Design Document

for

IITKart

Version 1.01

Prepared by

Group 5: **Group Name: Thunderbolts**

Hardik Tiwari	230437	hardikt23@iitk.ac.in
Pranjali Deshpande	240336	deshpandep24@iitk.ac.in
Mahi Mittal	240608	mahii24@iitk.ac.in
Mayukh Chowdhury	240642	cmayukh24@iitk.ac.in
Palak Bandhu	240719	palakb24@iitk.ac.in
Praveen	240790	praveenas24@iitk.ac.in
Shreshthraj Bhidodiya	240996	sbhidodiya24@iitk.ac.in
Sneha Kumari	241024	snehak24@iitk.ac.in
Virendra Kala	241172	virendrak24@iitk.ac.in
Yash Dabi	241195	yashd24@iitk.ac.in

Course: CS253

Mentor TA: *George T L*

Date: 06/02/2026

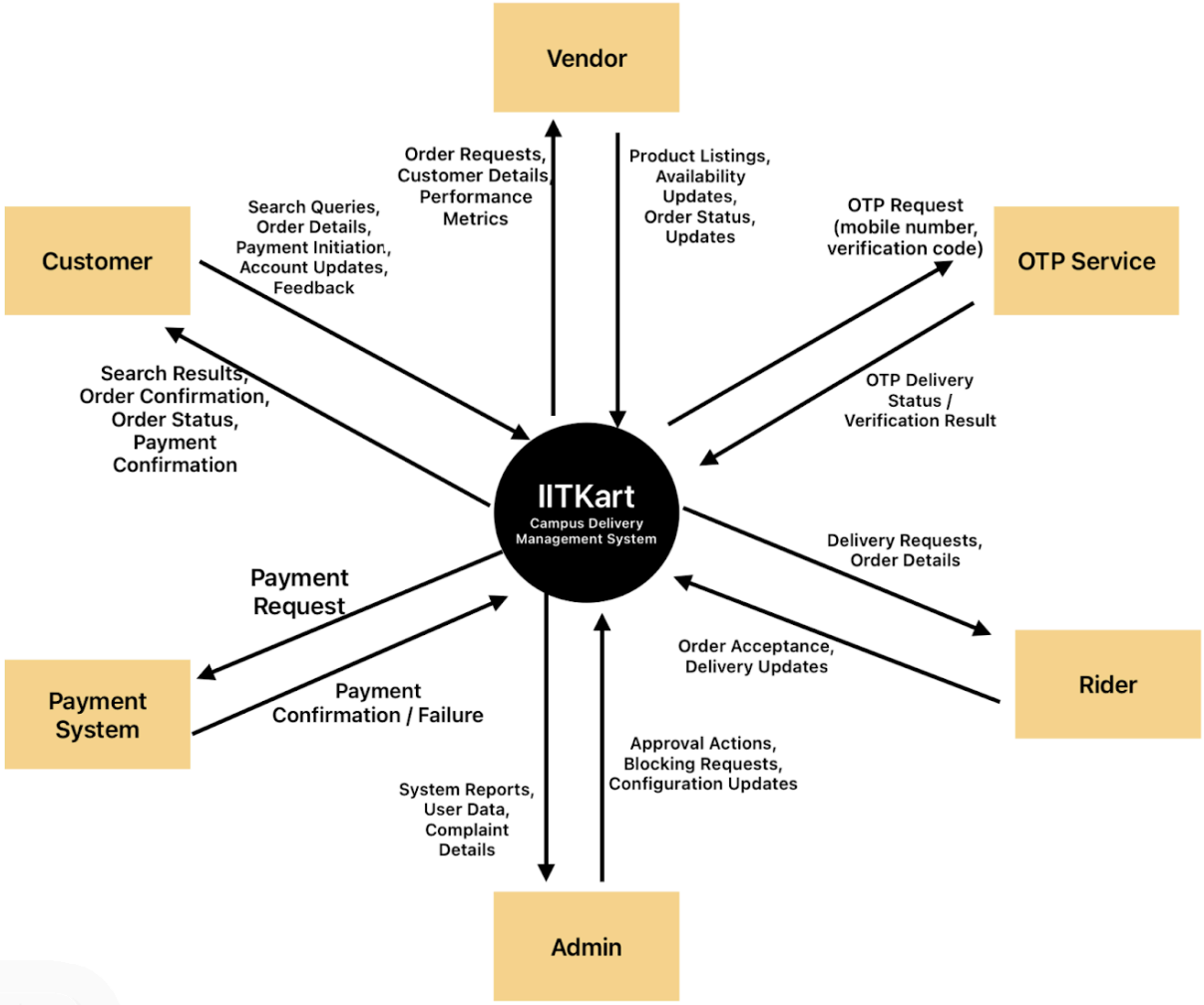
INDEX

Revisions

Version	Primary Author(s)	Description of Version	Date Completed
1.00	Thunderbolts	First draft of full design document completed	05/02/2026
1.01	Thunderbolts	Document proofread,changes and modifications implemented.	06/02/2026

1 Context Design

1.1 Context Model

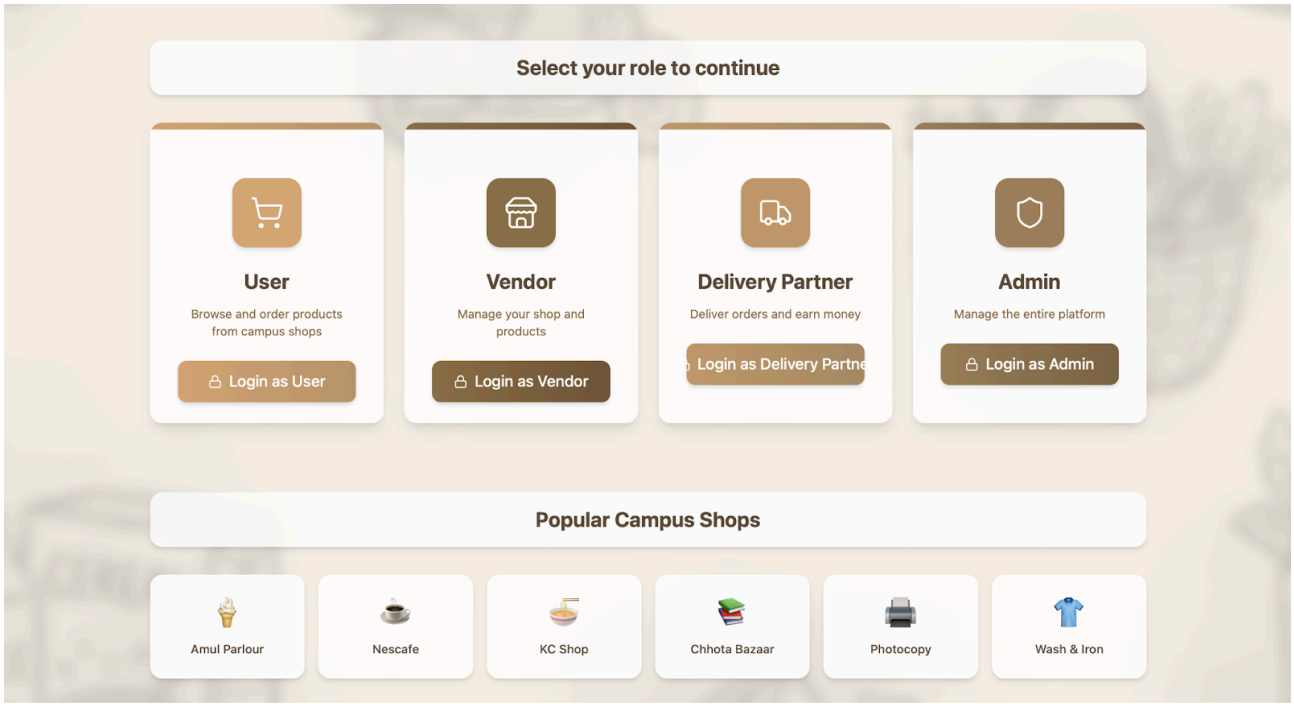


The context model diagram presents IITKart as a single, self-contained system and illustrates its interaction with external entities. It defines the system boundary and identifies all external actors, stakeholders and services that exchange information with the system.

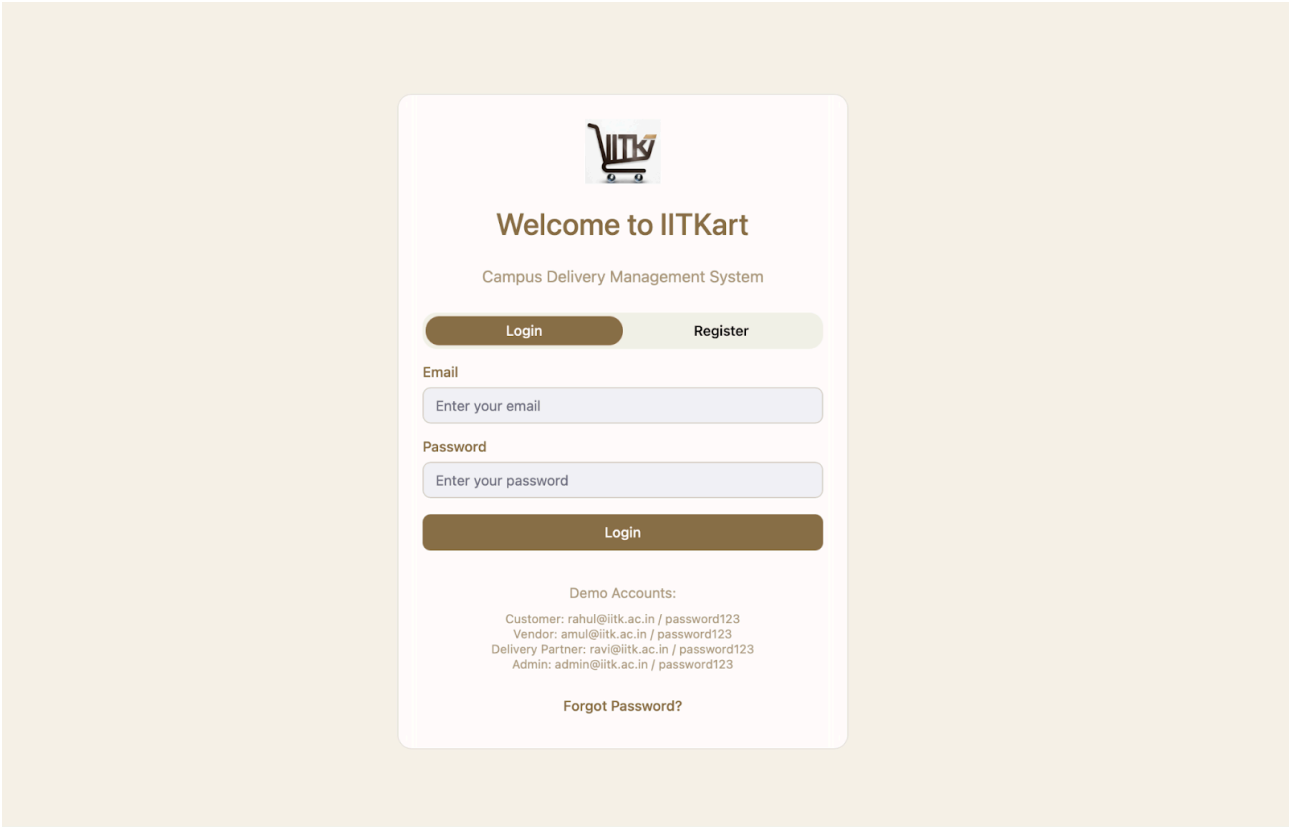
- **Customers** will initiate product searches, order placements, payments, reviews and feedback .
- **Vendors** will be managing product listings, availability, and order fulfillment.
- **Rider** will accept/reject delivery requests, update delivery status and report delivery issues (if any).
- **Admin** will oversee overall system governance, user management, and complaint handling.
- **Payment System** is an external service/gateway responsible for processing order payments.
- **OTP Service** is an external service to be used for account verification during registration.

1.2 Human Interface Design

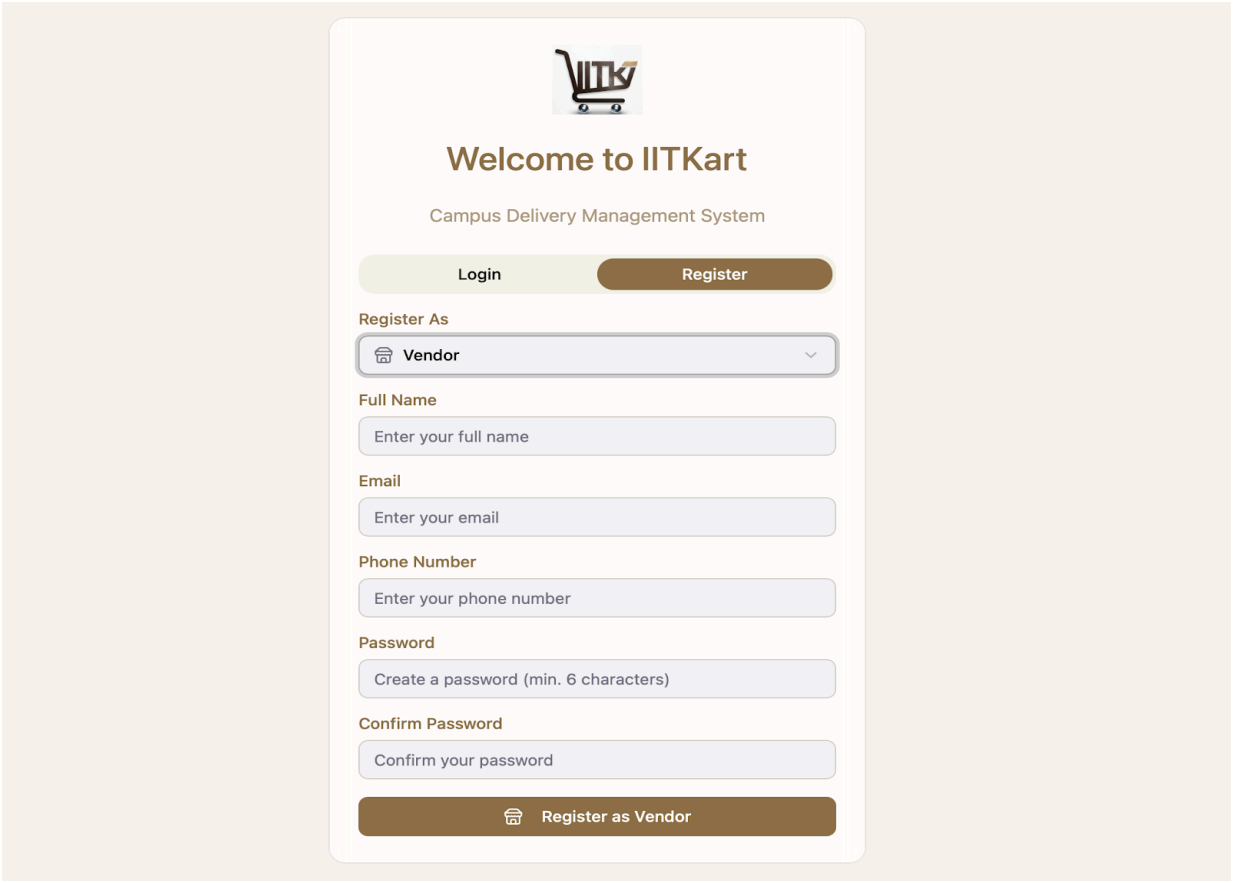
1.2.1 Home Page



1.2.2 Login Page

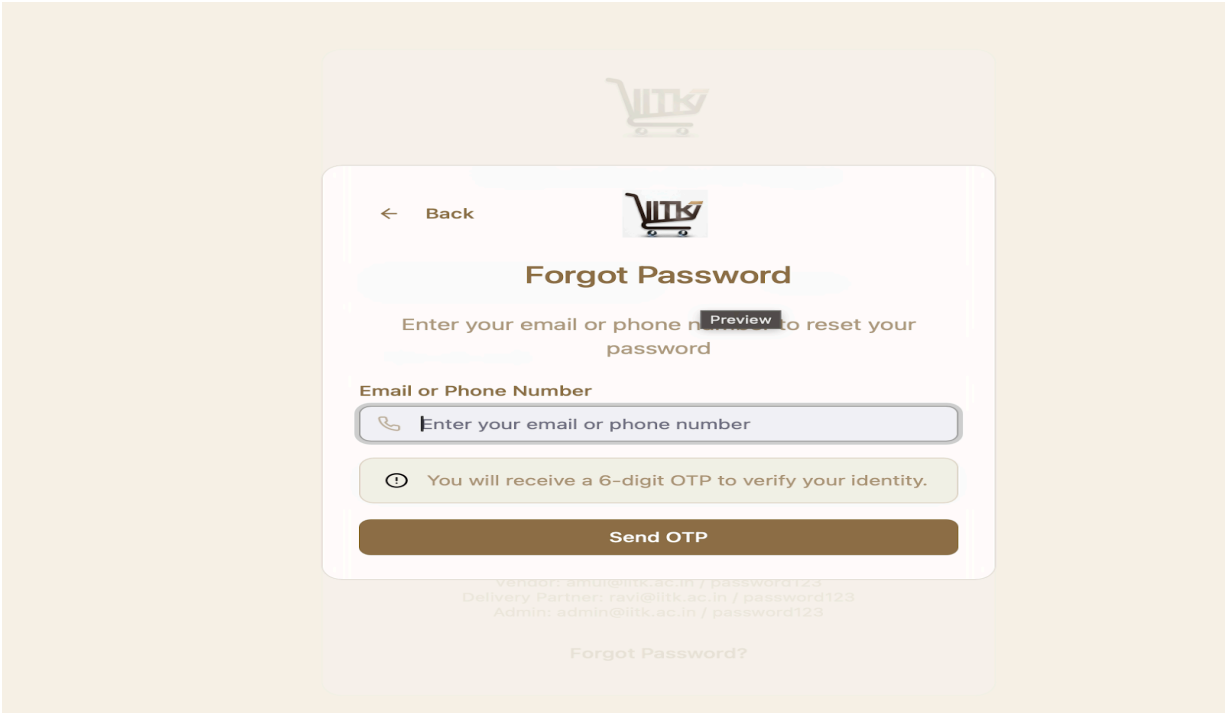


1.2.3 Registration Page

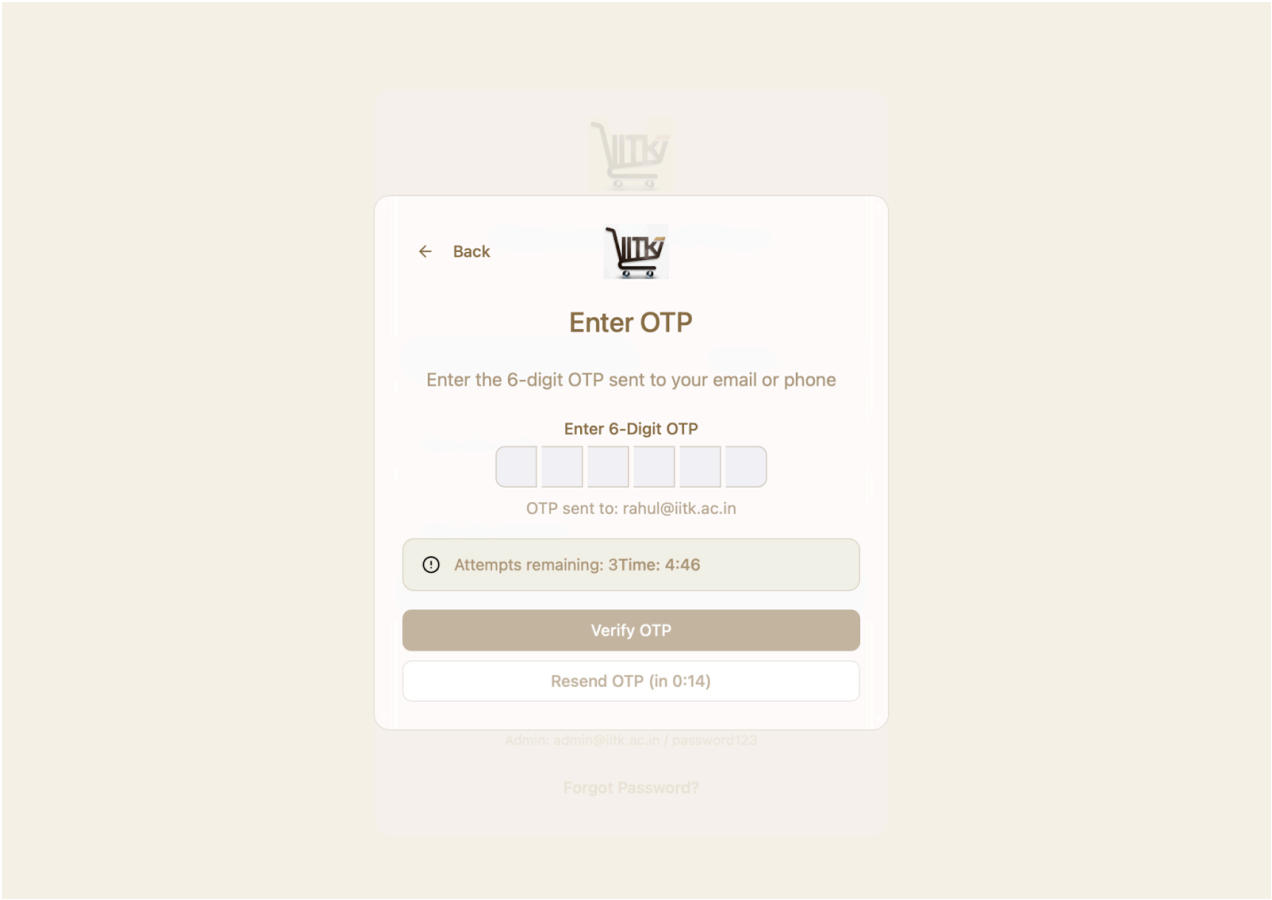


1.2.4 Reset Password

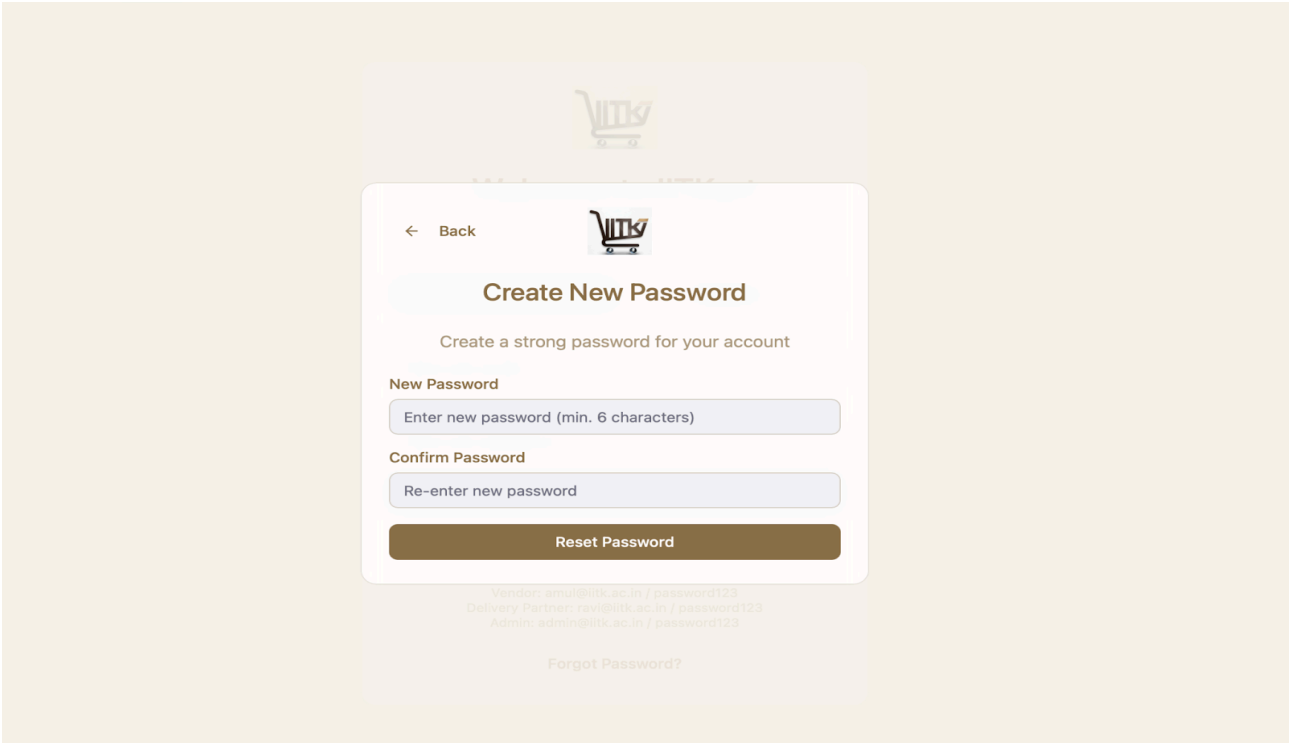
- **Password Reset Request**



- **OTP Verification**

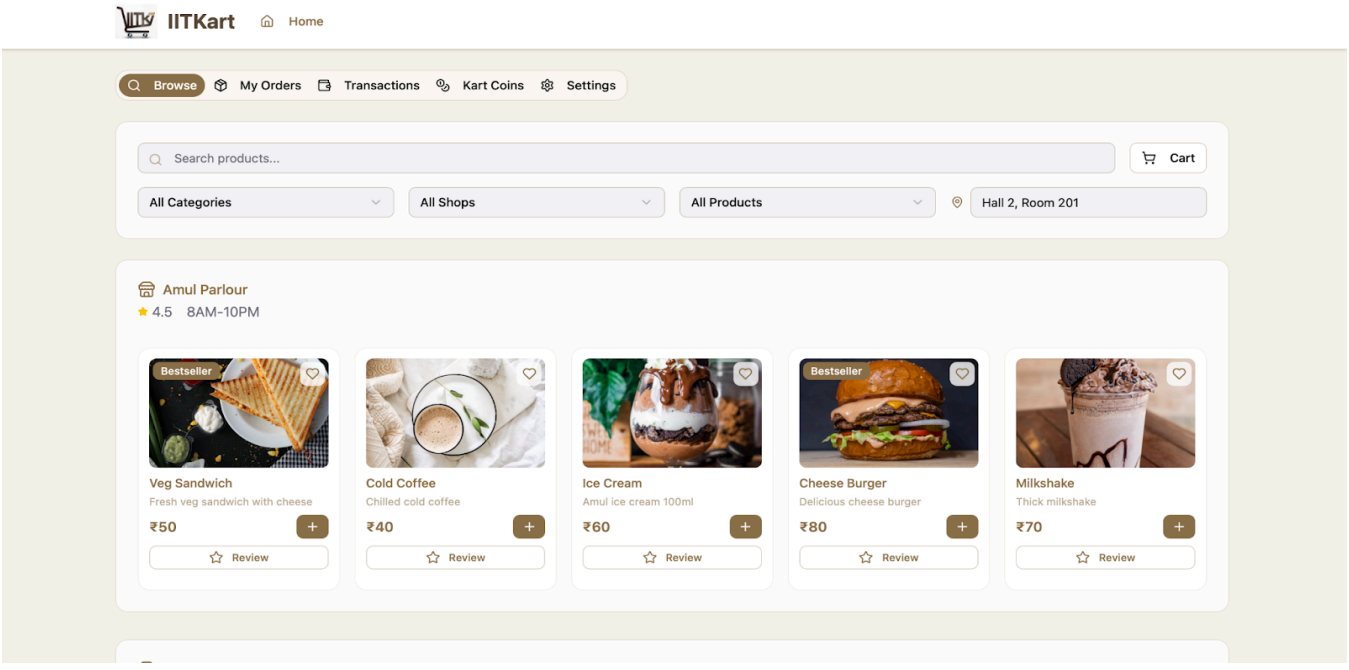


- **Create New Password**

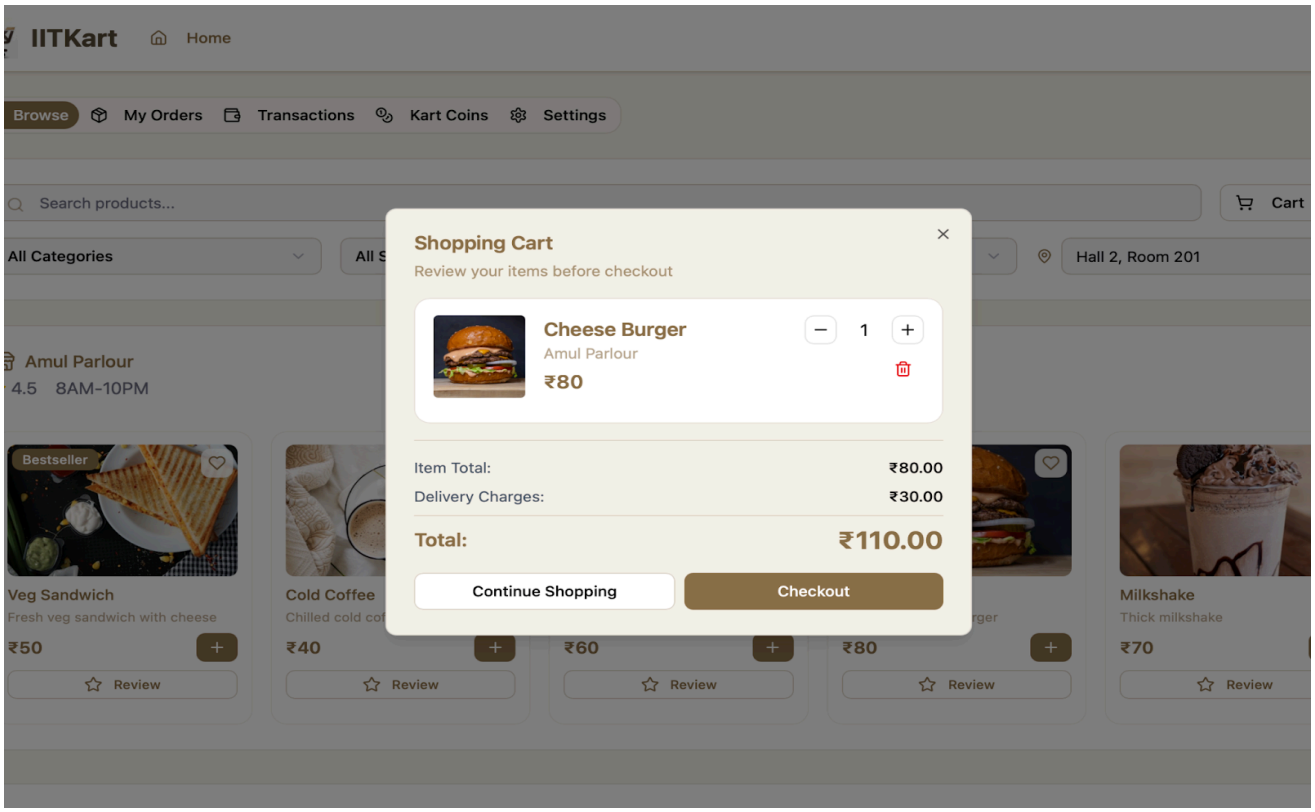


User Module

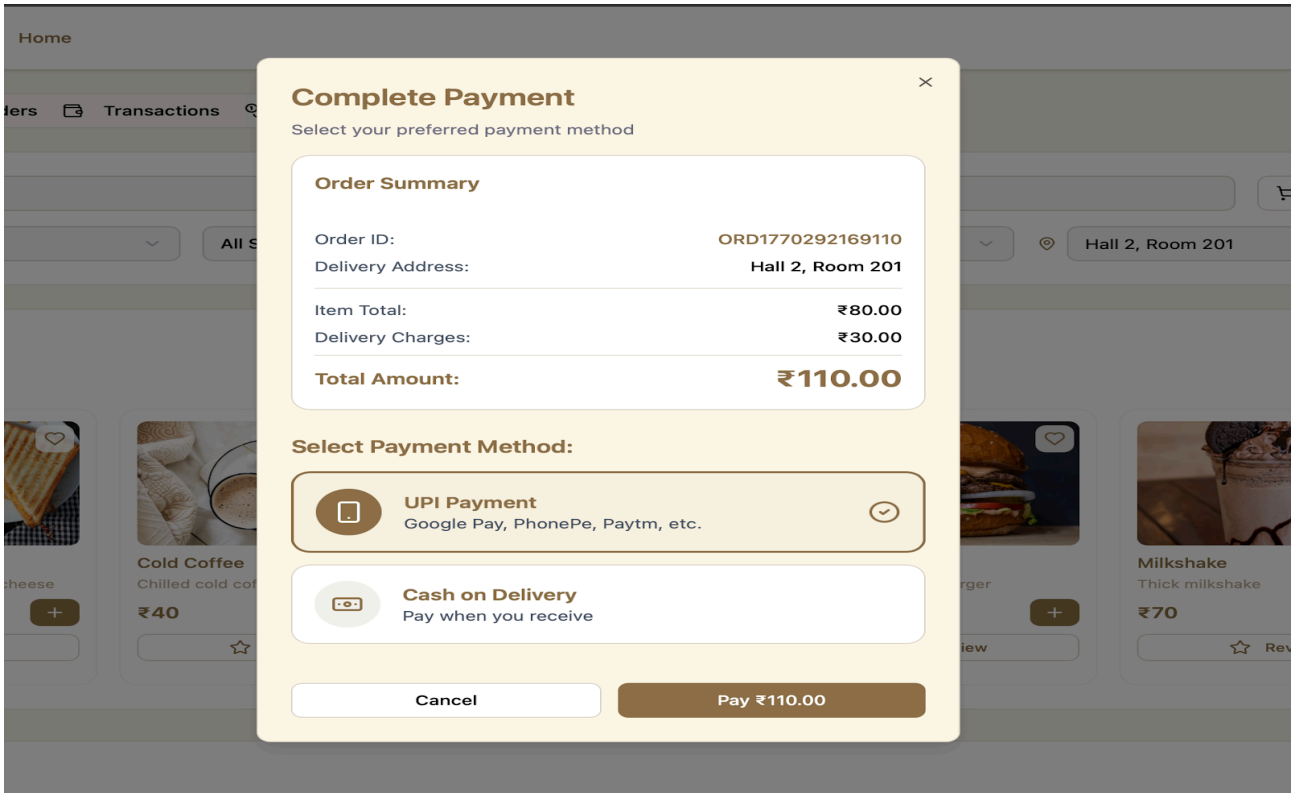
- **Home Page**



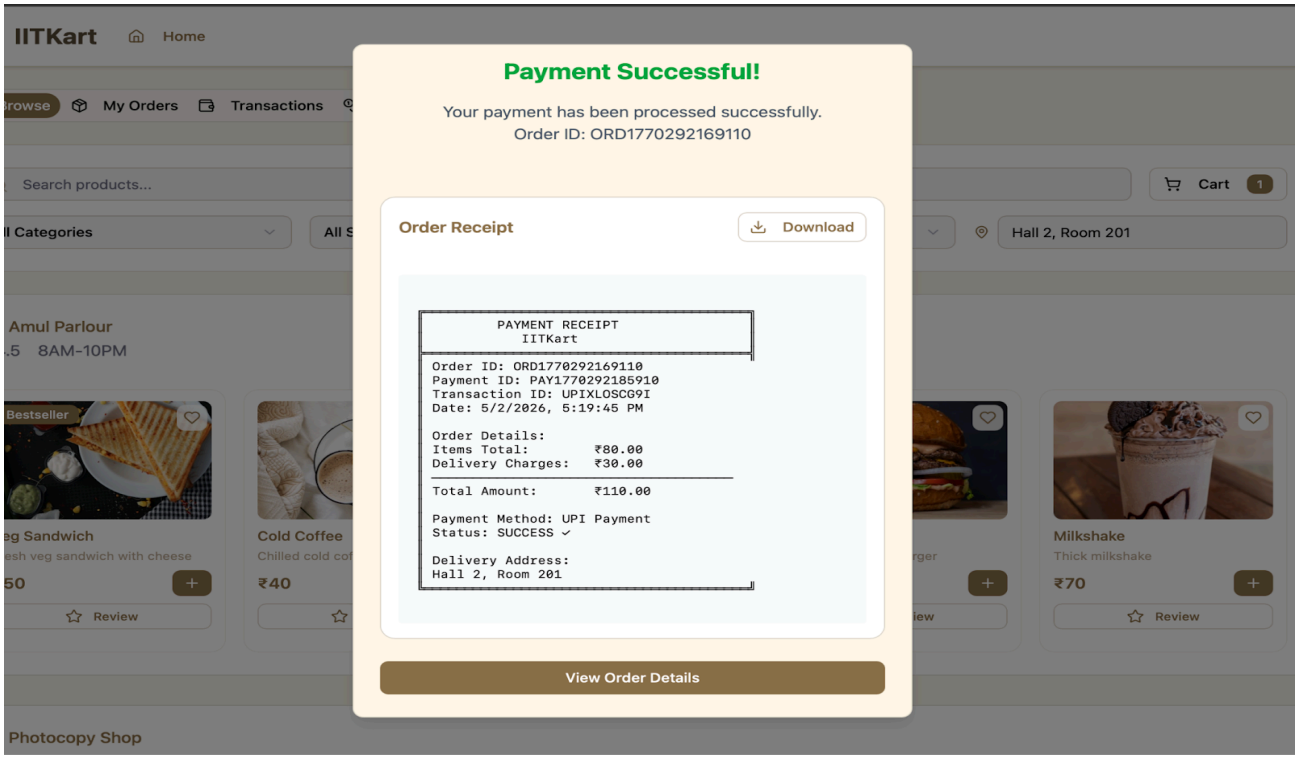
- **Kart**



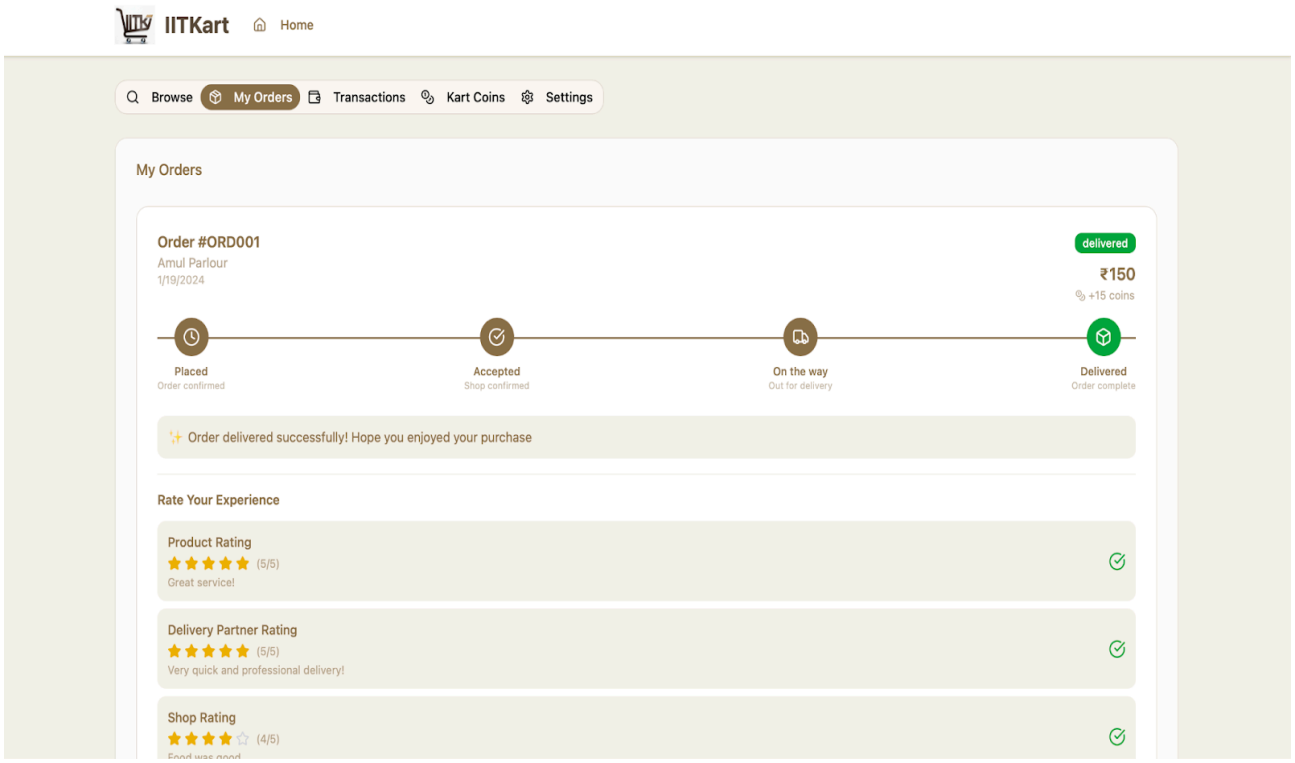
- **Checkout And Payment Selection**



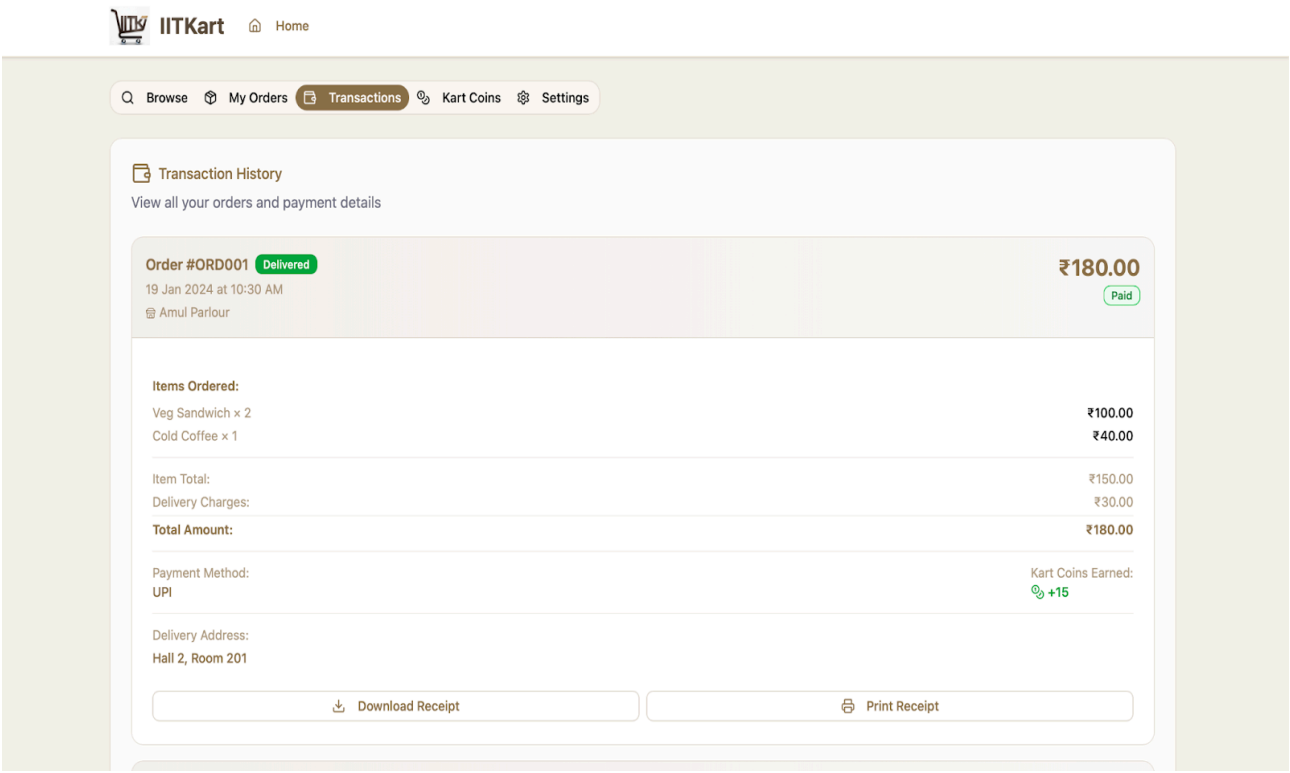
- **Confirmation Receipt**



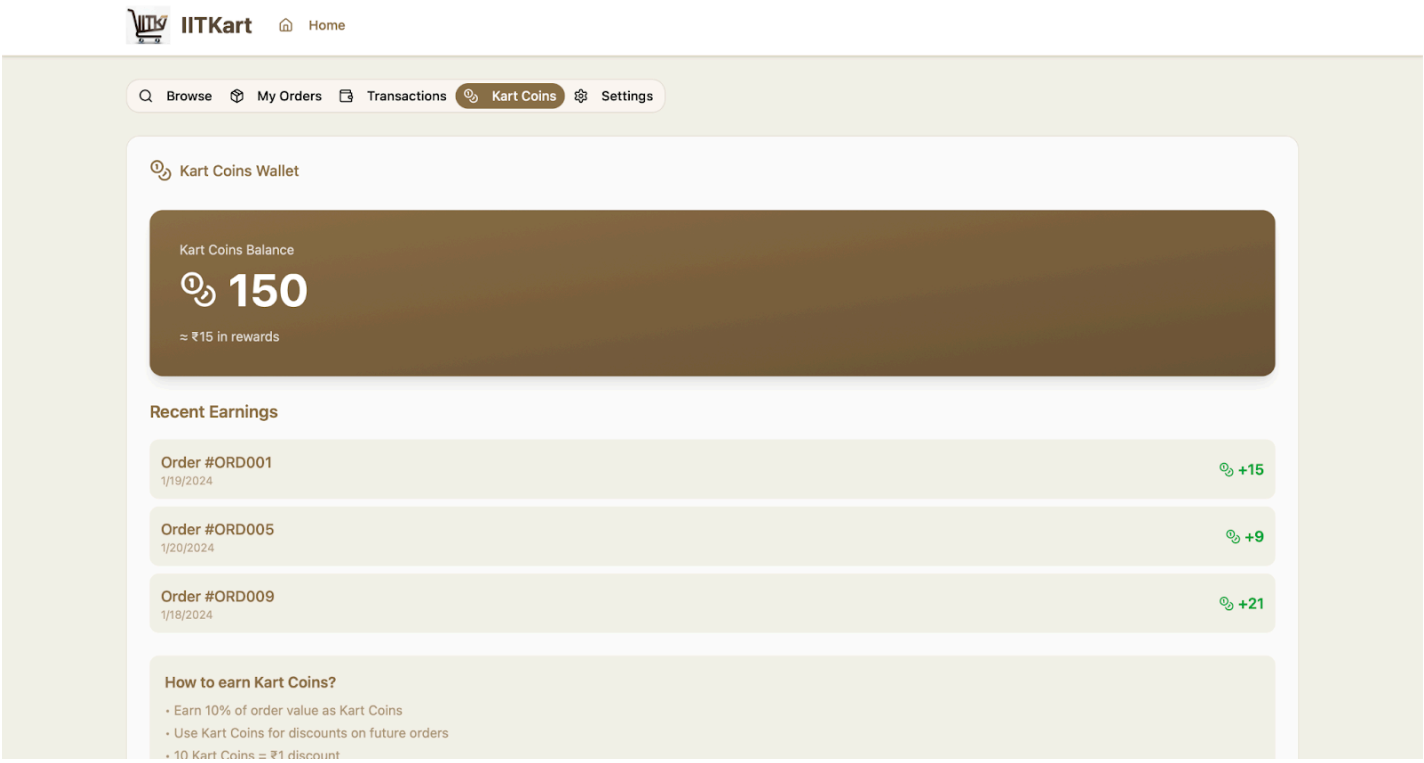
- **Order History**



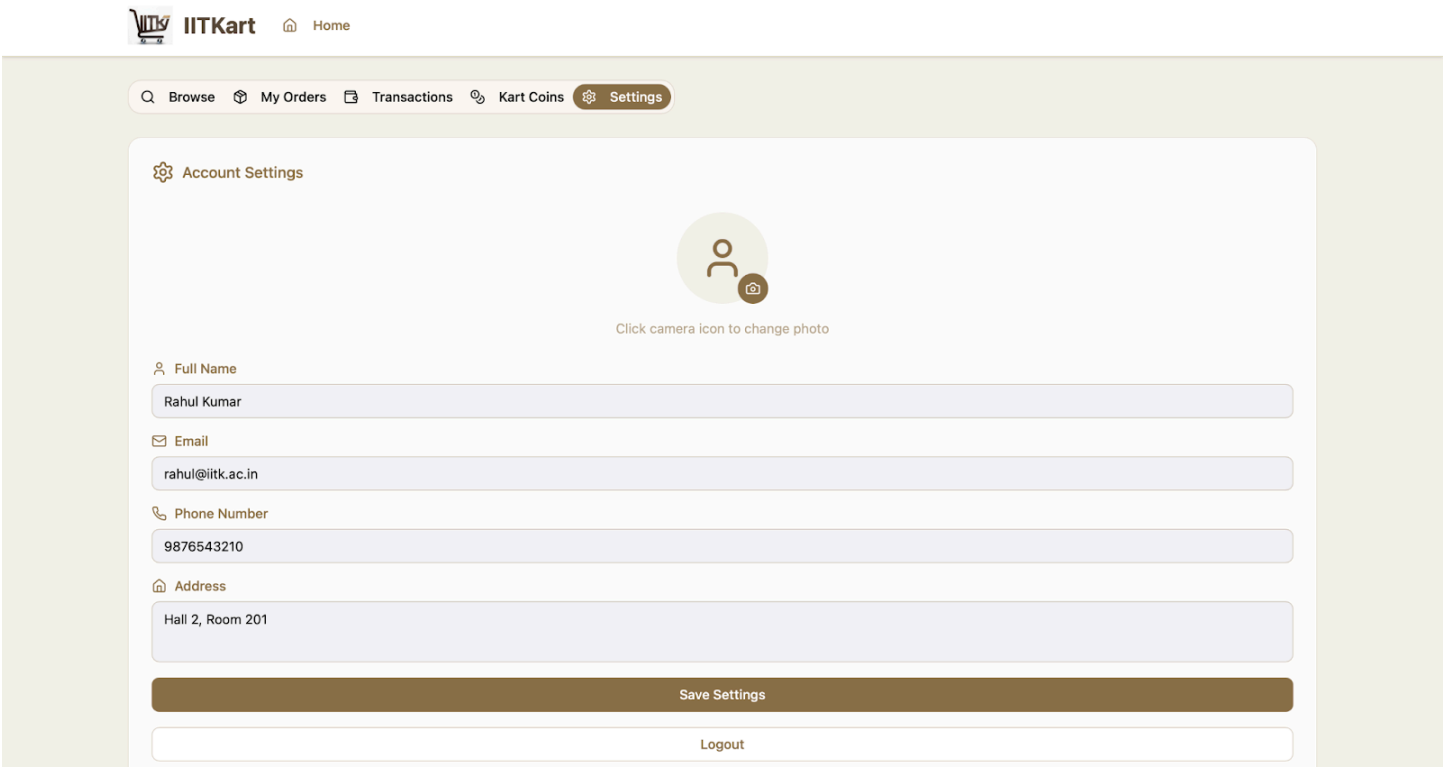
- **Transaction Records**



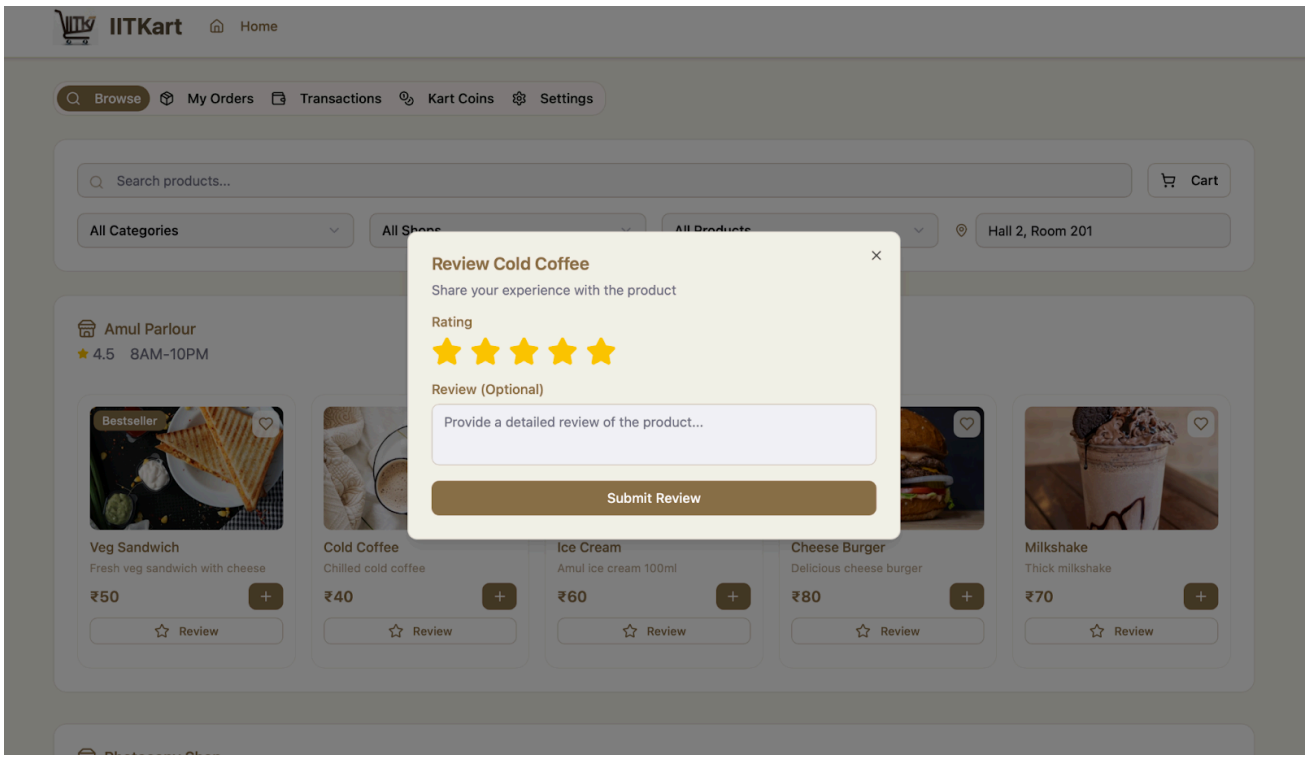
- **Kart Coins(wallet)**



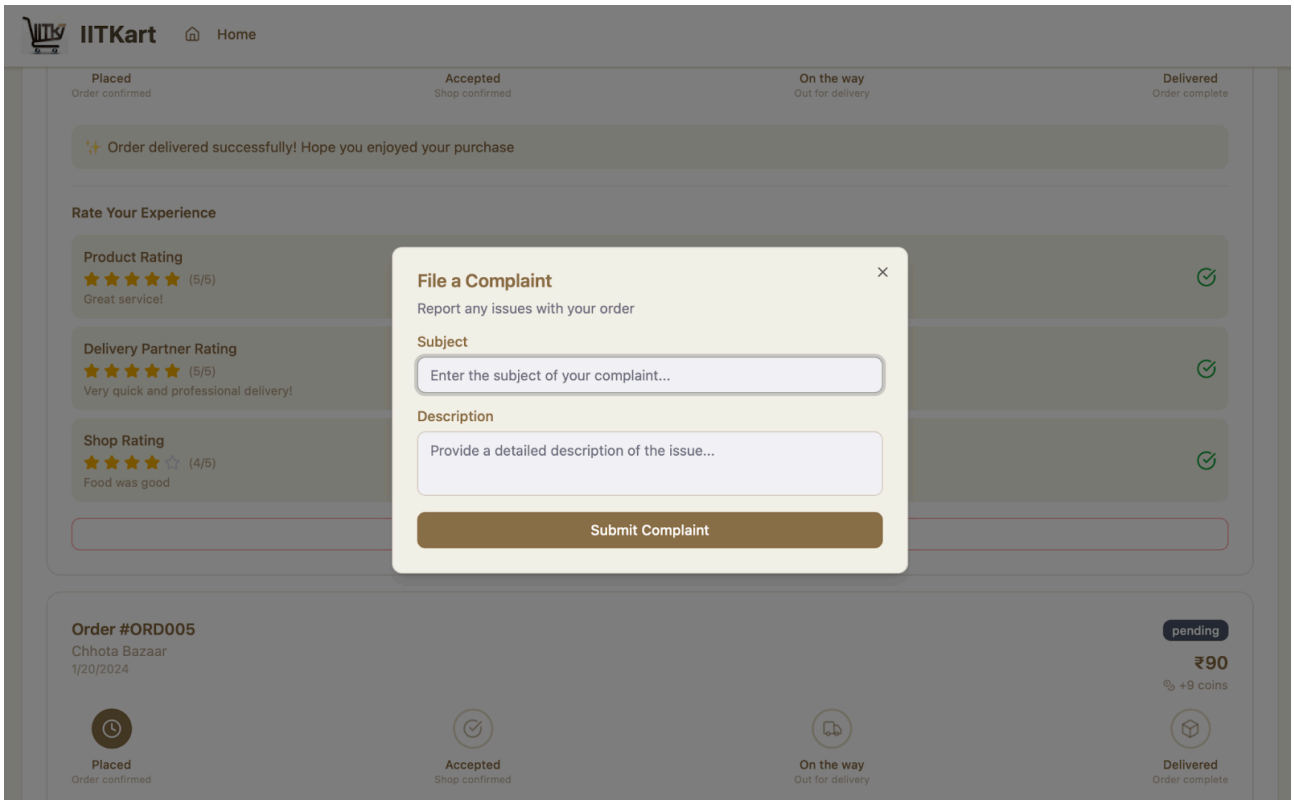
- **Account Settings**



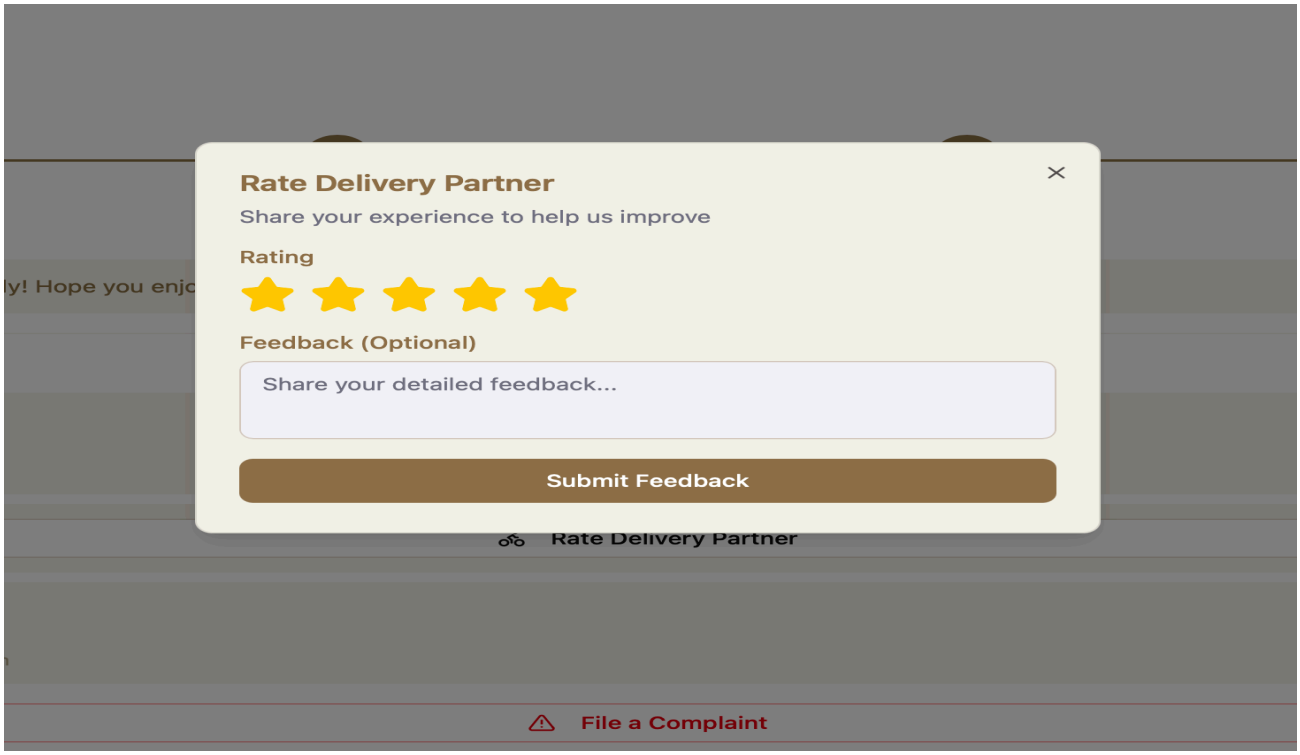
- **Rate and Review Order**



- **File complaint**

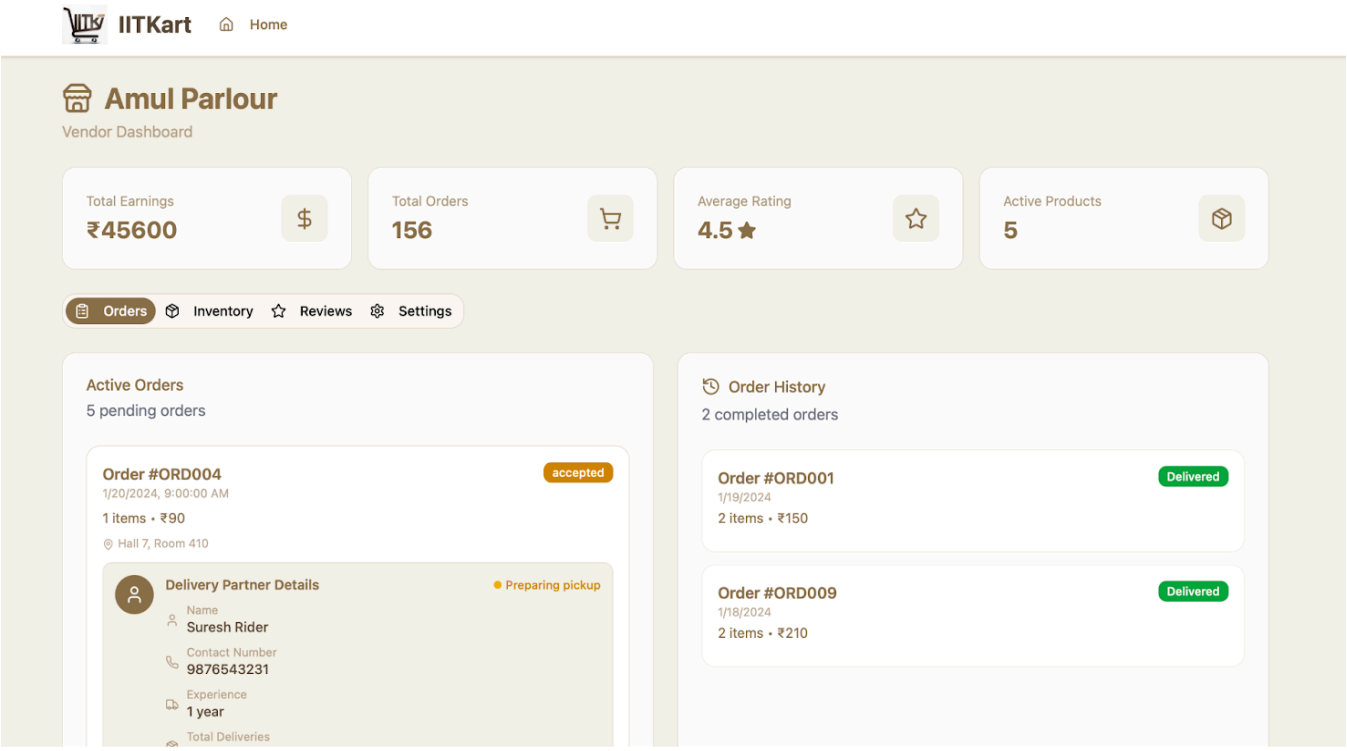


- **Rate Delivery Partner**

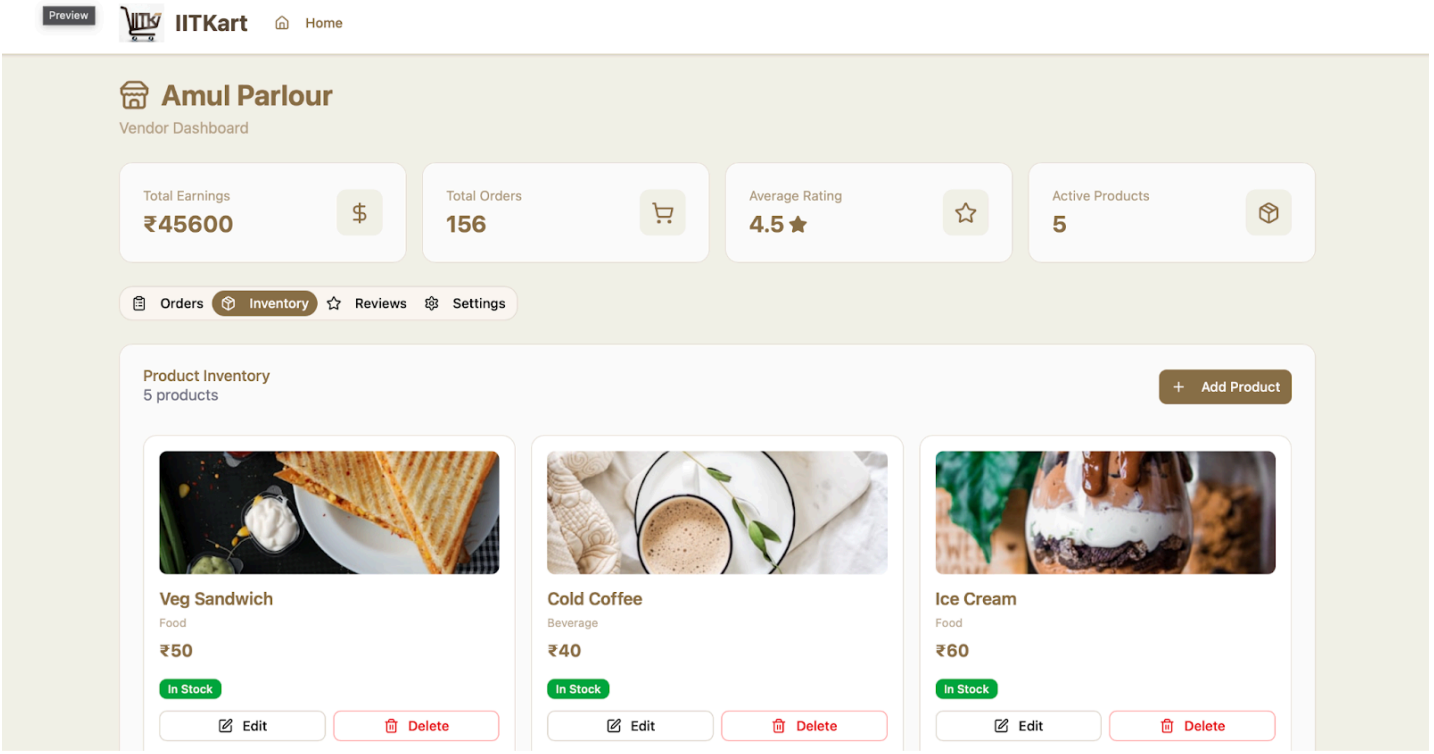


Vendor Module

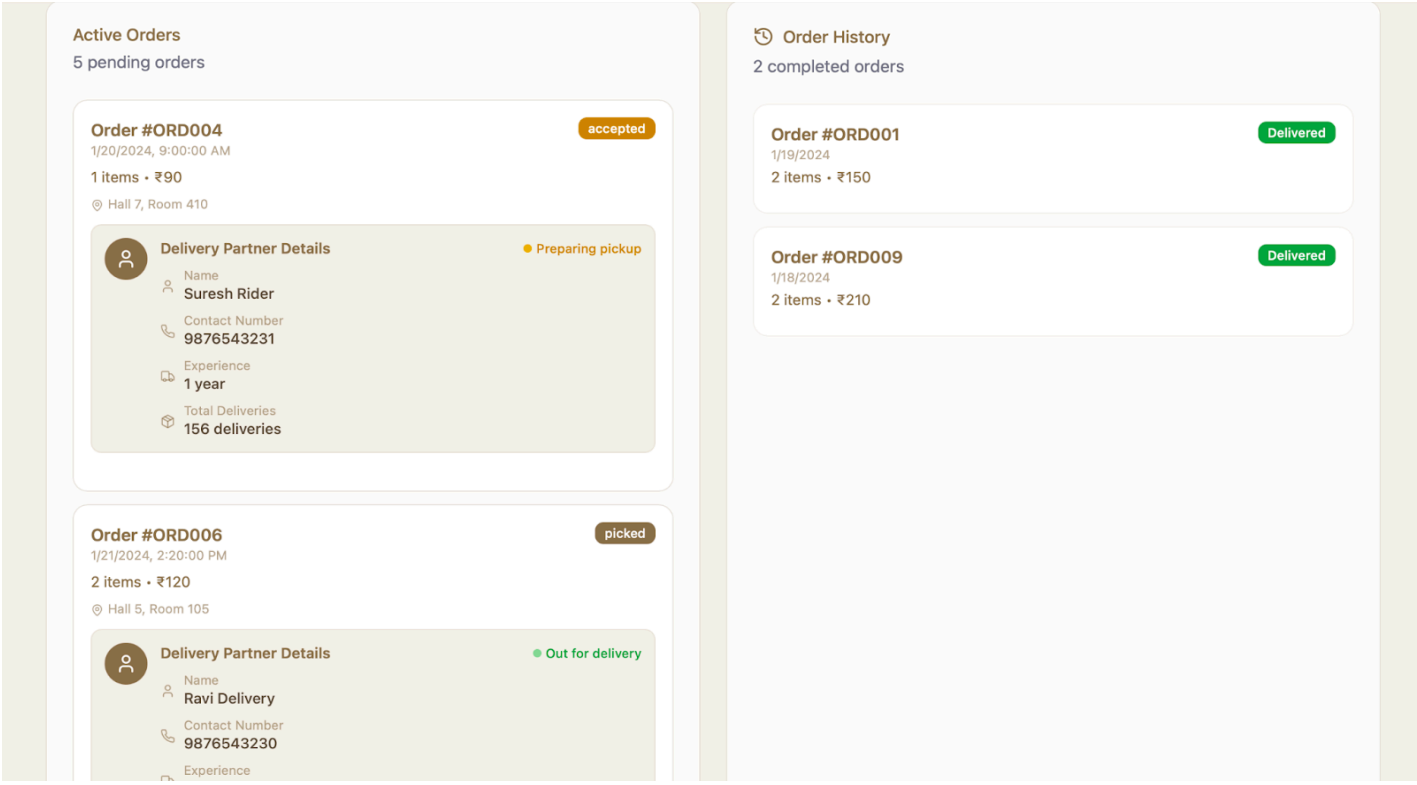
- **Vendor Dashboard**



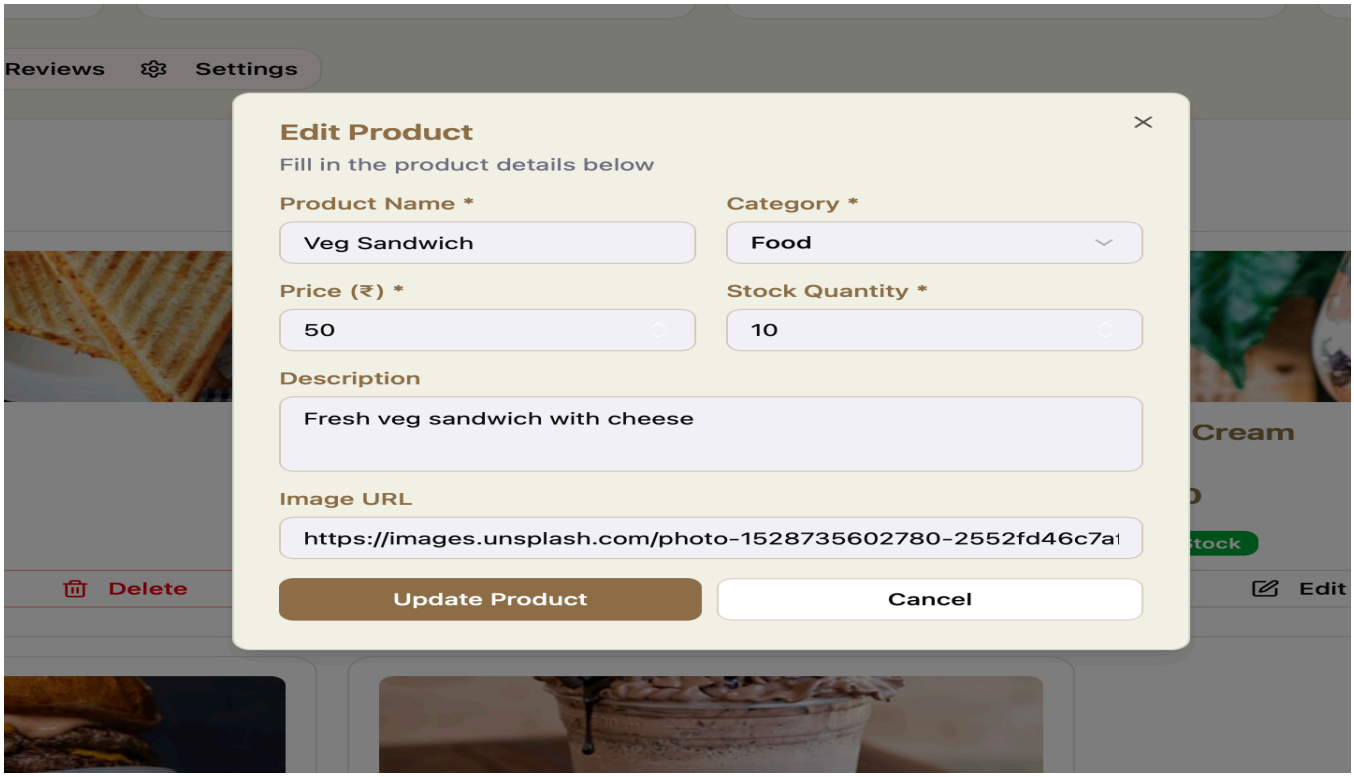
- **Inventory(Products Management Page)**



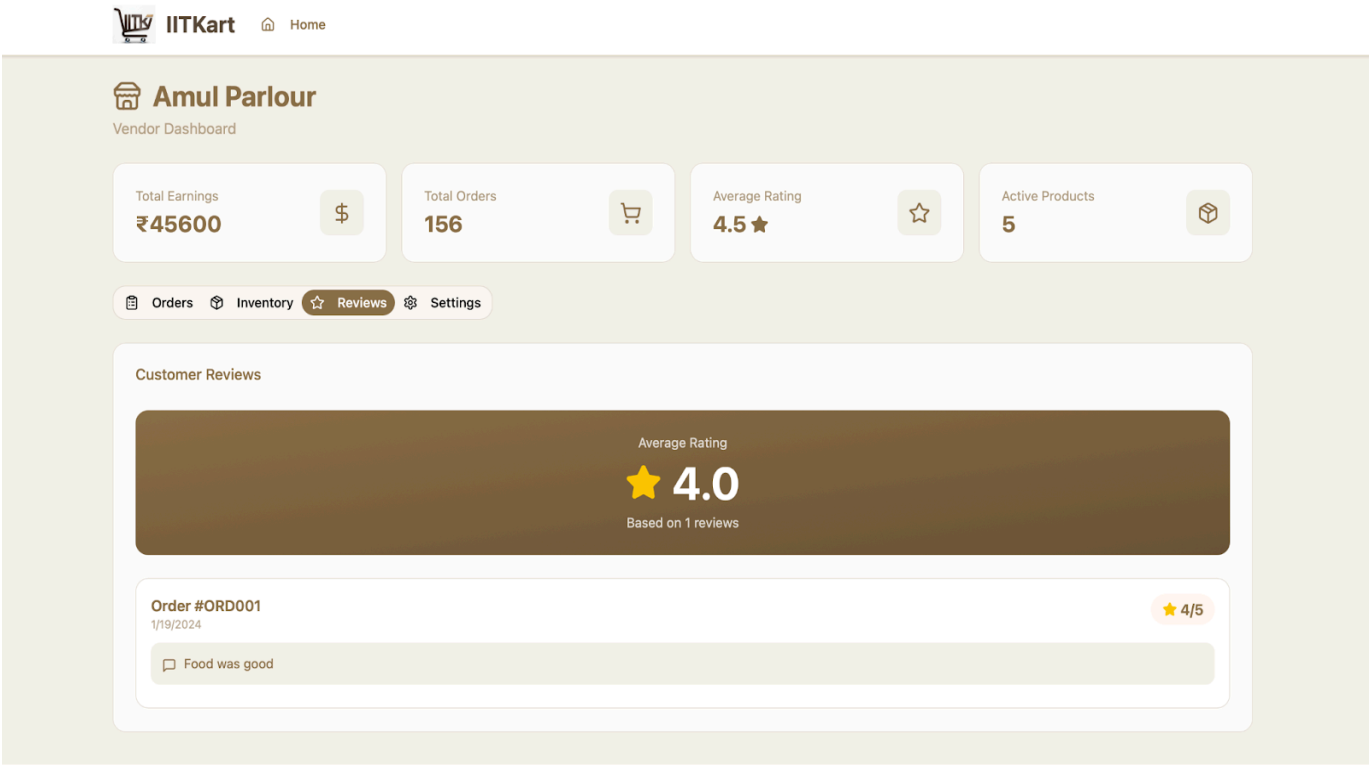
- **Active Order and Order History**



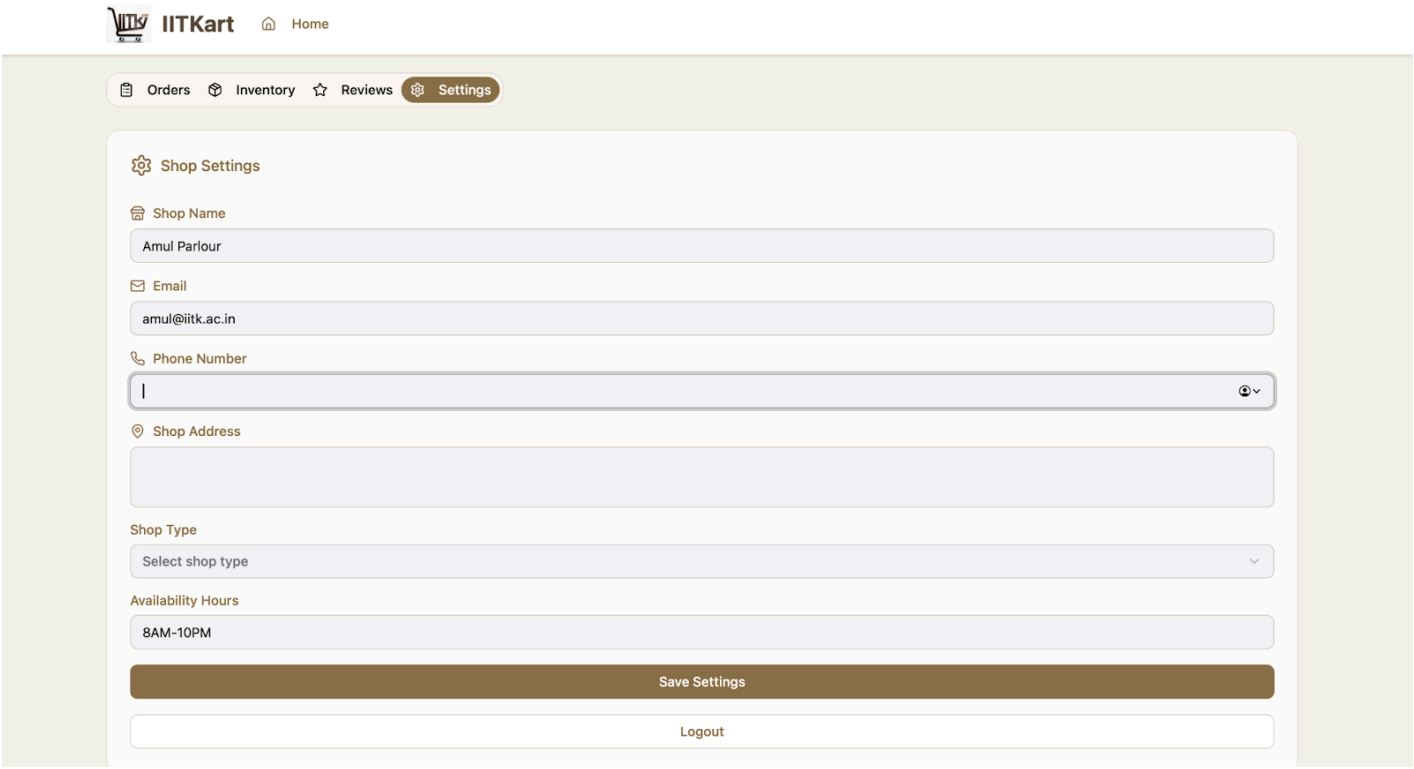
- **Adding and Editing Inventory**



- **Review and feedback Page**

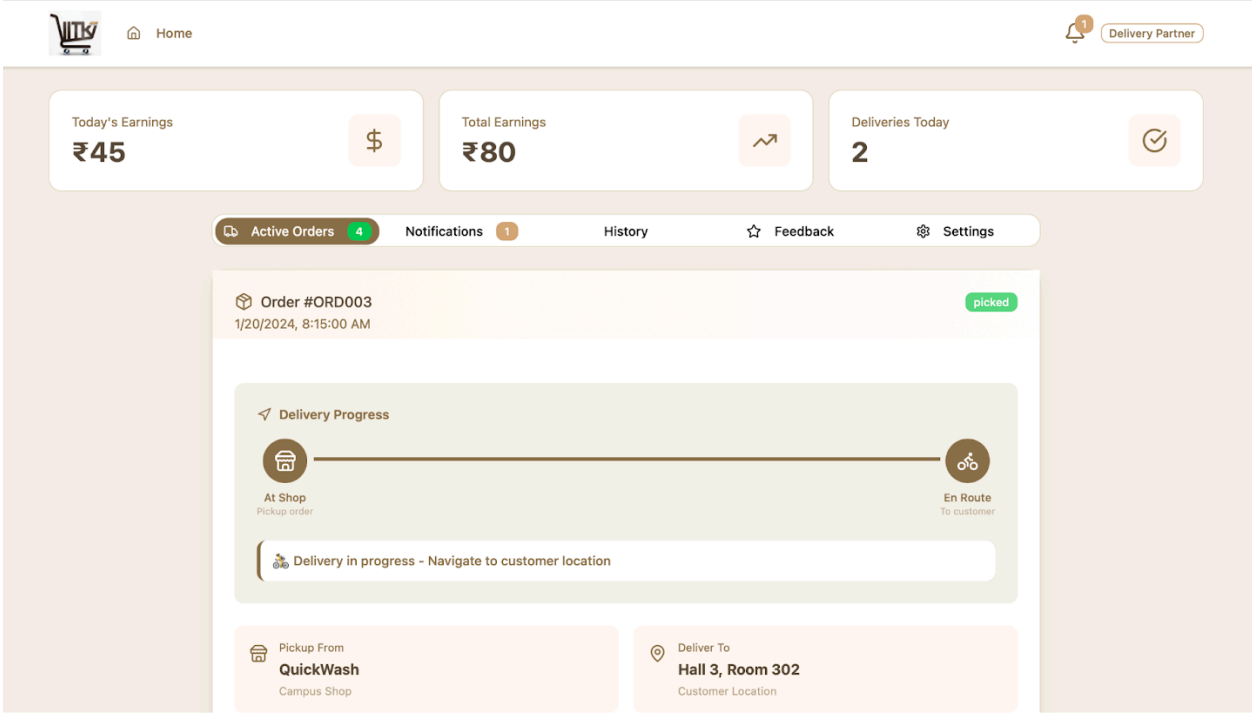


- **Shop Profile**

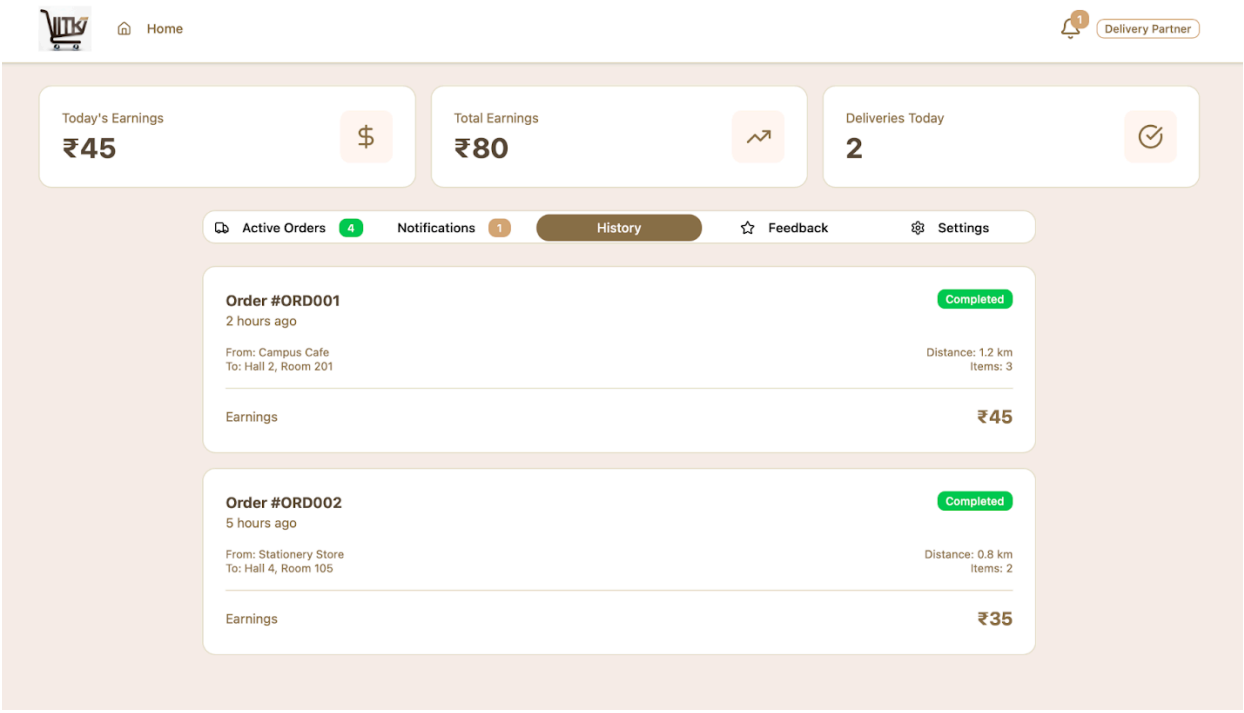


Rider(Courier) Module

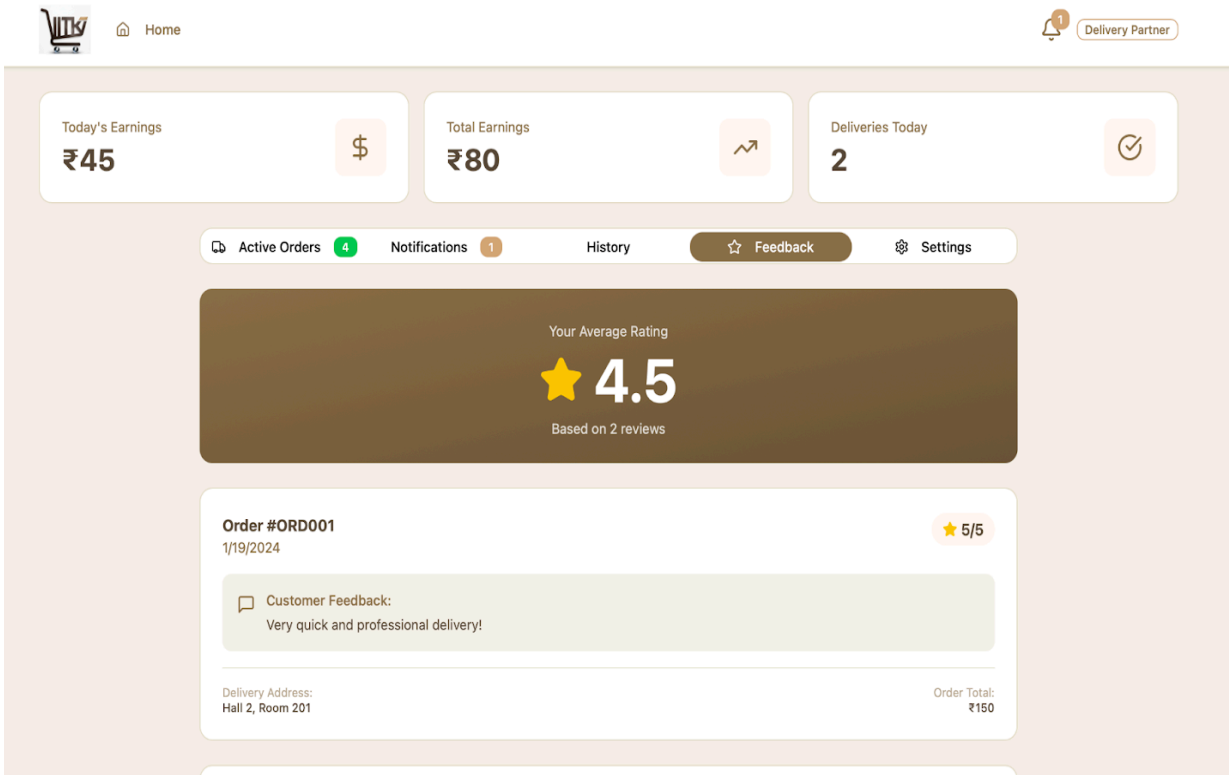
- **Rider Dashboard**



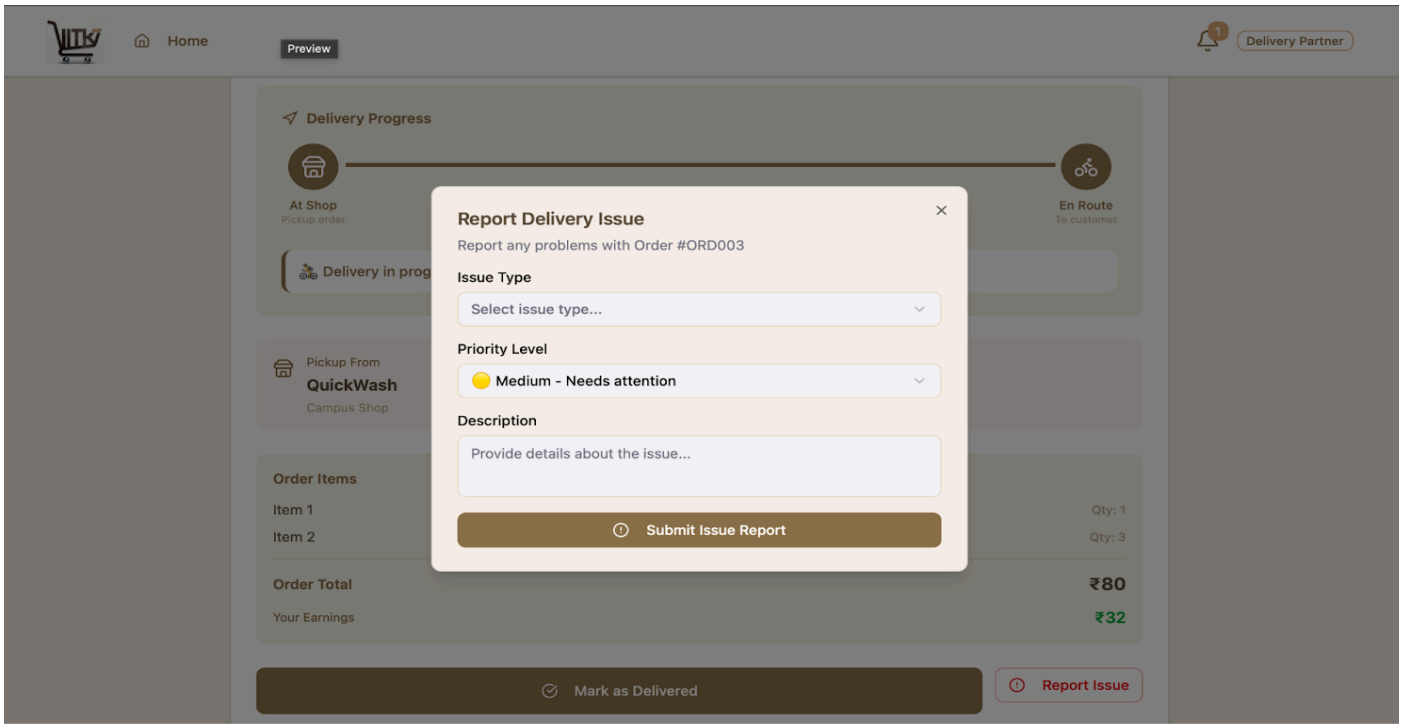
- **Order History Page**



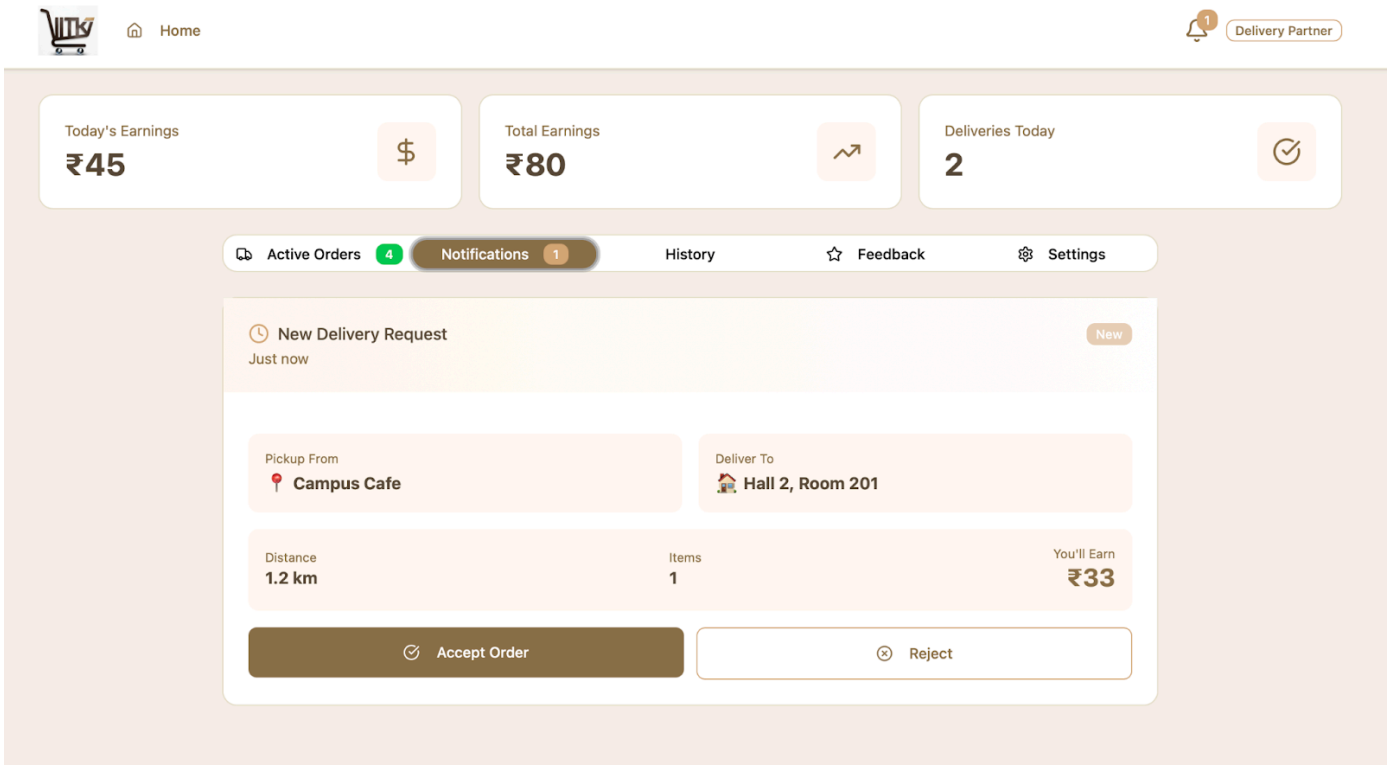
- **Reviews and Feedback Page**



- **Report issue and Emergency**

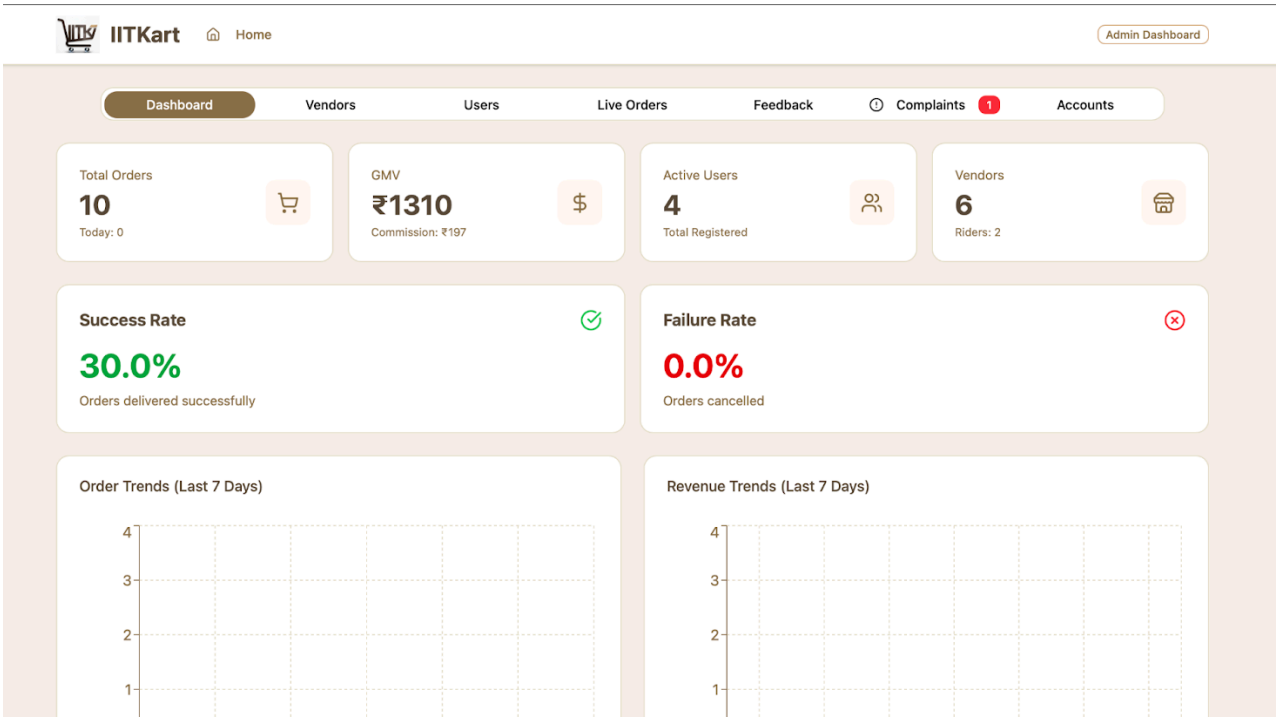


- **Notifications Page**

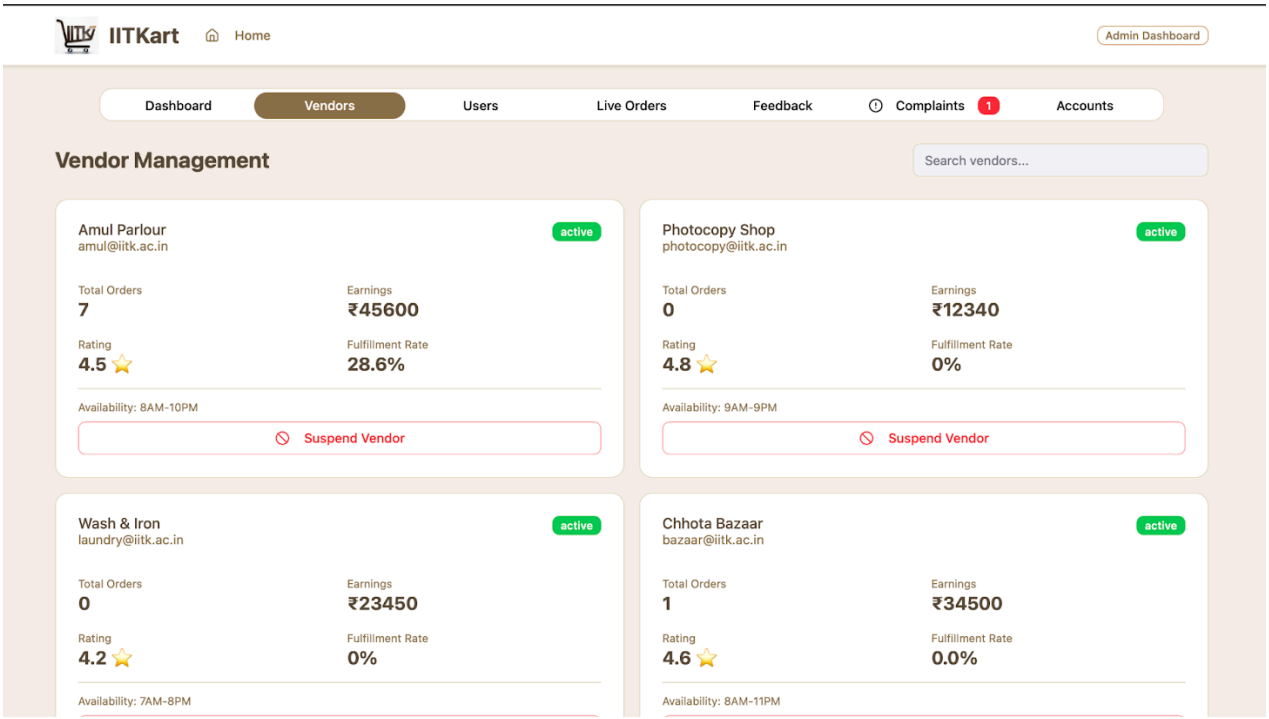


Admin Module

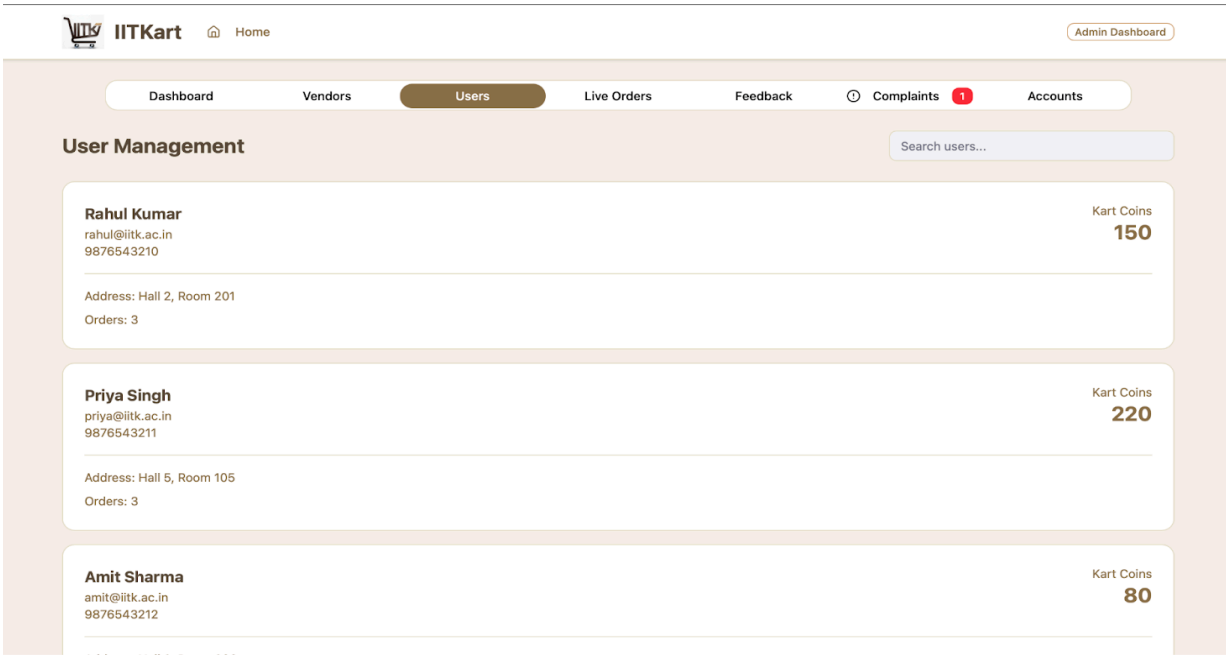
- **Admin Dashboard**



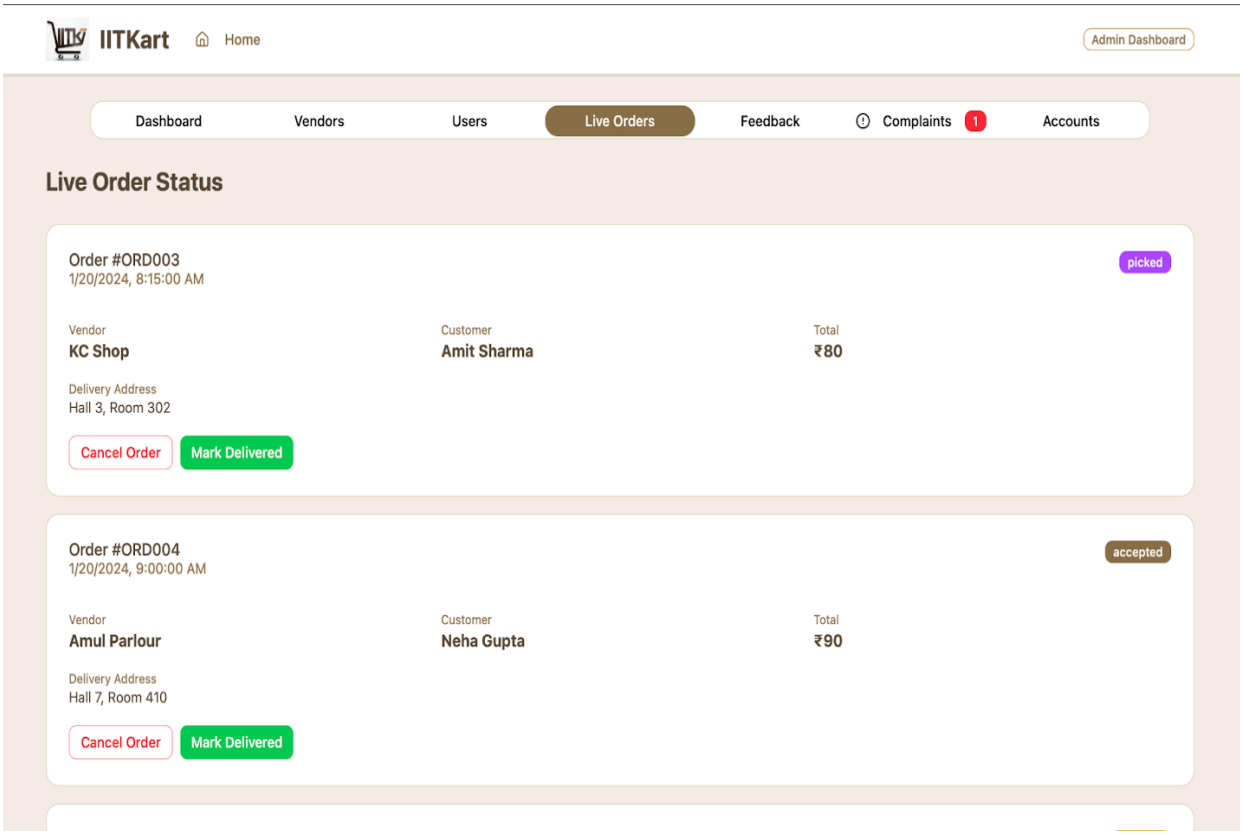
- **Vendor Management Page**



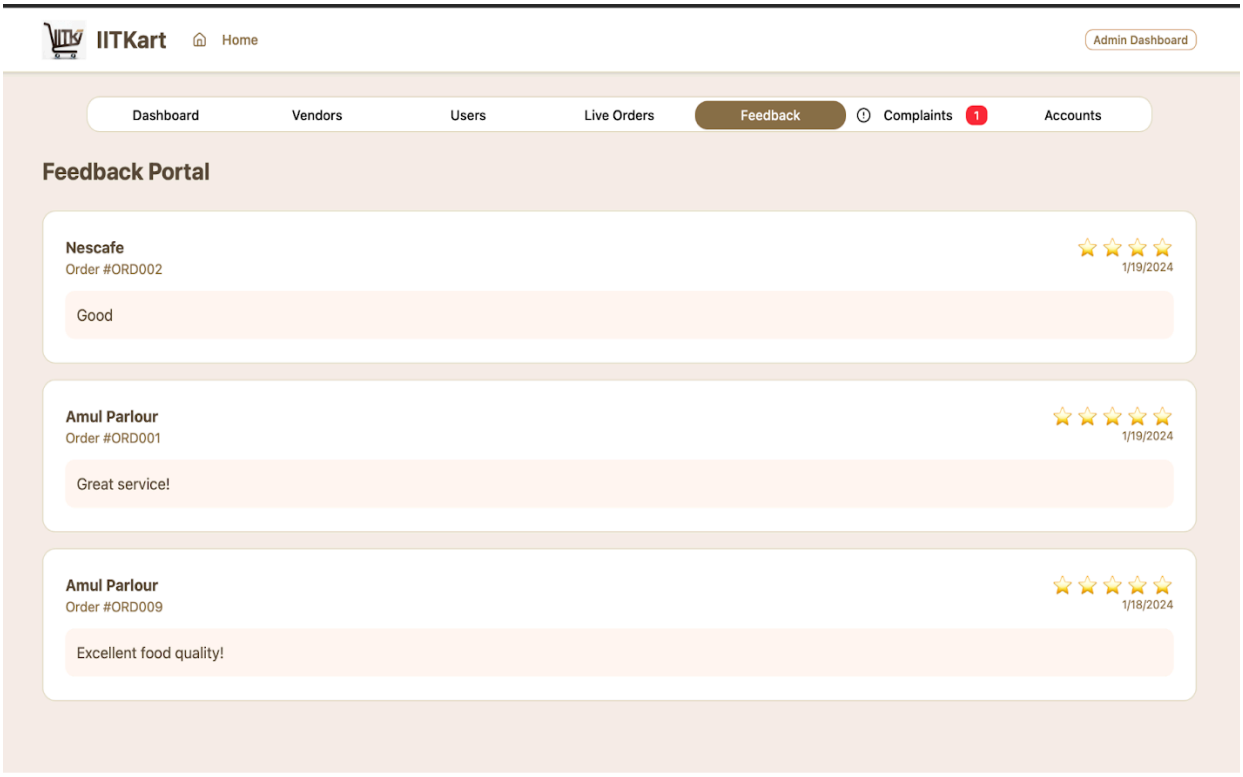
- **User Management Page**



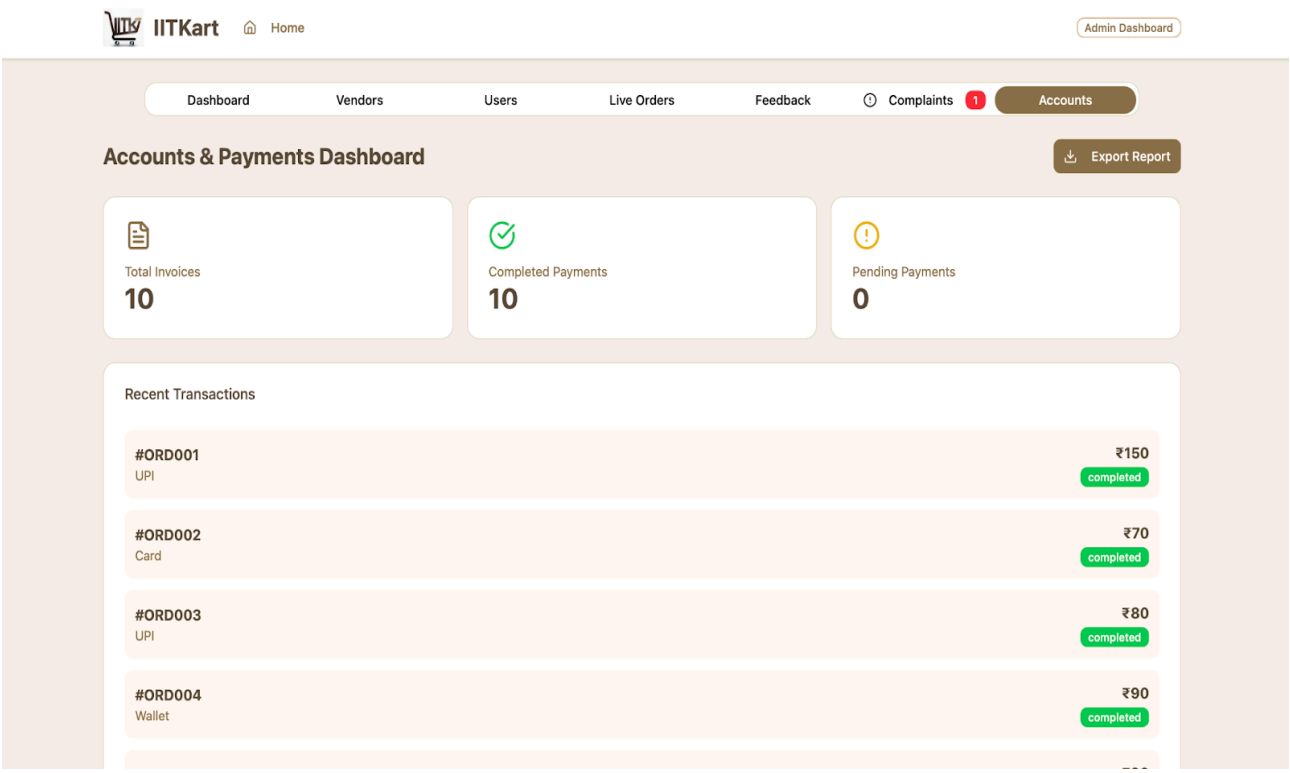
- **Order Status Page**



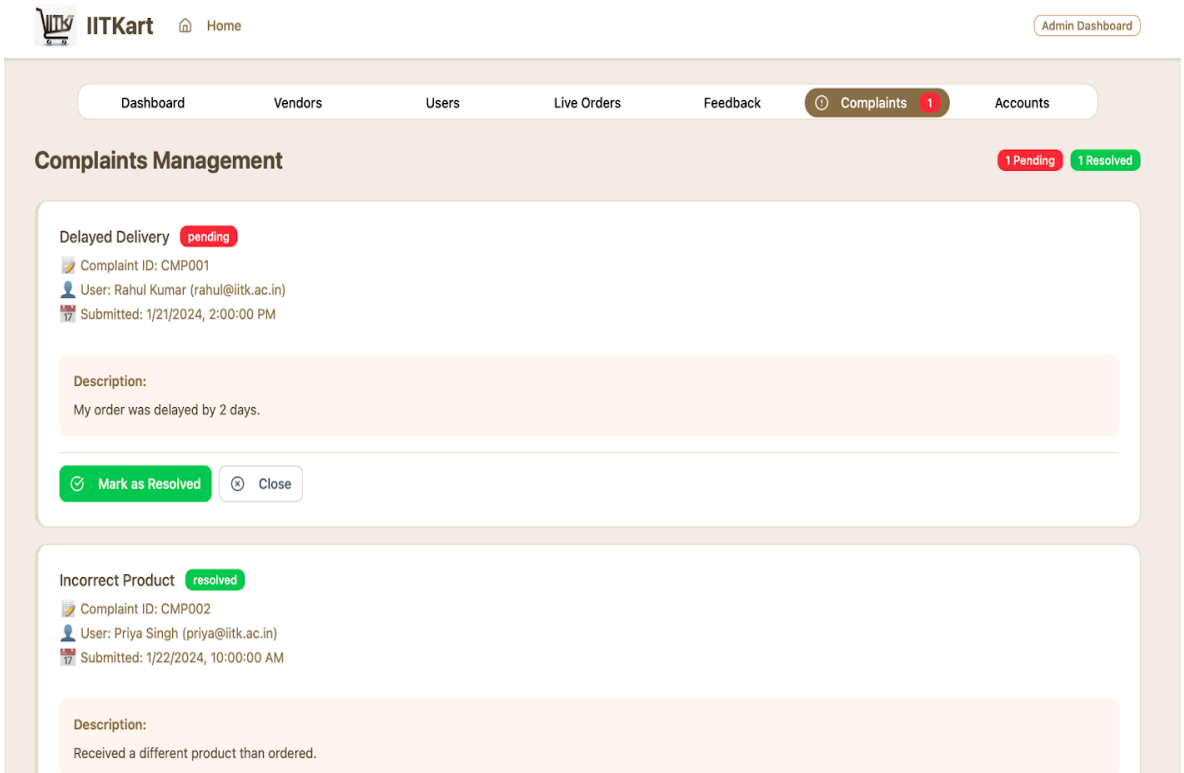
- **Feedback Page**



- **Accounts and Payment Dashboard**

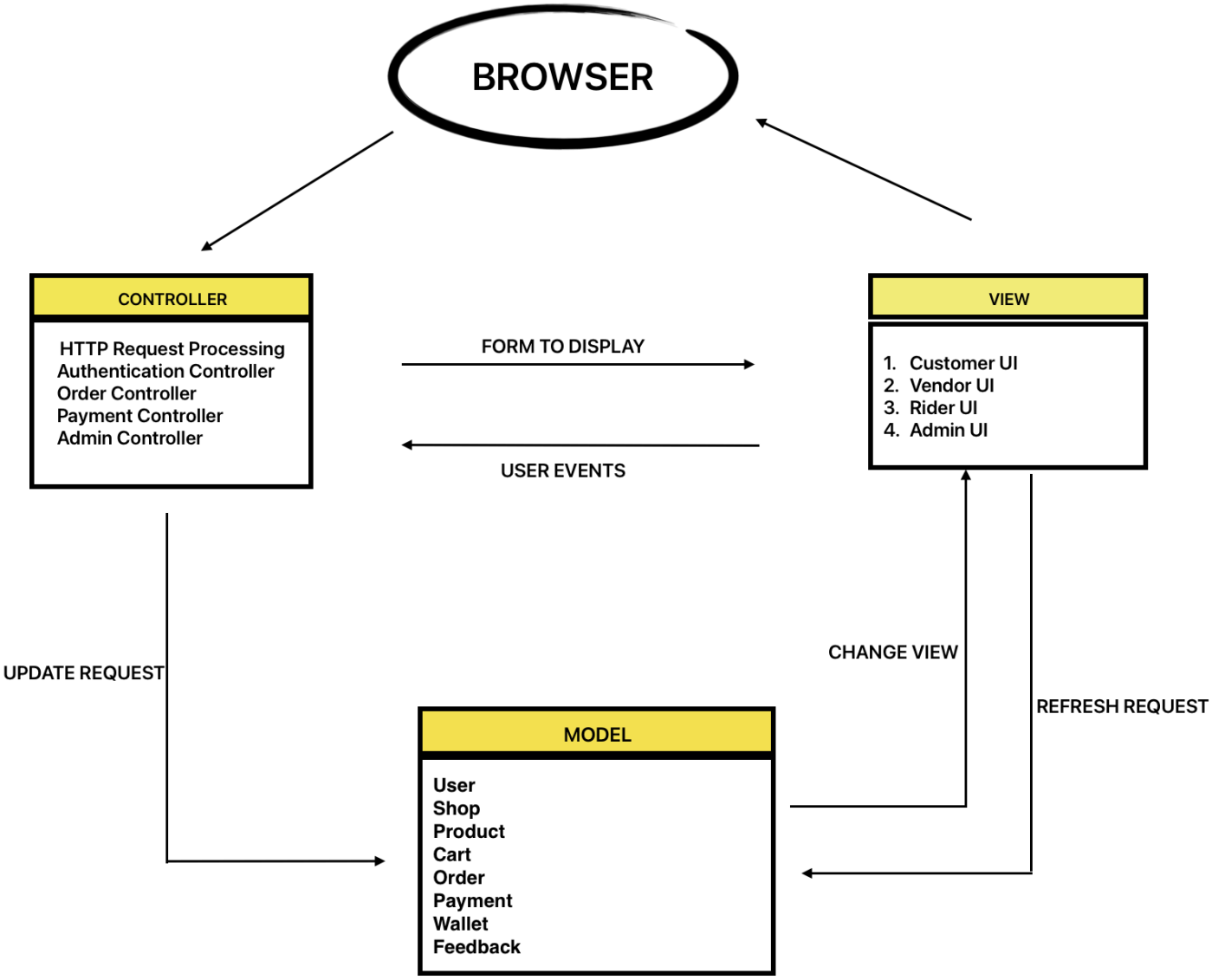


- **Complaint Management**



2 Architecture Design

The application intends to be a web-application, intended for use in modern browsers,offering a clean frontend to be intended for use in the browser, connecting to the backend,powered by servers, to allow for centralized control and processing of data.



The IITKart system adopts the **Model View Controller (MVC)** architectural pattern to structure its overall design. MVC is a widely accepted architectural model that separates the application into three distinct components based on responsibility: presentation(View), control logic(Controller), and data/business logic(Model). This separation enables better organization of the system, improves maintainability, and supports scalable development for applications involving multiple user roles and complex workflows.

IITKart serves four categories of users, namely, customers, vendors, riders, and administrators ,each with distinct interaction patterns but shared business logic.Thus MVC provides a natural and effective way to structure the system.

Overview

At a high level, the system operates as follows:

1. Users interact with the system through role-specific user interfaces.
2. User actions are captured and forwarded to the appropriate controllers.
3. Controllers interpret these actions and perform relevant operations on the model.
4. The model updates or retrieves system state according to business rules.
5. Processed data is returned to the controller, which determines the appropriate view to be rendered.
6. The view presents updated information back to the user.

This interaction flow ensures that each architectural component remains focused on its designated responsibility.

View Layer

- Responsible for all **presentation and user interface** aspects of the system
- Provides **role-specific interfaces** for Customers , Vendors ,Delivery Partners (Riders) and Administrators
- Displays system data and **collects user input**
- Supports maintainability and flexibility in user interface design

Controller Layer

- Acts as the **central coordination component** of the MVC architecture
- Handles **user requests** originating from the View layer
- Serves as an intermediary between **View and Model**
- Responsibilities include:
    - Interpreting user actions
    - Performing input validation
    - Enforcing access control and authorization
    - Invoking appropriate operations in the Model layer

- Controllers are **functionally organized**, such as:
    - Authentication Controller (login and registration)
    - Order Controller (order placement and tracking)
    - Payment Controller (payment initiation and confirmation)
    - Admin Controller (user management and complaint handling)

- Does **not** store persistent data or implement core business rules
- Centralizes request handling, improving:
    - Modularity
    - Debugging
    - Testing

Model Layer

- Represents the **core domain and business logic** of the IITKart system
- Encapsulates key entities, including:
    - User
    - Shop
    - Product
    - Cart
    - Order
    - Payment
    - Wallet
    - Feedback

- Responsible for:
    - Maintaining system state
    - Enforcing data consistency
    - Validating business constraints (e.g., product availability, order status)
    - Executing core operations such as order creation and loyalty point updates

- Remains **independent of user interface concerns**
- Does not directly interact with users or presentation logic
- Provides a **single source of truth** for system data, ensuring correctness and reliability

Alignment with Non-Functional Requirements

Maintainability

The modular structure of MVC makes the system easier to maintain. Changes to one layer,such as modifying business rules or redesigning user interfaces can be made with minimal impact on other layers.

Scalability

As IITKart evolves, new user roles, features, or interfaces can be added by extending Views and Controllers while reusing existing Model logic. This supports both functional growth and increased system usage.

Security

Sensitive operations and data are confined to the Model layer, while authentication and authorization checks are enforced at the Controller level. This reduces exposure of critical logic and helps maintain secure access control.

Testability

MVC improves testability by allowing individual layers to be tested independently. Business logic in the Model can be unit-tested without user interfaces, while Controllers can be tested for correct request handling.

Performance

By keeping Views lightweight and assigning computation heavy tasks to the Model, the system avoids unnecessary processing at the presentation level, contributing to efficient request handling.

3 Object Oriented Design

3.1 Use Case Diagrams

3.1.1 U1- Stakeholder Onboarding and Profile Management

Author - Sneha Kumari

Purpose - To provide a unified entry point for all campus stakeholders (Students, Faculty, Vendors, and Riders) to establish identity, verify credentials via OTP, and maintain profile-specific metadata such as delivery addresses or shop locations.

Requirements Traceability - Users shall be able to register using mobile/email, verify via OTP, and manage personal account information including saved addresses and shop settings.

Priority - High. Essential for Role-Based Access Control (RBAC) and system security.

Preconditions - The user must have access to a valid email or mobile number. Riders/Vendors must have valid identification documents (Aadhaar/PAN/GSTIN) for verification.

Post conditions - A secure account is created; the user is redirected to a role-specific dashboard (Consumer, Vendor, Rider, or Admin).

Actors - User(Consumer, Vendor, Rider, Admin).

Exceptions -

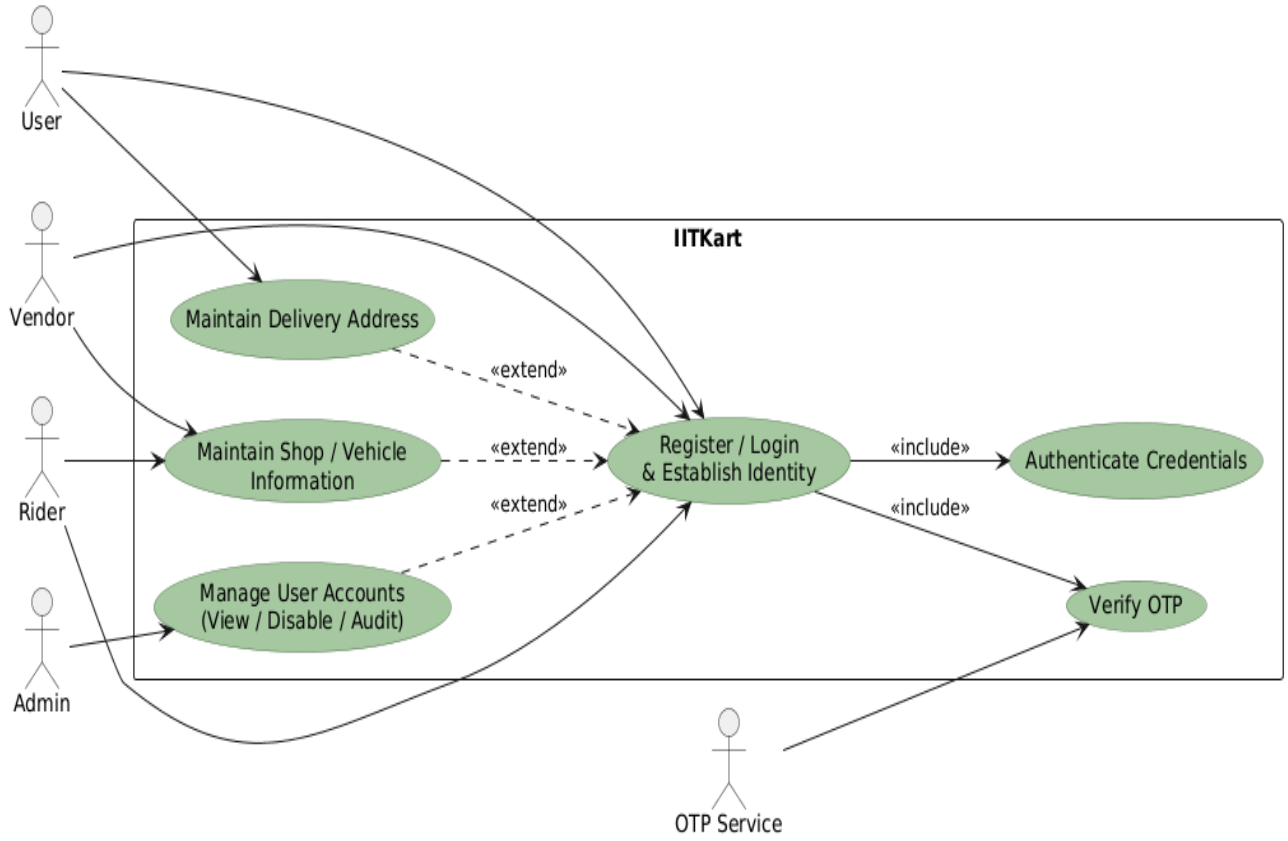
E1: OTP verification failure;

E2: Duplicate registration;

E3: Invalid document upload format.

Includes - User Authentication.

Notes/Issues - Sensitive operations like password changes require hashing. Profile picture uploads are planned for future versions.



3.1.2 U2- Product Discovery and Recommendation Engine

Author - Sneha Kumari

Purpose - To facilitate the discovery of campus goods and services through keyword searching, category-based filtering, and a recommendation mechanism driven by user activity and history.

Requirements Traceability - Users shall be able to browse products and vendors, apply category-based filters (Price, Bestsellers, Availability), and receive recommendations based on prior activity.

Priority - High. Core functionality for the e-commerce discovery phase.

Preconditions - Consumer is successfully logged in. At least one vendor must have "In Stock" inventory listed.

Post conditions - A dynamically sorted list of products or stores is displayed on the UI.

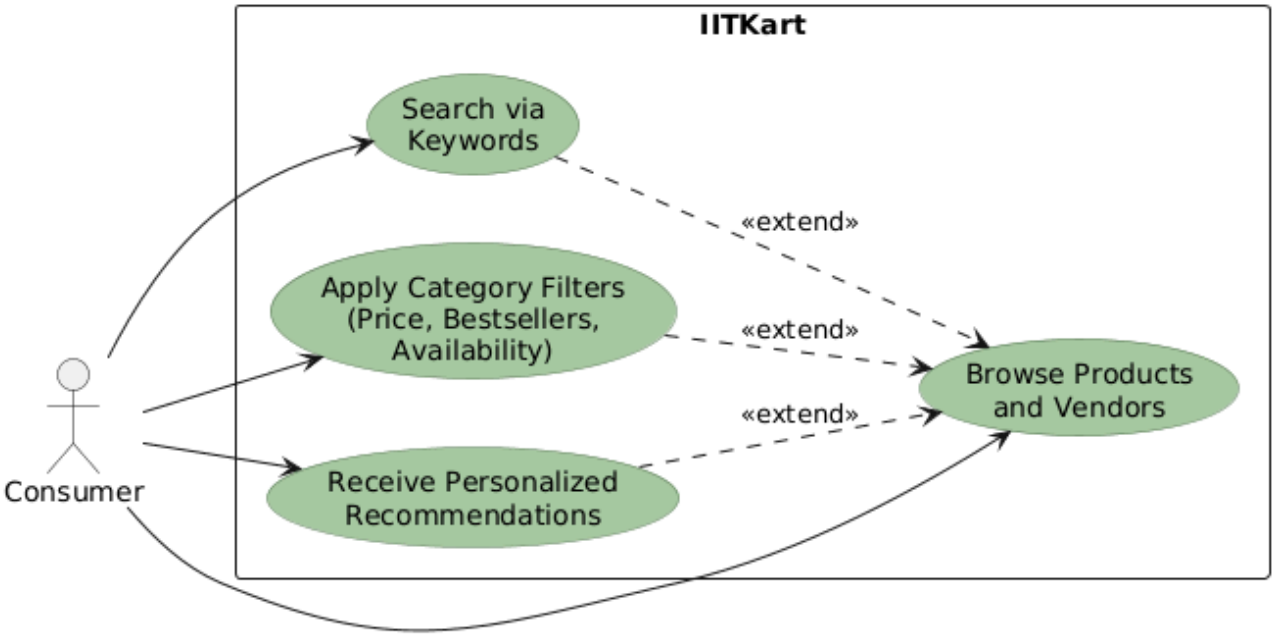
Actors - Consumer

Exceptions -

E1: No matching products or vendors found.

Includes - None

Notes/Issues - Recommendation facilities require the integration of Machine Learning based techniques.



3.1.3 U3- Transactional Workflow and Payment Integration

Author - Sneha Kumari

Purpose - To manage the addition of items to the cart, calculation of total billing (including delivery fees), and secure payment processing via external gateways.

Requirements Traceability - Users shall be able to place orders from the cart and complete online payments using UPI, net banking, or cards through an external gateway.

Priority - High. Critical for order finalization and financial integrity.

Preconditions - Selected items must be in stock. A delivery rider must be available to accept the task.

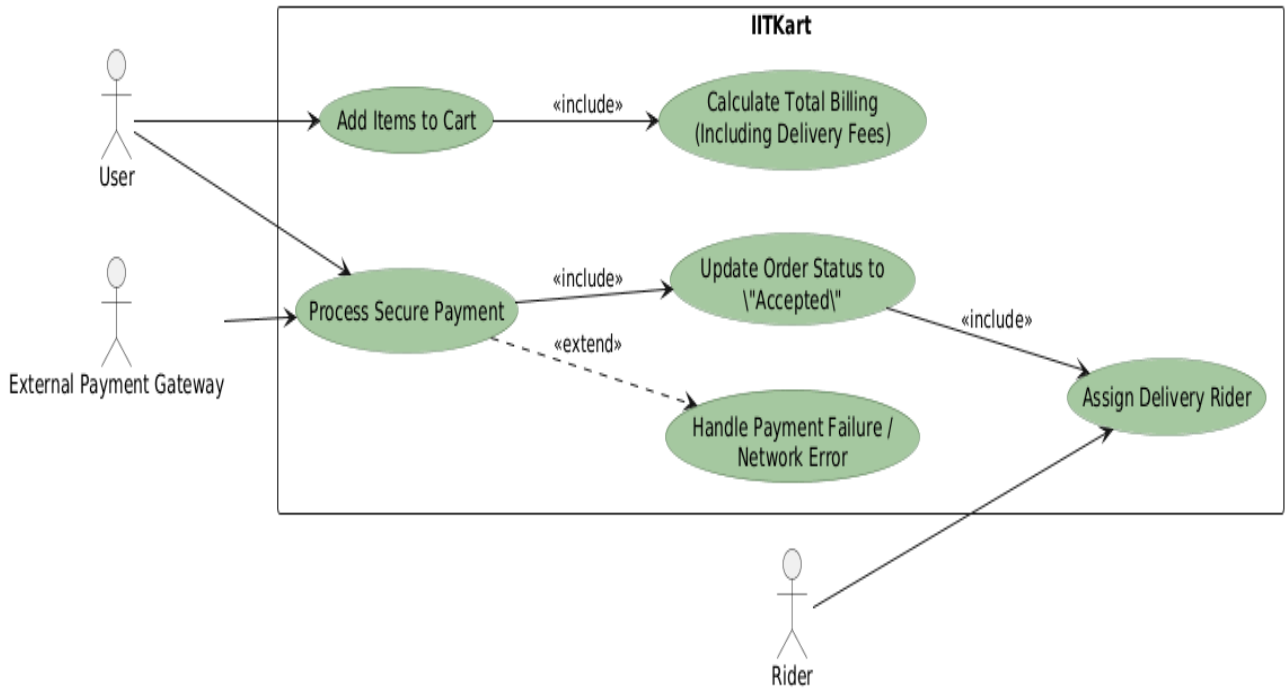
Post conditions - Order status moves to "Accepted". A payment confirmation and transaction record are generated.

Actors - User(Consumer, Vendor, Rider, External Payment Gateway).

Exceptions -
E1: Payment declined by gateway (network failure);
E2: No riders available at the moment.

Includes - Customer completes payment.

Notes/Issues - System uses Prisma ORM for type-safe database queries. Payments must be HTTPS secured.



3.1.4 U4- Vendor Shop and Inventory Operations

Author - Sneha Kumari

Purpose - To empower campus vendors to manage their digital store-front, update product availability, adjust pricing, and monitor real-time business KPIs.

Requirements Traceability -Vendors shall be able to add/edit/delete inventory items, set shop timings/details, and view metrics like total earnings, order volume, and ratings.

Priority - High. Essential for strategic oversight and quality assurance within the campus ecosystem.

Preconditions - Vendor is successfully authenticated. Stores must be identified by a unique `vendorID`.

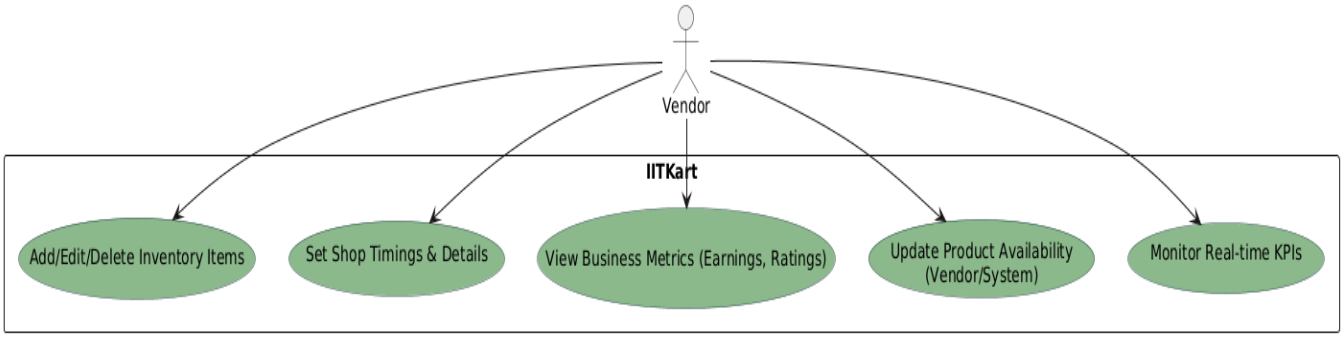
Post conditions - Shop inventory/profile is updated in the database and immediately visible on the customer UI.

Actors - Vendor (Primary), System (Secondary).

Exceptions -
E1: Image upload error;
E2: Non-numeric price input;
E3: Null data state for metrics.

Includes - UC1: User Authentication.

Notes/Issues - Data isolation ensures each vendor sees only their specific metrics upon login.



3.1.5 U5- Real-time Logistics and Order Fulfillment

Author - Sneha Kumari

Purpose - To synchronize the physical delivery process through rider notifications, order acceptance, status tracking, and confirmed handover to the consumer.

Requirements Traceability - Riders shall be able to view active delivery orders, accept/reject delivery requests, and mark orders as "Delivered" upon handover.

Priority - High. Core functionality for delivery execution and managing ongoing logistics.

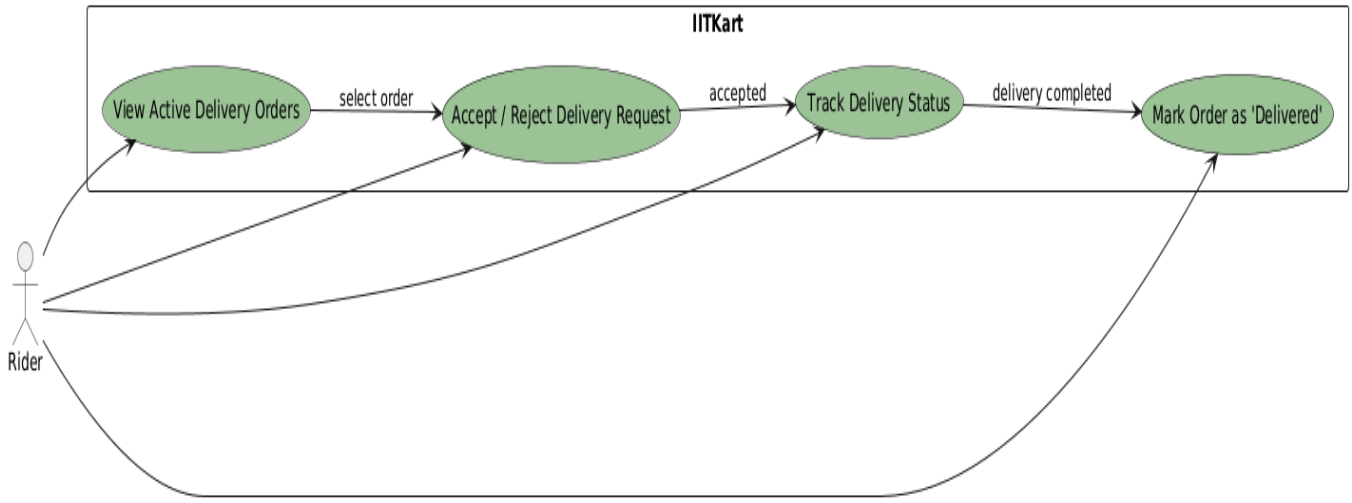
Preconditions - Order is fully paid. Rider is logged in and "Active".

Post conditions - Order status moves to "Delivered". Rider earnings and delivery counts are updated.

Actors - User(Vendor, Rider)

Exceptions -
E1: Order already accepted by another rider;
E2: Rider does not respond in the time window.

Includes - UC2: Customer Places Order; UC3: Customer Completes Payment.



3.1.6 U6- Reputation, Performance, and Loyalty System

Author - Sneha Kumari

Purpose -To manage service quality through multi-party ratings and incentivize usage via the internal "Kart Coins" reward mechanism.

Requirements Traceability - Users shall be able to rate vendors and riders, earn Kart Coins (loyalty rewards) for completed orders, and view performance metrics.

Priority - Medium. Vital for long-term engagement and quality control.

Preconditions - Order must be successfully marked as "Delivered".

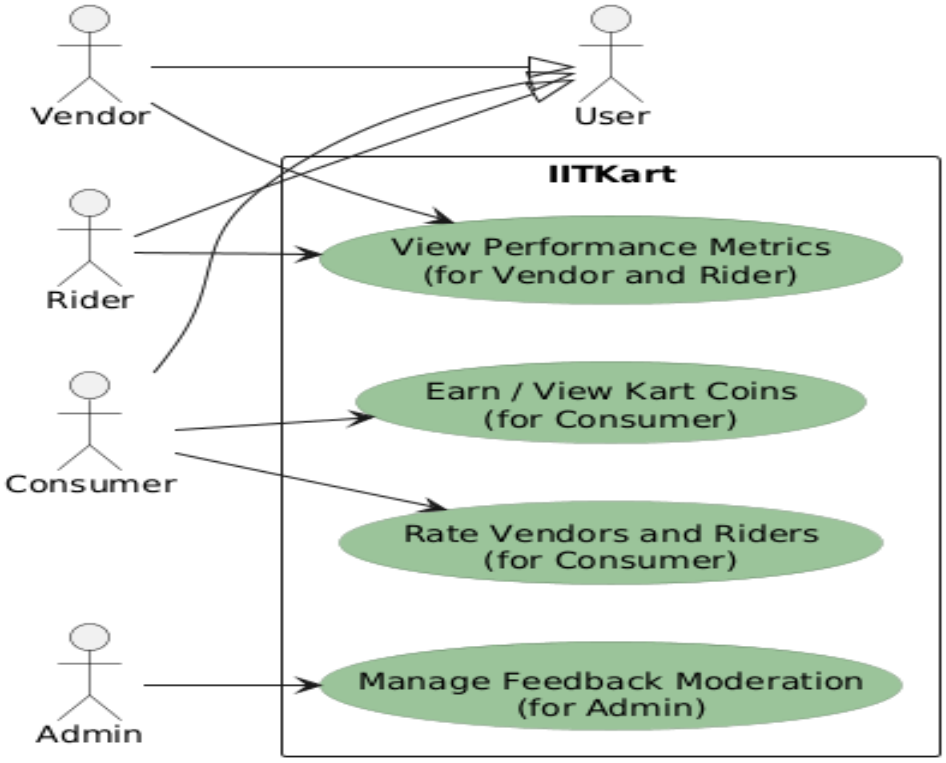
Post conditions -Wallet balance updated. Average ratings updated in the Vendor and Rider profiles.

Actors - User(Consumer, Vendor, Rider and admin).

Exceptions -
E1: No reviews available (display 0.0 rating);
E2: Data sync delay.

Includes - UC5: Viewing Account/Settings.

Notes/Issues - Feedback moderation is required to prevent inappropriate content. Kart Coins have no external financial regulation.



3.1.7 U7- Support and Conflict Resolution

Author - Sneha Kumari

Purpose - To handle exceptional delivery scenarios through a formal issue reporting channel for riders and consumers.

Requirements Traceability - Riders shall be able to report delivery issues (customer unreachable, vendor delay);Customers should also be able to complain about order or delivery issues; Admins shall handle user complaints and feedback.

Priority - Medium. Essential for system robustness and real-world reliability**.**

Preconditions - User must be logged in. Riders must have an "Active" order assigned.

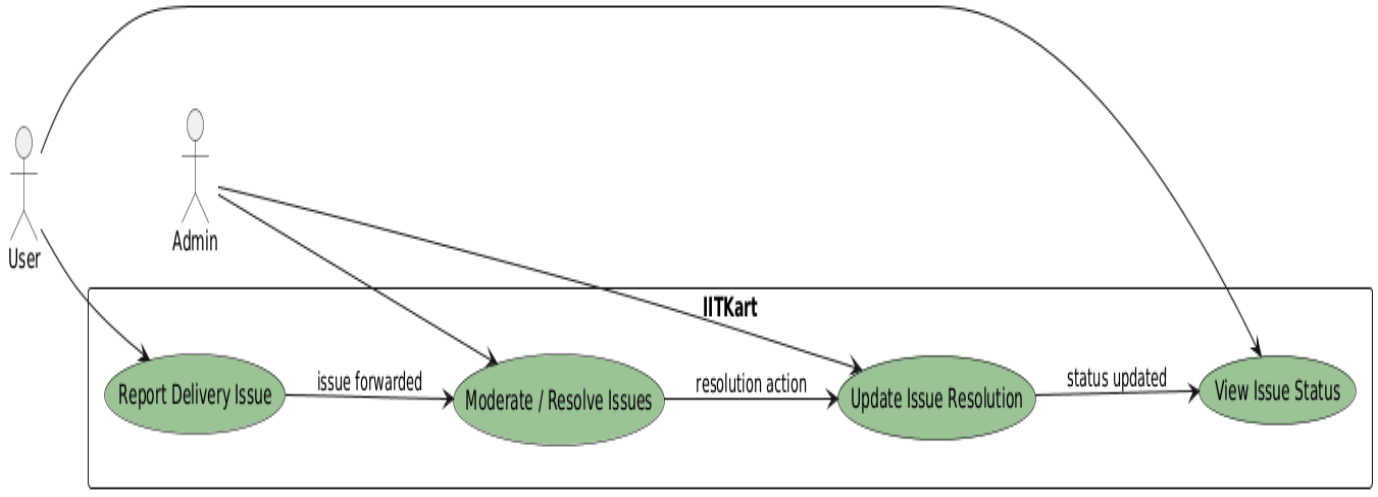
Post conditions - Issue logged with `order_id`. Admin notified; order may be temporarily put “On Hold” OR “Issue Reported”.

Actors - User(Consumer, Rider,Admin.)

Exceptions -
E1: Order cancelled before report;
E2: Incomplete issue details.

Includes - UC7: Admin Manages Users/Vendors/Riders.

Notes/Issues - Prevents riders from being penalized for genuine obstacles like unreachable customers or damaged items.



3.1.8 U8- Platform Governance and Business Analytics

Author - Sneha Kumari

Purpose - To provide Admin oversight of system health metrics (GMV, Success/Failure rates) and moderate stakeholder status.

Requirements Traceability - Admin shall be able to monitor platform metrics, manage user/vendor/rider status (Approve/Suspend), and audit financial records.

Priority - Medium to High. Critical for administrative control and financial auditing**.**

Preconditions - Admin must be logged into the secure Admin portal with elevated privileges.

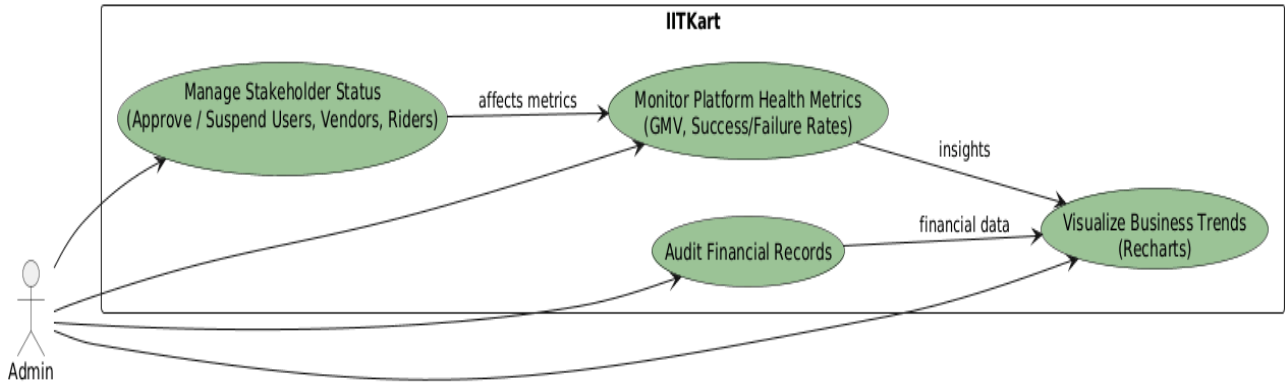
Post conditions - Financial records rendered; user/vendor status updated.

Actors - Admin

Exceptions -
E1: Temporary server unavailability;
E2: Incomplete transaction records.

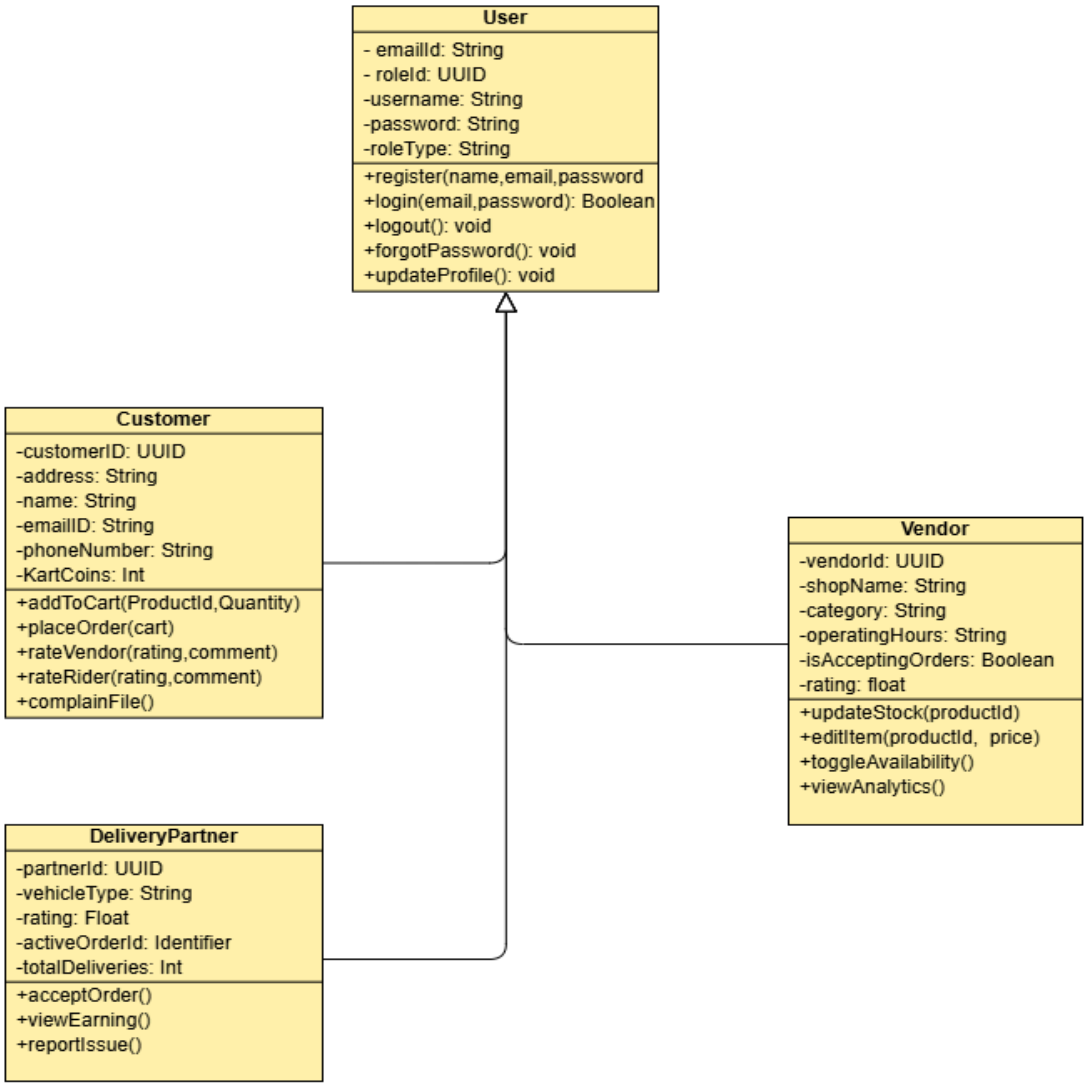
Includes - None

Notes/Issues - Admin accounts are immutable and cannot be deleted. Analytics use Recharts for visual trends.



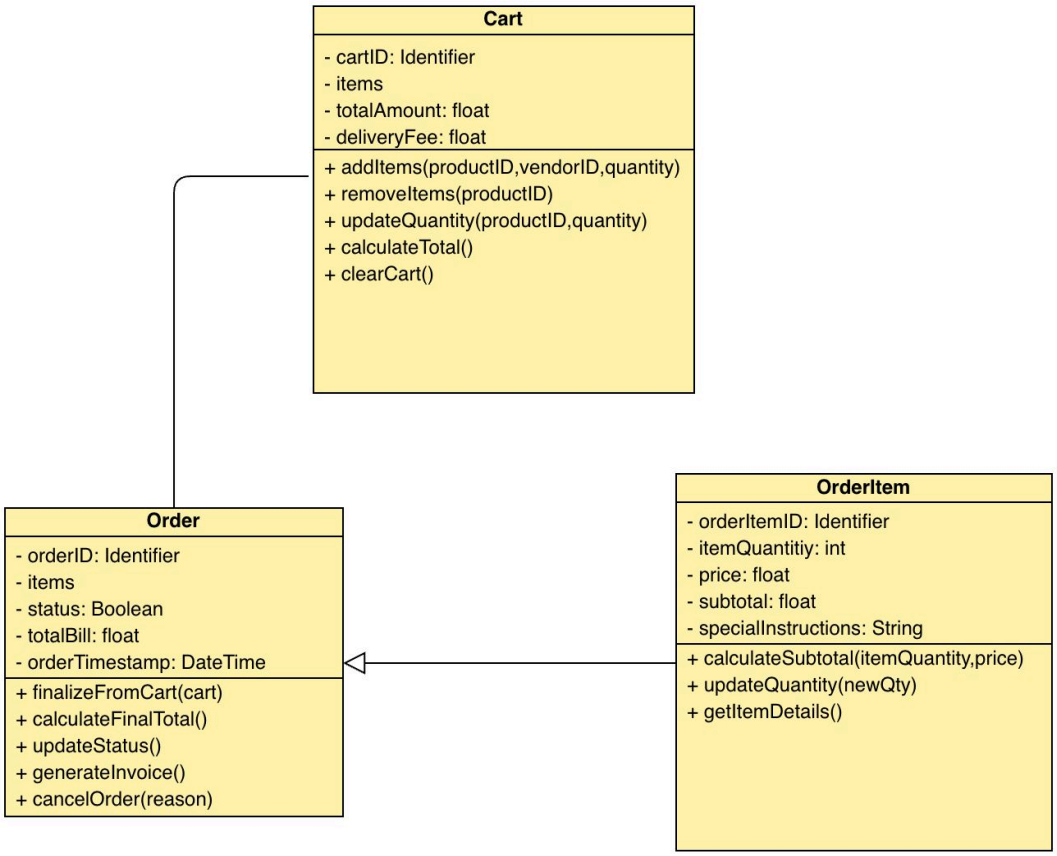
3.2 Class Diagrams

3.2.1 Class Diagram 1

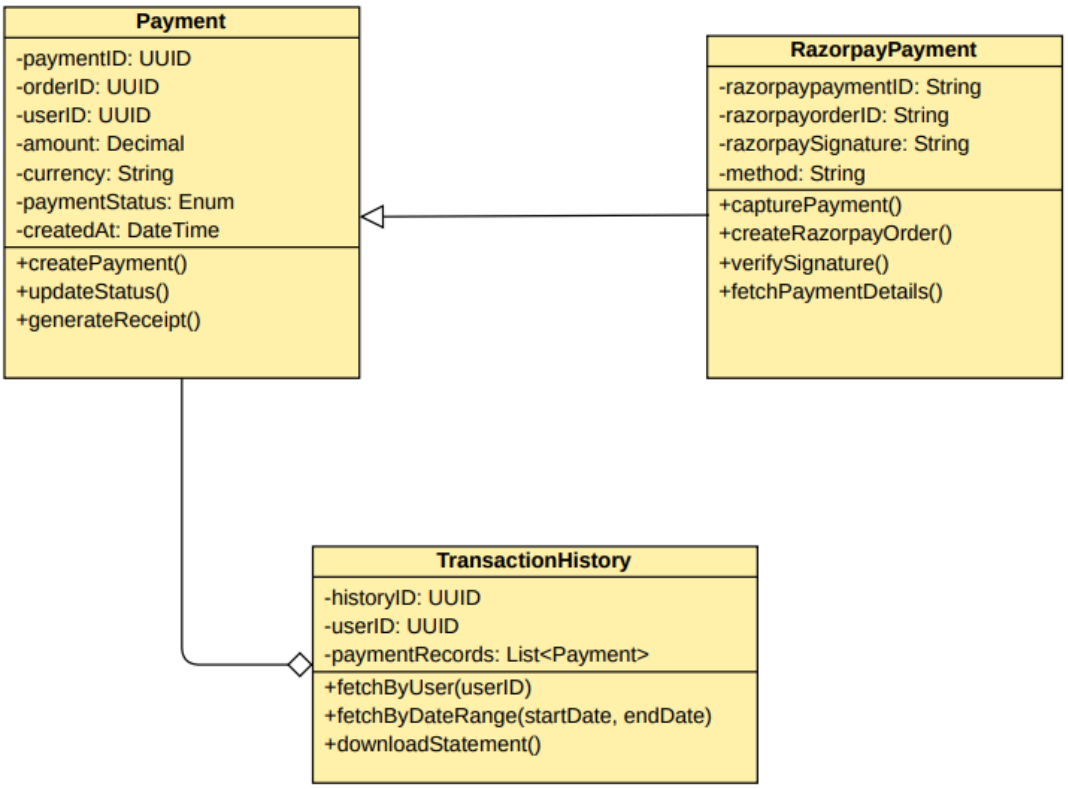


3.2.2 Class Diagram 2

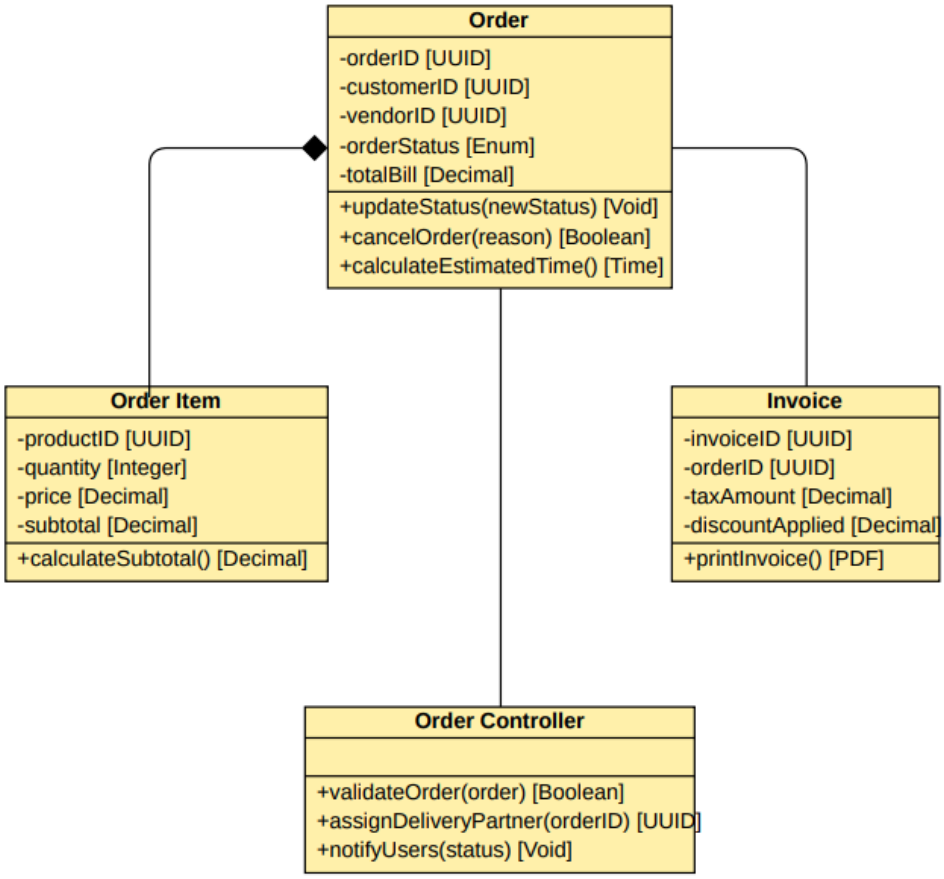
3.2.3 Class Diagram 3



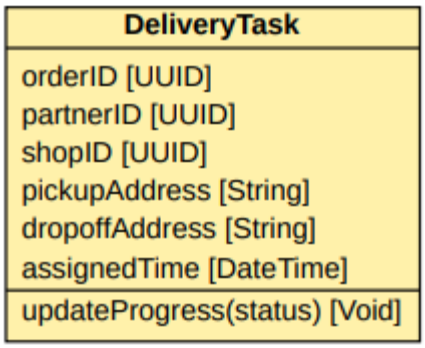
3.2.4 Class Diagram 4



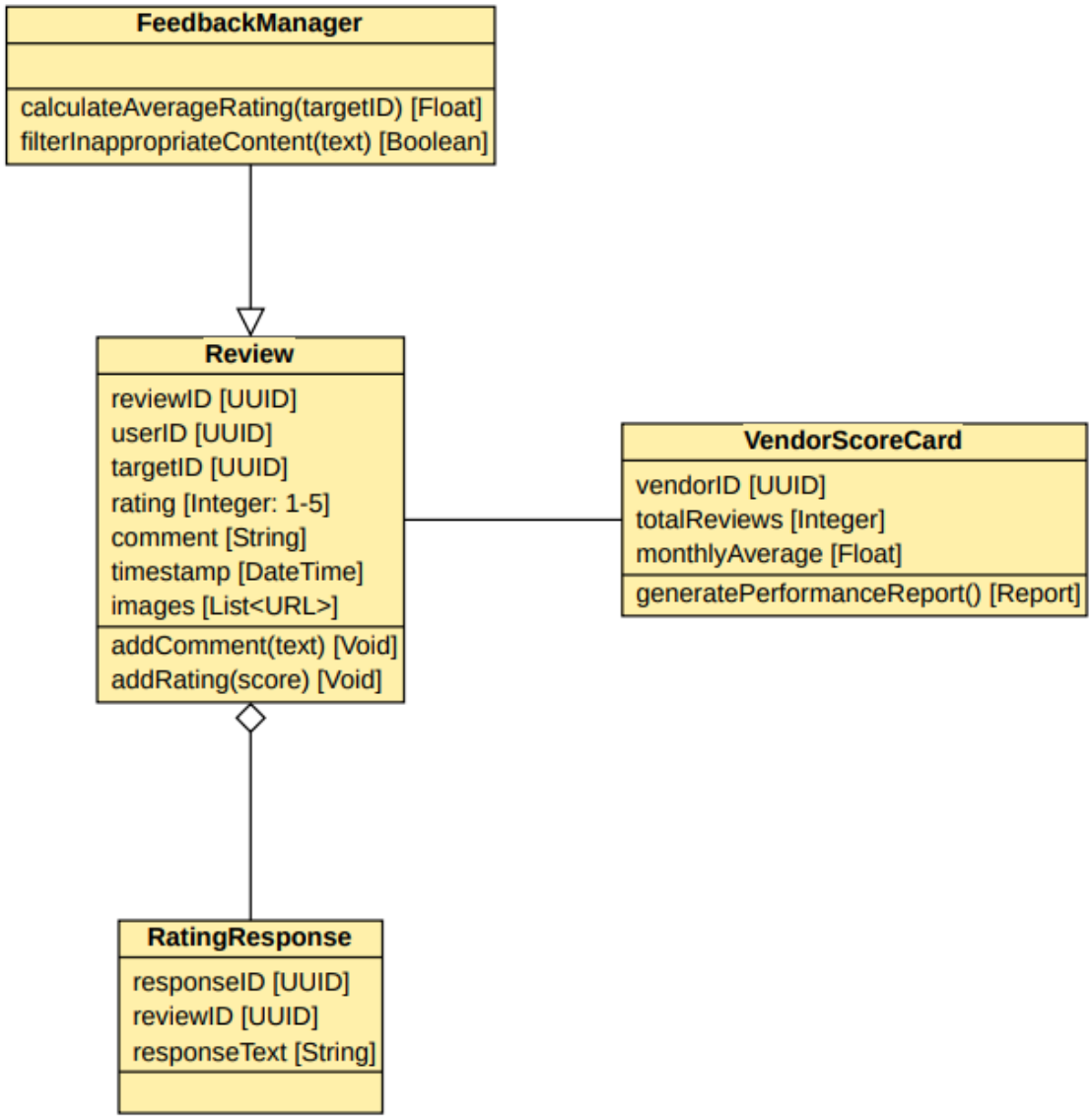
3.2.5 Class Diagram 5



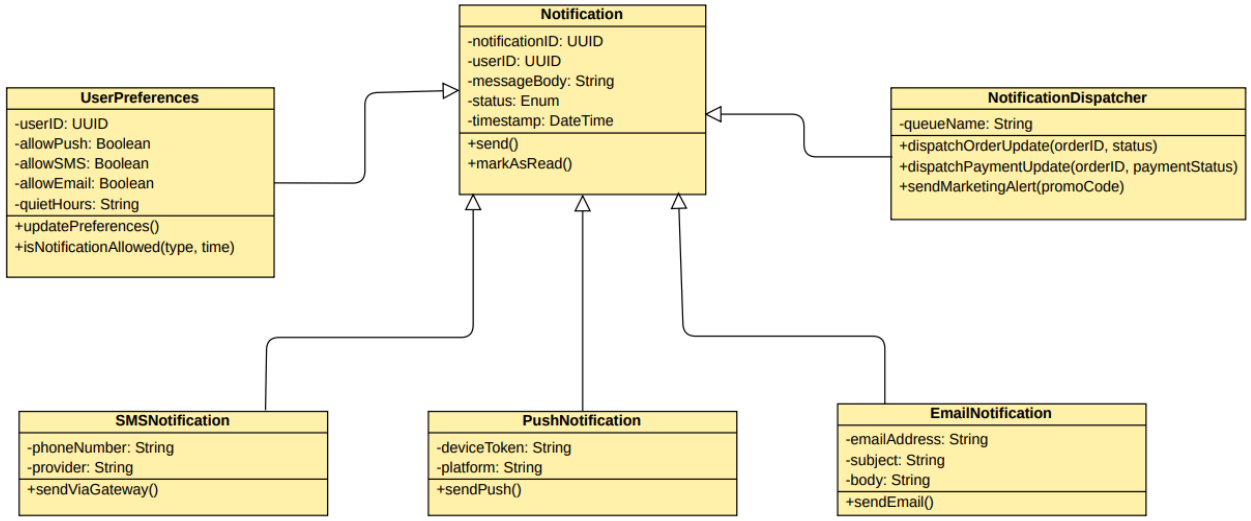
3.3.6 Class Diagram 6



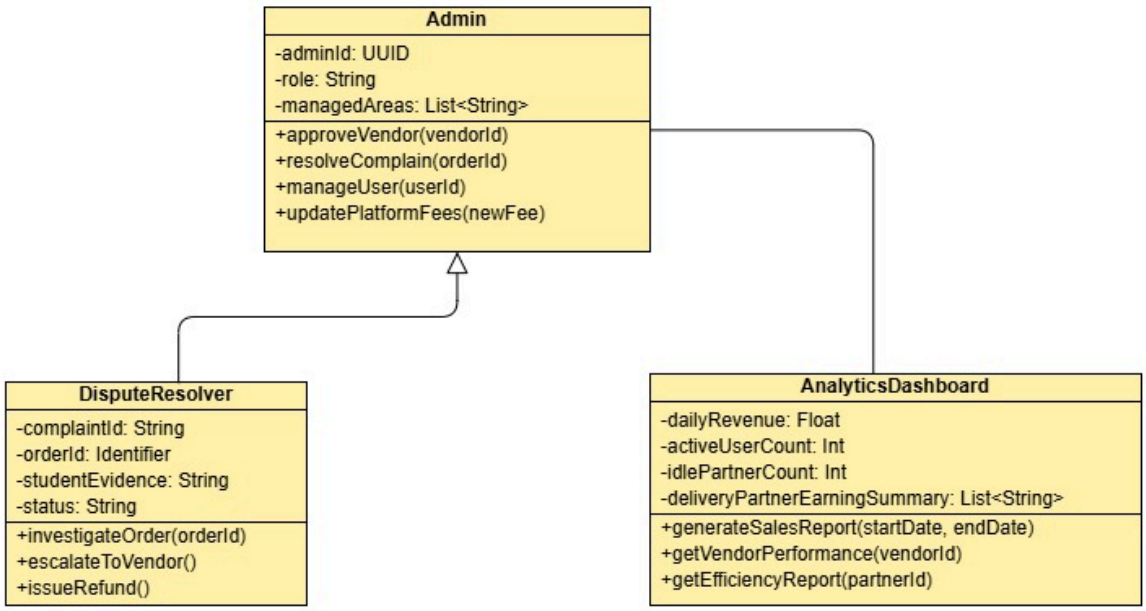
3.2.7 Class Diagram 7



3.2.8 Class Diagram 8



3.2.9 Class Diagram 9



1. **User Class (Base class)**

    - **Attributes:**
        - **emailId:** A unique string representing the user's email address.
        - **roleId:** A Universally Unique Identifier (UUID) used to distinguish the user in the system.
        - **username:** The display name or handle for the user.
        - **password:** The secret string used for authentication**.**
        - **roleType:** A string identifying if the user is a Customer, Vendor, or Delivery Partner.
    - **Operations:**
        - **register(name, email, password):** Handles account creation for new users.
        - **login(email, password):** Authenticates the user and returns a Boolean (true/false) based on success.
        - **logout():** Terminates the user's active session**.**
        - **forgotPassword():** Initiates a credential recovery process.
        - **updateProfile():** Allows the user to modify their account details.

2. **Customer Class**

    - **Attributes:**
        - **customerID**: A specific UUID for the customer profile.
        - **address**: The physical location for delivery.
        - **name**: The legal or full name of the customer.
        - **phoneNumber**: The contact number for delivery updates.
        - **KartCoins**: An integer representing loyalty points or digital currency.
    - **Operations:**
        - **addToCart(ProductId, Quantity)**: Adds a specific item to the virtual shopping basket.
        - **placeOrder(cart)**: Finalizes the purchase of items currently in the cart.
        - **rateVendor(rating, comment)**: Provides feedback on the merchant.
        - **rateRider(rating, comment)**: Provides feedback on the delivery partner.
        - **complainFile()**: Initiates a formal dispute or support request.

3. **Vendor Class**

    - **Attributes:**
        - **vendorId**: A specific UUID for the vendor entity.
        - **shopName**: The commercial name of the store.
        - **category**: The type of products sold (e.g., Grocery, Electronics).
        - **operatingHours**: The time range during which the shop is active.

- **isAcceptingOrders**: A Boolean toggle to show if the shop is currently open for business.
- **rating**: A floating-point number representing the average customer feedback score.

- **Operations:**
  - **updateStock(productId)**: Modifies the quantity of a specific item in the inventory.
  - **editItem(productId, price)**: Updates the details or pricing of an existing product.
  - **toggleAvailability()**: Quickly switches the isAcceptingOrders status.
  - **viewAnalytics()**: Provides data-driven insights into sales and performance.

4. **Delivery Partner Class**

- **Attributes:**
  - **partnerId**: A specific UUID for the delivery personnel.
  - **vehicleType**: The mode of transport (e.g., Bike, Van).
  - **rating**: The average performance score from customers.
  - **activeOrderId**: An identifier for the order currently being handled.
  - **totalDeliveries**: An integer tracking the partner's lifetime completed tasks.
- **Operations:**
  - **acceptOrder()**: Claims a pending delivery request.
  - **viewEarning()**: Shows the financial breakdown of completed deliveries.
  - **reportIssue()**: Flags problems like traffic, vehicle breakdown, or customer unavailability.

5. **Shop Class**

- **Attributes:**
  - **shopID**: A unique identifier for the shop.
  - **shopName**: The name of the store.
  - **shopType**: The category (e.g., Stationery, Grocery).
  - **openingTime / closingTime**: Operational hours.
  - **isOpen**: A true/false status indicating if it's currently trading.
  - **contactNumber**: Telephone or mobile details.
- **Operations:**
  - **updateStatus()**: Manages the isOpen toggle.
  - **getAverageRating()**: Calculates customer feedback scores.
  - **getMenu() / getLocationDetails()**: Retrieves descriptive information about the shop’s offerings or physical spot.

6. **Product Class**

- **Attributes:**
  - **productID**: The unique serial or identification number for the item.
  - **name**: The display name of the product.
  - **description**: A text summary of the product's features.
  - **averageRating**: The quality score assigned by users.
  - **price**: The monetary value (stored as a String, likely for currency formatting).
  - **isAvailable**: Indicates if the item is currently in stock.
  - **imageURL**: The path to the product's visual representation.
  - **Category**: The group the product belongs to (e.g., "Stationery").
- **Operations:**
  - **updatePrice()**: Modifies the **price** attribute.
  - **editDetails()**: Allows for updating the **name** and **description**.
  - **toggleAvailability()**: Switches the **isAvailable** status.

7. **Stock Class**

- **Attributes:**
  - **StockID**: Unique identifier for the inventory record.
  - **items**: A List containing multiple **Product** objects.
  - **lastUpdated**: Records the time of the most recent inventory change.
  - **isActive**: A boolean showing if this specific stock list is currently live.
- **Operations:**
  - **getAvailableItems()**: Returns only those products where isAvailable is true.
  - **filterByCategory()**: Groups and displays items based on their **Category**.
  - **updateItemAvailability()**: Batch updates the status of multiple items.
  - **searchItem()**: Locates a specific product within the list.
  - **applyDiscount()**: Modifies the price for a group of items.

8. **Cart Class**

- **Attributes:**
  - **cartID**: A unique identifier for the user's active shopping session.
  - **items**: A collection of products currently held in the cart.
  - **totalAmount**: The running sum of the prices of all items in the cart.
  - **deliveryFee**: The calculated cost for transporting the items to the user.
- **Operations:**
  - **addItems()**: Adds a specific quantity of a product from a vendor to the cart.
  - **removeItems()**: Completely deletes a product from the cart using its ID.

- **updateQuantity()**: Adjusts the number of units for a specific product.
- **calculateTotal()**: Computes the sum of all items plus the deliveryFee.
- **clearCart()**: Wipes all items from the cart, usually after an order is placed or cancelled.

9. OrderItem Class

- **Attributes:**
  - **orderItemID**: A unique identifier for this specific entry in an order.
  - **itemQuantity**: The number of units purchased for this specific item.
  - **price**: The individual cost of the item at the time of the transaction.
  - **subtotal**: The result of price multiplied by itemQuantity.
  - **specialInstructions**: User-provided notes
- **Operations:**
  - **calculateSubtotal()**: Logic to determine the total cost for this specific line item.
  - **updateQuantity()**: Changes the quantity, which would subsequently trigger a subtotal recalculation.
  - **getItemDetails()**: Fetches descriptive data about the product for the invoice.

10. Order Class

- **Attributes:**
  - **orderID**: The permanent tracking number for the transaction.
  - **items**: A list of OrderItem objects associated with this purchase.
  - **status**: A boolean or state indicator (e.g., Pending, Completed, or Cancelled).
  - **totalBill**: The final amount charged to the user.
  - **orderTimestamp**: The exact date and time the order was finalized.
- **Operations:**
  - **finalizeFromCart()**: The process of converting the temporary Cart data into a permanent Order.
  - **calculateFinalTotal()**: Sums all OrderItem subtotals and applies any final taxes or fees.
  - **updateStatus()**: Moves the order through the workflow (e.g., "Placed" to "Delivered").
  - **generateInvoice()**: Creates a bill or receipt for the user.
  - **cancelOrder()**: Terminates the order and requires a reason for record-keeping.

11. Payment Class

- **Attributes:**
  - **paymentID**: A unique identifier (UUID) for the specific transaction.

- **orderID**: Connects the payment to a specific order from the Order class.
- **userID**: Identifies which user is making the payment.
- **amount**: The total numerical value to be charged.
- **currency**: The type of money used (e.g., INR).
- **paymentStatus**: An Enum representing the current state (e.g., PENDING, SUCCESS, FAILED).
- **createdAt**: A timestamp of when the payment attempt was initiated.

- **Operations:**
  - **createPayment()**: Initializes a new payment record in the database.
  - **updateStatus()**: Refreshes the paymentStatus based on the gateway response.
  - **generateReceipt()**: Produces a confirmation document for the user.

12. RazorpayPayment Class

- **Attributes:**
  - **razorpaypaymentID**: The specific ID returned by Razorpay's servers.
  - **razorpayorderID**: The order token generated by the gateway.
  - **razorpaySignature**: A security hash used to verify that the payment is authentic.
  - **method**: The specific payment mode (e.g., UPI, Net Banking, Card).
- **Operations:**
  - **capturePayment()**: Confirms the receipt of funds from the gateway.
  - **createRazorpayOrder()**: Communicates with the Razorpay API to start a transaction.
  - **verifySignature()**: A security check to ensure the transaction data hasn't been tampered with.
  - **fetchPaymentDetails()**: Retrieves technical logs about the transaction from the gateway.

13. TransactionHistory Class

- **Attributes:**
  - **historyID**: A unique ID for the history record.
  - **userID**: Links the history to a specific account.
  - **paymentRecords**: A List containing multiple Payment objects associated with the user.
- **Operations:**
  - **fetchByUser(userID)**: Retrieves all transaction records for a specific person.
  - **fetchByDateRange(startDate, endDate)**: Filters the history for a specific time period.
  - **downloadStatement()**: Exports the transaction list into a portable format like a PDF.

14. **Invoice Class**

- **Attributes:**
  - **invoiceID**: A unique UUID for the tax/billing document.
  - **orderID**: A reference linking the invoice to a specific **Order**.
  - **taxAmount**: The calculated tax applied to the transaction.
  - **discountApplied**: Any price reductions (e.g., promo codes) applied to the final bill.
- **Operations:**
  - **printInvoice()**: Exports the billing details into a PDF format for the user.

15. **Order Controller Class**

- **Operations:**
  - **validateOrder():** A security check to ensure the order data (items, prices) is correct before processing.
  - **assignDeliveryPartner():** A logic-heavy method that finds and assigns a delivery person to the specific orderID.
  - **notifyUsers():** Sends real-time status updates (like "Order Placed" or "Out for Delivery") to the customer and vendor.

16. **FeedbackManager Class**

- **Operations:**
  - **calculateAverageRating:** Logic to compute the mean score for a specific item.
  - **filterInappropriateContent:** A validation check to block or flag offensive text.

17. **Review Class**

- **Attributes:**
  - **reviewID**: The unique identity of a specific feedback submission.
  - **userID**: The identity of the customer/user writing the review.
  - **targetID**: The identity of the specific product or entity being reviewed.
  - **rating**: The numerical value (1-5) assigned to the experience.
  - **comment**: The written feedback or opinion.
  - **timestamp**: The exact point in time when the review was submitted.
  - **images**: The visual media attached to the review.
- **Operations:**
  - **addComment**: The action of inputting or updating the text.
  - **addRating**: The action of setting or updating the score.

18. VendorScoreCard Class

- **Attributes:**
  - **vendorID**: The unique identity of the business being measured.
  - **totalReviews**: The count of every review received by the vendor.
  - **monthlyAverage**: The calculated mean score for the current month.
- **Operations:**
  - **generatePerformanceReport**: The process of creating a summary of the vendor's metrics.

19. RatingResponse

- **Attributes:**
  - **responseID**: The unique identity of a reply made to a review.
  - **responseText**: The written content of the reply

20. Notification

- **Attributes:**
  - **notificationID**: A unique identifier for the specific notification instance.
  - **userID**: The identifier for the user who is the recipient of the notification.
  - **messageBody**: The main text or content contained within the notification.
  - **status**: The current condition or phase of the notification, such as sent or pending.
  - **timestamp**: The specific date and time the notification was generated.
- **Operations:**
  - **send()**: The operation that triggers the delivery of the notification.
  - **markAsRead()**: The operation that updates the notification to show it has been viewed.

21. UserPreferences

- **Attributes**:
  - **allowPush**: A setting determining if mobile or web push alerts are enabled.
  - **allowSMS**: A setting determining if text message alerts are enabled.
  - **allowEmail**: A setting determining if email alerts are enabled.
  - **quietHours**: A designated timeframe where notifications are restricted or silenced.
- **Operations:**
  - **updatePreferences()**: The process used to change a user's notification settings.
  - **isNotificationAllowed(type, time)**: A check to see if a notification can be sent based on its category and the current time.

22. SMSNotification

- **Attributes**:
  - **phoneNumber**: The specific mobile number designated to receive the SMS.
  - **provider:** The service or carrier used to route the text message.
- **Operations:**
  - **sendViaGateway()**: The specific method for transmitting the message through an SMS gateway.

23. PushNotification

- **Attributes:**
  - **deviceToken**: A unique string used to identify and target a specific mobile device.
  - **platform**: The operating system or environment, such as iOS or Android, receiving the push.
- **Operations:**
  - **sendPush()**: The specific method used to deliver a push alert.

24. EmailNotification

- **Attributes**:
  - **emailAddress**: The digital destination where the email is sent.
  - **subject**: The brief summary line of the email's content.
  - **body**: The full text or HTML content of the email message.
- **Operations:**
  - **sendEmail()**: The specific method used to transmit the email.

25. NotificationDispatcher

- **Attributes:**
  - **queueName:** The identifier for the specific line or buffer where notifications are held for processing.
- **Operations:**
  - **dispatchOrderUpdate(orderID, status):** Coordinates the sending of an alert regarding changes to a customer's order.
  - **dispatchPaymentUpdate(orderID, paymentStatus):** Coordinates the sending of an alert regarding a financial transaction.
  - **sendMarketingAlert(promoCode):** Manages the distribution of promotional messages containing a discount or offer code.

26. Admin

- **Attributes:**

- **adminId**: A unique identifier for the administrative user.
- **role**: The specific designation or permission level assigned to the admin.
- **managedAreas**: A list of specific categories or departments the admin oversees.

- **Operations:**
  - **approveVendor(vendorId)**: The operation to grant a vendor permission to operate on the platform.
  - **resolveComplain(orderId)**: The process of addressing and closing customer complaints.
  - **manageUser(userId)**: An operation to update, suspend, or modify user accounts.
  - **updatePlatformFees(newFee)**: An operation to change the service fees charged by the platform.

27. DisputeResolver

- **Attributes:**
  - **complaintId:** The unique identifier for a specific complaint or dispute.
  - **orderId**: The identifier for the transaction being disputed.
  - **studentEvidence:** Text or documentation provided by a student to support their claim.
- **Operations:**
  - **investigateOrder(orderId)**: The process of reviewing the details of a contested transaction.
  - **escalateToVendor()**: The action of passing the dispute to the vendor for further response.
  - **issueRefund()**: The operation to return money to a user if a dispute is settled in their favor.

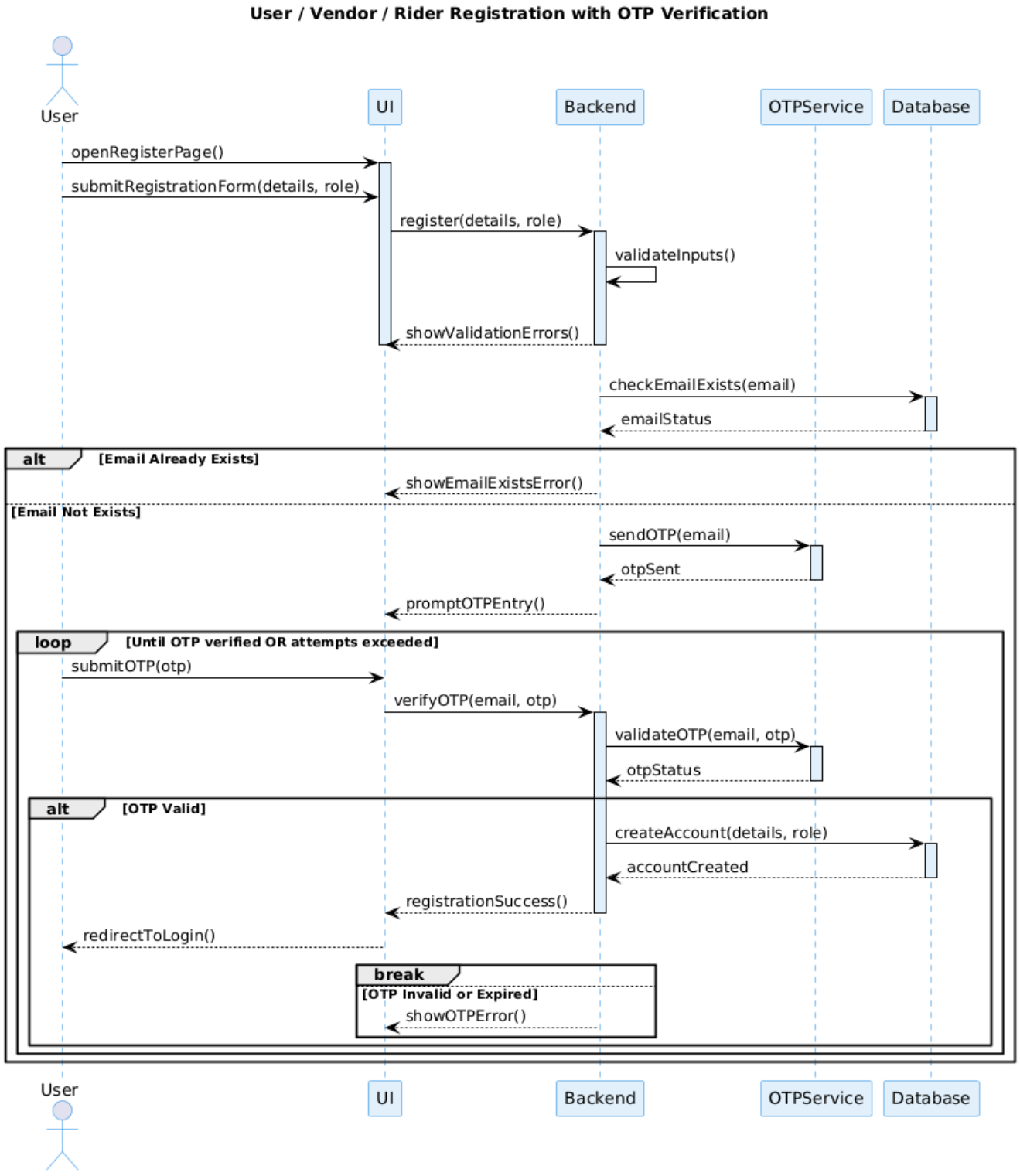
28. AnalyticsDashboard

- **Attributes:**
  - **dailyRevenue:** The total amount of money earned by the platform in a single day.
  - **activeUserCount:** The number of users currently interacting with the platform.
  - **idlePartnerCount**: The number of delivery or business partners who are currently inactive.
  - **deliveryPartnerEarningSummary**: A summarized list of earnings for various delivery partners.
- **Operations:**
  - **generateSalesReport(startDate, endDate):** An operation to compile financial data over a specific period.
  - **getVendorPerformance(vendorId)**: Retrieves metrics and ratings for a specific vendor.

- **getEfficiencyReport(partnerId)**: Analyzes the speed or success rate of a specific partner.

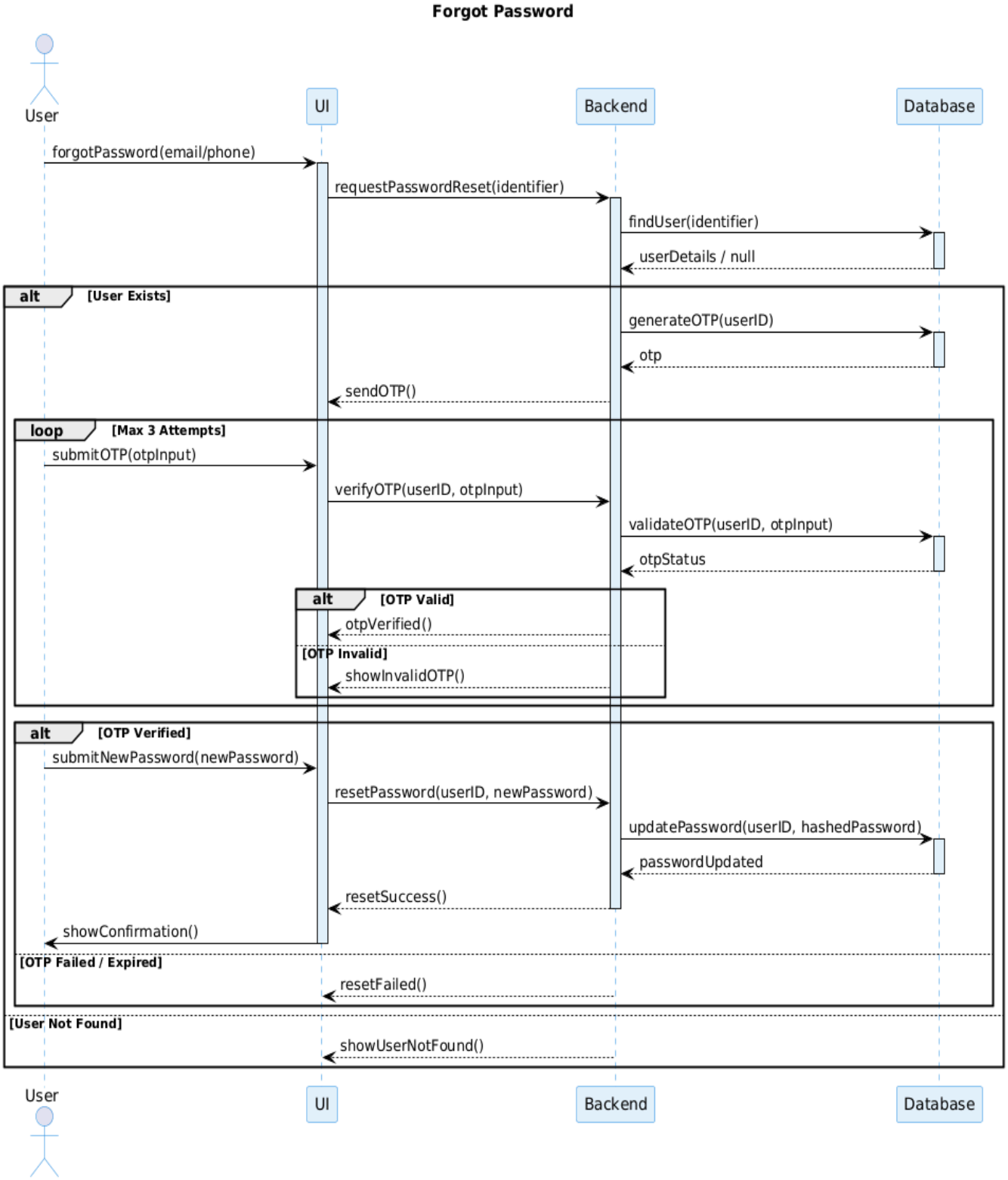
3.3 Sequence Diagrams

3.3.1 Customer/Vendor/Rider/Admin – Register

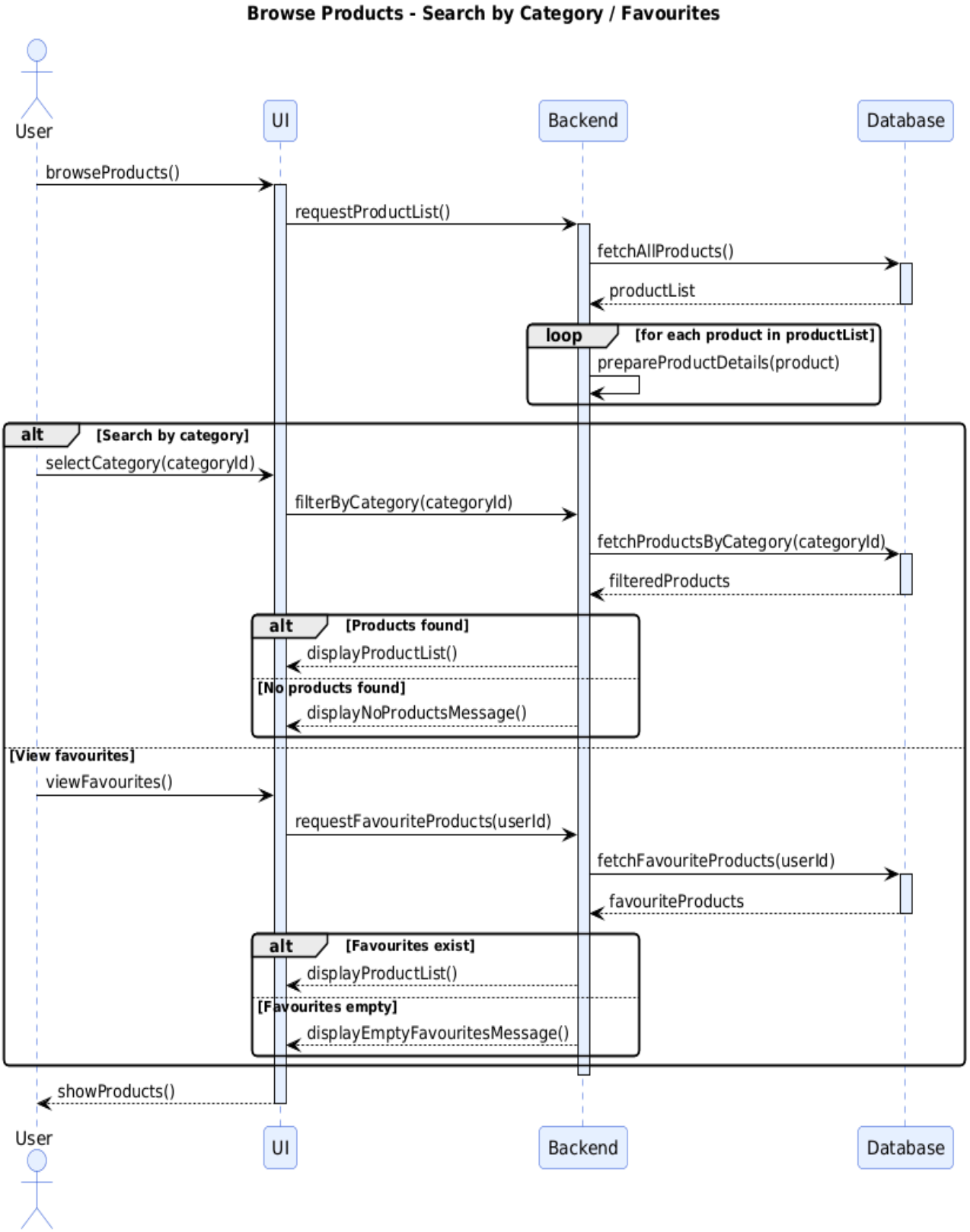


3.3.2 Customer/Vendor/Rider/Admin – Login

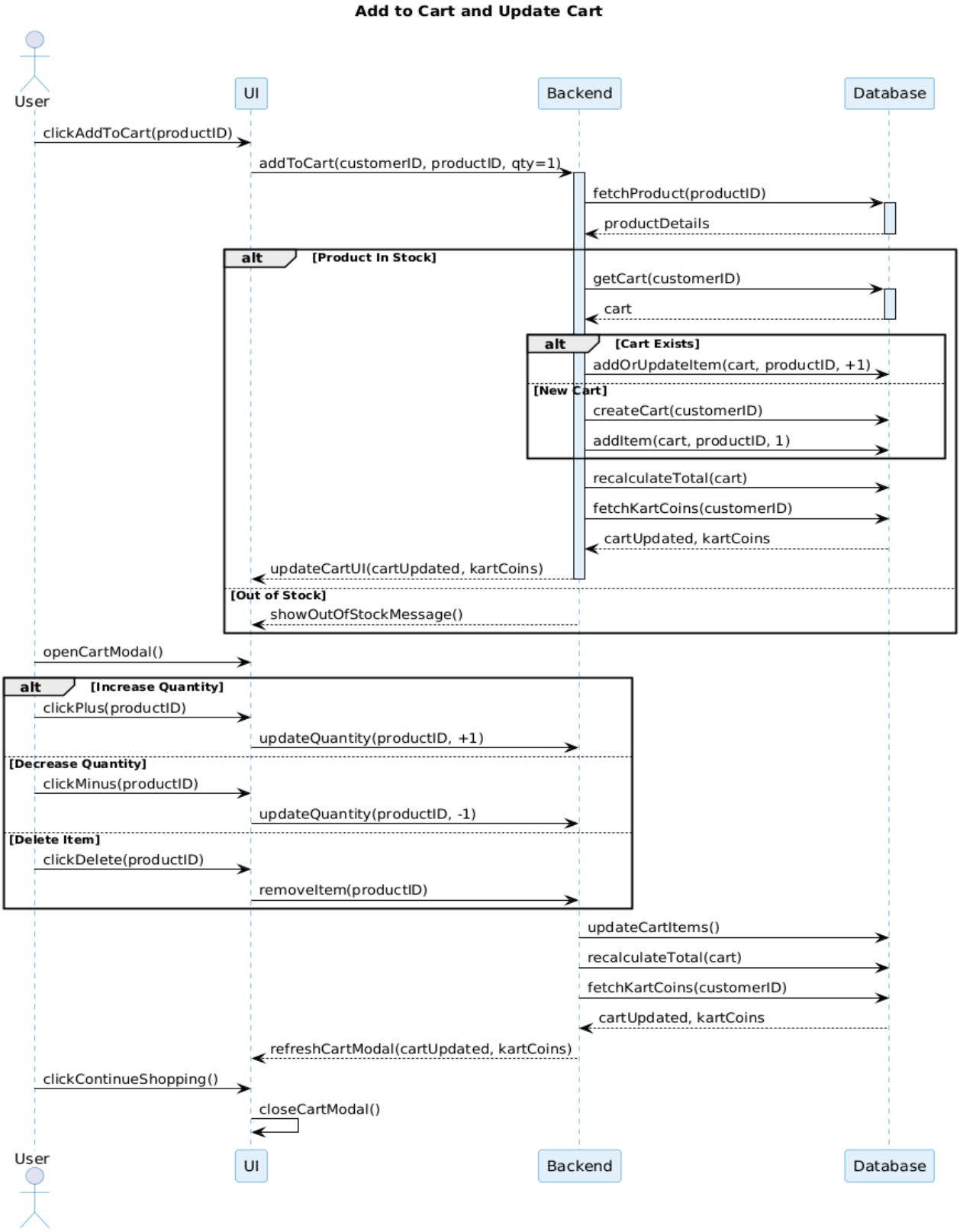
3.3.3 Password Reset



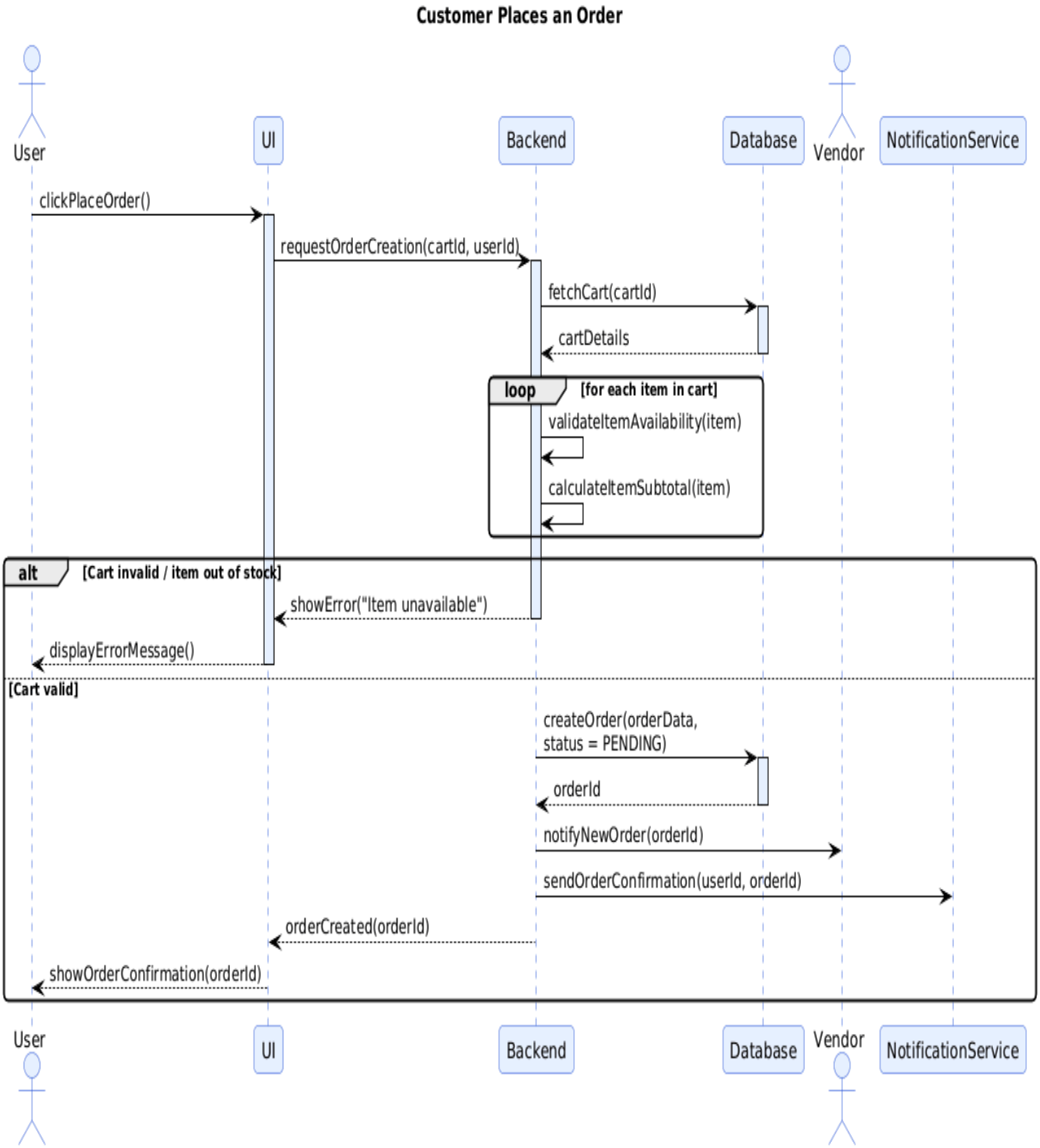
3.3.4 Browse Products – Search by Category/Favourites



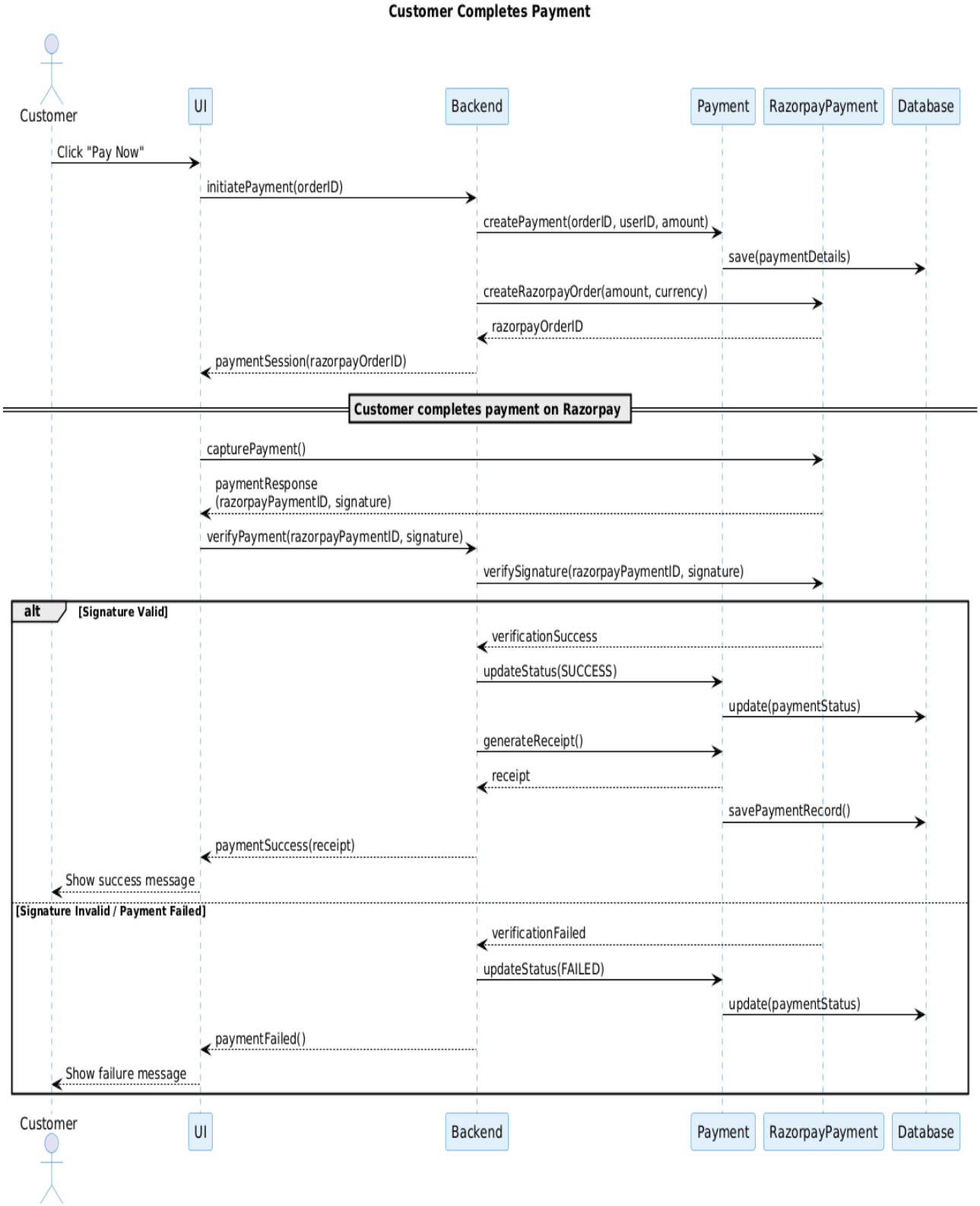
3.3.5 Add to Cart and Update Cart (Kart Coins)



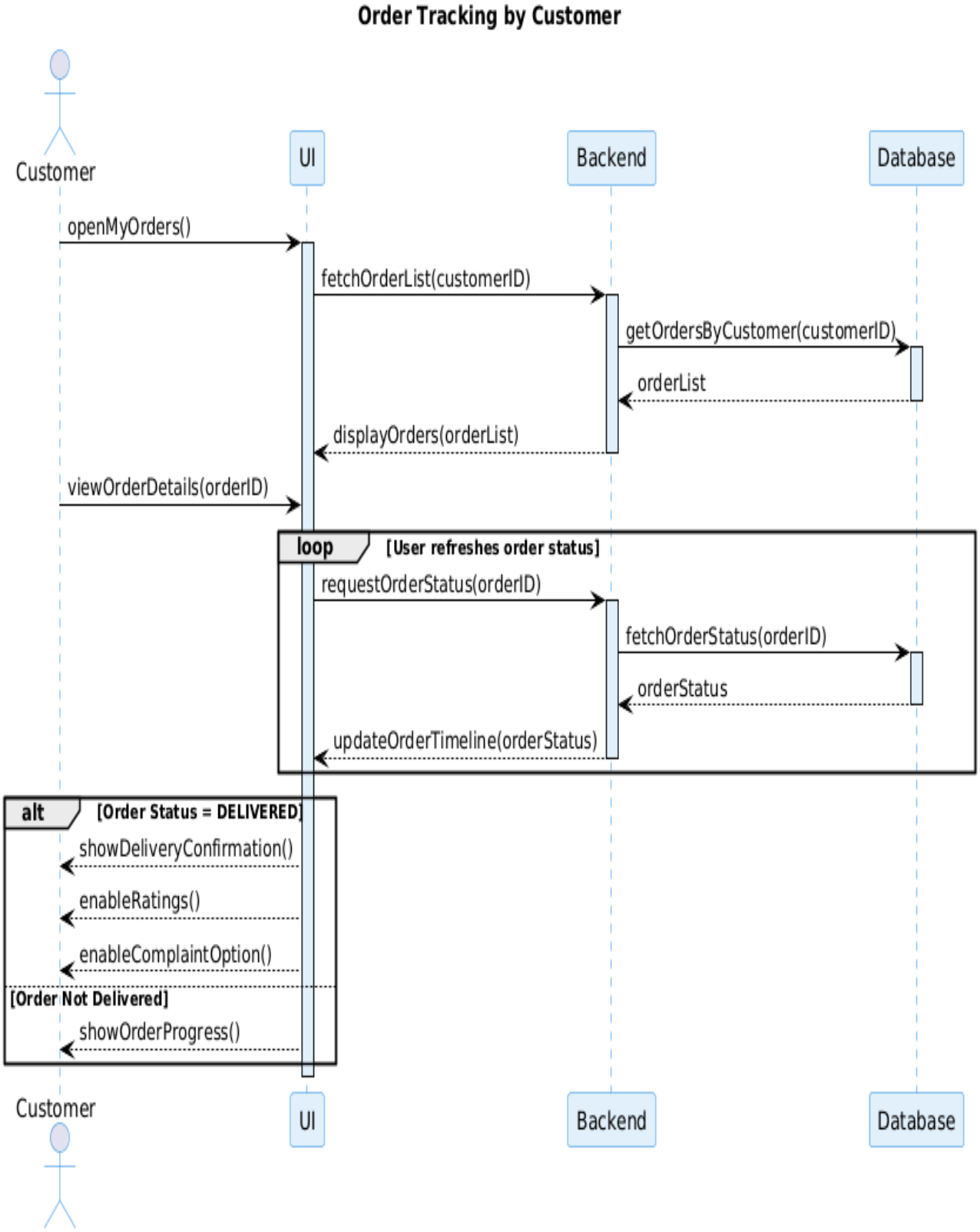
3.3.6 Customer Places an Order



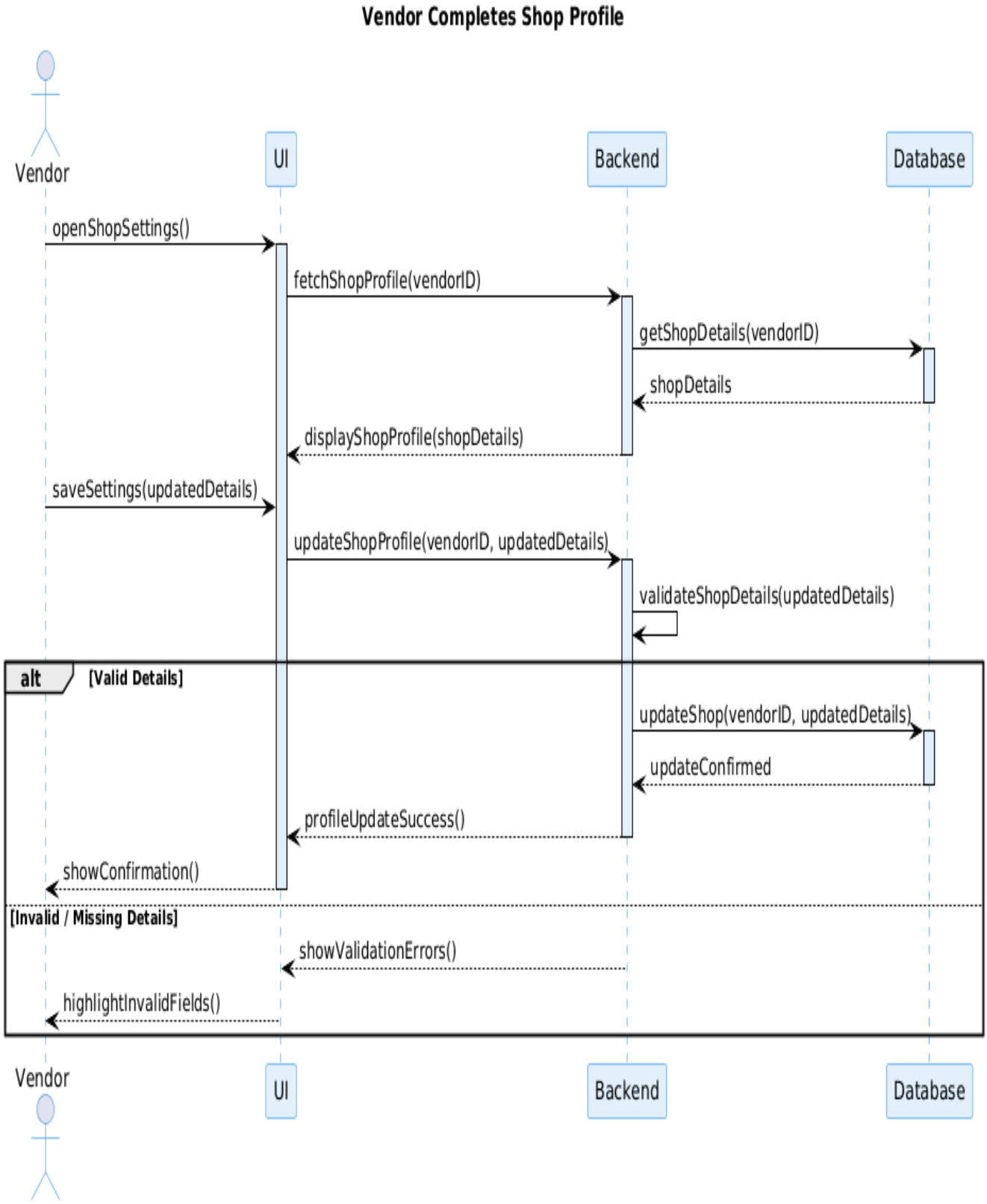
3.3.7 Customer Completes Payment



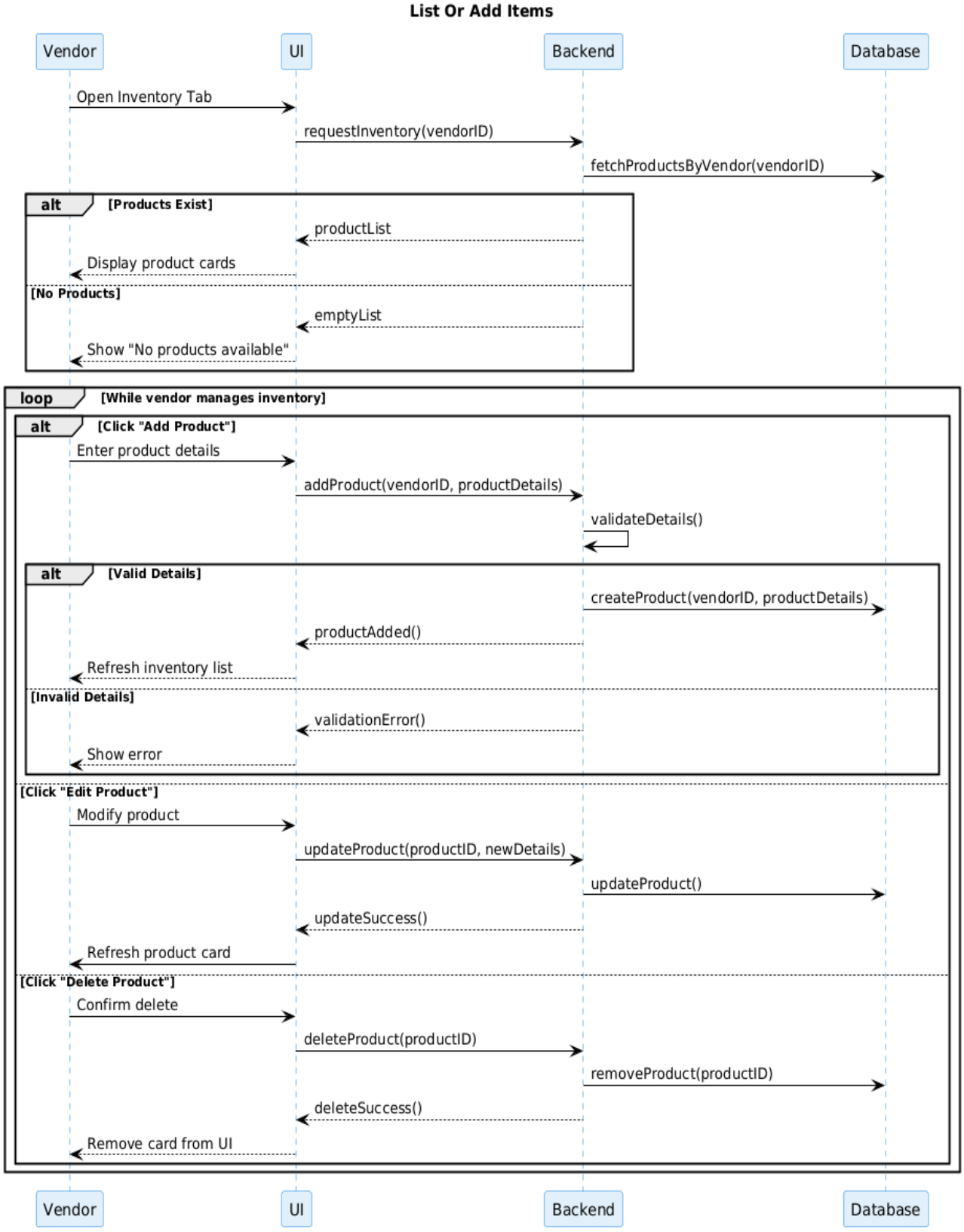
3.3.8 Order Tracking by Customer



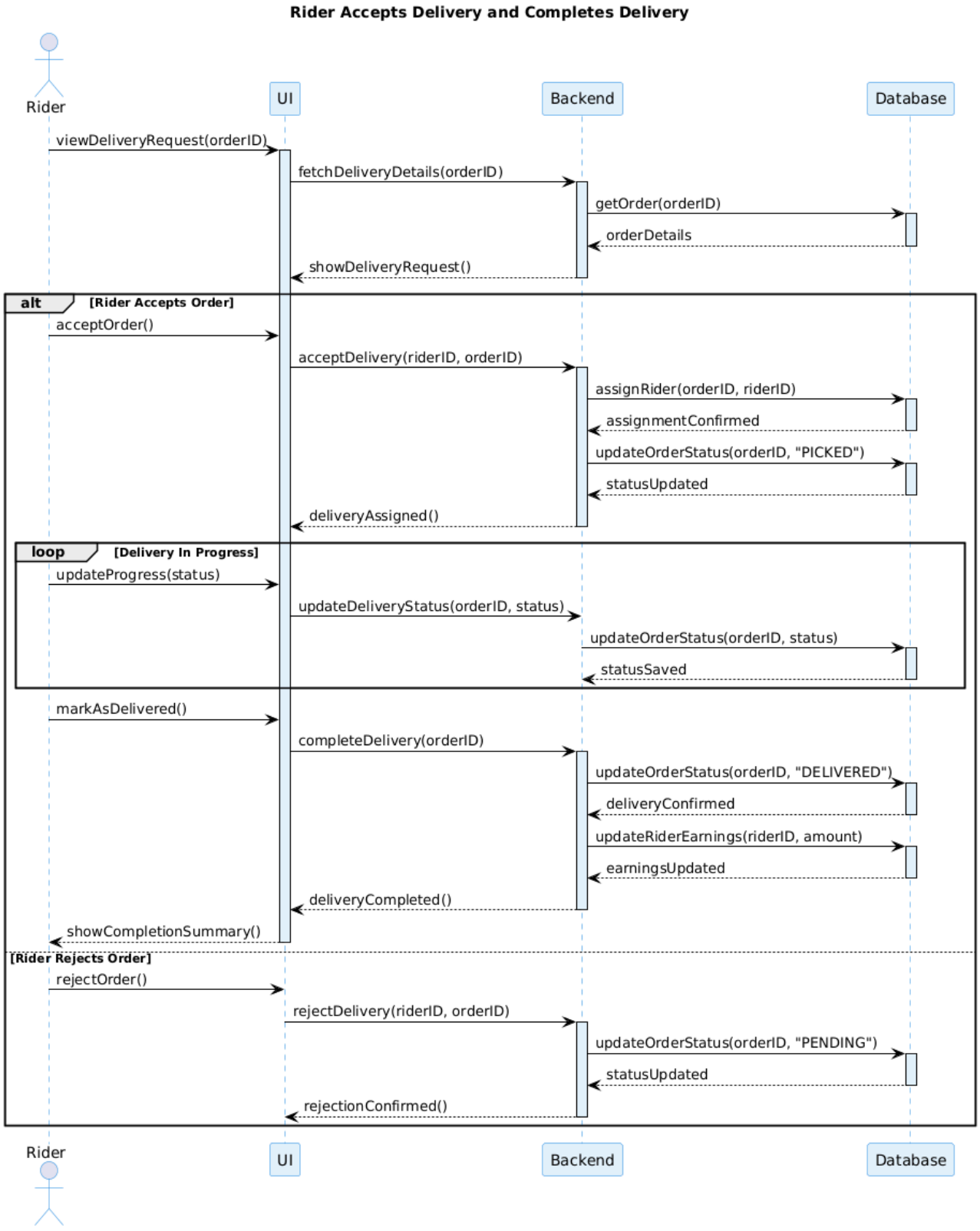
3.3.9 Vendor Completes Shop Profile



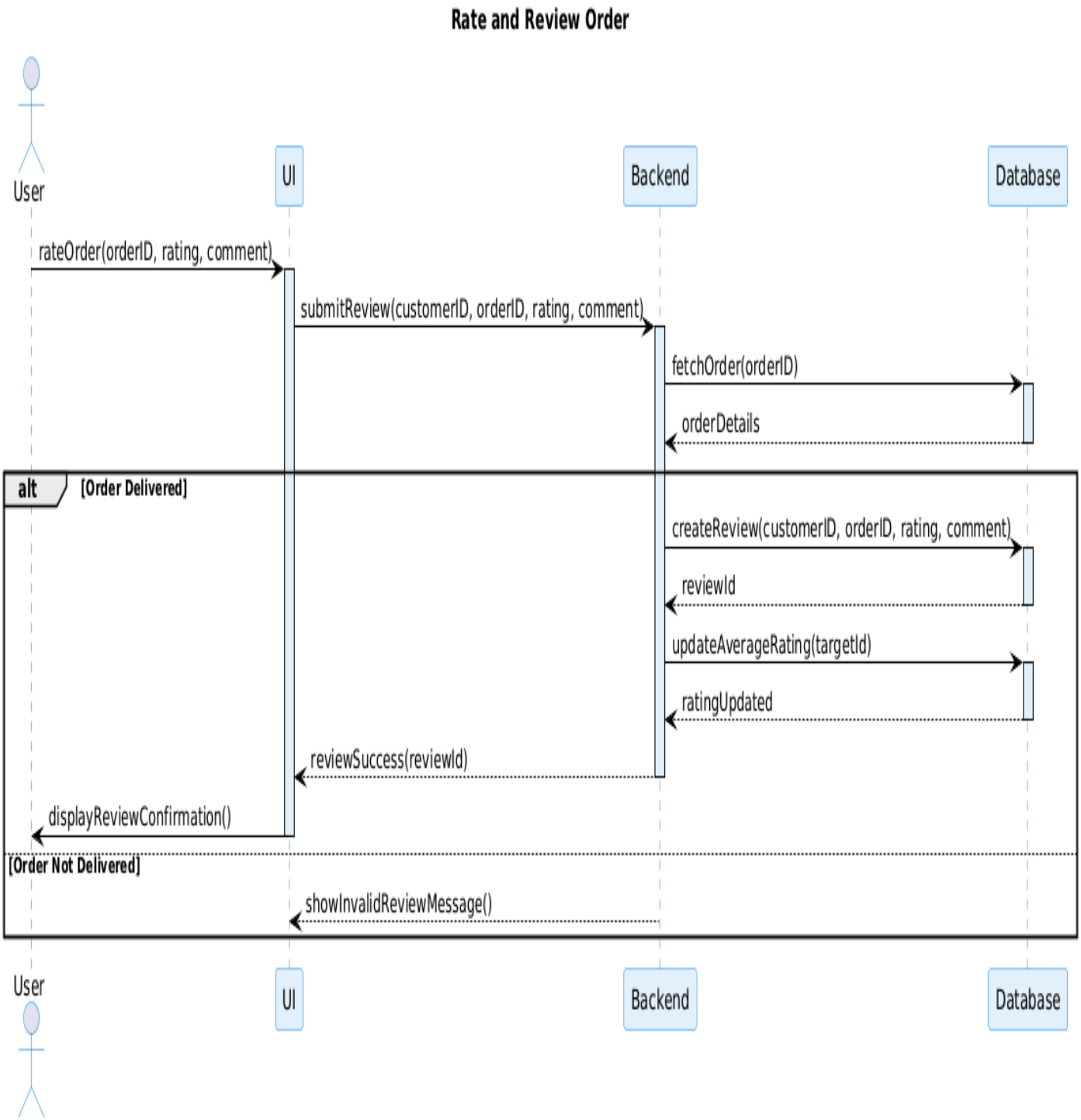
3.3.10 Vendor List Items / Manages Inventory



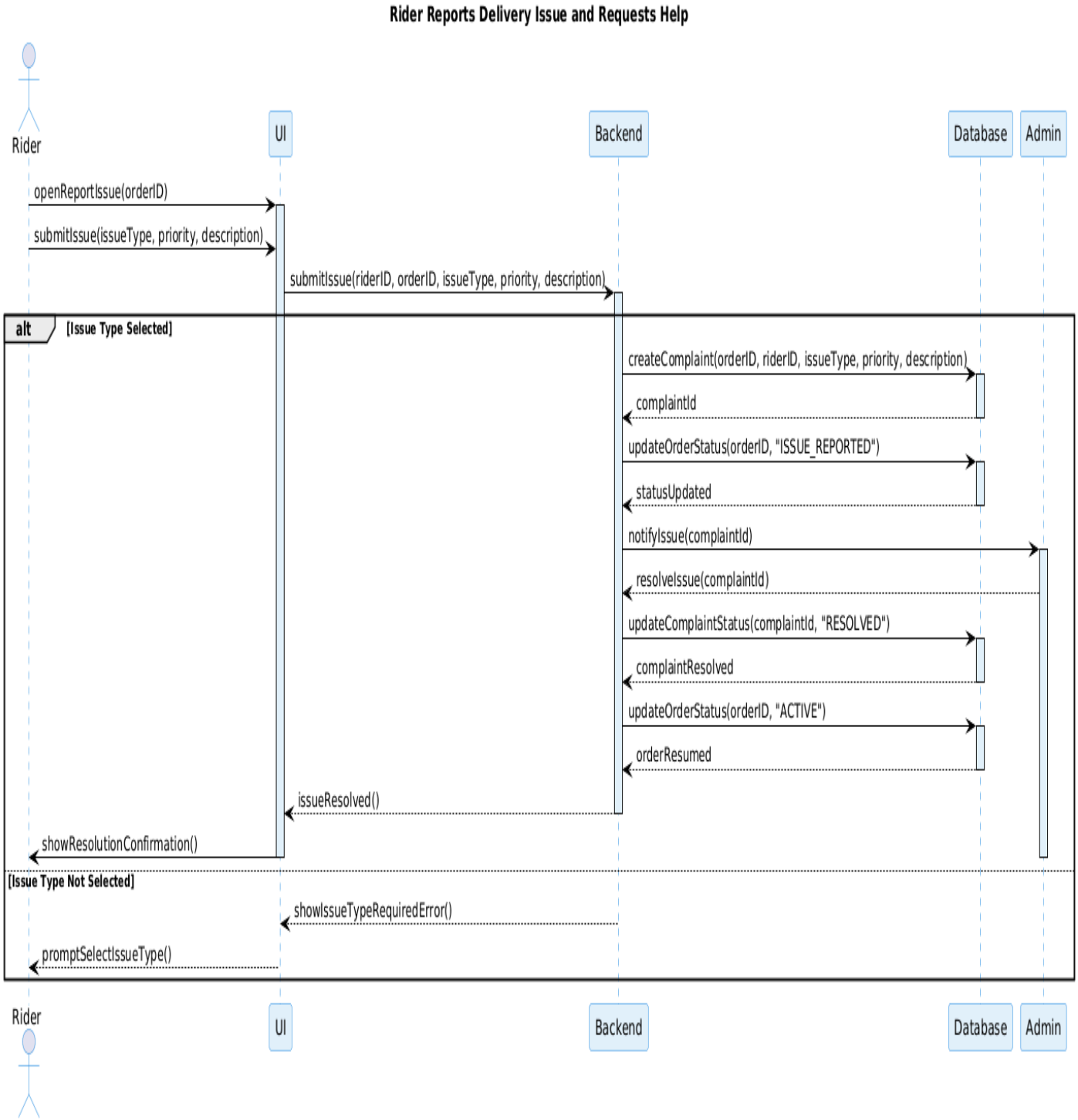
3.3.11 Rider Accepts Delivery and Completes Delivery



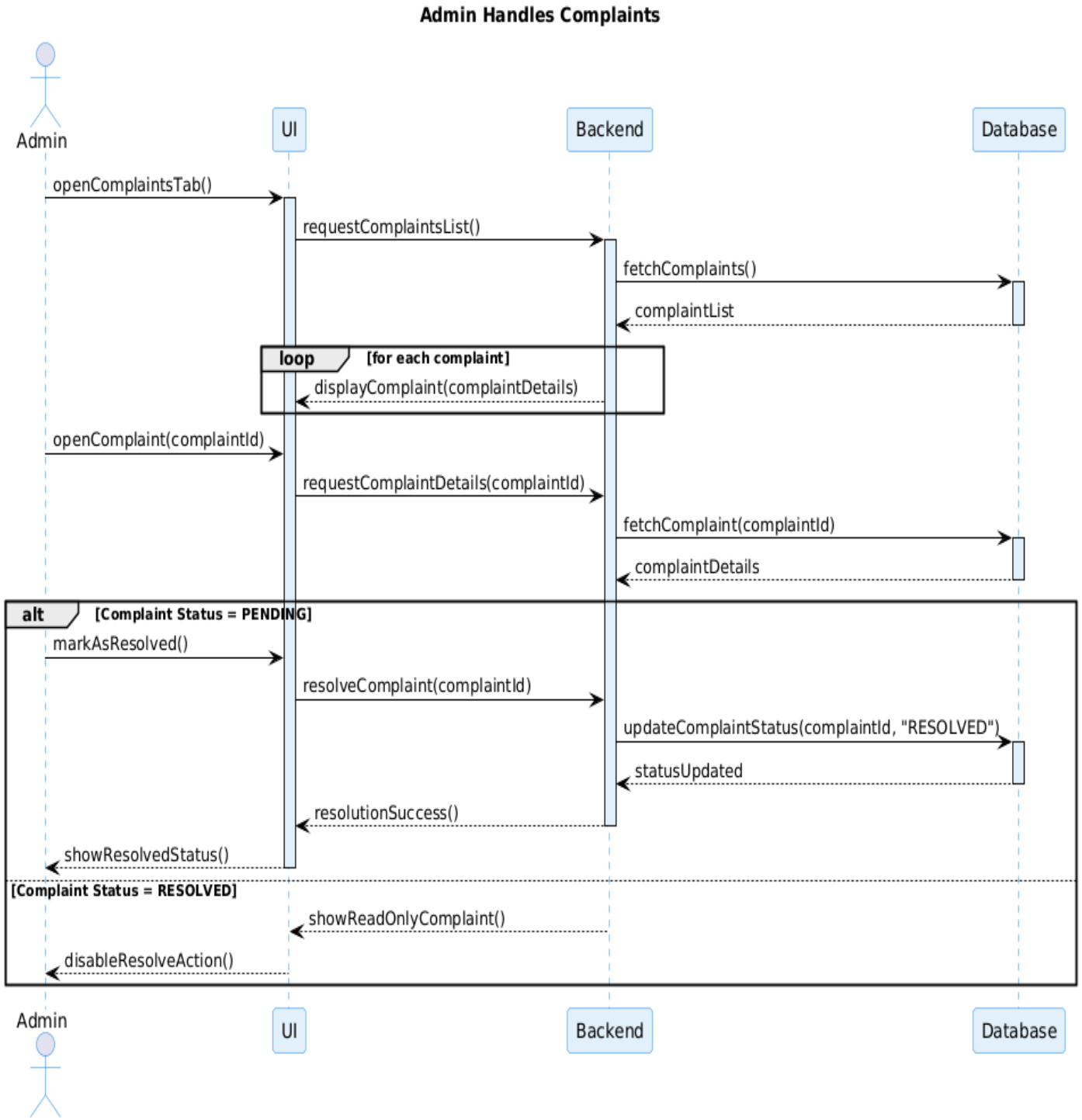
3.3.12 Rate and Review Order



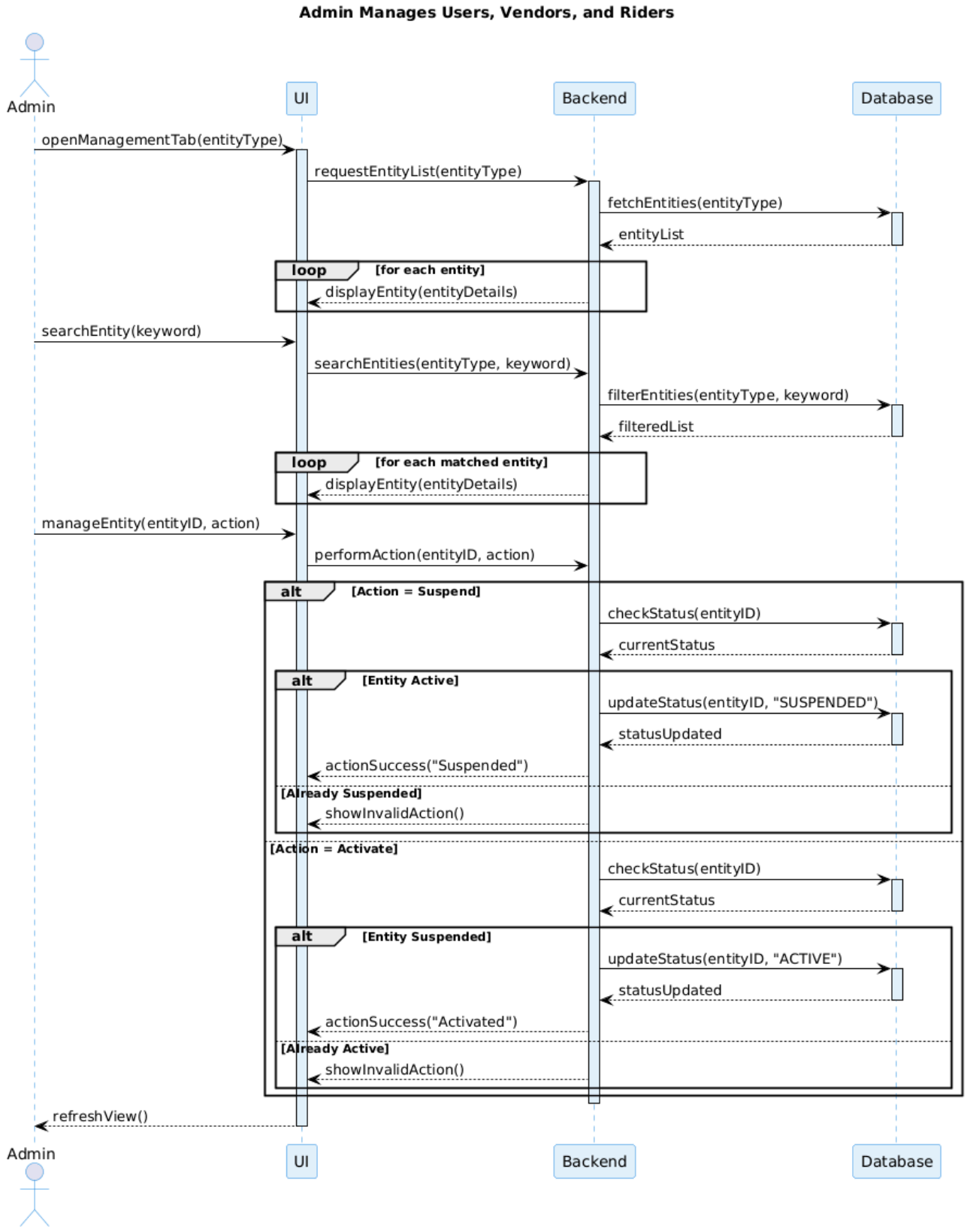
3.3.13 Rider Report Delivery issue and Requests Help



3.3.14 Admin Handles Complaints

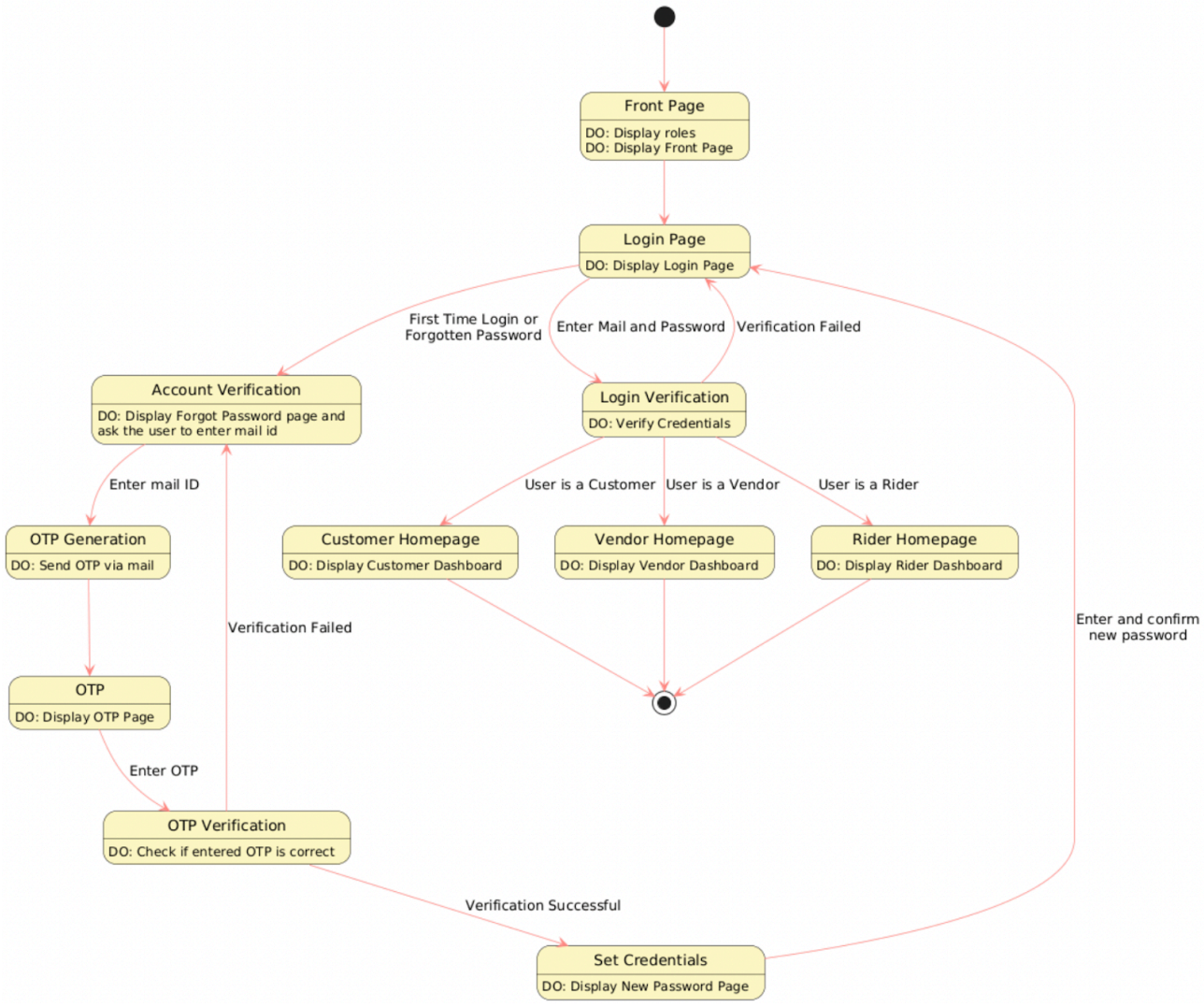


3.3.15 Admin Manages Users, Vendors, and

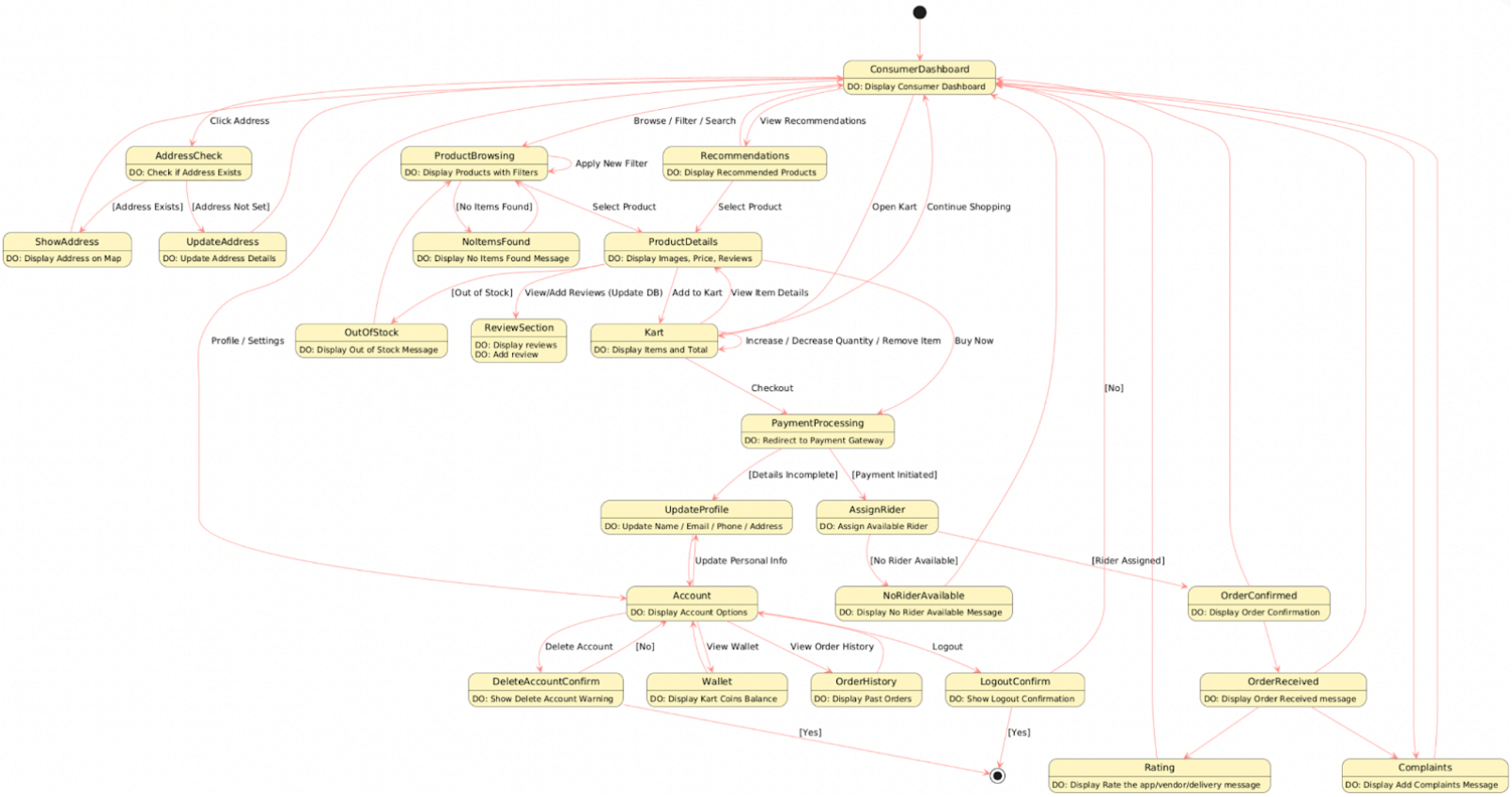


3.4 State Diagrams

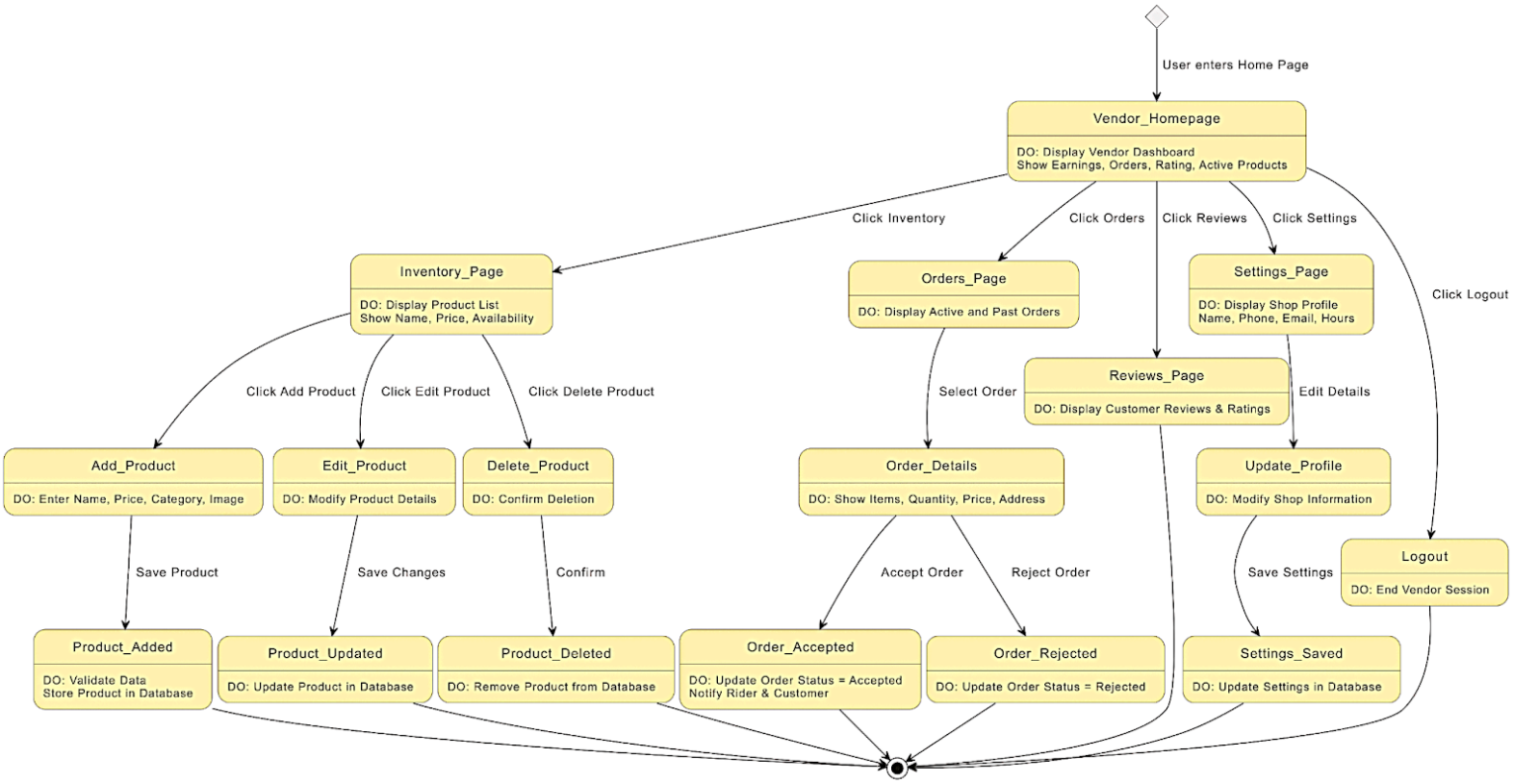
3.4.1 LOGIN AND DASHBOARD



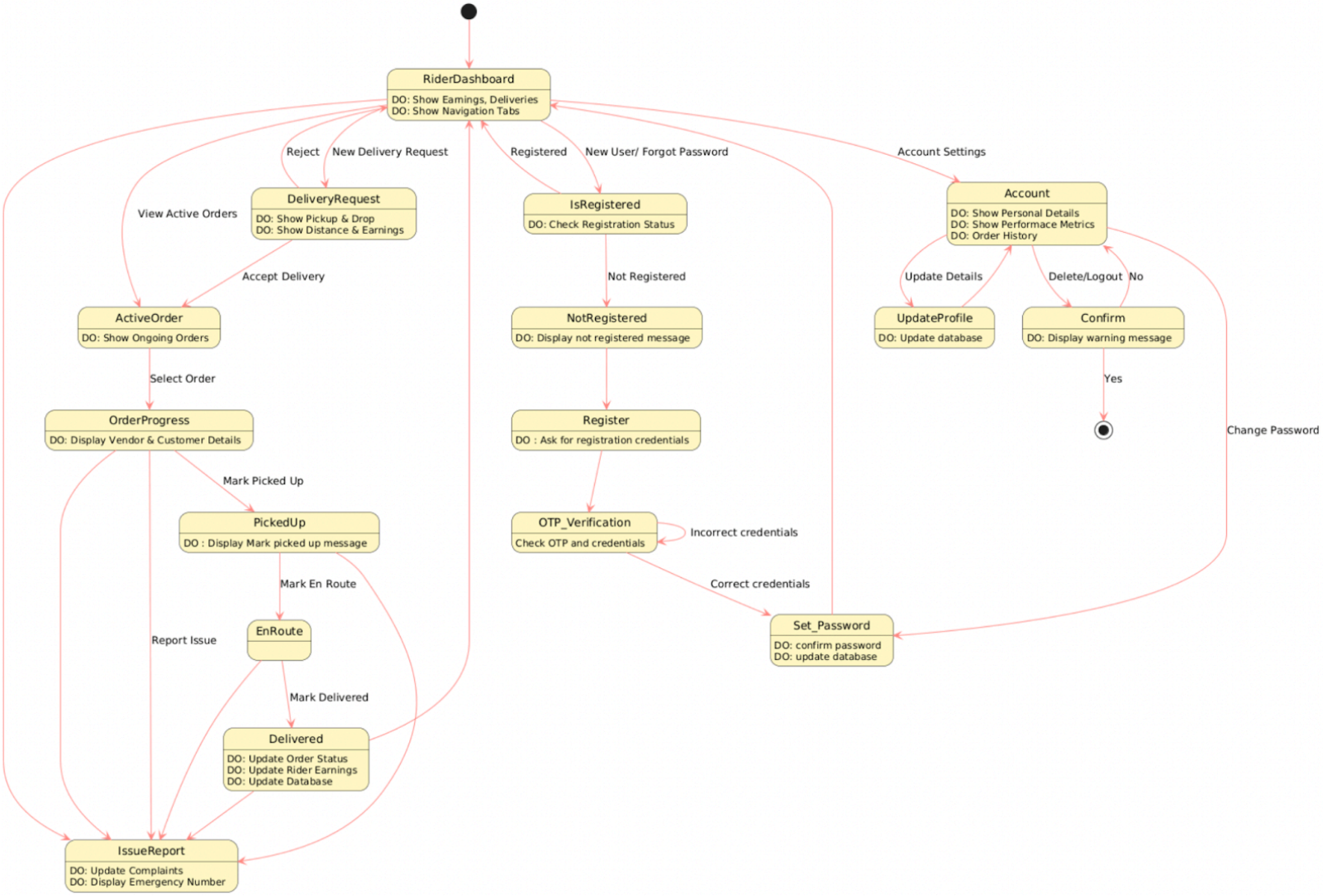
3.4.2 CONSUMER DASHBOARD



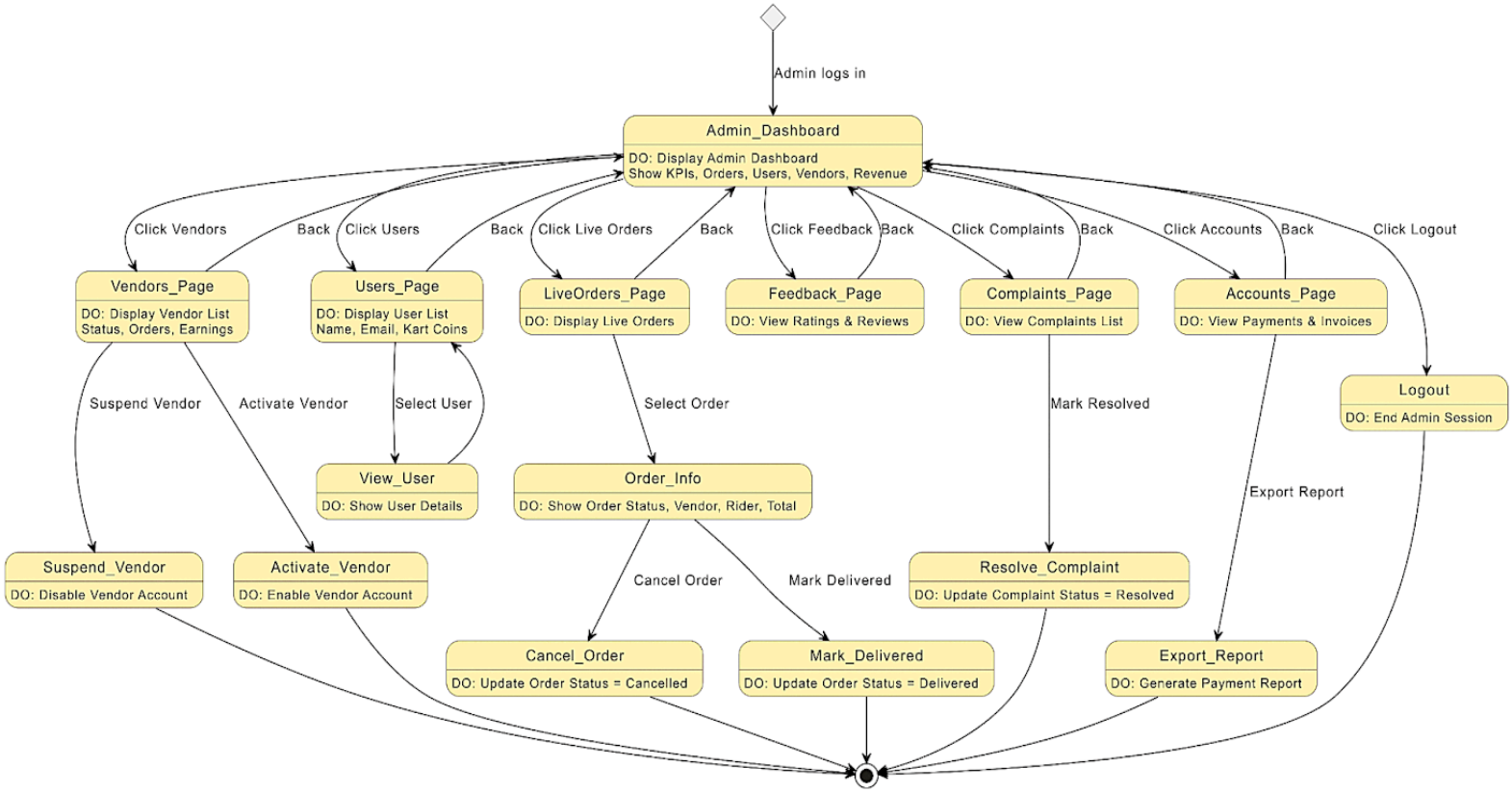
3.4.3 VENDOR'S DASHBOARD



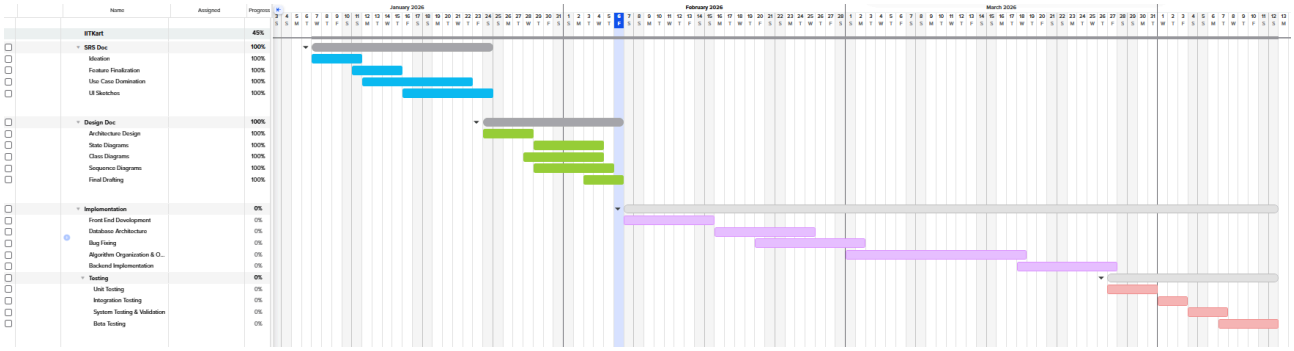
3.4.4 RIDER'S DASHBOARD



3.4.5 ADMIN'S DASHBOARD



4 Project Plan



MEMBERS	TASK
Virendra Kala	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, ALPHA TESTING
Sneha Kumar	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, UNIT TESTING
Praveen	BACKEND DEVELOPMENT, CODE IMPROVEMENT, ALPHA TESTING
Mahi Mittal	FULL STACK IMPLEMENTATION, CODE IMPROVEMENT, UNIT TESTING
Pranjali Deshpande	FULL STACK IMPLEMENTATION, SYSTEM TESTING, BETA TESTING
Hardik Tiwari	BACKEND DEVELOPMENT, INTEGRATION TESTING, ALPHA TESTING
Mayukh Chowdhury	FRONTEND DEVELOPMENT, UNIT TESTING, ADDRESSING FEEDBACK
Palak Bandhu	FRONTEND DEVELOPMENT, SYSTEM TESTING, ADDRESSING FEEDBACK
Yash Dabi	BACKEND DEVELOPMENT, INTEGRATION TESTING, CODE IMPROVEMENT
Shreshthraj Bhidodiya	FRONTEND DEVELOPMENT, MANUAL FOR BETA TESTING, BETA TESTING

Appendix A - Group Log

DATE	TOPICS DISCUSSED
28.01.2026	First discussion meeting for Design Document, initial deliverables were understood and work distributed
01.02.2026	Class Diagrams and Sequence Diagrams were proofread , changes suggested and edits made
02.02.2026	All Interfaces prepared and finalised , Architecture design model discussed
02-04.02.2026	Everyone Individually worked on their respective divisions , Designs and Context , Architecture Diagrams completed.Use Cases written down.
06.02.2026	Final meet before submission.Project Plan and division of further tasks decided. Design Document proofread.