AgroTech
Software Design Specification

Divyanshu Madhav(IIT2022267)
Sandarbh Kansal(IIB2022007)
Kushal Singh(IIB2022022)
Dhruv Gupta(IIB2022025)

April 3, 2024

Table of Contents

Contents

1 Introduction

1.1 Purpose of this document

The purpose of this document is to provide a comprehensive overview of the software design for our e-commerce application. It serves as a guide for the development team, detailing how the software should be built and providing documentation for software development processes.

1.2 Scope of the development project

The scope of the development project includes the design and implementation of an e-commerce application that purchases high-quality agricultural products directly from farmers and produces various products such as flour, spices, and tea leaves. The application will focus on sourcing products from the best locations and providing consumers with access to premium-quality goods.

1.3 Definitions, acronyms, and abbreviations

User Interface (UI): The visual elements and interactive components through which users interact with the software. **SDS**: Software Design Specification, a document detailing the design of the software system. **IEEE**: Institute of Electrical and Electronics Engineers, an organization setting standards for software engineering. **CRS**: Campus Recruitment System, an application for job/internship applications by students. **Mieten**: Name of a CRS, an android application. **References**: External sources referenced for information and guidance.

1.4 References

1. R. S. Pressman, *Software Engineering: A Practitioner's Approach*, 5th Ed, McGraw-Hill, 2001.
2. IEEE SDS template.

1.5 Overview of document

This Software Design Specification (SDS) is structured into several sections, each providing detailed information about different aspects of the software design. These sections include:

1. Introduction: Overview of the document, purpose, scope, and definitions.
2. Conceptual Architecture/Architecture Diagram: Overview of components, modules, structure, relationships, and user interface issues.
3. Logical Architecture: Description of logical architecture and components.
4. Execution Architecture: Runtime environment, processes, and deployment view.

5. Design Decisions and Trade-offs: Decisions taken during design and reasoning behind them.
6. Pseudocode for components: Description of pseudocode for various components.
7. Appendices: Additional subsidiary matter, if any.

Architectural Diagram 1

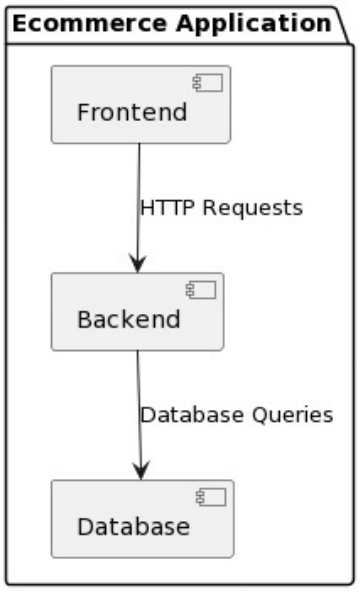


Figure 1: Architectural Diagram 1

Architectural Diagram 2

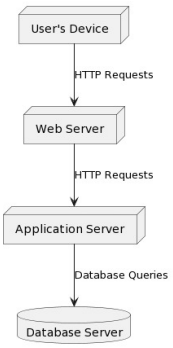


Figure 2: Architectural Diagram 2

Overview of Modules

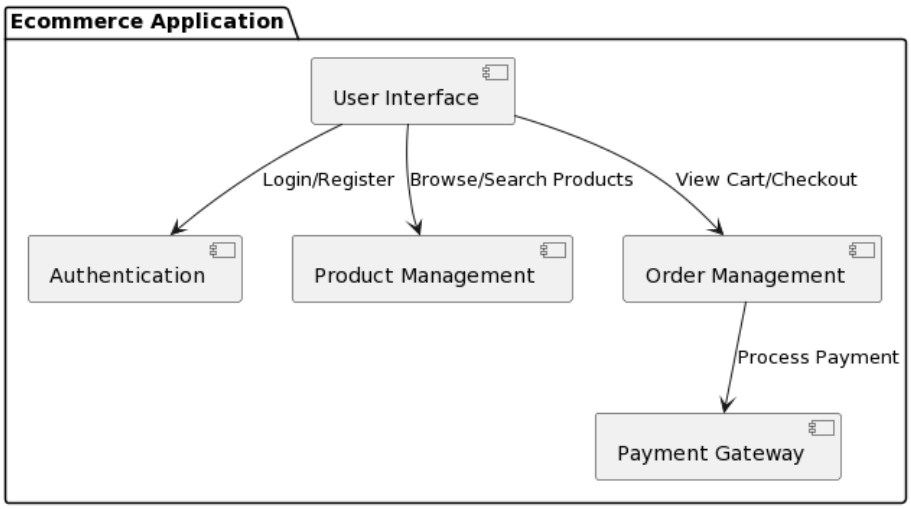


Figure 3: Overview of Modules

Structure and Relationships

Admin Side

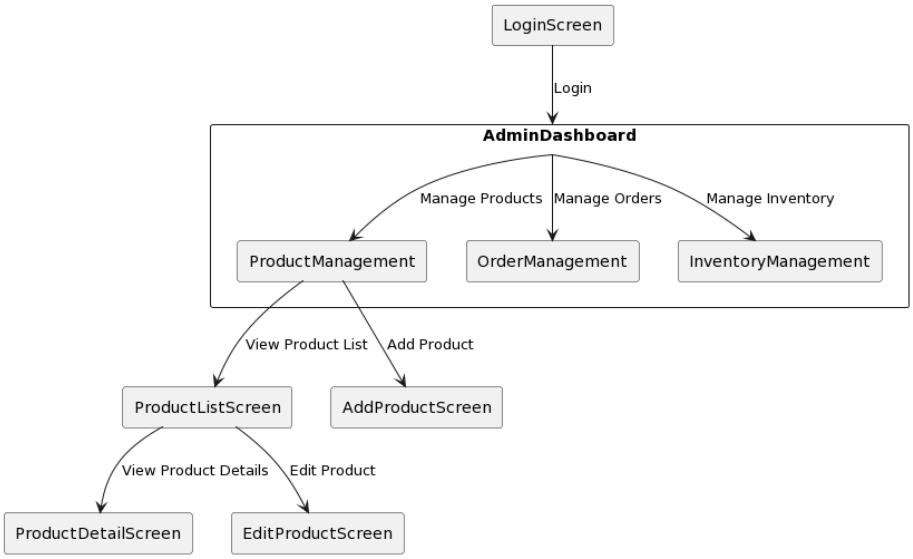


Figure 4: Admin Side

Customer Side

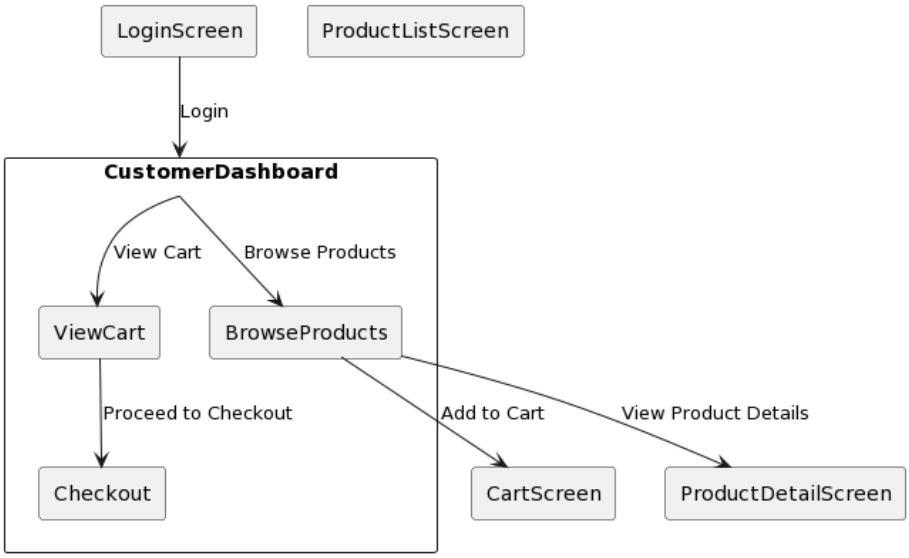


Figure 5: Customer Side

User Interface Issues

In the conceptual architecture of our ecommerce application, several user interface issues need to be addressed. Here are some of the key concerns:

Responsive Design

The user interface should be designed to be responsive, ensuring optimal viewing and interaction experience across various devices, including desktops, tablets, and smartphones.

Intuitive Navigation

Navigation within the application should be intuitive, allowing users, both admins and customers, to easily find the desired functionalities and information.

Consistent Layout

Maintaining a consistent layout throughout the application is essential for providing a cohesive user experience. Elements such as headers, footers, menus, and buttons should have a uniform appearance and placement across different screens.

Accessibility

The user interface should be accessible to all users, including those with disabilities. This includes providing alternative text for images, ensuring sufficient color contrast, and supporting keyboard navigation.

Feedback Mechanisms

Incorporating feedback mechanisms, such as error messages, confirmation dialogs, and progress indicators, helps users understand the outcome of their actions and provides a smoother interaction experience.

Performance Optimization

Efforts should be made to optimize the performance of the user interface, ensuring fast loading times and smooth transitions between screens. This involves minimizing unnecessary animations, reducing server requests, and implementing caching strategies.

Security Measures

Security features, such as HTTPS encryption, secure authentication mechanisms, and protection against cross-site scripting (XSS) and SQL injection at-

tacks, should be implemented to safeguard user data and prevent unauthorized access.

Localization and Internationalization

The user interface should support localization and internationalization, allowing users from different regions to interact with the application in their preferred language and format.

These user interface issues should be carefully considered and addressed during the design and development phases of the ecommerce application to ensure a positive user experience.

3. Logical Architecture (Class Diagram, Sequence Diagram, State Diagram)

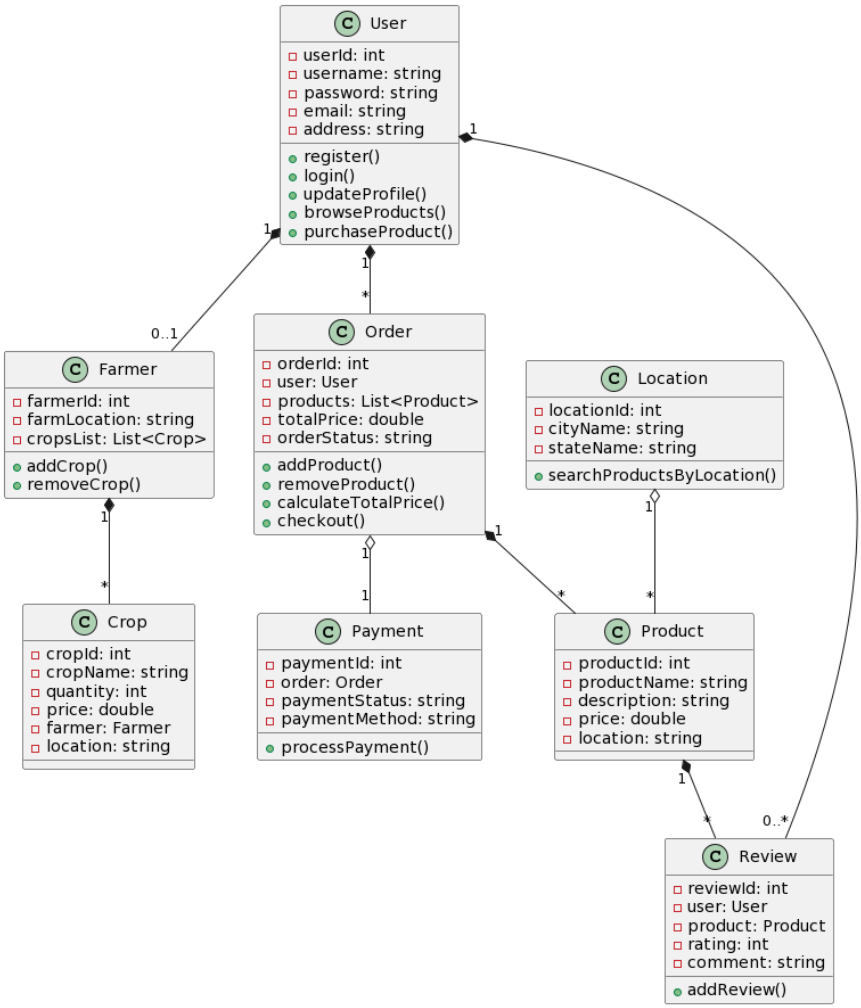


Figure 6: Class Diagram

Sequence Diagrams:

Sequence Diagram: User Registration

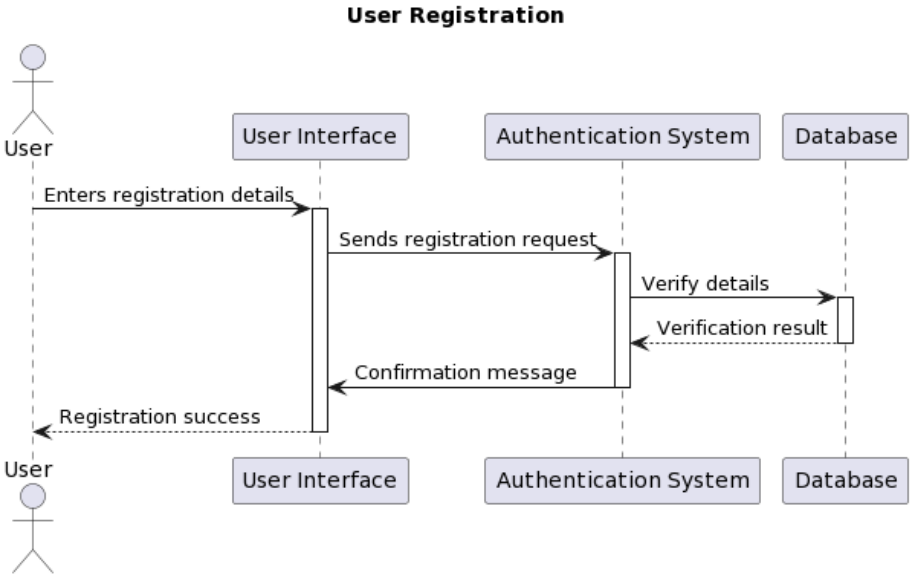


Figure 7: Sequence Diagram for User Registration

Sequence Diagram: User login

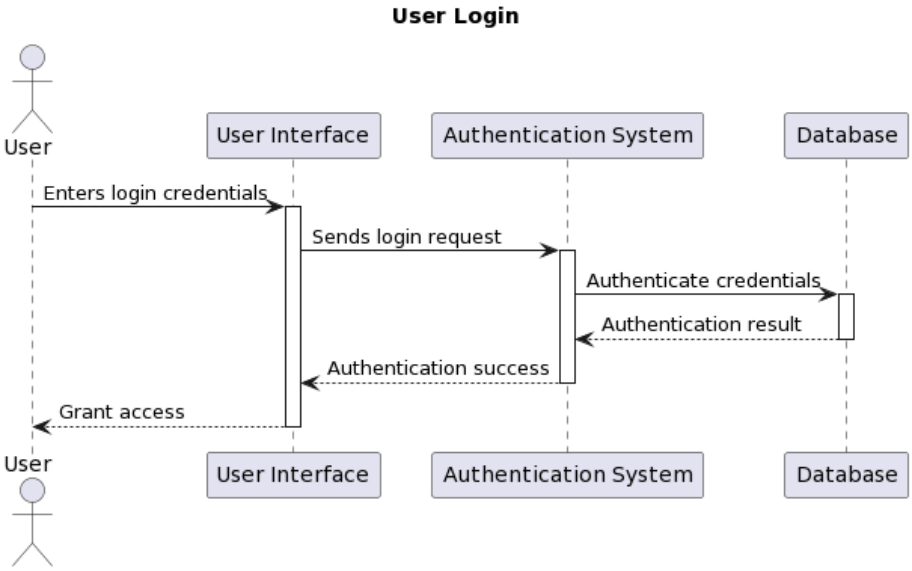


Figure 8: Sequence Diagram for User login

Sequence Diagram: Product search and purchase

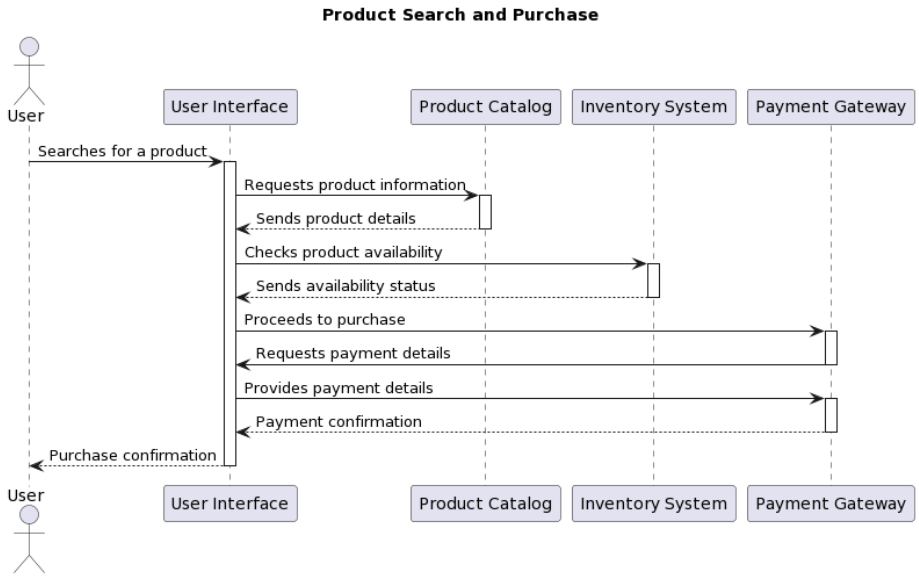


Figure 9: Sequence Diagram for Product search and purchase

Sequence Diagram: Order confirm

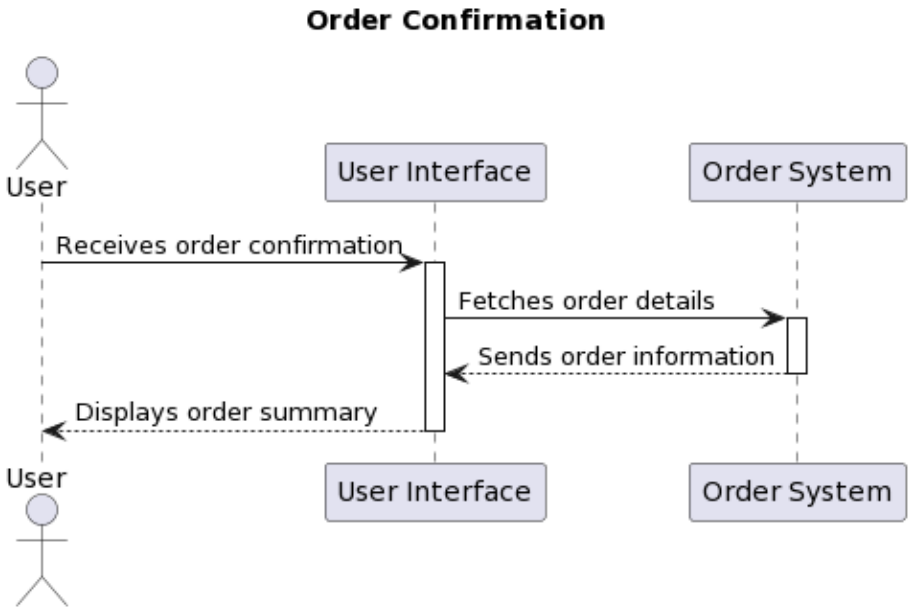


Figure 10: Sequence Diagram for Order confirm

Sequence Diagram: Add product to cart

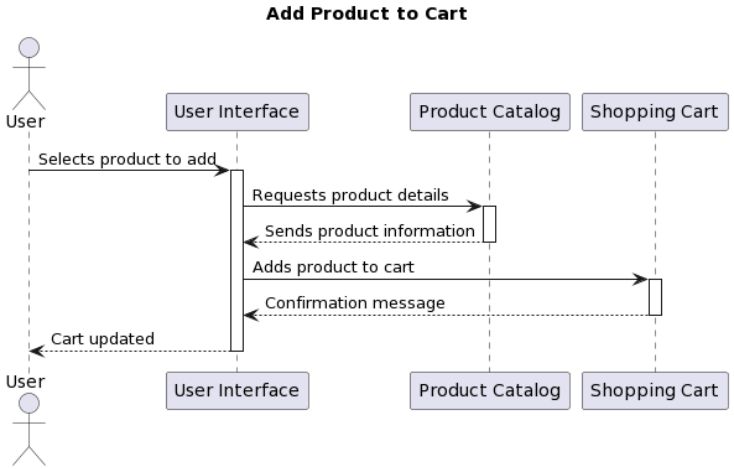


Figure 11: Sequence Diagram for Add product to cart

Sequence Diagram: Check out process

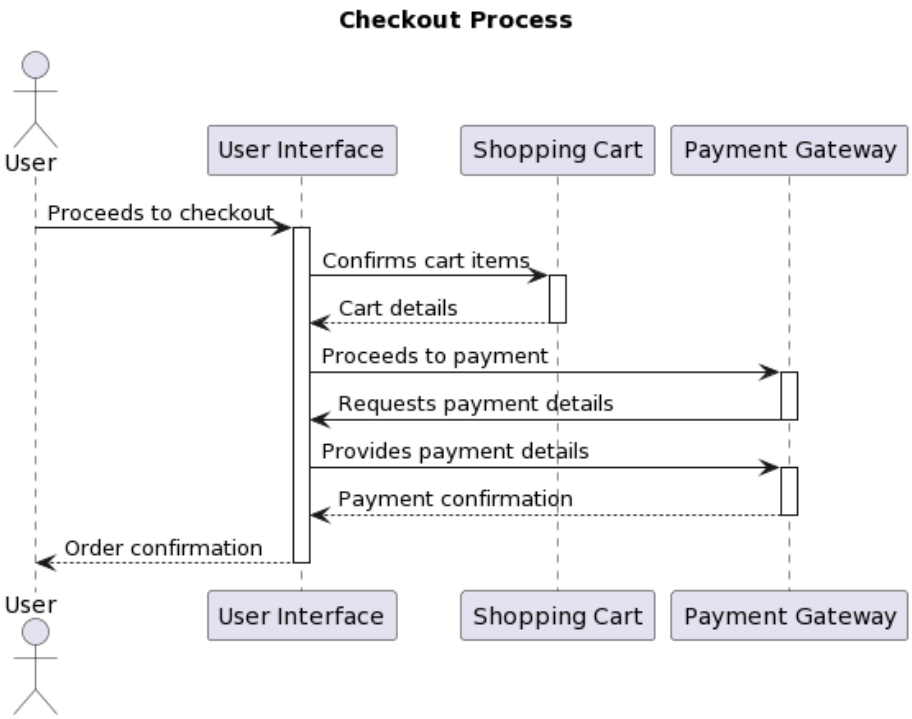


Figure 12: Sequence Diagram for Check out process

State Diagrams:

State Diagram: User profile

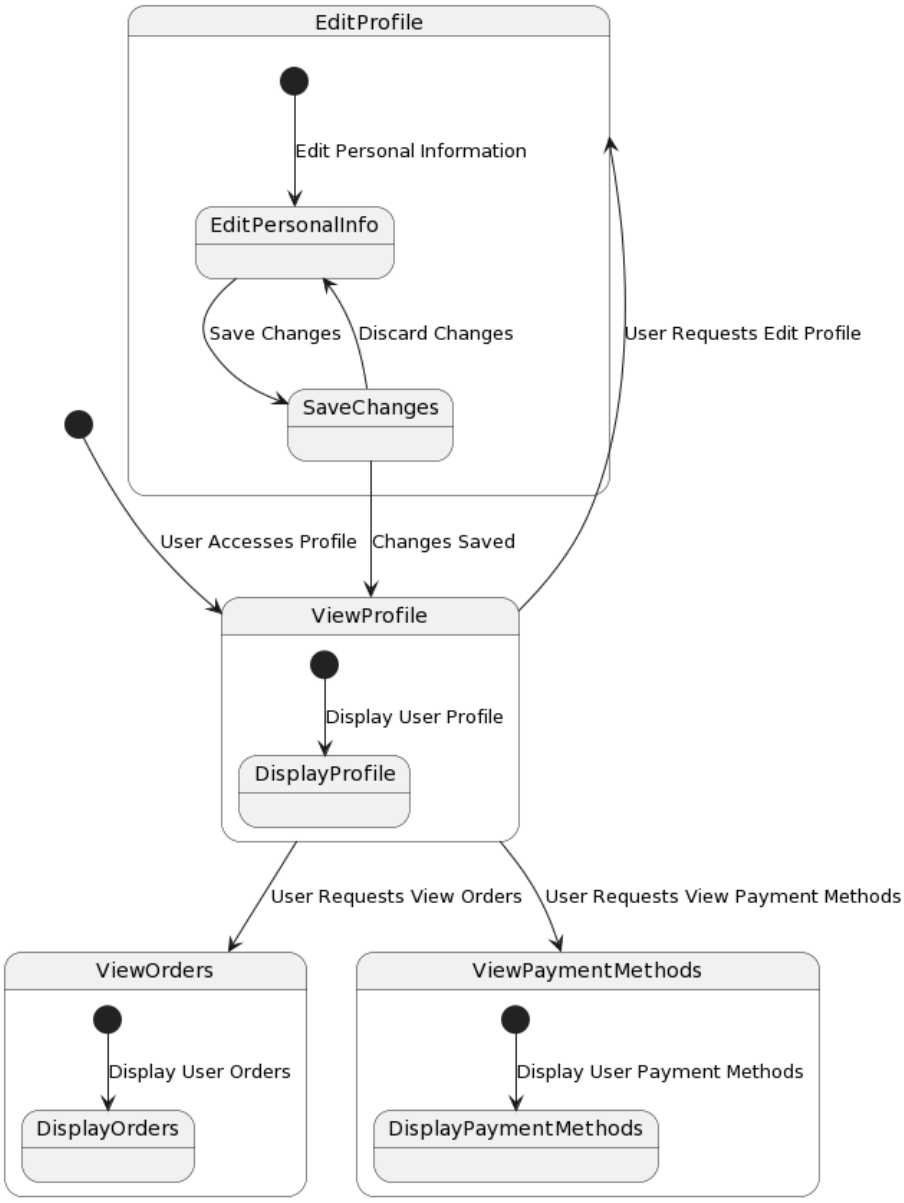


Figure 13: State Diagram for User profile

State Diagram: User authentication

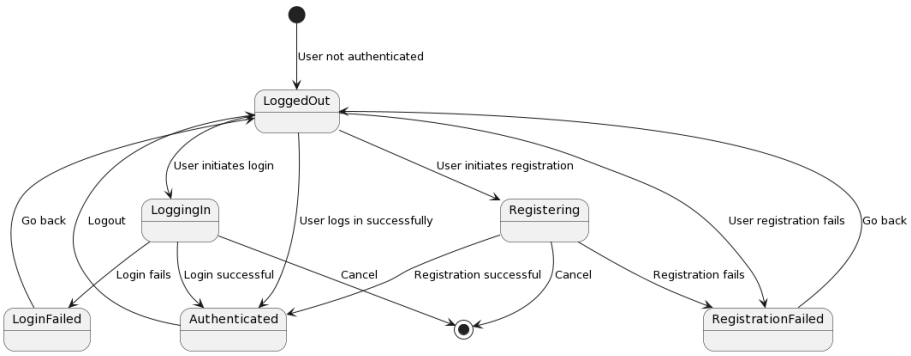
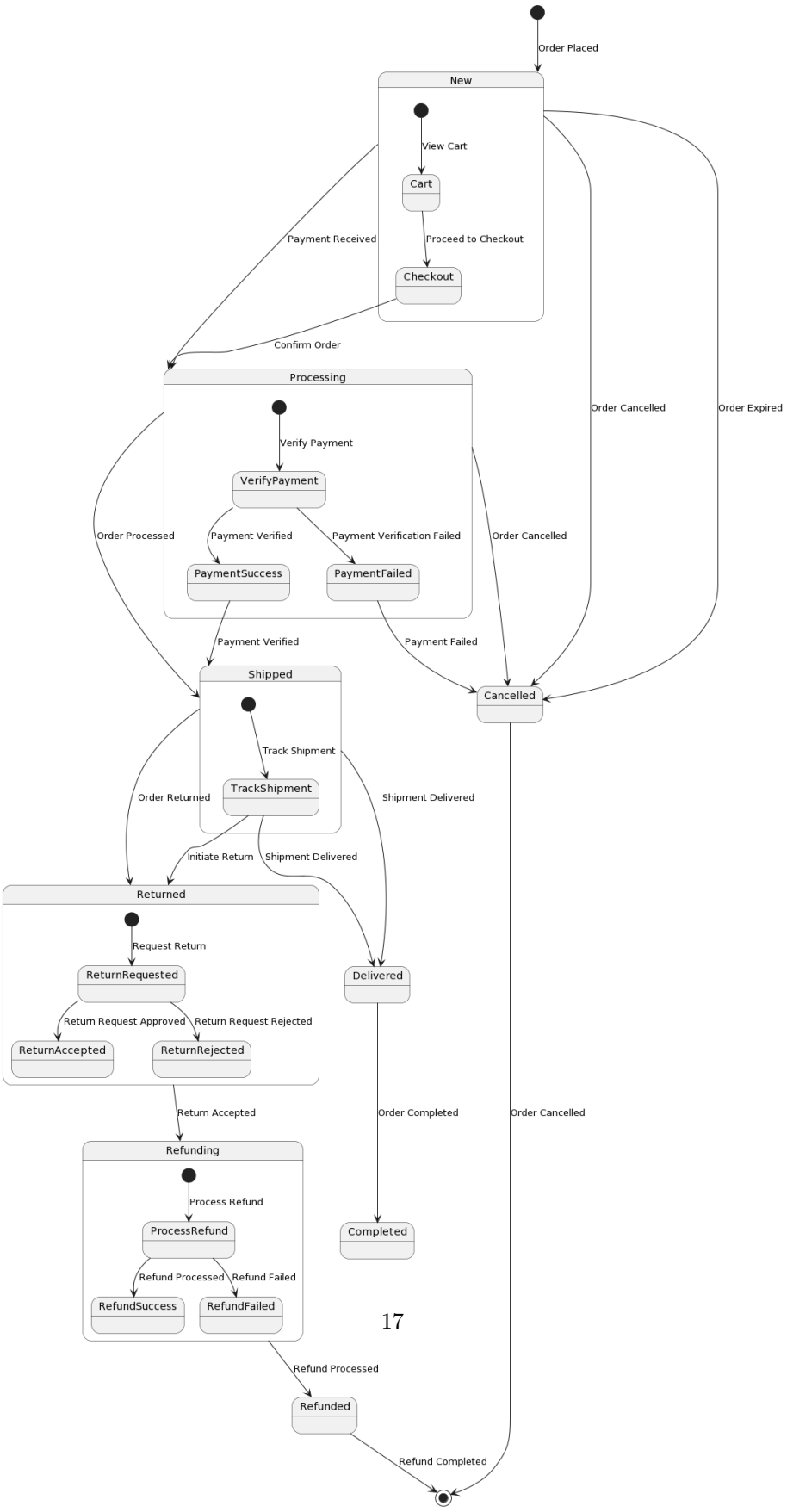


Figure 14: State Diagram for User authentication

State Diagram: Product order


Order Placed
New
View Cart
Cart
Proceed to Checkout
Checkout
Payment Received
Confirm Order
Order Cancelled
Order Expired
Processing
Verify Payment
VerifyPayment
Order Processed
Payment Verified
Payment Verification Failed
Order Cancelled
PaymentSuccess
PaymentFailed
Payment Verified
Payment Failed
Shipped
Cancelled
Track Shipment
TrackShipment
Order Returned
Shipment Delivered
Initiate Return
Shipment Delivered
Returned
Request Return
ReturnRequested
Delivered
Return Request Approved
Return Request Rejected
ReturnAccepted
ReturnRejected
Return Accepted
Order Completed
Order Cancelled
Refunding
Process Refund
ProcessRefund
Completed
Refund Processed
Refund Failed
RefundSuccess
RefundFailed
Refund Processed
Refunded
Refund Completed

State Diagram: Product availability

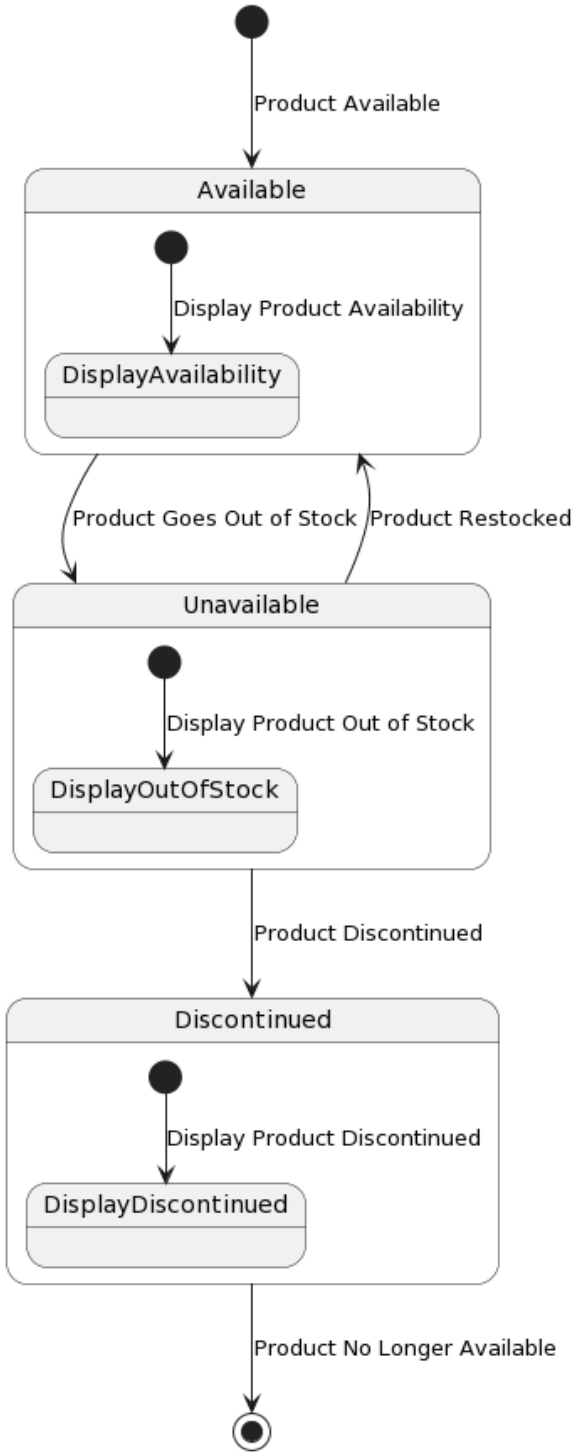


Figure 16: State Diagram for Product availability

State Diagram: Shopping cart

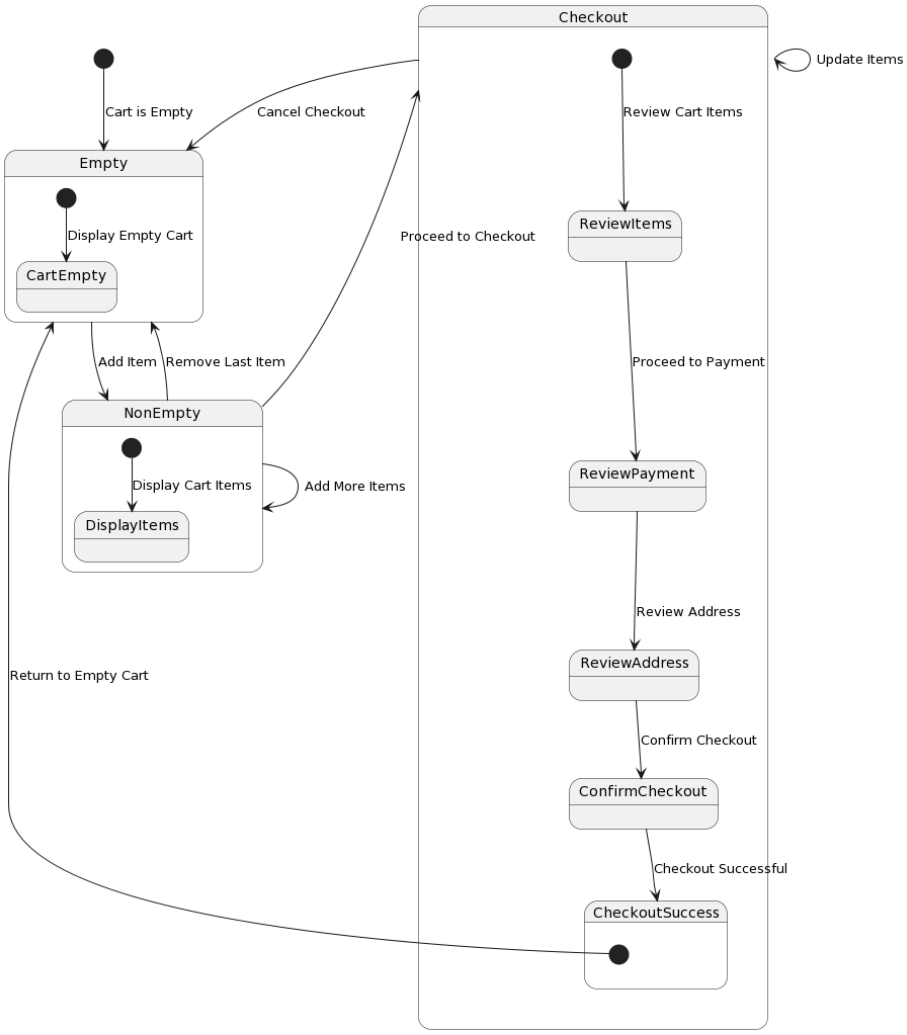


Figure 17: State Diagram for Shopping cart

State Diagram: Payment

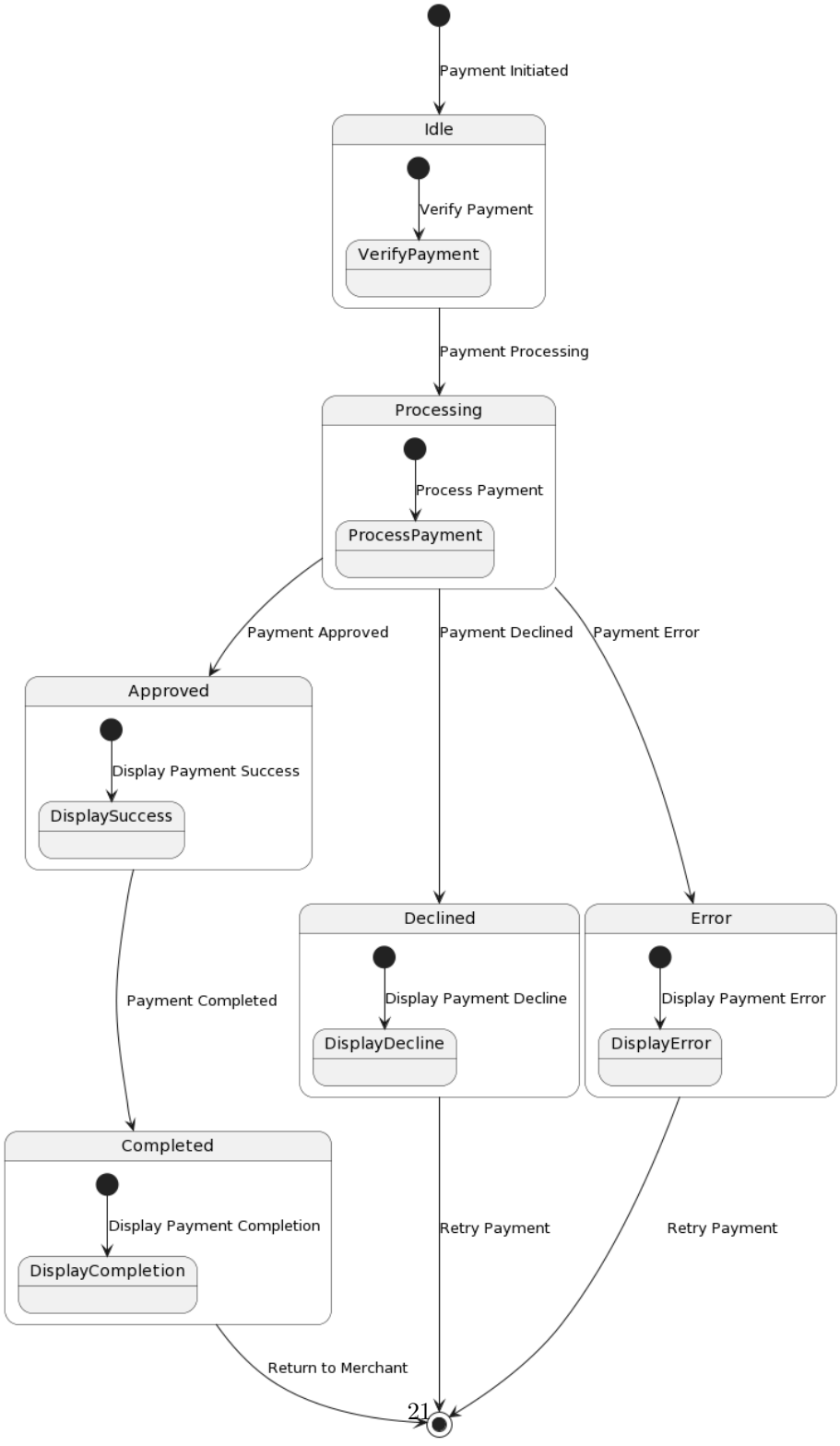


Figure 18: State Diagram for Payment

State Diagram: Review

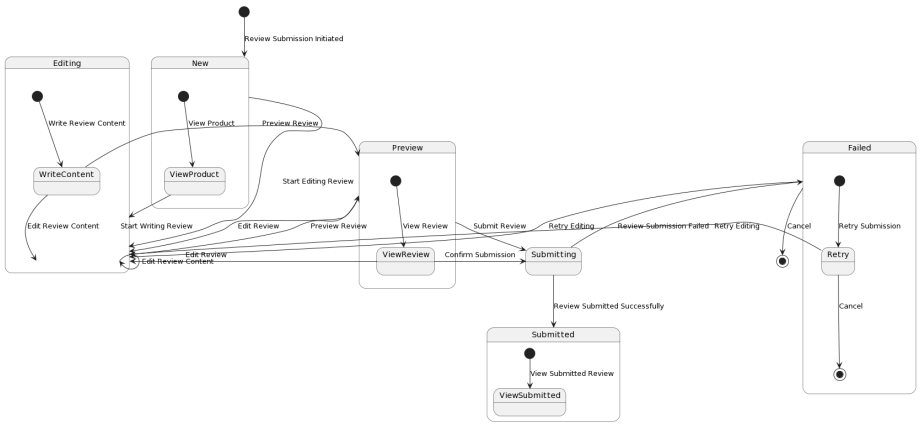


Figure 19: State Diagram for Review

3.1 Logical Architecture Description

3.1.1 Class Diagram explanation:

Here's a brief explanation of the class diagram:

User Class: Represents users of the e-commerce platform, with attributes such as user ID, username, password, email, and address. Users can register, log in, update their profiles, browse products, and purchase products.

Farmer Class: Represents farmers who cultivate crops, with attributes such as farmer ID, farm location, and a list of crops they cultivate. Farmers can add and remove crops from their list.

Crop Class: Represents individual crops available for sale, with attributes such as crop ID, name, quantity, price, and location. Each crop is associated with a farmer who cultivates it.

Product Class: Represents products available for sale on the e-commerce platform, with attributes such as product ID, name, description, price, and location.

Order Class: Represents orders placed by users, with attributes such as order ID, user, list of products, total price, and order status. Users can add or remove products from their orders, calculate the total price, and proceed to checkout.

Review Class: Represents reviews submitted by users for products, with attributes such as review ID, user, product, rating, and comment. Users can add reviews for products they have purchased.

Payment Class: Represents payments made for orders, with attributes such as payment ID, order, payment status, and payment method. This class handles the processing of payments for orders.

Location Class: Represents locations where products are available, with attributes such as location ID, city name, and state name. Users can search for products available in specific locations.

Here are the relationships depicted in the class diagram:

- User to Farmer Relationship:
  - A user can have an optional association with a farmer, represented by the "1" and "0..1" multiplicity on the association line from User to Farmer. This indicates that a user may or may not be associated with a farmer, implying that not all users are necessarily farmers.
- Farmer to Crop Relationship:
  - A farmer can have multiple crops associated with them, as indicated by the "*" multiplicity on the association line from Farmer to Crop. This means that a farmer can cultivate multiple crops.
- User to Order Relationship:
  - A user can have multiple orders associated with them, as indicated by the "*" multiplicity on the association line from User to Order. This means that a user can place multiple orders.
- Order to Product Relationship:
  - An order can contain multiple products, as indicated by the "*" multiplicity on the association line from Order to Product. This means that an order can consist of multiple products.
- Product to Review Relationship:
  - A product can have multiple reviews associated with it, as indicated by the "*" multiplicity on the association line from Product to Review. This means that multiple users can review the same product.
- User to Review Relationship:
  - A user can submit multiple reviews, as indicated by the "0..*" multiplicity on the association line from User to Review. This means that a user can submit zero or more reviews.

- Order to Payment Relationship:
  - An order is associated with one payment, as indicated by the "1" multiplicity on the association line from Order to Payment. This means that each order is associated with a single payment.
- Location to Product Relationship:
  - A location can have multiple products associated with it, as indicated by the "*" multiplicity on the association line from Location to Product. This means that multiple products can be available in the same location.

These relationships define how the various entities in the system are connected and interact with each other, providing a comprehensive view of the system's structure and functionality.

2 Sequence Diagram

Arrow line signifies there is a send message taken place. Response is being shown by dotted arrows.

2.1 User Registration

1. The user initiates the registration process by providing necessary details. 2. The system verifies the provided information and creates a new user account. 3. Upon successful registration, the system sends a confirmation message to the user.

2.2 User Login

1. The user enters login credentials (username and password). 2. The system verifies the credentials and authenticates the user. 3. Upon successful authentication, the system grants access to the user's account.

2.3 Product Search and Purchase

1. The user searches for a product by entering keywords or selecting categories. 2. The system retrieves relevant products based on the search criteria. 3. The user selects a product and adds it to the shopping cart. 4. The system calculates the total cost and generates an invoice. 5. The user proceeds to checkout, enters shipping and payment details. 6. The system processes the payment and updates the order status. 7. Finally, the system confirms the purchase and sends a confirmation email to the user.

2.4 Adding Product to Cart

1. The user interacts with the user interface to select a product to add to the cart. 2. The user interface requests product details from the product catalog. 3. The product catalog provides the product information to the user interface. 4. The user interface adds the product to the shopping cart. 5. The shopping cart sends a confirmation message back to the user interface. 6. The user interface notifies the user that the cart has been updated.

2.5 Checkout Process

1. The user indicates the intention to proceed to checkout through the user interface. 2. The user interface confirms the items in the shopping cart. 3. Once confirmed, the user interface proceeds to the payment gateway for payment. 4. The payment gateway requests payment details from the user interface. 5. The user provides the payment details to the payment gateway. 6. The payment gateway confirms the payment, and the user interface notifies the user of the payment confirmation.

2.6 Order Confirmation

1. The user receives an order confirmation through the user interface. 2. The user interface fetches the order details from the order system. 3. The order system sends the order information to the user interface. 4. The user interface displays the order summary to the user.

3 State Diagram

Initial state is being shown by starting with a black dot. Final State is being shown by the black dot surrounded by an empty circle.

3.1 Product order

States such as New, Processing, Shipped, Delivered, Completed, Cancelled, Returned, and Refunding represent different stages in the order lifecycle. Transitions between states illustrate actions or events triggering state changes. Substates within states provide more detailed actions or processes occurring within that state. For example, within the Processing state, there are substates for verifying payment. Various transitions cover scenarios like payment success or failure, order cancellation, shipment tracking, return initiation, refund processing, etc.

3.2 Shopping Cart

States such as Empty, NonEmpty, and Checkout represent different stages in the shopping cart lifecycle. Transitions between states illustrate actions or events

triggering state changes. Substates within states provide more detailed actions or processes occurring within that state. For example, within the Checkout state, there are substates for reviewing items, payment, address, and confirming checkout. Various transitions cover scenarios like adding or removing items from the cart, proceeding to checkout, reviewing items and payment during checkout, and confirming checkout.

3.3 Payment

States such as Idle, Processing, Approved, Declined, Error, and Completed represent different stages in the payment process lifecycle. Transitions between states illustrate actions or events triggering state changes. Substates within states provide more detailed actions or processes occurring within that state. For example, within the Processing state, there is a substate for processing payment. Various transitions cover scenarios like initiating payment, processing payment, payment approval or decline, handling payment errors, and completing the payment process.

3.4 User profile

States such as ViewProfile, EditProfile, ViewOrders, and ViewPaymentMethods represent different stages in the user profile lifecycle. Transitions between states illustrate actions or events triggering state changes. Substates within states provide more detailed actions or processes occurring within that state. For example, within the EditProfile state, there are substates for editing personal information and saving changes. Various transitions cover scenarios like requesting to view profile, editing profile information, viewing orders, and viewing payment methods.

3.5 Product availability

States such as Available, Unavailable, and Discontinued represent different states of product availability. Transitions between states illustrate actions or events triggering state changes. For example, transitioning from Available to Unavailable occurs when the product stock is depleted. Various transitions cover scenarios like product availability changes due to stock updates, product discontinuation, or changes in product status.

3.6 User authentication

States such as LoggedOut, LoggingIn, LoggedIn, LoggingOut, and Registration represent different states in the user authentication process. Transitions between states illustrate actions or events triggering state changes. For example, transitioning from LoggedOut to LoggingIn occurs when the user initiates the login process. Various transitions cover scenarios like user login, logout, and registration processes.

3.7 Review

States such as ViewReviews, WriteReview, and EditReview represent different states in the review lifecycle. Transitions between states illustrate actions or events triggering state changes. For example, transitioning from ViewReviews to WriteReview occurs when the user selects to write a review for a product. Various transitions cover scenarios like viewing existing reviews, writing new reviews, and editing or deleting existing reviews.

4 Execution Architecture

The execution architecture of our e-commerce application defines the runtime environment, processes, and deployment view of the system.

4.1 Runtime Environment

The runtime environment of the e-commerce application consists of the following components:

- **Web Server**: Responsible for serving web pages to users and handling HTTP requests.
- **Application Server**: Hosts the application logic and processes business logic requests from clients.
- **Database Server**: Stores and manages data related to products, users, orders, and transactions.

These components work together to ensure the proper functioning of the application during runtime.

4.2 Processes

The execution of the e-commerce application involves several processes, including:

- **User Authentication**: Process of verifying the identity of users before granting access to the application.
- **Product Management**: Process of adding, updating, and removing products from the system.
- **Order Processing**: Process of handling orders placed by users, including payment processing and order fulfillment.
- **Reporting and Analytics**: Process of generating reports and analyzing data to gain insights into user behavior and sales performance.

These processes are executed sequentially or concurrently to ensure the smooth operation of the application.

4.3 Deployment View

The deployment view of the e-commerce application describes how the software components are deployed across different environments, such as development, testing, and production. It includes details about server configurations, network topology, and deployment strategies.

4.4 Reuse and relationships to other products

Our e-commerce application leverages existing technologies and services to enhance its functionality and performance. Some of the key relationships to other products include:

- **Payment Gateways**: Integration with third-party payment gateways to facilitate secure and efficient payment processing.
- **Shipping Providers**: Integration with shipping providers to calculate shipping costs, generate shipping labels, and track shipments.
- **Cloud Services**: Utilization of cloud services for hosting, storage, and scalability to handle varying levels of user traffic.
- **Content Delivery Networks (CDNs)**: Use of CDNs to cache and deliver static assets such as images, CSS, and JavaScript files to users worldwide, reducing latency and improving performance.

These relationships enable our e-commerce application to provide a seamless and reliable shopping experience to users while optimizing resource utilization and cost efficiency.

5 Design Decisions and Trade-offs

This section discusses the design decisions made during the development of our e-commerce application and any trade-offs considered.

5.1 Scalability vs. Cost

One of the key design decisions was to prioritize scalability while managing costs effectively. We opted for a cloud-based infrastructure that allows us to scale resources up or down based on demand. However, this decision comes with a trade-off as scaling resources can incur additional costs. To mitigate this, we employ auto-scaling policies and cost optimization strategies to ensure efficient resource utilization.

5.2 Monolithic vs. Microservices Architecture

Another important decision was choosing between a monolithic and microservices architecture. We ultimately decided to adopt a microservices architecture to improve modularity, scalability, and flexibility. However, this approach introduces complexities such as service discovery, inter-service communication, and managing distributed systems. We carefully weighed the benefits and trade-offs before committing to this architectural style.

5.3 Consistency vs. Performance

Ensuring data consistency across distributed systems while maintaining high performance is a constant trade-off. We implemented eventual consistency mechanisms and caching strategies to achieve a balance between consistency and performance. However, there are scenarios where immediate consistency is required, leading to potential performance overhead. We evaluate each use case carefully to determine the appropriate consistency model.

5.4 Technology Stack

Choosing the right technology stack was crucial for the success of our e-commerce application. We evaluated various frameworks, libraries, and tools based on factors such as performance, scalability, community support, and developer experience. While some technologies may offer superior performance, they may come with a steeper learning curve or limited community support. We prioritized technologies that strike a balance between performance and ease of development, ensuring long-term maintainability and scalability.

5.5 User Experience vs. Security

Balancing user experience with security was a critical consideration in the design of our application. While we strive to provide a seamless and intuitive user experience, we also prioritize security measures to protect user data and prevent unauthorized access. This involves implementing secure authentication mechanisms, data encryption, and robust access control policies. However, stringent security measures may sometimes introduce friction in the user experience. We aim to find the right balance between usability and security to deliver a secure yet user-friendly application.

5.6 Trade-offs Evaluation

Throughout the design process, we continuously evaluate trade-offs and make informed decisions based on factors such as project requirements, technical constraints, and stakeholder feedback. We prioritize trade-offs that align with our project goals and objectives while mitigating risks and maximizing benefits.

6 Pseudo code for components

6.1 Class Name: Login

6.1.1 Method 1: onCreate()

Pseudo-code:

```
Input: savedInstanceState, Email, Password
Output: Launch the Activity, SignIn
1. super.onCreate(savedInstanceState);
2. setContentView(R.layout.activity_login);
3. mAuth = get FirebaseAuth instance
4. editTextEmail = stores the ID of EditText for email
5. editTextPassword = stores the ID of EditText for password
6. buttonSignIn = stores the ID of SignInButton
7. progressDialog = create new ProgressDialog object
8. buttonSignIn.setOnClickListener(new View.OnClickListener()
9. public void onClick(View v)
10. signIn() // User sign in
```

6.1.2 Method 2: Admin_Login()

Pseudo-code:

```
Input: view, email, password
Output: Admin landing page if login successful
1. String email = get email from the user
2. String password = get password from the user
3. if email != null and password != null then4.    progressDialog.setMessage("Logging in, pl
6.    mAuth.signInWithEmailAndPassword(email, password)
7.          .addOnCompleteListener(new OnCompleteListener<AuthResult>()
8.          {
9.              public void onComplete(@NonNull Task<AuthResult> task)
10.             {
11.                 progressDialog.dismiss();
12.                 if (task.isSuccessful())
13.                 {
14.                     // Redirect to Admin landing page
15.                 }
16.                 else
17.                 {
18.                     // Display login failed message
19.                 }
20.             }
21.         });
22. else23.     // Display error message for empty email or password
```

6.1.3 Method 3: consumer_Login()

Pseudo-code:

```
Input: view, email, password
Output: Consumer landing page if login successful
1. String email = get email from the user
2. String password = get password from the user
3. if email != null and password != null then4.    progressDialog.setMessage("Logging in, pl
6.    mAuth.signInWithEmailAndPassword(email, password)
7.          .addOnCompleteListener(new OnCompleteListener<AuthResult>()
8.          {
9.              public void onComplete(@NonNull Task<AuthResult> task)
10.             {
11.                 progressDialog.dismiss();
12.                 if (task.isSuccessful())
13.                 {
14.                     // Redirect to Consumer landing page
15.                 }
16.                 else
17.                 {
18.                     // Display login failed message
19.                 }
20.             }
21.         });
22. else23.     // Display error message for empty email or password
```

6.2 Class Name: Admin

6.2.1 Method 1: login(String username, String password)

Pseudo-code:

```
Input: username - the username of the admin
       password - the password of the admin
Output: Confirmation of successful login or error message
1. Check if the username and password match the credentials of an admin in the database
2. If the credentials are valid:3.     Log the admin into the system4.     Return a success
5. Else:6.     Return an error message indicating invalid credentials
```

6.2.2 Method 2: addProduct(Product product)

Pseudo-code:

```
Input: product - the product to be added to the inventory
Output: Confirmation of successful addition of the product or error message
1. Add the product to the inventory database
2. Return a success message indicating the product was added successfully
```

6.2.3 Method 3: removeProduct(Product product)

Pseudo-code:

```
Input: product - the product to be removed from the inventory
Output: Confirmation of successful removal of the product or error message
1. Check if the product exists in the inventory
2. If the product exists:3.     Remove the product from the inventory database4.     Return
5. Else:6.     Return an error message indicating the product does not exist
```

6.2.4 Method 4: updateProduct(Product product)

Pseudo-code:

```
Input: product - the product to be updated in the inventory
Output: Confirmation of successful update of the product or error message
1. Check if the product exists in the inventory
2. If the product exists:3.     Update the product details in the inventory database4.     R
5. Else:6.     Return an error message indicating the product does not exist
```

6.3 Class Name: Notification

6.3.1 Method 1: Notification()

Pseudo-code:

```
Input: mAuthListener
Output: handle authentication state changes
1. String strtitle = set notification title
2. String strtext = set notification text
```

6.3.2 Method 2: getNotification(Bundle savedInstanceState)

Pseudo-code:

```
Input: savedInstanceState { object of bundle
Output: Displaying the notification
1. super.onCreate(savedInstanceState);
2. setContentView(R.layout.activity_main);
3. NotificationManager notificationmanager = create new NotificationManager
4. Intent i = retrieve data from MainActivity
5. title = i.getStringExtra("title");
6. text = i.getStringExtra("text");
7. txttitle.setText(title);
8. txttext.setText(text);
```

6.4 Class Name: FindItems

6.4.1 Method 1: searchItems(String query)

Pseudo-code:

```
Input: query - search query entered by the user
Output: List of items matching the search query
1. Initialize an empty list to store search results
2. Iterate over the database of available items
3. For each item in the database:4.     Check if the item matches the search query5.     If
6. Return the list of search results
```

6.4.2 Method 2: filterItems(String category)

Pseudo-code:

```
Input: category - category selected by the user for filtering items
Output: List of items belonging to the selected category
1. Initialize an empty list to store filtered items
2. Iterate over the database of available items
3. For each item in the database:4.     Check if the item belongs to the selected category5.
6. Return the list of filtered items
```

6.4.3 Method 3: sortItems(String sortBy)

Pseudo-code:

```
Input: sortBy - sorting criteria selected by the user
Output: Sorted list of items based on the selected criteria
1. Initialize an empty list to store sorted items
2. Iterate over the database of available items
3. Add all items to the list of sorted items
4. Sort the list of items based on the selected criteria (e.g., price, popularity)
5. Return the sorted list of items
```

6.5 Class Name: AddToCart

6.5.1 Method 1: addToCart(Item item, int quantity)

Pseudo-code:

```
Input: item - the item to be added to the cart
       quantity - the quantity of the item to be added
Output: Updated shopping cart with the added item
1. Check if the item is available in stock
2. If the item is available:3.     If the quantity requested is less than or equal to the av
5.         Update the stock level of the item in the inventory
6.         Return a success message indicating the item was added to the cart
```

```
7.     Else:
8.         Return an error message indicating insufficient stock
9. Else:10.    Return an error message indicating the item is out of stock
```

6.5.2 Method 2: updateCart(Item item, int newQuantity)

Pseudo-code:

```
Input: item - the item in the cart to be updated
       newQuantity - the new quantity of the item
Output: Updated shopping cart with the quantity of the specified item updated
1. Check if the item is present in the shopping cart
2. If the item is present:3.     If the new quantity is less than or equal to the available
5.         Update the stock level of the item in the inventory
6.         Return a success message indicating the cart was updated
7.     Else:
8.         Return an error message indicating insufficient stock
9. Else:10.    Return an error message indicating the item is not in the cart
```

6.5.3 Method 3: removeFromCart(Item item)

Pseudo-code:

```
Input: item - the item to be removed from the cart
Output: Updated shopping cart with the specified item removed
1. Check if the item is present in the shopping cart
2. If the item is present:3.     Remove the item from the shopping cart4.     Return a succe
5. Else:6.     Return an error message indicating the item is not in the cart
```

6.6 Class Name: HomePage

6.6.1 Method 1: displayHomePage()

Pseudo-code:

```
Input: None
Output: Display the home page UI to the user
1. Load the home page layout
2. Display a welcome message to the user
3. Display featured products or categories to attract user attention
4. Provide navigation options to different sections of the application (e.g., browse product
5. Implement functionality to handle user interactions (e.g., click events, navigation)
6. Optionally, display promotional banners or announcements
```

6.6.2 Method 2: navigateToProductPage(Product product)

Pseudo-code:

```
Input: product - the product selected by the user
Output: Navigate to the product page for the selected product
1. Retrieve details of the selected product
2. Load the product page layout
3. Display product information (e.g., name, description, price)
4. Provide options for the user to add the product to their cart or view more details
5. Implement functionality to handle user interactions (e.g., add to cart button click)
```

6.6.3 Method 3: searchProducts(String query)

Pseudo-code:

```
Input: query - the search query entered by the user
Output: Display search results for the specified query
1. Retrieve products matching the search query from the database
2. Display search results to the user
3. Provide options for the user to refine their search or view more details about specific p
4. Implement functionality to handle user interactions (e.g., click events)
```

6.6.4 Method 4: viewCart()

Pseudo-code:

```
Input: None
Output: Navigate to the shopping cart page
1. Load the shopping cart page layout
2. Display the items currently in the user's cart
3. Provide options for the user to update quantities, remove items, or proceed to checkout
4. Implement functionality to handle user interactions (e.g., update quantity, remove item)
```

6.7 Class Name: Payment

6.7.1 Method 1: processPayment(Cart cart, PaymentMethod paymentMethod)

Pseudo-code:

```
Input: cart - the user's shopping cart containing items to be purchased
       paymentMethod - the method of payment chosen by the user
Output: Confirmation of successful payment or error message
1. Calculate the total amount to be paid based on the items in the cart
2. Verify the selected payment method and proceed accordingly
3. If paymentMethod is CreditCard:4.     Prompt the user to enter credit card details (e.g.,
6.     If the payment is successful:
7.         Deduct the amount from the user's account
8.         Update the order status to "Paid"
9.         Send a confirmation email to the user
10.        Return a success message
```

```
11.    Else:
12.        Return an error message indicating payment failure
13. Else if paymentMethod is CashOnDelivery:14.    Generate an order invoice and notify the
16.    Return a success message
17. Else:18.    Return an error message indicating unsupported payment method
```

6.7.2 Method 2: verifyPayment(PaymentDetails paymentDetails)

Pseudo-code:

```
Input: paymentDetails - details of the payment to be verified (e.g., credit card informatio
Output: Confirmation of payment verification or error message
1. Check the validity of the paymentDetails provided by the user
2. If the paymentDetails are valid:3.     Verify the payment with the payment gateway4.
5.         Return a success message
6.     Else:
7.         Return an error message indicating payment verification failure
8. Else:9.     Return an error message indicating invalid paymentDetails
```

6.8 Class Name: CustomerReview

6.8.1 Method 1: addReview(Product product, String reviewText, int rating)

Pseudo-code:

```
Input: product - the product being reviewed
       reviewText - the text of the review submitted by the customer
       rating - the numerical rating assigned by the customer (e.g., on a scale of 1 to 5)
Output: Confirmation of successful addition of the review or error message
1. Create a new review object with the provided reviewText and rating
2. Associate the review with the specified product
3. Add the review to the product's list of customer reviews
4. Return a success message indicating the review was added successfully
```

6.8.2 Method 2: viewReviews(Product product)

Pseudo-code:

```
Input: product - the product for which reviews are to be viewed
Output: List of customer reviews for the specified product
1. Retrieve the list of customer reviews associated with the specified product
2. Display the reviews to the user, including the review text and rating
3. Optionally, provide options for sorting or filtering the reviews (e.g., by date, by ratin
```

6.9 Class Name: FarmersToContactUs

6.9.1 Method 1: submitInquiry(String farmerName, String farmerLocation, String contactInfo, String inquiry)

Pseudo-code:

```
Input: farmerName - the name of the farmer making the inquiry
       farmerLocation - the location of the farmer (e.g., address, city)
       contactInfo - contact information of the farmer (e.g., phone number, email)
       inquiry - the inquiry or message from the farmer
Output: Confirmation of successful submission of the inquiry or error message
1. Create a new inquiry object with the provided details (farmerName, farmerLocation, contac
2. Save the inquiry in the database or send it via email to the appropriate department
3. Return a success message indicating the inquiry was submitted successfully
```

6.9.2 Method 2: viewInquiries()

Pseudo-code:

```
Input: None
Output: List of inquiries received from farmers
1. Retrieve the list of inquiries from the database
2. Display the inquiries to the appropriate department for further action
3. Provide options for responding to inquiries (e.g., contacting farmers, providing informat
```

7 Appendices