



UE21CS352B - Object Oriented Analysis & Design using Java

Mini Project Report

Online Grocery Store Management System

Submitted by:

Neeharika Anand	PES1UG21CS372
Babitha M	PES1UG21CS905
Naveen George Jacob	PES1UG21CS366
R Naveen Kumar	PES1UG21CS367

6th Semester F Section

Prof. Bhargavi Mokashi

January - May 2024

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY**

(Established under Karnataka Act No. 16 of 2013)
100ft Ring Road, Bengaluru – 560 085, Karnataka, India

Problem Statement: Nature's Basket - Online Grocery Store Management System

Nature's Basket is a comprehensive Online Grocery Store Management System which facilitates efficient operations and enhanced user experiences. The system empowers customers to browse, select, and purchase groceries online seamlessly while enabling administrators to manage products, orders, and inventory effectively. Specifically, administrators will have the capability to oversee product inventory, update stock levels, add/remove products, and manage order fulfillment.

Key Features:

1. User-Friendly Interface:

- Developed an intuitive and responsive web interface for customers to browse products, add items to cart, and place orders effortlessly.
- Consists of an easy-to-navigate dashboard for administrators to manage products, orders, and inventory efficiently.

2. Product Management:

- Enables administrators to add new products, update existing product details (e.g., name, description, price), and remove products as necessary.
- Implemented categorization and filtering options to assist customers in locating products based on categories, brands, or keywords.

3. Cart and Order Management:

- Allows customers to add products to their cart, review their cart contents, and proceed to checkout to place orders securely.
- Provides mailing functionality for customers to monitor the status of their orders, specifically when the order is placed and shipped.
- Enables administrators to view, process, and manage orders efficiently, including order confirmation, packaging, and shipping.

4. Admin Inventory Management:

- Empowers administrators with the ability to manage product inventory effectively, including updating stock levels, tracking product availability, and receiving notifications for low stock items.
- Consists of inventory management features such as adding new stock, adjusting stock levels, and removing obsolete products from inventory.

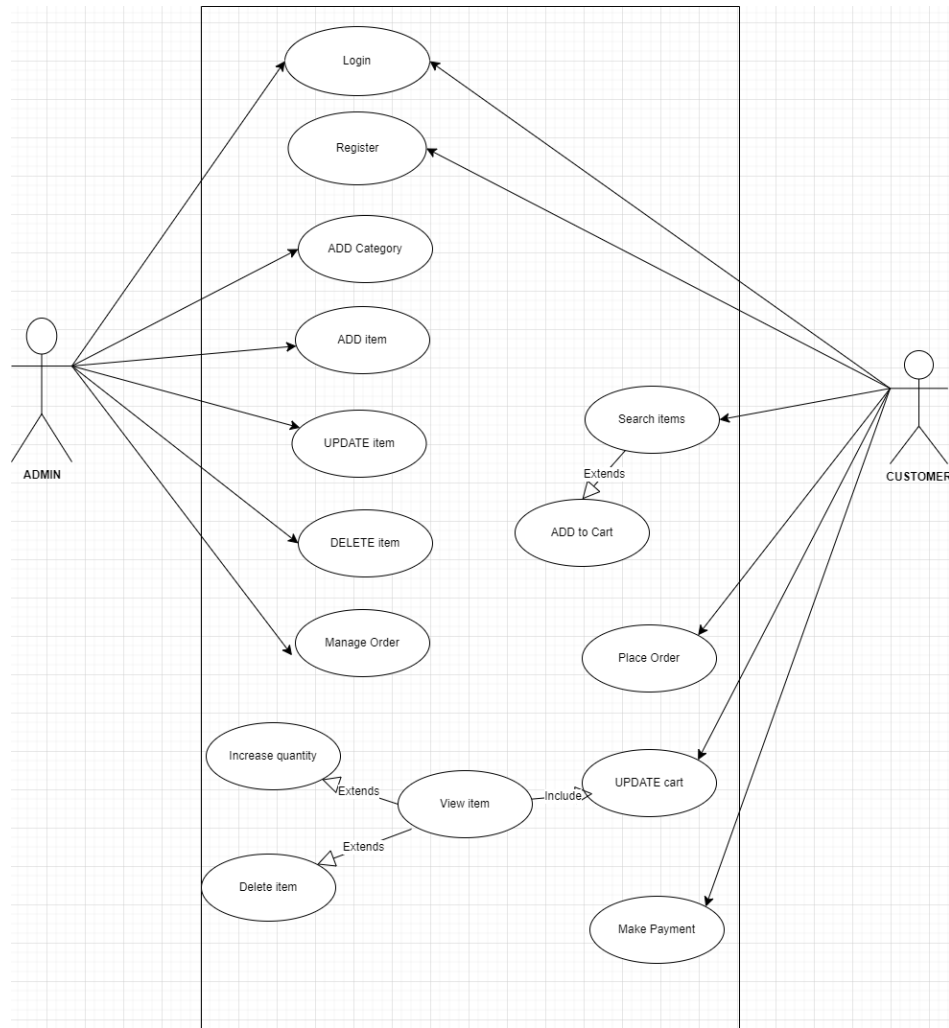
5. User Authentication and Security:

- Implements separate and secure authentication mechanisms for users and administrators, ensuring that access to sensitive functionalities is appropriately restricted.
- Enables customers to create accounts, manage profiles, and securely store payment details for convenient future purchases.
- Provides administrators with exclusive access to administrative functionalities through a separate login process, maintaining clear separation between user and admin privileges.

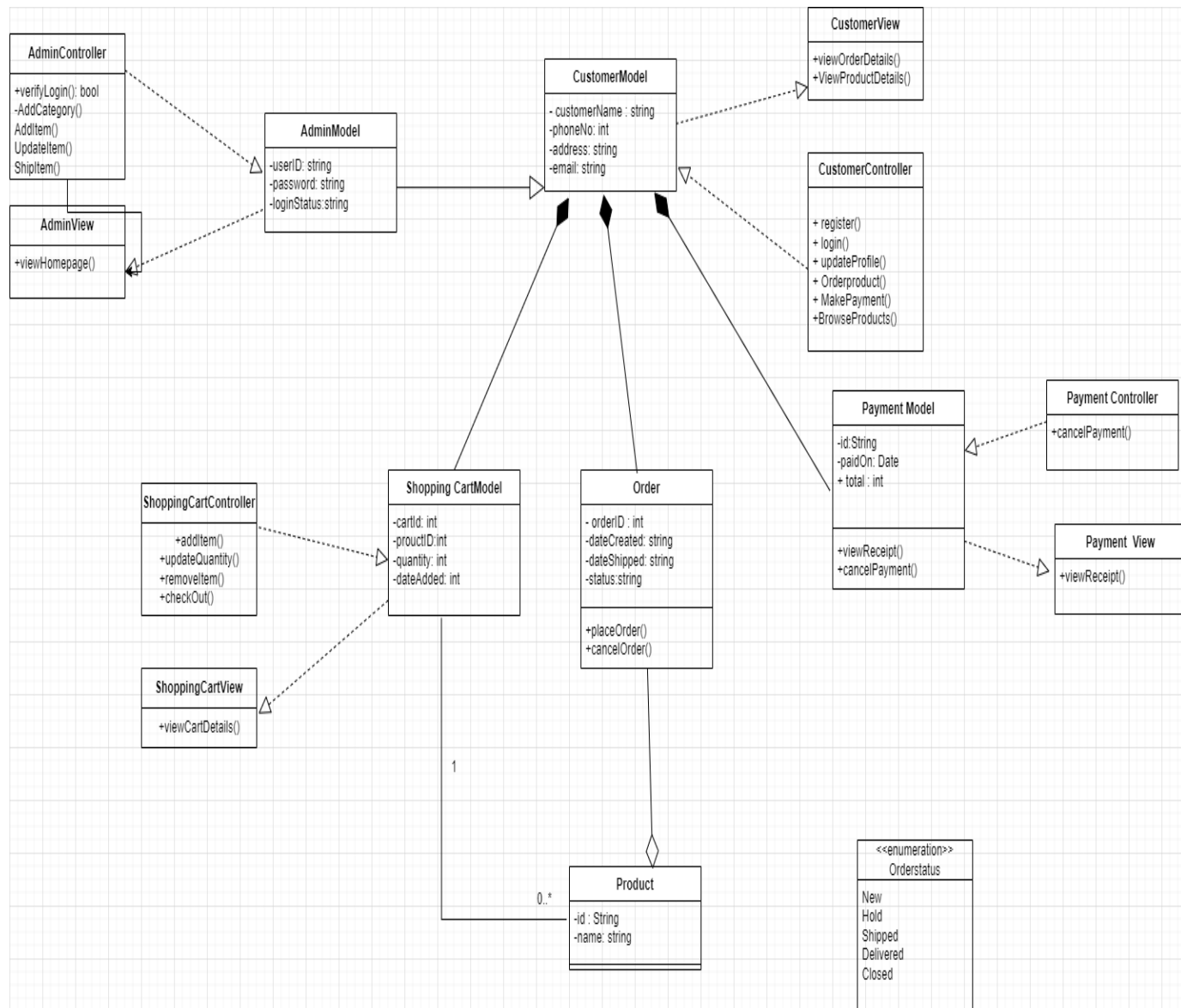
Models:

Use Case Diagram:

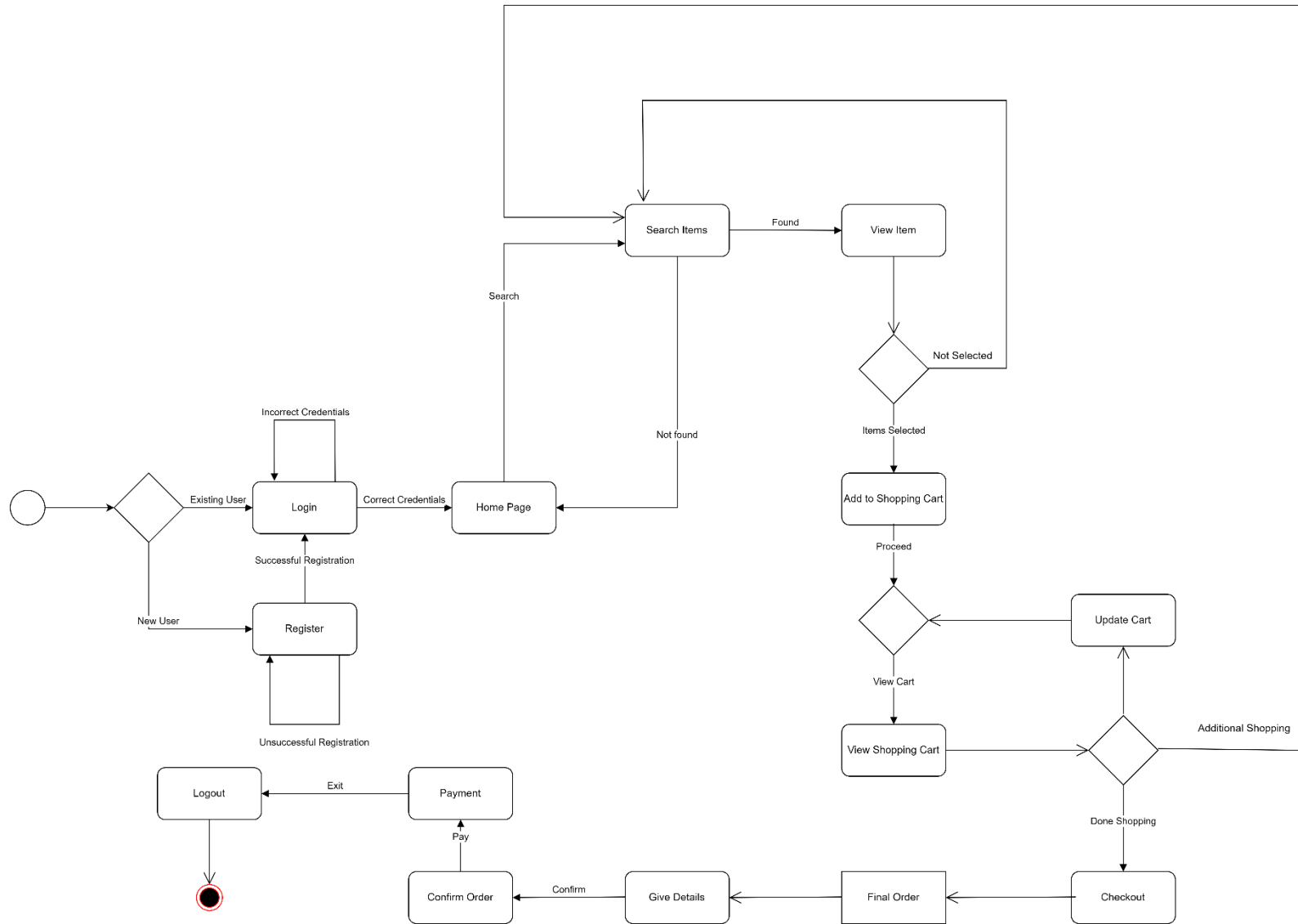
ONLINE GROCERY STORE MANAGEMENT SYSTEM USECASE



Class Diagram:

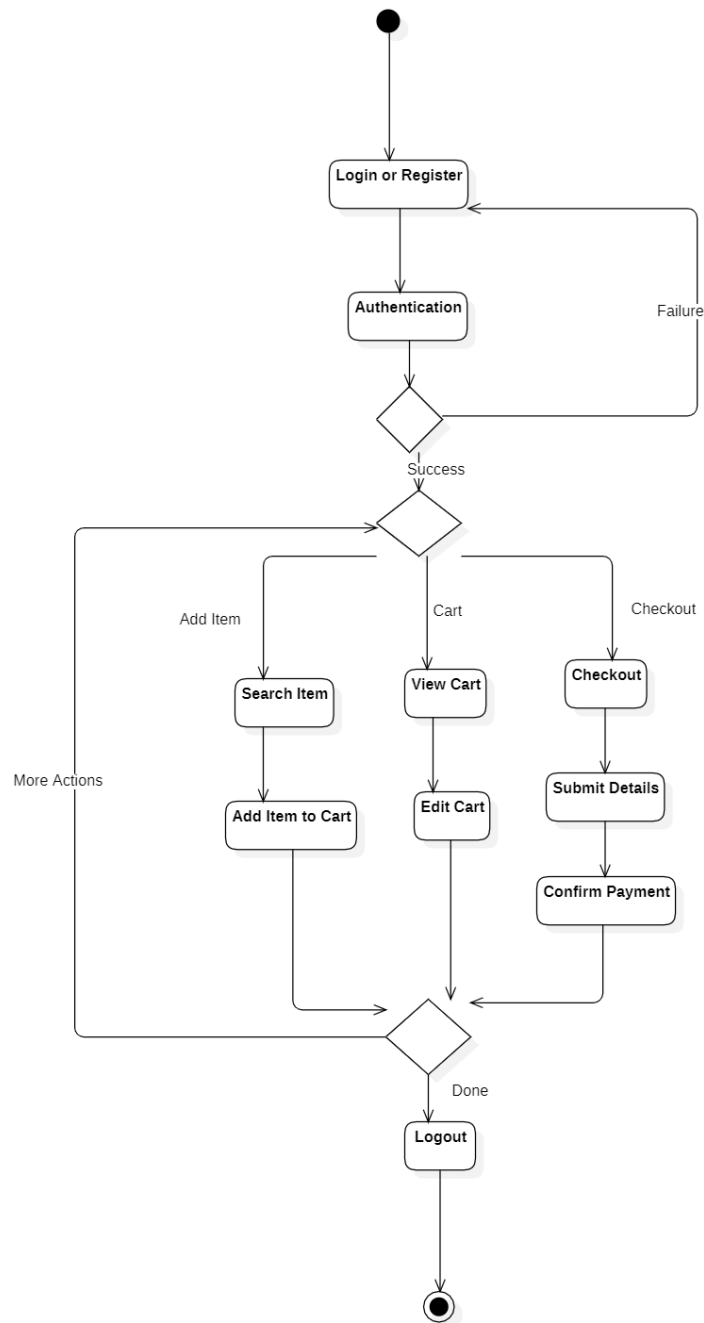


State Diagram:

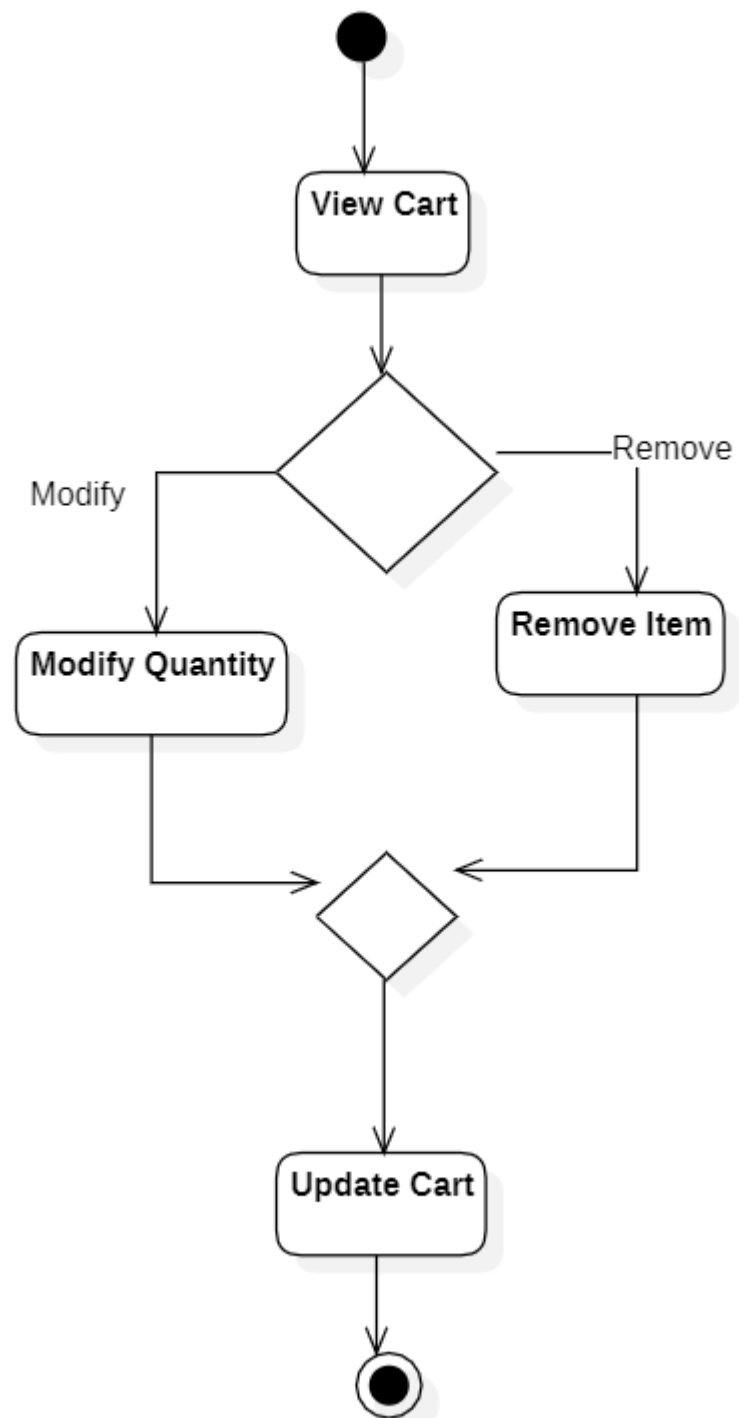


Activity Diagrams:

1. Major Usecase



2. Minor Use case



Architecture Patterns, Design Principles, and Design Patterns:

Architecture Patterns

1. Facade Design Pattern for Login Functionality

The Facade pattern is a **structural design pattern** that provides a simplified interface to a complex system of classes, hiding its internal implementation details. It promotes loose coupling between clients and subsystems, improving maintainability and ease of use.

In our online grocery store management system, we've implemented the Facade design pattern to simplify the login process and encapsulate the complexities involved in authenticating users. The Facade acts as a unified interface to a set of interfaces in the subsystem, providing a higher-level interface that makes it easier to use.

The intent of the Facade pattern in our system is to provide a simple and unified interface to handle the various types of logins (user login and admin login), abstracting away the details of the authentication process and providing a seamless experience for users.

Implementation:

Our Facade class, named LoginFacade, exposes a set of high-level methods corresponding to the different types of logins supported by the system:

userLogin: Handles user authentication.

adminLogin: Handles admin authentication.

How Facade Helps:

- Simplified Interface: Clients interact with a single, easy-to-use interface, hiding the complexities of the underlying authentication subsystem.
- Encapsulation: The Facade encapsulates the login process, promoting information hiding and reducing dependency on internal subsystem components.

- Flexibility: Allows for changes in the authentication mechanism without affecting client code, as long as the Facade interface remains unchanged.

2. Singleton Design Pattern for Database Connection

In our Online Grocery Store Management System, we have implemented the **Singleton** design pattern to manage the database connection effectively. The Singleton pattern ensures that **only one instance** of the **DBUtil** class exists throughout the lifetime of the application, providing a global point of access to the database connection.

Importance of Singleton Pattern:

- Resource Optimization: By restricting the instantiation of the DbUtil class to a single instance, we optimize resource usage, especially in scenarios where establishing multiple database connections simultaneously could lead to resource exhaustion.
- Global Access: The Singleton pattern provides a centralized and consistent way to access the database connection across different components of the system. This promotes code consistency and facilitates communication between various modules of the system.
- Thread Safety: Our implementation of the Singleton pattern incorporates thread-safe initialization using synchronization, ensuring that the Singleton instance is created only once and is safely accessible from multiple threads.

Drawbacks of Multiple Connections:

- Resource Consumption: Without proper management, creating multiple database connections can lead to excessive resource consumption, such as memory and CPU usage. This can degrade the performance of the system and affect the overall system stability.
- Connection Pooling Overhead: Establishing and tearing down database connections repeatedly can incur additional overhead, particularly in distributed systems or environments with high concurrency. This overhead can impact the responsiveness and scalability of the system.

How Singleton Helps:

- Resource Conservation: By restricting the instantiation of the DBUtil class to a single instance, the Singleton pattern mitigates the risk of resource exhaustion associated with creating multiple database connections. It ensures that resources are utilized efficiently, optimizing the performance of the system.
- Connection Pooling: The Singleton instance of DbUtil encapsulates connection pooling logic, enabling the reuse of existing connections and minimizing the

overhead of establishing new connections. This enhances the scalability and responsiveness of the system, especially in scenarios with a high volume of database operations.

3. Builder Design Pattern for Product Creation

In our Online Grocery Store Management System, we have employed the Builder design pattern to facilitate the creation of complex Product objects with varying attributes. The Builder pattern separates the construction of a complex object from its representation, allowing for flexible and fluent object creation.

Importance of Builder Pattern:

- Simplified Object Creation: The Builder pattern simplifies the process of creating Product objects by providing a step-by-step approach to set individual attributes. This promotes readability and maintainability, especially when dealing with objects with many optional parameters.
- Flexible Configuration: With the Builder pattern, clients can configure Product objects with different combinations of attributes without the need for multiple constructors or setter methods. This flexibility accommodates diverse use cases and enhances the reusability of the Product class.
- Encapsulation of Construction Logic: The Builder encapsulates the construction logic of Product objects within the Builder class, abstracting away the complexity from the client code. This promotes encapsulation and reduces code duplication, resulting in cleaner and more modular codebase.

Drawbacks of Direct Object Instantiation:

- Constructor Overloading: In the absence of the Builder pattern, constructors with multiple parameters may lead to constructor overloading, complicating the class interface and making it challenging to maintain.
- Inflexible Configuration: Direct object instantiation with setters may result in an inflexible configuration process, especially when dealing with immutable objects or objects with complex initialization requirements.

How Builder Helps:

- Clear and Fluent API: The Builder pattern provides a clear and fluent API for constructing Product objects, allowing clients to specify attributes in a natural and readable manner.
- Complex Object Construction: With the Builder pattern, clients can construct Product objects with complex initialization requirements, such as default values, optional parameters, and conditional logic, in a systematic and organized manner.
- Immutable Product Objects: The Builder pattern facilitates the creation of immutable Product objects by enforcing parameter validation and ensuring that the object state remains consistent throughout the construction process.

The implementation of this pattern consists of the following components:

- Product Class: The Product class represents the main entity in our system, encapsulating information about grocery products such as name, price, category, and quantity. The Product class has a private constructor to enforce immutability and a public static inner class named ProductBuilder for constructing Product objects.
- ProductBuilder Class: The ProductBuilder class serves as the builder for constructing Product objects step by step. It provides setter methods for configuring individual attributes of the Product, such as name, price, category, and quantity. Each setter method returns the builder instance itself, allowing for method chaining to set multiple attributes fluently.
- build() Method: The build() method in the ProductBuilder class finalizes the construction process by instantiating a new Product object with the configured attributes. This method creates a new instance of the Product class using the values set by the client through the setter methods.

4. Command Design Pattern for Order Processing

In our Online Grocery Store Management System, we have integrated the Command design pattern to encapsulate order processing actions, specifically focusing on the payment process. This pattern allows us to separate the request for an action (command) from its execution, providing flexibility and extensibility in handling order-related tasks.

Importance of Command Pattern:

- **Decoupling:** The Command pattern decouples the client (OrderController) from the implementation details of order processing actions, such as payment. This separation ensures that the client remains agnostic to the specifics of how orders are processed, promoting maintainability and modularity.
- **Encapsulation:** Each order processing action is encapsulated within a concrete command class (e.g., PaymentCommand). This encapsulation shields the client from the complexities of the payment process, improving code readability and reducing potential bugs caused by tightly coupled components.
- **Extensibility:** The Command pattern facilitates extensibility by allowing new order processing commands to be added seamlessly. Additional concrete command classes can be created without modifying existing client code, enabling the incorporation of new features and functionalities into the system without disrupting existing functionality.

Implementation of Command:

We used the Command Pattern to encapsulate order processing actions, specifically the payment process, using the components:

- Command Interface (OrderCommand):

This interface defines the contract for executing order processing commands. OrderCommand is the abstract class representing the command interface, with the execute() method signature.

- Concrete Command (PaymentCommand):

The PaymentCommand class extends OrderCommand and represents a specific order processing action . It encapsulates the details of the payment process, including the username and paid amount, and implements the execute() method.

- OrderController:

The OrderController acts as the client, which creates and invokes the command objects based on user actions.

The intent of the command pattern is to encapsulate a request in an object , allow the parameterization of clients with different requests,allow saving the requests in a queue.

5. Model – View – Controller Pattern (MVC)

The Model-View-Controller (MVC) architecture pattern is widely used in software development to separate the concerns of an application into three interconnected components: the Model, View, and Controller. Each component has a specific role and responsibility within the system.

Components of MVC:

1. Model:

- The Model represents the application's data and business logic. It encapsulates the data access, manipulation, and validation operations, independent of the user interface or presentation logic.
- In our Grocery Management System, the Model includes entities such as User, Cart , Order,Product , Transaction, Demand along with corresponding services for data retrieval, manipulation, and persistence.

2. View:

- The View represents the presentation layer of the application, responsible for displaying data to the user and handling user input. It presents the information retrieved from the Model in a user-friendly format and communicates user actions back to the Controller.
- In our system , the View comprises the user interface components such as web pages, forms, and dashboards, which have been designed using **JSP**, that enable interaction with the users and provide access to the functionalities offered by the system.

3. Controller:

- The Controller acts as an intermediary between the Model and the View, handling user input, processing requests, and updating the Model accordingly. It

orchestrates the flow of data and controls the application's behavior based on user interactions.

- In our system , the Controller contains the logic to interpret user requests, invoke appropriate services from the Model layer, and render the corresponding views to the user based on the requested actions.

Benefits of MVC:

- **Separation of Concerns:** MVC promotes a clear separation of concerns, allowing each component to focus on its specific responsibilities. This enhances code maintainability, reusability, and testability by minimizing dependencies and facilitating modular design.
- **Parallel Development:** The modular structure of MVC enables parallel development of different components by different teams or developers, promoting collaboration and accelerating the development process.
- **Flexibility and Scalability:** MVC facilitates flexibility and scalability by allowing modifications or extensions to one component without affecting the others. This enables the system to adapt to changing requirements and scale efficiently as the system evolves.

Architecture Principles

1. Single Responsibility Principle (SRP)

The Single Responsibility Principle (SRP) is one of the SOLID principles of object-oriented design, proposed by Robert C. Martin. SRP states that a class should have only one reason to change, meaning that it should have only one responsibility or job within the system.

Importance of SRP:

- **Enhanced Maintainability:** By adhering to SRP, classes become more focused and less complex. Each class is responsible for a single aspect of the system, making it easier to understand, modify, and maintain.
- **Improved Reusability:** Classes with well-defined responsibilities are more likely to be reusable in other parts of the system or in different projects. They

encapsulate specific functionalities, making them versatile and adaptable to various scenarios.

- **Facilitates Testing:** When each class has a single responsibility, it becomes simpler to write unit tests for individual components. Testing becomes more focused and comprehensive, leading to better test coverage and more robust code.

Implementation of SRP in the Online Grocery Store Management System:

In our Online Grocery Store Management System, we have adhered to the Single Responsibility Principle (SRP) to enhance maintainability, readability, and extensibility of our codebase. SRP dictates that a class should have only one reason to change, meaning it should have only one responsibility or purpose.

1. User Class:

- The User class is responsible for representing user entities within the system.
- Its responsibilities include managing user information such as username, email, password, and address.
- The User class adheres to SRP by solely focusing on managing user-related data and behavior. It does not contain logic unrelated to user management.

2. Product Class:

- The Product class encapsulates product entities in the system.
- Its responsibilities include storing product information such as name, price, category, and quantity.
- The Product class conforms to SRP by concentrating solely on product-related functionalities, such as retrieving product details, updating product information, and managing inventory.

3. Cart Class:

- The Cart class represents the shopping cart functionality in the system.
- Its responsibilities include managing the items added to the cart, calculating the total price, and facilitating the checkout process.
- The Cart class follows SRP by handling cart-related operations exclusively, ensuring separation of concerns and modularity.

4. Order Class:

- The Order class is responsible for representing orders placed by users.

- Its responsibilities include storing order details such as order ID, user ID, order items, total price, and order status.
- The Order class conforms to SRP by focusing solely on order-related functionalities, such as processing orders, updating order status, and generating order reports.

2. Interface Segregation Principle (ISP)

The Interface Segregation Principle (ISP) is one of the five SOLID principles of object-oriented design. It states that a client should not be forced to depend on interfaces it does not use. In other words, interfaces should be specific to the needs of the clients that use them, rather than being overly general and including methods that are not relevant to all clients.

Importance of Interface Segregation Principle (ISP):

- Reduced Dependency: By segregating interfaces based on client-specific functionalities, ISP reduces the dependency of clients on irrelevant methods. This promotes loose coupling between components and facilitates easier maintenance and evolution of the codebase.
- Improved Cohesion: ISP leads to interfaces that are more focused and cohesive, containing only the methods that are relevant to a specific client or group of clients. This improves the readability and understandability of the code, as well as making it easier to reason about and maintain.
- Enhanced Testability: Segregating interfaces based on client-specific requirements allows for more targeted and efficient testing. Each client can be tested independently, leading to more robust and reliable testing processes.
- Flexible Evolution: ISP makes the system more flexible and adaptable to change. When new clients or functionalities are introduced, it's easier to extend existing interfaces or create new ones tailored to their specific needs without impacting existing clients or interfaces.

Implementation of ISP in the Online Grocery Store Management System:

In our Online Grocery Store Management System, we have applied the Interface Segregation Principle (ISP) to ensure that client-specific interfaces are tailored to the

needs of the clients, thereby preventing clients from being forced to depend on interfaces they do not use.

1. UserService Interface:

- The UserService interface defines methods related to user management, such as createUser(), updateUser(), and deleteUser().
- By segregating user-specific operations into a UserService interface, we adhere to ISP, ensuring that clients requiring user management functionalities depend only on this interface.

2. ProductService Interface:

- The ProductService interface declares methods for product management, including addProduct(), updateProduct(), and removeProduct().
- By segregating product-related operations into a ProductService interface, we follow ISP, allowing clients needing product management capabilities to rely solely on this interface.

3. CartService Interface:

- The CartService interface specifies methods for cart management, such as addToCart(), removeFromCart(), and viewCart().
- Adhering to ISP, we segregate cart-specific functionalities into a CartService interface, enabling clients requiring cart management features to interact solely with this interface.

4. OrderService Interface:

- The OrderService interface encompasses methods related to order processing, including placeOrder(), cancelOrder(), and trackOrder().
- By segregating order-specific operations into an OrderService interface, we conform to ISP, ensuring that clients needing order management functionalities depend exclusively on this interface.

3. Open Closed Principle (OCP)

The OCP is one of the five SOLID principles which states that classes, modules, microservices, and other code units should be open for extension but closed for modification. We should be able to extend the existing code using OOP features like inheritance via subclasses and interfaces. While adding a new feature extend the code rather than modifying it, so that the risk of failure is minimized.

Importance of Open-Closed Principle(OCP):

- Flexibility: OCP enables greater flexibility in software design and architecture. Changes in requirements or business logic can be accommodated by adding new functionality through extension rather than modifying existing code.
- Reuse: By designing components to be open for extension, developers can create reusable building blocks that can be applied in different contexts without modification. This promotes code reuse and reduces duplication.
- Testability: OCP supports better testability by encouraging the creation of modular and loosely coupled components. Unit testing becomes easier as each component can be tested in isolation without relying on the implementation details of other components.

Implementation of OCP in the Online Grocery Store Management System:

In our Online Grocery Store Management System, we have implemented the Open-Closed Principle (OCP) to enhance the flexibility and scalability of our codebase so that the software entities are open for extension but closed for modification, meaning that they should allow for new functionality to be added through extension without requiring changes to existing code.

1 . AdminServiceImpl:

- The AdminServiceImpl class adheres to the Open-Closed Principle by implementing the UserService interface, which defines a contract for user management operations.
- The class is open for extension, as new functionality related to user management can be added by extending the UserService interface, while the existing functionality is closed for modification.

2 . UserServiceImpl:

- The UserServiceImpl class adheres to the Open-Closed Principle by implementing the UserService interface, which defines a contract for user management operations.
- Each method in the UserServiceImpl class encapsulates a specific aspect of user management, such as user registration, login authentication, retrieving user details, and obtaining user address. This encapsulation allows for the extension of behavior without modifying the existing codebase, adhering to the OCP.

3 . CartServiceImpl:

- The methods defined in the 'CartService' interface, such as addProductToCart, removeProductFromCart, updateProductToCart, getAllCartItems, getCartCount, and getCartItemCount, provide extension points for adding new functionality related to the cart management.
- Each method encapsulates a specific behavior, allowing for easy extension or modification without altering other parts of the codebase which adheres to the OCP.

4. **Liskov Substitution Principle (LSP)**

It states that objects of a superclass should be replaceable with objects of a subclass without affecting the correctness of the program. In other words, a derived class must be substitutable for its base class without altering the desirable properties of the program.

Importance of Liskov Substitution Principle (LSP):

- Maintainability: By adhering to the LSP, classes within a hierarchy become more interchangeable, making it easier to maintain and extend the system over time. Subclasses can be added or modified without affecting the existing behavior of the system.
- Flexibility: LSP promotes a flexible design where objects can be substituted seamlessly, allowing for easier reuse of code and components. This flexibility enables developers to build systems that are more modular, scalable, and extensible.

- Design Clarity: LSP encourages a clear and understandable class hierarchy by promoting the consistency of behavior across subclasses. This clarity enhances code readability and comprehensibility.

Implementation of LSP in the Online Grocery Store Management System:

In our Online Grocery Store Management System, we have also applied the Liskov Substitution Principle (LSP) to ensure the integrity and consistency of our codebase. Here the objects of a superclass are replaceable with objects of its subclasses without affecting the correctness of the program.

- The 'PaymentCommand' and 'OrderCommand' classes adhere to the Liskov Substitution Principle by ensuring that subclasses can be substituted for their superclasses.
- The PaymentCommand class extends the functionality provided by the OrderCommand class by introducing payment-specific behavior. However, it does so without modifying the existing behavior of the superclass.

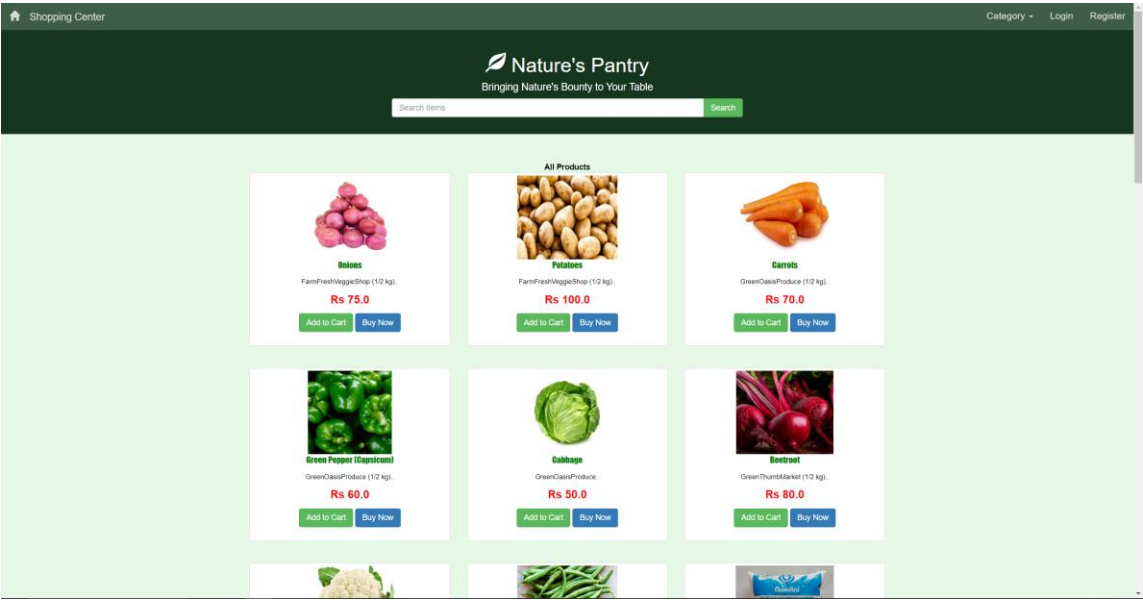
Github link to the Codebase:

<https://github.com/neecharika-anand/grocery-store>

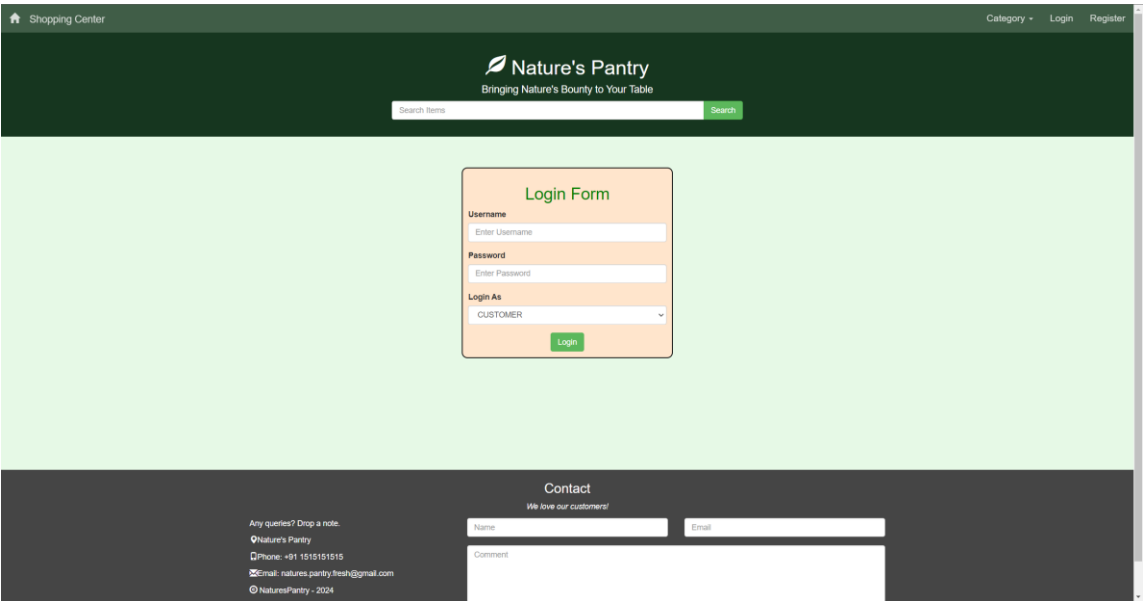
Screenshots

UI:

Landing Page:

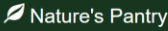


User Login:



Register User:


[Shopping Center](#)[Category](#)[Login](#)[Register](#)


Bringing Nature's Bounty to Your Table

Registration Form

Name	Email
<input type="text"/>	<input type="text"/>
Address	
<input type="text"/>	
Mobile	Pin Code
<input type="text"/>	<input type="text"/>
Password	Confirm Password
<input type="text"/>	<input type="text"/>
<input type="button" value="Reset"/>	<input type="button" value="Register"/>

Any queries? Drop a note.

 Nature's Pantry

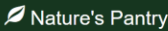
Contact

We love our customers!

<input type="text" value="Name"/>	<input type="text" value="Email"/>
<input type="text" value="Comment"/>	

Admin Login:


[Shopping Center](#)[Category](#)[Login](#)[Register](#)



Bringing Nature's Bounty to Your Table


Login Form


Username
<input type="text" value="Enter Username"/>
Password
<input type="text" value="Enter Password"/>
Login As
<input type="text" value="ADMIN"/>
<input type="button" value="Login"/>

Any queries? Drop a note.

 Nature's Pantry

 Phone: +91 1515151515

 Email: nature.pantry.fresh@gmail.com

 © Nature'sPantry - 2024

Contact

We love our customers!

<input type="text" value="Name"/>	<input type="text" value="Email"/>
<input type="text" value="Comment"/>	

User registering:

The screenshot shows the Nature's Pantry website with a registration form in the center. The form is titled "Registration Form" and contains fields for Name, Email, Address, Mobile, Pin Code, Password, and Confirm Password. The Name field is filled with "User", the Email field with "pes1202101219@pesu.pes.edu", the Address field with "PES University", the Mobile field with "123456789", and the Pin Code field with "560035". The Password and Confirm Password fields are filled with "****". There are "Reset" and "Register" buttons at the bottom of the form. The website header includes "Shopping Center", "Category", "Login", and "Register" links. The footer includes a "Contact" section with a "Name" and "Email" field, and a "Nature's Pantry" logo.

Registration Form

User Registered Successfully!

Sign up email:

Registration Successful External Inbox x



natures.pantry.fresh@gmail.com

to me ▾

Welcome to Nature's Pantry

Hi User,

Thanks for signing up with Nature's Pantry.

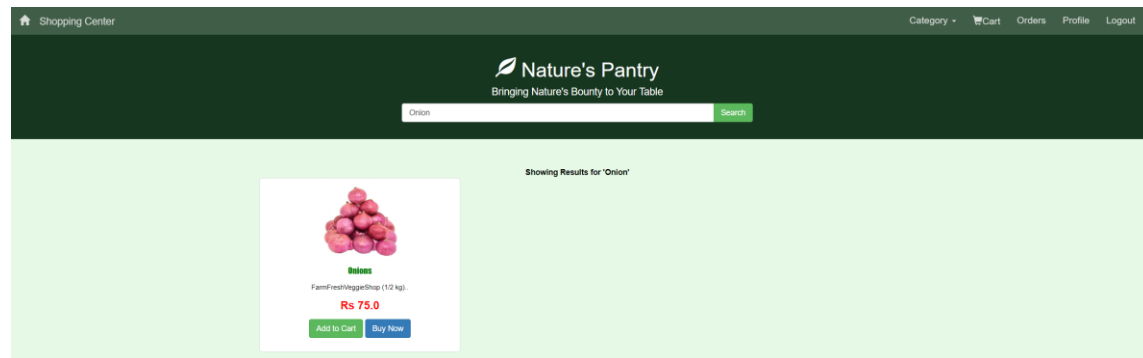
We are glad that you choose us. We invite you to check out our plethora of fresh and organic groceries.

There is an exciting sale going on, so please visit our site and explore our products!

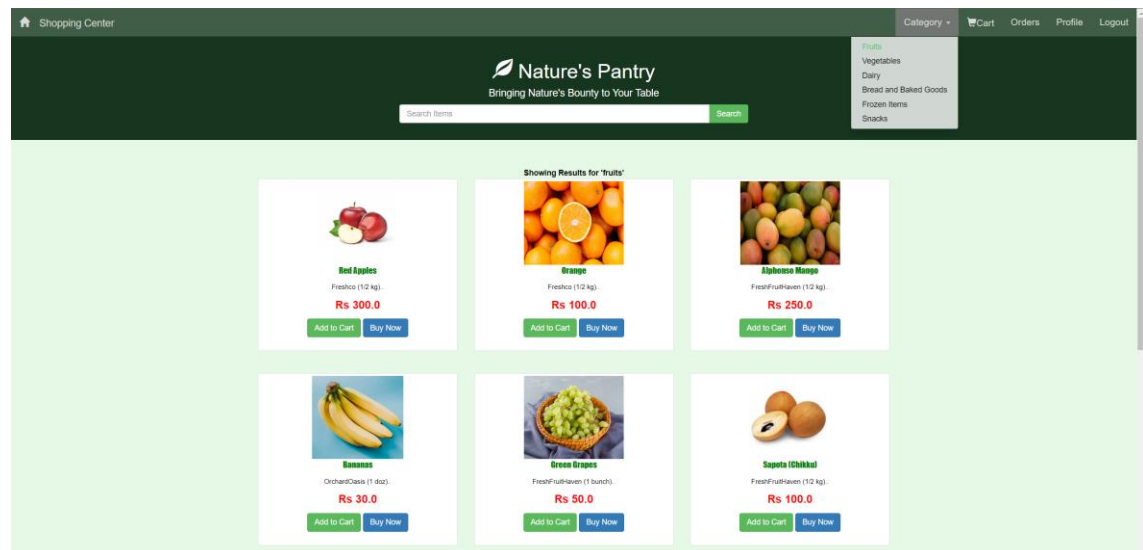
We deliver products to your house with no extra delivery charges

Have a good day!

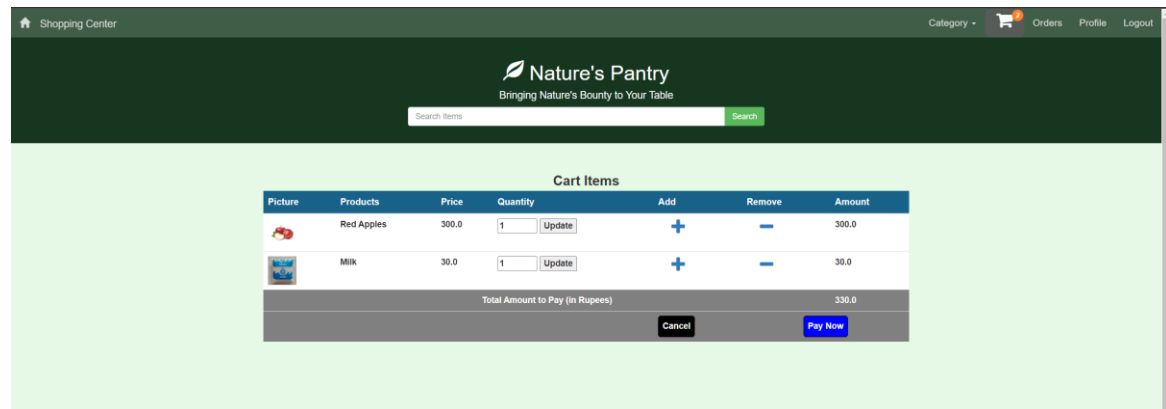
Searching for product by name:



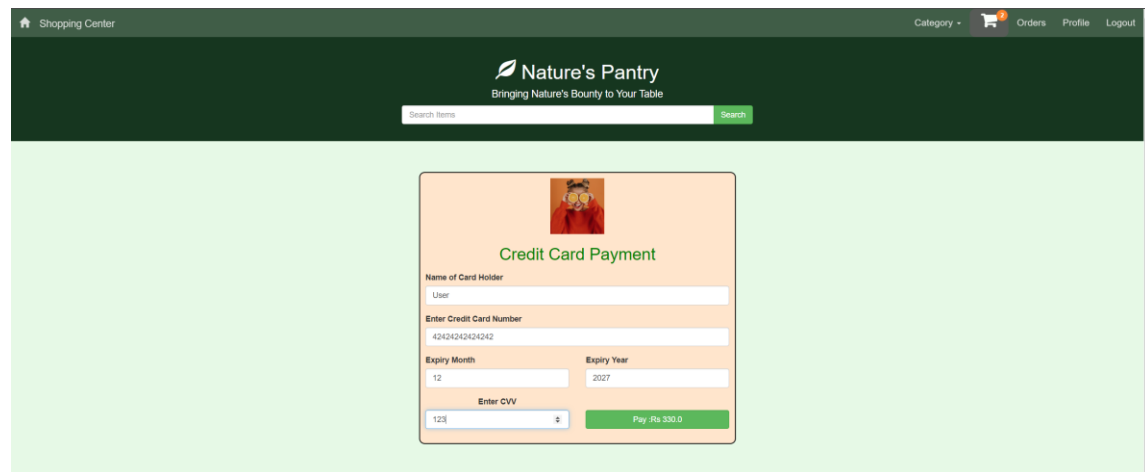
Filtering by Category:



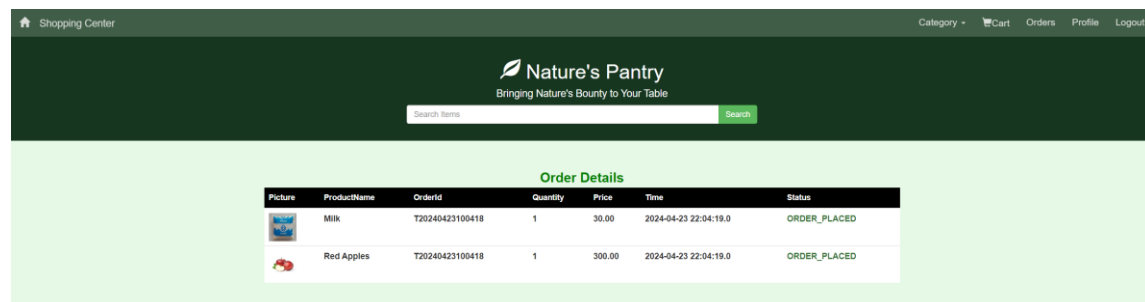
Cart View:



Checkout and Payment:



Successful Order with 'Order Placed' status:



Email Confirmation:

Order placed successfully at Nature's Pantry External Inbox x



natures.pantry.fresh@gmail.com

to me ▾

Hello User,

Thank you for shopping with Nature's Pantry!

Your order has been placed successfully and in process to be shipped.

Please Note that this is a demo project email and you have not made any real transaction with us till now!

Here are your Transaction Details:

Order Id: T20240423095309

Amount Paid: 330.0

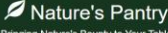
Thanks for shopping with us!

Please do visit us again!


Nature's Pantry

Admin View:


Shopping CenterProductsCategoryStockShippedOrdersUpdate ItemsLogout


Bringing Nature's Bounty to Your Table


Search




Onion (P20240405010251)
FarmFreshVeggieShop (12 kg)
Rs 75.0
Remove Product Update Product




Potatoes (P20240405010100)
FarmFreshVeggieShop (12 kg)
Rs 100.0
Remove Product Update Product




Carrots (P20240405010101)
GreenClassProduce (12 kg)
Rs 70.0
Remove Product Update Product



Green Pepper (Bajajicand) (P20240405010256)
GreenClassProduce (12 kg)
Rs 60.0
Remove Product Update Product











Cabbage (P20240405010129)
GreenClassProduce
Rs 50.0
Remove Product Update Product



Beetroot (P20240405010127)
GreenThumbMarket (12 kg)
Rs 80.0
Remove Product Update Product


Stock View:

Nature's Pantry Bringing Nature's Bounty to Your Table									
Stock Products									
Image	ProductId	Name	Type	Price	Sold Qty	Stock Qty	Actions		
	P20240405010251	Onions..	VEGETABLES	75.0	8	8	<button>Update</button>	<button>Remove</button>	
	P20240405010709	Potatoes..	VEGETABLES	100.0	4	26	<button>Update</button>	<button>Remove</button>	
	P20240405010919	Carrots..	VEGETABLES	70.0	0	30	<button>Update</button>	<button>Remove</button>	
	P20240405011256	Green Pepper (Capsicum)..	VEGETABLES	60.0	0	30	<button>Update</button>	<button>Remove</button>	
	P20240405011529	Cabbage..	VEGETABLES	50.0	0	30	<button>Update</button>	<button>Remove</button>	
	P20240405011927	Beetroot..	VEGETABLES	80.0	0	30	<button>Update</button>	<button>Remove</button>	
	P20240405012119	Cauliflower..	VEGETABLES	60.0	0	30	<button>Update</button>	<button>Remove</button>	
	P20240405012312	Beans..	VEGETABLES	100.0	0	30	<button>Update</button>	<button>Remove</button>	

Updating product:

Nature's Pantry Bringing Nature's Bounty to Your Table																	
Product Update Form																	
Product Name		Product Type															
<input type="text" value="Onions"/>		<input type="text" value="Vegetables"/>															
Product Description																	
<input type="text" value="FarmFreshVeggieShop (12 kg)"/>																	
Unit Price		Stock Quantity															
<input type="text" value="75.0"/>		<input type="text" value="30"/>															
<button>Cancel</button>		<button>Update Product</button>															

Updated Stock View:

Nature's Pantry Bringing Nature's Bounty to Your Table									
Stock Products									
Image	ProductId	Name	Type	Price	Sold Qty	Stock Qty	Actions		
	P20240405010251	Onions..	VEGETABLES	75.0	8	20	<button>Update</button>	<button>Remove</button>	

Adding Product:

Shopping Center

ProductsCategory - StockShippedOrdersUpdate Items - Logout

Nature's Pantry

Bringing Nature's Bounty to Your Table

Search Items

Search

Product Addition Form

Product Name

Fresh Cream

Product Type

Dairy

Product Description

Amul (0.5L)

Unit Price

70

Stock Quantity

30

Product Image

Choose Filefreshcream.jpg

Reset

Add Product

Added Successfully:

Shopping Center

ProductsCategory - StockShippedOrdersUpdate Items - Logout

Nature's Pantry

Bringing Nature's Bounty to Your Table

Search Items

Search

Showing Results for 'Fresh Cream'

Fresh Cream (P202404231000001)

Amul (0.5L)

Rs 70.0

Remove Product

Update Product

Deleting Product:

Product Deletion Form

Product Removed Successfully!

Shopping Center

ProductsCategory - StockShippedOrdersUpdate Items - Logout

Nature's Pantry

Bringing Nature's Bounty to Your Table

Search Items

Search

No Items found for the search 'Fresh Cream'

Viewing Orders:

Shopping Center

ProductsCategoryStockShippedOrdersUpdate ItemsLogout

Nature's Pantry

Bringing Nature's Bounty to Your Table

Search Items

Search

UnShipped Orders

TransactionId	ProductId	User Email Id	Address	Quantity	Status	Action
T20240423100418	P20240405012700	pes1202101219@pesu.pes.edu	PES University	1	READY_TO_SHIP	SHIP NOW
T20240423100418	P20240405013834	pes1202101219@pesu.pes.edu	PES University	1	READY_TO_SHIP	SHIP NOW

Shipping Orders:

Shopping Center

ProductsCategoryStockShippedOrdersUpdate ItemsLogout

Nature's Pantry

Bringing Nature's Bounty to Your Table

Search Items

Search

Order Has been shipped successfully

Shipped Orders

TransactionId	ProductId	Username	Address	Quantity	Amount	Status
T20240423100418	P20240405012700	pes1202101219@pesu.pes.edu	PES University	1	Rs. 30.0	SHIPPED
T20240423100418	P20240405013834	pes1202101219@pesu.pes.edu	PES University	1	Rs. 300.0	SHIPPED

Change in status:

Shopping Center

CategoryCartOrdersProfileLogout



Nature's Pantry

Bringing Nature's Bounty to Your Table

Search Items

Search


Order Details

Picture	Product Name	OrderId	Quantity	Price	Time	Status
	Milk	T20240423100418	1	30.00	2024-04-23 22:04:19.0	ORDER_SHIPPED
	Red Apples	T20240423100418	1	300.00	2024-04-23 22:04:19.0	ORDER_SHIPPED

Shipped Items confirmation email:

Hurray!! Your Order has been shipped from Nature's Pantry

ExternalInbox x



natures.pantry.fresh@gmail.com

to me

Hey User,

Thank you for shopping with Nature's Pantry!

Your order has been shipped successfully and on the way to be delivered.

Please Note that this is a demo project email and you have not made any real transaction with us till now!

Here are your Transaction Details:


Order Id: T20240423100418

Amount Paid: 30.0

Thanks for shopping with us!

Please do visit us again!

Nature's Pantry



natures.pantry.fresh@gmail.com

to me

Hey User,

Thank you for shopping with Nature's Pantry!

Your order has been shipped successfully and on the way to be delivered.

Please Note that this is a demo project email and you have not made any real transaction with us till now!

Here are your Transaction Details:

Order Id: T20240423100418

Amount Paid: 300.0

Individual contributions of the team members:

Name	Module worked on
Neeharika Anand	Products module
Babitha M.	Orders module
Naveen George Jacob	User module
R Naveen Kumar	Cart module