Cash on Delivery	Wadala
2	Immanuel	Compostable Cutlery Set - ₹100 Bamboo Tableware Se...	400	Net Banking	Wadala
3	James	Compostable Cutlery Set - ₹100 Bamboo Tableware Se...	220	UPI	Thane
4	James	Mushroom Compost - ₹65 Vermicompost - ₹70 Coffee G...	305	Cash on Delivery	Mumbai
5	James	Compostable Cutlery Set - â‚¹100 Bamboo Tableware ...	555	Cash on Delivery	Mumbai
6	James	Compostable Cutlery Set - â‚¹100 Bamboo Tableware ...	220	Net Banking	Home
7	James	Compostable Cutlery Set - Rs. 100 Bamboo Tableware...	690	Net Banking	Mumbai
8	James	Compostable Cutlery Set - Rs. 100 Bamboo Tableware...	390	Cash on Delivery	Mumbai
9	Immanuel	Vegetable Compost - Rs. 50 Vermicompost - Rs. 70 Vermicompost - Rs. 70 Coffee Grounds Compost - Rs. 30 Mushroom Compost - Rs. 65 Vegetable Compost - Rs. 50 Energy-Efficient LED Bulbs - Rs. 30 Biodegradable Garbage Bags - Rs. 50 Recycled Printing Paper - Rs. 250 Bamboo Tableware Set - Rs. 120		Delivery	Mumbai

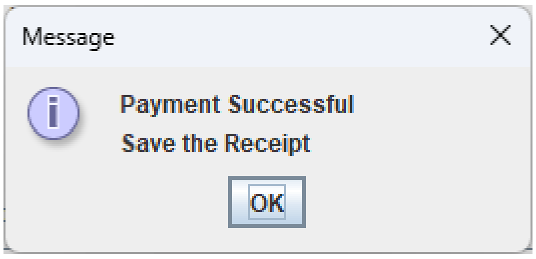

Message

Order details saved successfully.

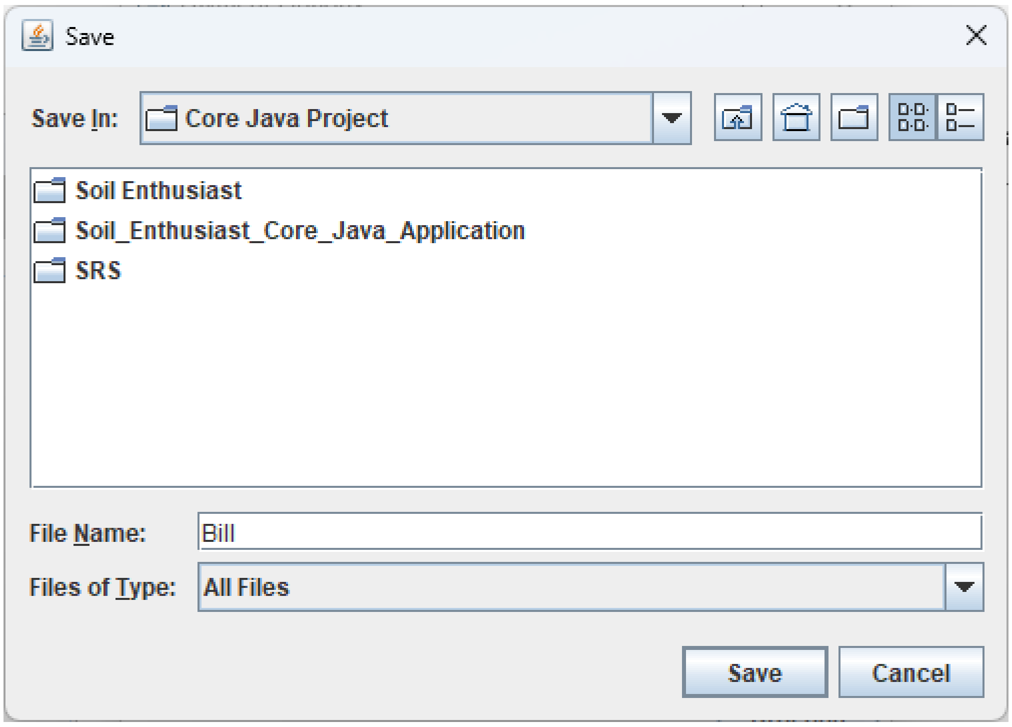
The PaymentOptionsFrame.java where order details are validated and processed, completing the data flow cycle within the application.


Message

Selected Option: Cash on Delivery


Message

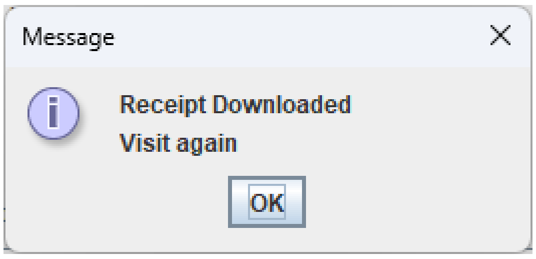
Payment Successful
Save the Receipt


Save

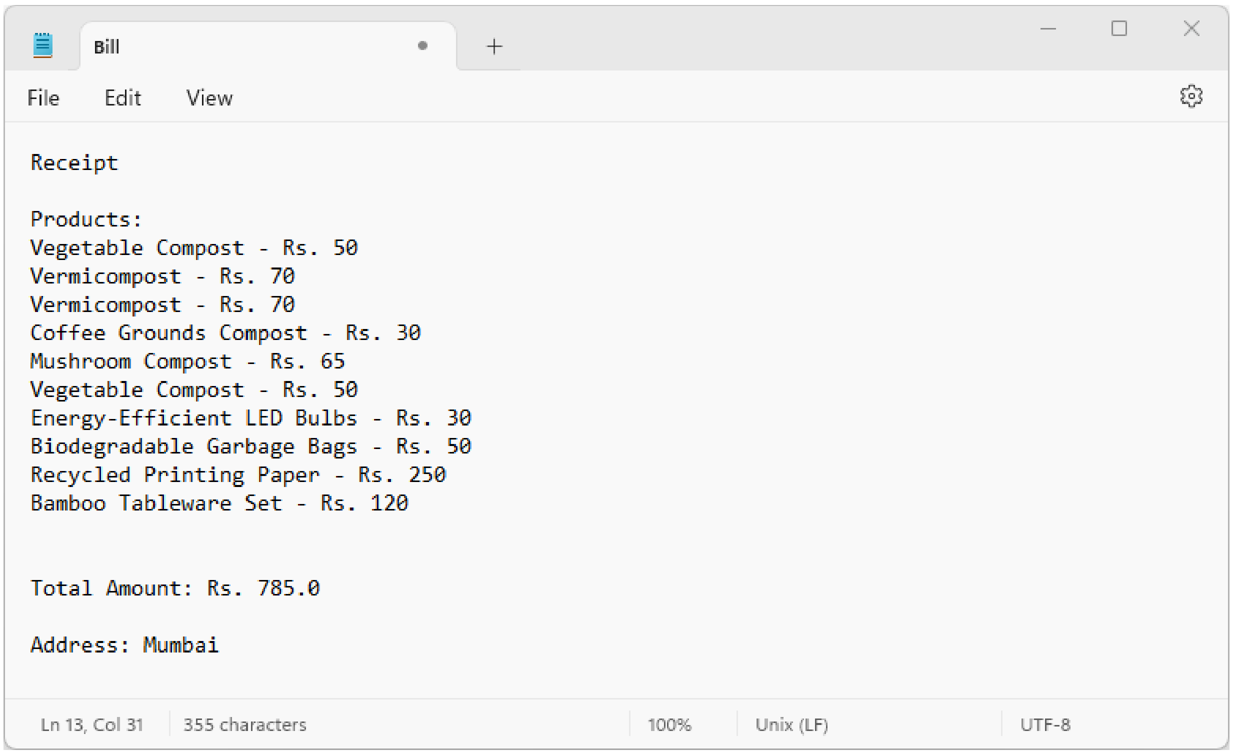
Save In: Core Java Project

Soil Enthusiast
Soil_Enthusiast_Core_Java_Application
SRS

File Name: Bill
Files of Type: All Files


Message

Receipt Downloaded
Visit again



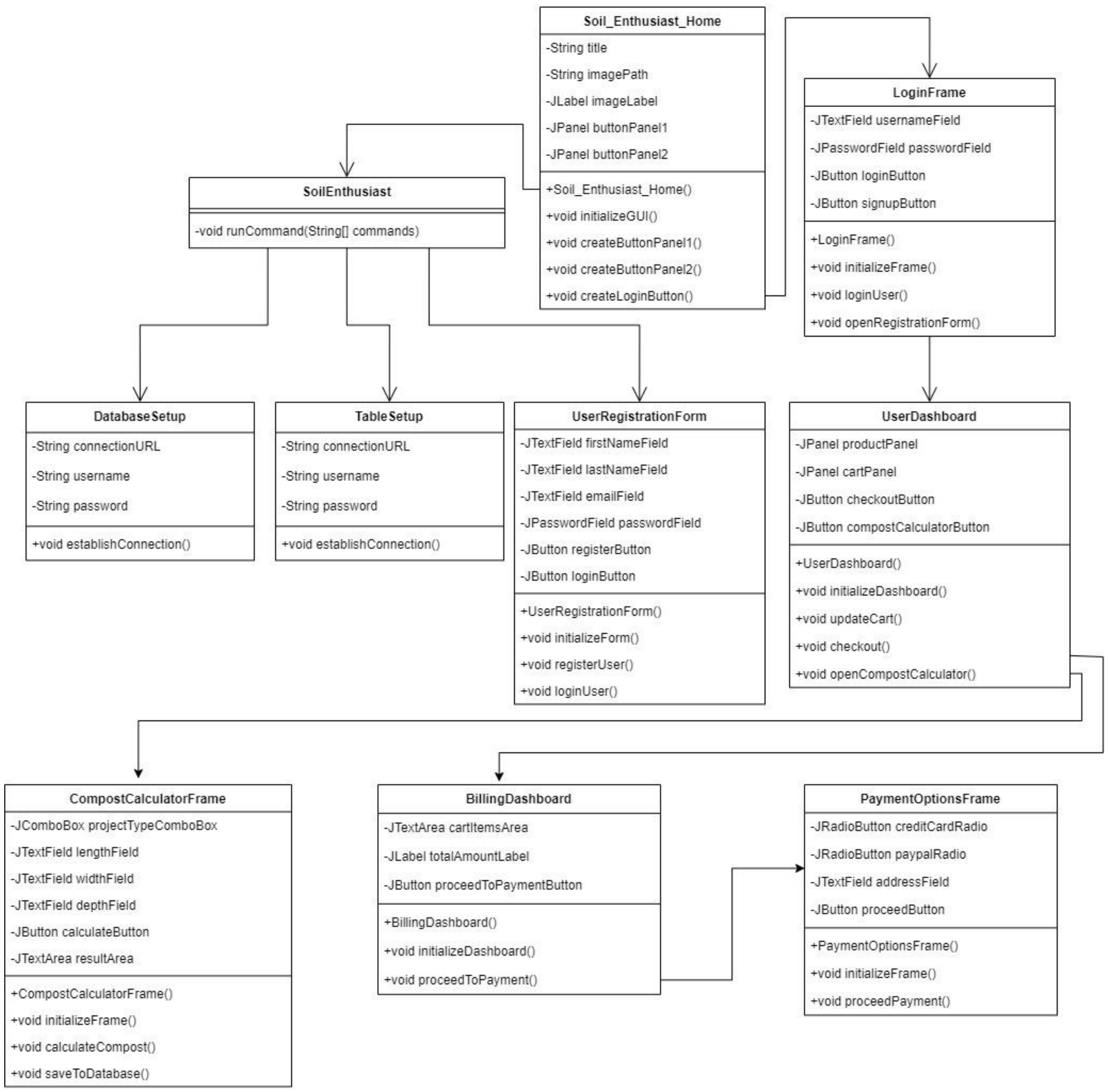
```
Receipt

Products:
Vegetable Compost - Rs. 50
Vermicompost - Rs. 70
Vermicompost - Rs. 70
Coffee Grounds Compost - Rs. 30
Mushroom Compost - Rs. 65
Vegetable Compost - Rs. 50
Energy-Efficient LED Bulbs - Rs. 30
Biodegradable Garbage Bags - Rs. 50
Recycled Printing Paper - Rs. 250
Bamboo Tableware Set - Rs. 120


Total Amount: Rs. 785.0

Address: Mumbai
```

Class diagram:

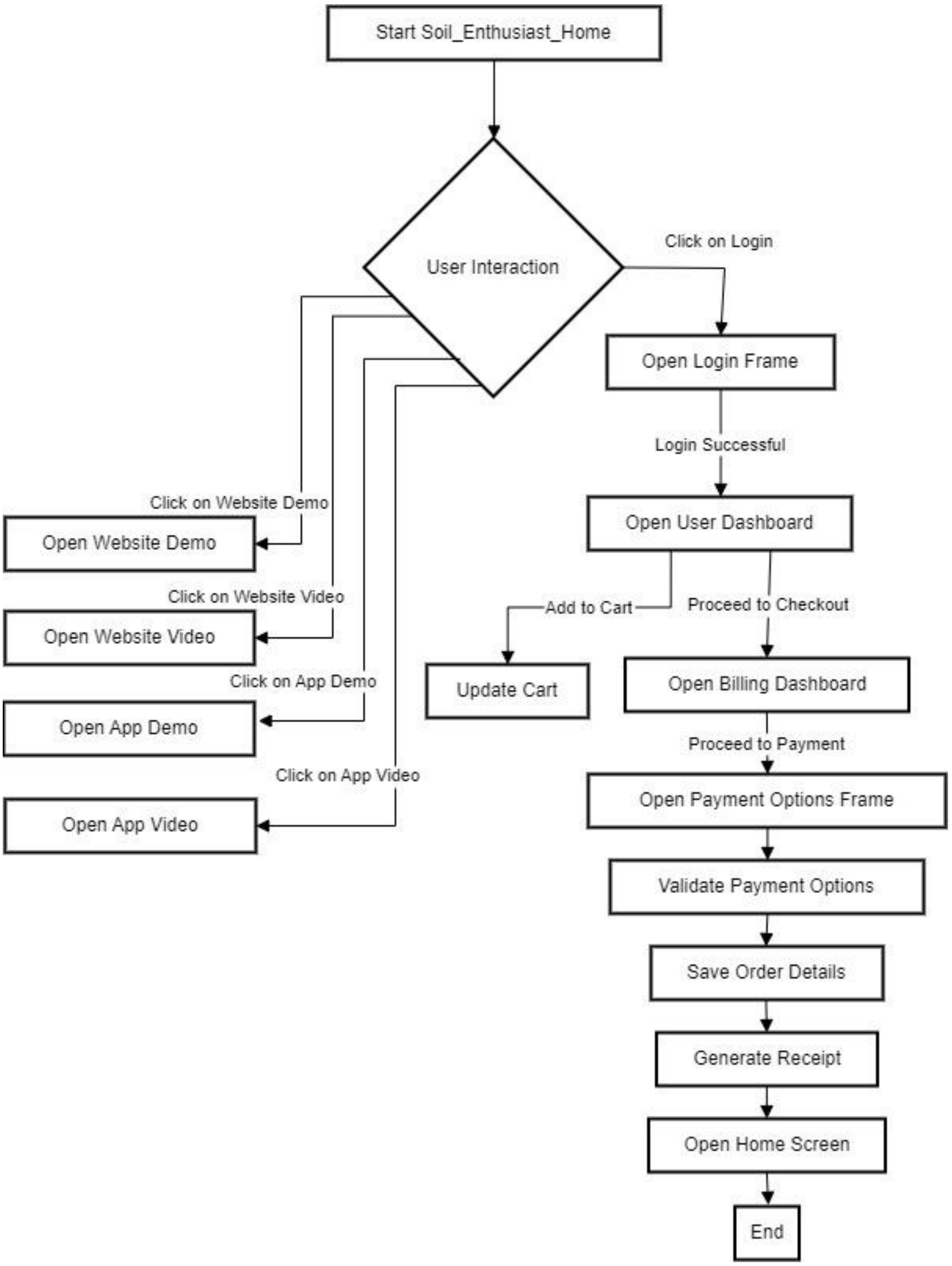
The class diagram for the Soil Enthusiast application encompasses several key classes and their relationships, highlighting the structure and interactions within the system.

1. **Soil_Enthusiast_Home.java:**
    - Extends JFrame to create the main GUI window with image labels and buttons.
    - Includes buttons for website demos, videos, and login.
2. **SoilEnthusiast.java:**
    - Executes commands related to database setup and user registration.
3. **DatabaseSetup.java:**
    - Establishes a connection to a MySQL database and creates required databases and tables.
4. **TableSetup.java:**
    - Establishes a connection to a MySQL database and creates necessary tables.
5. **UserRegistrationForm.java:**
    - Creates a user registration form with input fields and validation.
6. **LoginFrame.java:**
    - Creates a login form for user authentication.
7. **UserDashboard.java:**
    - Implements the user dashboard with product categories, cart functionality, checkout, and compost calculator.

8. **CompostCalculatorFrame.java:**
   - Creates a frame for compost calculation based on user inputs.
9. **BillingDashboard.java:**
   - Creates a billing dashboard for cart summary and payment options.
10. **PaymentOptionsFrame.java:**
    - Creates a frame for payment options with validation and order details processing.

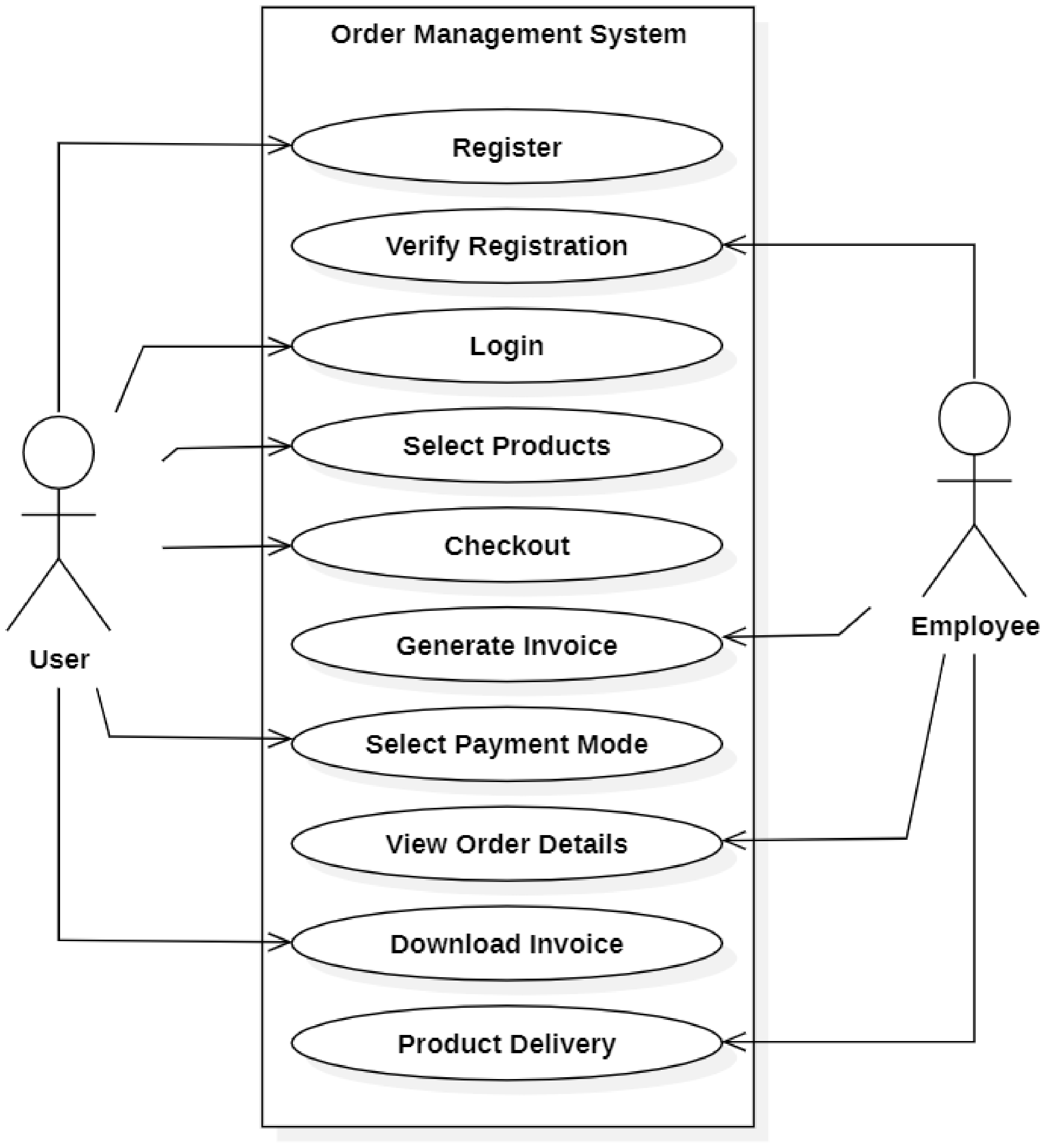


Flow Chart:



Start Soil_Enthusiast_Home
User Interaction
Click on Login
Open Login Frame
Login Successful
Open User Dashboard
Add to Cart
Update Cart
Proceed to Checkout
Open Billing Dashboard
Proceed to Payment
Open Payment Options Frame
Validate Payment Options
Save Order Details
Generate Receipt
Open Home Screen
End
Click on Website Demo
Open Website Demo
Click on Website Video
Open Website Video
Click on App Demo
Open App Demo
Click on App Video
Open App Video

Use Case Diagram:



Algorithm

Soil_Enthusiast_Home.java:

- Creates the main GUI window with image labels and buttons for website demos, videos, and login.

SoilEnthusiast.java:

- Executes commands related to database setup and user registration.

DatabaseSetup.java:

- Establishes a connection to a MySQL database and creates required databases and tables.

TableSetup.java:

- Establishes a connection to a MySQL database and creates necessary tables.

UserRegistrationForm.java:

- Creates a user registration form with input fields and validation.

LoginFrame.java:

- Creates a login form for user authentication.

UserDashboard.java:

- Implements the user dashboard with product categories, cart functionality, checkout, and compost calculator.

CompostCalculatorFrame.java:

- Creates a frame for compost calculation based on user inputs.

BillingDashboard.java:

- Creates a billing dashboard for cart summary and payment options.

PaymentOptionsFrame.java:

- Creates a frame for payment options with validation and order details processing.

File Name: Soil_Enthusiast_Home.java

1. Start the Soil_Enthusiast_Home class which extends JFrame.
2. Set the title of the JFrame to "Soil Enthusiast".
3. Set the default close operation to EXIT_ON_CLOSE.
4. Set the layout of the JFrame to a 4x1 grid layout.
5. Create an ImageIcon by resizing the "welcome_image.jpg" to 65% of its original size and create a JLabel with this ImageIcon. Add the JLabel to the JFrame.
6. Create buttonPanel1 with a 1x2 grid layout and a horizontal and vertical gap of 20 pixels.
7. Resize "website_image.png" to 70% of its original size and "website_video.png" to 80% of its original size to create ImageIcons.
8. Create websiteButton with text "Website Demo" and the resized websiteImageIcon. Add an ActionListener to open the webpage "https://soil-enthusiast-2024.netlify.app/" when clicked.
9. Create websiteVideoButton with text "Website Video" and the resized websiteVideoIcon. Add an ActionListener to open the webpage "https://youtu.be/vrqANy-PgS0" when clicked.
10. Create buttonPanel2 with a 1x2 grid layout and a horizontal and vertical gap of 20 pixels.
11. Resize "app_image.jpg" and "app_video.jpg" to 70% of their original size to create ImageIcons.
12. Create appButton with text "App Demo" and the resized appImageIcon. Add an ActionListener to open the webpage "https://youtu.be/j24O1qllG2o" when clicked.
13. Create appVideoButton with text "App Video" and the resized appVideoIcon. Add an ActionListener to open the webpage "https://youtu.be/Y3IPMGGflmY" when clicked.
14. Add appVideoButton and websiteVideoButton to buttonPanel1, and add appButton and websiteButton to buttonPanel2.
15. Add buttonPanel1 and buttonPanel2 to the JFrame.
16. Resize "login_image.jpg" to 65% of its original size to create an ImageIcon for the loginButton. Add an ActionListener to open a new instance of SoilEnthusiast when clicked.
17. Set the size of the JFrame to 1280x720 pixels.
18. Set the location of the JFrame to the center of the screen and make it visible.

File Name: SoilEnthusiast.java

1. **Define SoilEnthusiast Class**:
    - Define a class named **SoilEnthusiast**.
    - Implement a constructor for the class to perform certain actions when an instance is created.
2. **Constructor Actions**:
    - Inside the constructor, perform the following actions:
        - Execute the **runCommand** method with arguments to compile and run **DatabaseSetup.java**.
        - Execute the **runCommand** method with arguments to compile and run **TableSetup.java**.
        - Execute the **runCommand** method with arguments to compile and run **UserRegistrationForm.java**.
3. **Run Command Method**:
    - Define a method named **runCommand** that takes a variable number of String arguments representing a command to be executed.
    - Inside the **runCommand** method, create a **ProcessBuilder** instance with the provided command arguments.
    - Try to start the process using **process.waitFor()** to wait for the process to complete.
    - Handle **IOException** and **InterruptedException** if they occur during process execution.
4. **Main Method**:
    - Define the **main** method.
    - Inside the **main** method, create an instance of **SoilEnthusiast** to run the commands specified in its constructor.
5. **Execution Flow**:
    - When the program is executed, it creates an instance of **SoilEnthusiast**.
    - The constructor of **SoilEnthusiast** is invoked, which in turn executes the specified commands using the **runCommand** method.
    - Each command compiles and runs a Java file (**DatabaseSetup.java**, **TableSetup.java**, **UserRegistrationForm.java**) using the command line.

File Name: DatabaseSetup.java

1. **Import Required Libraries**:
    - Import necessary Java libraries for JDBC (Java Database Connectivity) and logging.
2. **Define DatabaseSetup Class**:
    - Define a class named **DatabaseSetup**.
3. **Class Variables**:
    - Define class variables for JDBC connection URL, username, password, and a Logger instance for logging.
4. **Main Method**:
    - Define the **main** method as the entry point of the class.
    - Inside the **main** method, establish a connection to the MySQL database using **DriverManager.getConnection**.
5. **Database Setup**:
    - Inside the try-with-resources block for the connection, create a **PreparedStatement** to execute SQL commands.
    - Use the **PreparedStatement** to execute an SQL command to create the database **soil_enthusiast** if it does not already exist (**CREATE DATABASE IF NOT EXISTS soil_enthusiast**).
6. **Exception Handling**:
    - Catch **SQLException** if any database-related errors occur during execution.
    - Use the logger to log the error message and stack trace at the **SEVERE** level.
7. **Execution Flow**:
    - When the program is executed, it attempts to establish a connection to the MySQL database using the specified URL, username, and password.
    - If the connection is successful, it executes an SQL command to create the database **soil_enthusiast** if it doesn't already exist.
    - If any errors occur during database setup, they are caught and logged using the logger.

File Name: TableSetup.java

1. **Import Required Libraries**:
    - Import necessary Java libraries for JDBC (Java Database Connectivity) and logging.
2. **Define TableSetup Class**:
    - Define a class named **TableSetup**.
3. **Class Variables**:
    - Define class variables for JDBC connection URL, username, password, and a Logger instance for logging.
    - Define SQL statements to create the required tables in the database.
4. **Main Method**:
    - Define the **main** method as the entry point of the class.
    - Inside the **main** method, establish a connection to the MySQL database using **DriverManager.getConnection**.
5. **Create Tables**:
    - Inside the try-with-resources block for the connection, call the **createTable** method for each SQL statement to create the required tables.
    - The **createTable** method takes a connection and an SQL statement as parameters and executes the statement to create the table.
6. **Exception Handling**:
    - Catch **SQLException** if any database-related errors occur during table creation.
    - Use the logger to log the error message and stack trace at the **SEVERE** level.
7. **Execution Flow**:
    - When the program is executed, it attempts to establish a connection to the MySQL database using the specified URL, username, and password.
    - If the connection is successful, it executes SQL statements to create the required tables (**login**, **register**, **order_details**, **CompostCalculated**) if they don't already exist.
    - If any errors occur during table creation, they are caught and logged using the logger.

File Name: UserRegistrationForm.java

1. **Import Required Libraries**:
   - Import necessary Java libraries for Swing components, event handling, and JDBC (Java Database Connectivity).
2. **Define UserRegistrationForm Class**:
   - Define a class named **UserRegistrationForm** that extends **JFrame** and implements the **ActionListener** interface.
3. **Instance Variables**:
   - Declare instance variables for the Swing components needed in the registration form, such as text fields, password field, buttons, and labels.
4. **Constructor**:
   - Implement the constructor for the registration form.
   - Set the title, default close operation, size, location, and layout (using **GridLayout**) for the frame.
   - Initialize all the Swing components and add them to the frame.
5. **Main Method**:
   - Define the **main** method to create an instance of **UserRegistrationForm** using **SwingUtilities.invokeLater**.
6. **ActionListener Implementation**:
   - Implement the **actionPerformed** method to handle button clicks and user registration logic.
   - Check which button is clicked using **e.getSource()** and perform the corresponding action.
   - If the register button is clicked:
     - Retrieve input values from the text fields and password field.
     - Validate the email format using a regular expression.
     - Establish a connection to the MySQL database using JDBC.
     - Prepare and execute SQL statements to insert user data into the **register** and **login** tables.
     - Display success or error messages using **JOptionPane**.
     - Clear the form fields after registration.
   - If the login button is clicked:
     - Open a new login frame (assuming the **LoginFrame** class exists).
     - Dispose of the registration frame to switch to the login frame.
7. **isValidEmail Method**:
   - Implement a method named **isValidEmail** to perform basic email validation using a regular expression.
8. **Execution Flow**:
   - When the program is executed, it creates an instance of **UserRegistrationForm** with the registration form GUI.
   - The user can input registration details, click the register button to store data in the database, or click the login button to switch to the login frame.
   - Input validation and database operations are performed based on user actions.

File Name: LoginFrame.java

1. **Import Required Libraries**:
   - Import necessary Java libraries for Swing components, event handling, and JDBC (Java Database Connectivity).
2. **Define LoginFrame Class**:
   - Define a class named **LoginFrame** that extends **JFrame** and implements the **ActionListener** interface.
3. **Instance Variables**:
   - Declare instance variables for the Swing components needed in the login frame, such as text fields, password field, and buttons.
4. **Constructor**:
   - Implement the constructor for the login frame.
   - Set the title, default close operation, size, and location for the frame.
   - Create a JPanel with a grid layout to organize the components.
   - Initialize and add the username label, username field, password label, password field, login button, and sign-up button to the panel.
   - Add the panel to the frame and set it visible.
5. **Main Method**:
   - Define the **main** method to create an instance of **LoginFrame** using **SwingUtilities.invokeLater**.
6. **ActionListener Implementation**:
   - Implement the **actionPerformed** method to handle button clicks and login/sign-up logic.
   - Check which button is clicked using **e.getSource()** and perform the corresponding action.
   - If the login button is clicked:
     - Retrieve the entered username and password.
     - Establish a connection to the MySQL database using JDBC.
     - Prepare and execute an SQL query to check if the username and password match in the **login** table.
     - If a match is found, show a success message and open the user dashboard frame (**UserDashboard**) passing the username.
     - Dispose of the login frame.
     - If no match is found, show an error message.
   - If the sign-up button is clicked:
     - Open a new user registration form (**UserRegistrationForm**).
     - Dispose of the login frame.
7. **Execution Flow**:
   - When the program is executed, it creates an instance of **LoginFrame** with the login form GUI.
   - The user can input login details, click the login button to authenticate, or click the sign-up button to register as a new user.
   - Input validation, database queries, and frame switching are performed based on user actions.

File Name: UserDashboard.java

1. **Import Required Libraries**:
    - Import necessary Java libraries for Swing components, event handling, image handling, and data structures.
2. **Define UserDashboard Class**:
    - Define a class named **UserDashboard** that extends **JFrame**.
    - Include instance variables for components like text areas, labels, buttons, total amount, product list, and username.
3. **Constructor**:
    - Implement the constructor to initialize the user dashboard.
    - Set the title, size, and layout of the frame.
    - Create panels for header, content, products, cart, and bottom buttons.
    - Add labels, combo box, scroll pane, buttons, and action listeners for various functionalities like category selection, adding/removing items to/from cart, resetting cart, checkout, logout, and compost calculator.
4. **Product Class**:
    - Define an inner class named **Product** to represent each product with attributes like name, image path, price, amount, and category.
5. **Update Product Panel**:
    - Implement the **updateProductPanel** method to refresh the products displayed based on the selected category.
    - Iterate over the product list and add product panels to the products panel if they match the selected category.
6. **Add Product Panel**:
    - Implement the **addProductPanel** method to create and add a panel for each product.
    - Include image handling using **ImageIO** to display product images.
    - Add buttons for adding/removing items to/from the cart with corresponding action listeners to update the cart and total amount.
7. **Reset Cart**:
    - Implement the **resetCart** method to clear the cart text area and reset the total amount to zero.
8. **Show Billing Dashboard**:
    - Implement the **showBillingDashboard** method to create and display the billing dashboard when the checkout button is clicked.
9. **Main Method**:
    - Define the **main** method to create an instance of **UserDashboard** with a sample username and make it visible.
10. **Execution Flow**:
    - When the program is executed, it creates an instance of **UserDashboard** with the user dashboard GUI.
    - Users can browse through different categories of eco-friendly products, add items to the cart, remove items from the cart, reset the cart, checkout, logout, and access the compost calculator.
    - Product information is displayed dynamically based on the selected category.
    - Cart updates and total amount calculations are performed based on user interactions with the buttons.

File Name: CompostCalculatorFrame.java

1. **Import Required Libraries**:
   - Import necessary Java libraries for Swing components, event handling, and database connectivity.
2. **Define CompostCalculatorFrame Class**:
   - Define a class named **CompostCalculatorFrame** that extends **JFrame**.
   - Include instance variables for components like labels, combo box, text fields, buttons, and result area.
   - Also, include a variable to store the username and the compost needed.
3. **Constructor**:
   - Implement the constructor to initialize the Compost Calculator frame.
   - Set the title, default close operation, layout, and visibility of the frame.
   - Create and add components such as project type combo box, text fields, calculate button, result area, and scroll pane.
   - Set default values for length, width, and depth based on inputs from the previous frame (if any).
   - Automatically calculate the compost needed upon frame initialization.
4. **Calculate Compost Method**:
   - Implement the **calculateCompost** method to calculate the compost needed based on project type, length, width, and depth.
   - Use switch cases to handle different project types and calculate the compost accordingly.
   - Update the result area with the calculated compost needed in cubic yards.
   - Save the calculated compost data to the database if the compost needed is greater than zero.
5. **Save to Database Method**:
   - Implement the **saveToDatabase** method to save the compost calculation data to the database.
   - Use JDBC to establish a connection to the MySQL database.
   - Prepare and execute an SQL INSERT statement to insert the compost calculation data into the database table **CompostCalculated**.
   - Catch and handle any SQL exceptions that may occur during database operations.
6. **Main Method**:
   - No main method is included in this class as it is meant to be used as a part of a larger application where the compost calculator frame is opened from another frame.

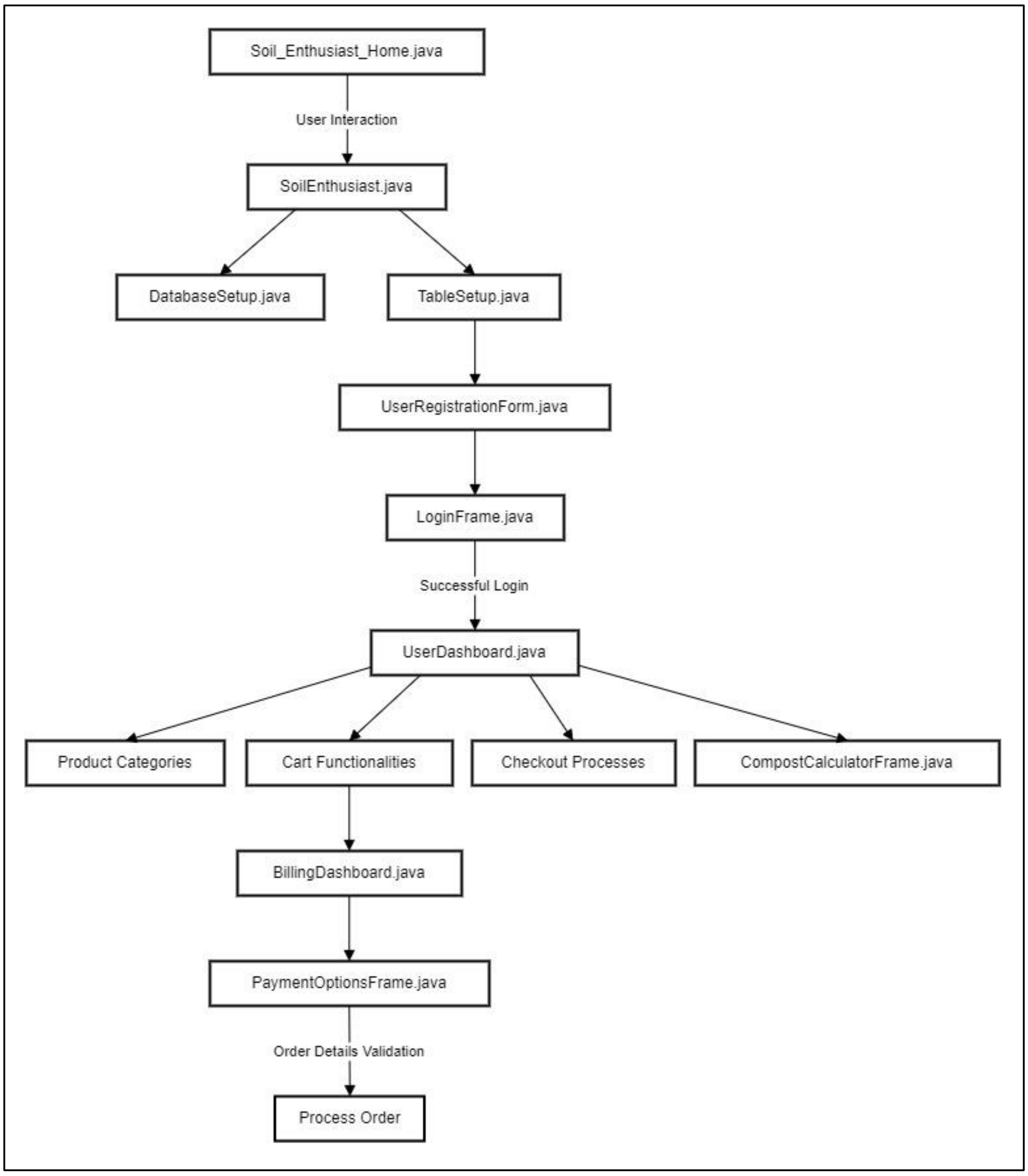
File Name: BillingDashboard.java

1. **Import Required Libraries**:
   - Import necessary Java libraries for Swing components and UI-related operations.
2. **Define BillingDashboard Class**:
   - Define a class named **BillingDashboard** that extends **JFrame**.
   - Include instance variables for components like **JTextArea** for cart items, **JLabel** for total amount, and the username.
3. **Constructor**:
   - Implement the constructor to initialize the Billing Dashboard frame.
   - Set the title, default close operation, size, and location of the frame.
   - Create and add components such as title label, cart items text area, total amount label, and proceed to payment button.
4. **Proceed to Payment Button ActionListener**:
   - Add an action listener to the "Proceed to Payment" button.
   - When the button is clicked, display a message dialog to prompt the user to select a payment option.
   - Open a new **PaymentOptionsFrame** passing the product list, total amount, and username.
   - Close the current **BillingDashboard** frame.
5. **Main Method**:
   - Include a main method that creates an instance of **BillingDashboard**.
   - Pass sample product data, total amount, and a username to the constructor to demonstrate the functionality.

File Name: PaymentOptionsFrame.java

1. **Import Required Libraries**:
    - Import necessary Java libraries for Swing components, UI-related operations, file handling, and database connectivity.
2. **Define PaymentOptionsFrame Class**:
    - Define a class named **PaymentOptionsFrame** that extends **JFrame**.
    - Include instance variables for components like **JTextField** for address input, **JLabel** for address label, **String** for username, and a **JPanel** for buttons.
3. **Constructor**:
    - Implement the constructor to initialize the Payment Options frame.
    - Set the title, default close operation, size, and location of the frame.
    - Create and add components such as payment option radio buttons, address input field, and a "Proceed" button.
4. **Proceed Button ActionListener**:
    - Add an action listener to the "Proceed" button.
    - When the button is clicked, validate the selected payment option and address input.
    - Save order details to the database using the **saveOrderDetails** method.
    - Show success messages, generate a receipt using the **generateReceipt** method, and dispose all frames to open the home screen.
5. **Database Connection and Saving Order Details**:
    - Implement the **saveOrderDetails** method to connect to the database and insert order details (username, products, total amount, payment mode, and address) into the **order_details** table.
6. **Generating Receipt**:
    - Implement the **generateReceipt** method to create a receipt file using a **BufferedWriter** and save it based on user choice using **JFileChooser**.
7. **Dispose All Frames and Open Home Screen**:
    - Implement the **disposeAllAndOpenHome** method to dispose all frames currently open and open the home screen.
8. **Main Method**:
    - Include a main method that creates an instance of **PaymentOptionsFrame** with sample data for demonstration purposes.

Order of Execution:



Overall Description

Product Perspective: The Soil Enthusiast application is a standalone system designed to provide users with a platform for managing eco-friendly products, user registrations, compost calculations, and payment processing. It operates independently and does not require integration with external systems.

Product Functions:

Soil_Enthusiast_Home.java: Creates the main GUI window with image labels and buttons for website demos, videos, and login.

SoilEnthusiast.java: Executes commands related to database setup and user registration.

DatabaseSetup.java: Establishes a connection to a MySQL database and creates required databases and tables.

TableSetup.java: Establishes a connection to a MySQL database and creates necessary tables.

UserRegistrationForm.java: Creates a user registration form with input fields and validation.

LoginFrame.java: Creates a login form for user authentication.

UserDashboard.java: Implements the user dashboard with product categories, cart functionality, checkout, and compost calculator.

CompostCalculatorFrame.java: Creates a frame for compost calculation based on user inputs.

BillingDashboard.java: Creates a billing dashboard for cart summary and payment options.

PaymentOptionsFrame.java: Creates a frame for payment options with validation and order details processing.

User Classes and Characteristics:

- Regular Users: Individuals interested in eco-friendly products, registration, and purchasing.
- Administrators: Manage database setup, user registration, and system functionalities.

Operating Environment:

The application is designed to run on Java-supported platforms (Windows, macOS, Linux) with Java Runtime Environment (JRE) installed. It requires access to a MySQL database server for data storage and retrieval.

Design and Implementation Constraints:

- The application is implemented using Java Swing for the graphical user interface (GUI).
- It relies on JDBC (Java Database Connectivity) for database operations, specifically with MySQL.
- System performance may be impacted by database response times and network connectivity.
- The GUI design is constrained by the capabilities and limitations of Java Swing components.

Assumptions and Dependencies:

- Assumption 1: Users have access to a compatible operating system and Java Runtime Environment (JRE).
- Assumption 2: The MySQL database server is accessible and properly configured.
- Dependency 1: Successful compilation and execution of Java files for database setup and user registration.
- Dependency 2: Internet connectivity for accessing external webpages and videos.

Glossary

- ❖ GUI: Graphical User Interface
- ❖ SRS: Software Requirements Specification
- ❖ FR: Functional Requirement
- ❖ NFR: Non-Functional Requirement

Conclusion

The Soil Enthusiast application aims to deliver a seamless user experience with features such as user registration, product management, compost calculation, and payment processing. By adhering to the outlined requirements and constraints, the application seeks to fulfill the needs of environmentally conscious users while ensuring usability, performance, reliability, and security.