**Introduction**

**Background**

In the era of digital transformation, urban mobility and last-mile delivery services are increasingly dependent on real-time logistics, transparency, and ease of use. Ethiopia, like many developing countries, is witnessing a rapid shift in how goods are transported and delivered. Despite this momentum, there remains a lack of structured platforms that connect customers and delivery motorists efficiently.

**Moteregna** is a smart delivery system that aims to digitize and streamline the package delivery process using mobile and web technologies. The platform connects customers who want to send packages with available registered motorists, while allowing admins and super admins to monitor deliveries, pricing, and motorist status.

## ****Statement of the Problem****

Currently, most delivery services in Ethiopia operate manually or through disconnected platforms like messaging apps. This leads to:

* Lack of real-time tracking
* No systematic delivery status updates
* Unverified or unregistered delivery personnel
* Poor transparency in pricing
* No notification or feedback mechanisms
* Difficulty managing multiple actors (customers, motorists, admins)

These limitations hinder trust, efficiency, and scalability of local delivery services.

## ****The Project's Purpose****

The main purpose of the Moteregna system is to offer an organized platform that facilitates communication and coordination between customers and registered motorists. It also empowers administrators to control delivery prices, locations, and track all users in the system in real time.

## ****Objectives****

**General Objective**

To develop a web and mobile-based delivery system that manages package delivery operations, user authentication, and real-time tracking using modern technologies.

**Specific Objectives**

* To allow customers to register, request delivery, and track packages
* To enable motorists to register, go online/offline, and accept deliveries
* To enable admins and super admins to manage users, pricing, and monitor deliveries
* To implement a notification system for delivery updates
* To provide secure login and role-based access control
* To ensure deliveries are traceable with timestamps and location data

## ****Scope and Limitations****

**Scope of the Project**

This project includes:

* Registration for customers, motorists, and admins
* Package creation and assignment
* Delivery status updates: Pending → Assigned → Picked → In Transit → Delivered
* Role-based access: Admin, Super Admin, Motorist and customers
* Location tracking of motorists
* Notification handling for both customers and motorists
* Price calculation based on distance and time

**Limitations of the Project**

* Real-time tracking depends on internet availability and GPS accuracy
* No payment gateway integration in this version (may be added in future)
* The motorist must be online for real-time tracking to function

**Project Tools**

**Coding Environment (IDE)**

* Visual Studio Code (VS Code) – for both web and back-end development.
* Android Studio – for Flutter mobile development.

**Frontend Development**

* Next.js – for building the responsive and optimized web client interface.
* Flutter– for building the cross-platform mobile application with a focus on performance and beautiful UI.

**Flutter Key Libraries**:

* **GetX** – for efficient state management and routing.
* **GetStorage** – for lightweight local storage handling.
* **Flutter Secure Storage** – for securely storing sensitive data like tokens.
* **HTTP** – for secure API handling and communication with the back-end.

**Backend Development**

* **Next.js API Routes** – used as the backend service layer.
* PostgreSQL – used as the relational database to store application data.
* **Prisma ORM** – used for database modeling, querying, and schema migrations.
* **JWT (JSON Web Token)** – for secure authentication and data encryption using JWT\_SECRET\_KEY.
* **Firebase Admin SDK** – for sending real-time push notifications to both customer and motorist apps using Firebase Cloud Messaging (FCM).
* **Hosted on a VPS (Virtual Private Server)** – to ensure full control over deployment, custom configurations, and secure API access.

**Keeping Track of Changes**

* **Git** – for version control.
* **GitHub** – for codebase collaboration and deployment pipelines.
* **Feature Branching** – used to isolate features and reduce merge conflicts during development.

**Backend Development**

* Next.js
* PostgreSQL for relational database
* Prisma ORM for database modeling and querying

**Keeping Track of Changes**

* GitHub for codebase collaboration
* Git branches for feature isolation

## ****1.7 Methodology****

**System Development Life Cycle (SDLC)**

This project follows the **Waterfall Development methodology** given its sequential and linear nature.

Phases:

* Requirement Gathering (Complete upfront)
* Design (Based on finalized requirements)
* Implementation (Build based on design)
* Testing (Performed after implementation)
* Deployment (Released as a single, complete unit)
* Maintenance (Post-deployment support)

**Requirements Gathering Methodology**

* Interviews with logistics personnel
* Review of existing local delivery apps
* Brainstorming with technical team

**System Design Paradigm: Object-Oriented Programming (OOP)**

The system uses OOP principles:

* Encapsulation for secure data handling
* Inheritance for role management
* Polymorphism for request handling flexibility
* Composition for modular code

# Proposed System

**Overview**

The proposed system, **Moteregna**, is a smart, web and mobile-based delivery management platform designed to address the inefficiencies of traditional package delivery methods. It aims to provide a streamlined, secure, and scalable solution for both customers and service providers in Ethiopia’s evolving urban logistics landscape.

Moteregna connects customers with available delivery motorists through a user-friendly interface, offering real-time updates and comprehensive delivery tracking. The system is structured to support different user roles — including **Customers**, **Motorists**, **Admins**, and **Super Admins** — each with role-specific features and permissions.

**Key Components of the Proposed System Include:**

* **Secure User Registration & Authentication**: The system ensures only verified users (customers and motorists) can register and use the platform through robust authentication mechanisms.
* **Real-Time Motorist Location Tracking**: Using GPS integration, admins and customers can monitor the live location of delivery motorists, enabling transparency and trust.
* **Delivery Management & Status Updates**: Each package delivery is assigned a unique status (PENDING, PICKED\_UP, IN\_TRANSIT, DELIVERED), and both customers and admins can view the delivery progress in real time.
* **Dynamic Price Calculation**: Based on configurable parameters such as distance (per kilometer), time (per minute), and base delivery fees, the system calculates total delivery costs. Prices can be managed and customized by admins.
* **Role-Based Access Control (RBAC)**:
  + **Customers** can request deliveries, track packages, and receive notifications.
  + **Motorists** receive package assignments, update delivery statuses, and manage availability.
  + **Admins** manage users, pricing rules, motorist verification, and delivery logs.
  + **Super Admins** have global access to the system for monitoring platform-wide performance and generating insights.
* **Package Categorization & Special Instructions**: Customers can define package types, weights, quantities, and provide special delivery instructions to ensure accurate service.
* **Notification System**: Push notifications are sent to users regarding delivery updates, status changes, and important announcements.
* **Analytics & Reporting Dashboard**: Admins and Super Admins have access to dashboards that show statistics on deliveries, earnings, motorist performance, and system usage patterns.
* **Responsive Design & Mobile Accessibility**: The system is accessible on both web and mobile devices, ensuring ease of use for all users regardless of their preferred platform.
* **Safety and Compliance**: Required documentation for motorists (driver's license, business permits) is securely uploaded, reviewed, and stored to maintain service quality and legal compliance.

Moteregna ultimately serves as a bridge between technology and transportation logistics, enhancing the reliability, visibility, and efficiency of package delivery services in Ethiopia. By digitizing the entire process, it eliminates ambiguity, fosters accountability, and empowers users with greater control over their delivery experience.

**Functional Requirement**

**All Users (Customers and Motorists):**

* **FR-101: User Registration:** The system allows users (customers and motorists) to register for an account using valid credentials and required details.
* **FR-102: User Login and Profile Management:** Registered users can log in securely and manage their personal information, password, and settings.
* **FR-103: Delivery Request Creation:** Customers can create a delivery request by entering pickup, drop-off locations, and package details.
* **FR-104: Delivery Status Tracking:** All users can view the status of ongoing and past deliveries (e.g., pending, picked up, in transit, delivered).
* **FR-105: Real-Time Notification:** The system provides users with real-time notifications regarding delivery updates or status changes.
* **FR-106: Delivery History Access:** Customers and motorists can view their respective delivery histories and related details.
* **FR-107: Contact Support:** Users can contact support through the platform for assistance or inquiries.

**Customers:**

* **FR-201: Package Detail Submission:** Customers can specify the package type, weight, quantity, and provide special instructions.
* **FR-202: Delivery Price Calculation:** The system automatically calculates the delivery fee based on the base rate, distance, and time.
* **FR-203: Available Motorist Matching:** The system assigns or allows customers to choose from available nearby motorists.
* **FR-204: Payment Confirmation:** Customers can confirm and proceed with delivery after reviewing price and motorist assignment.
* **FR-205: Delivery Feedback Submission:** Customers can rate the motorist and provide feedback after the delivery is complete.

**Motorists:**

* **FR-301: Availability Toggle:** Motorists can set their status as available or unavailable to receive delivery assignments.
* **FR-302: Delivery Acceptance or Rejection:** Motorists receive notifications of new delivery requests and can accept or reject them.
* **FR-303: Status Update on Delivery:** Motorists can update the delivery status at each stage (picked up, in transit, delivered).
* **FR-304: Delivery Start:** Motorists can start a delivery by viewing the pickup and drop-off locations on a map.
* **FR-304: Delivery Complete:** Motorists can complete a delivery after arriving at the drop-off location
* **FR-305: Daily/Weekly/Monthly Statistics:** Motorists can view a summary of their delivery performance and earnings over different periods.

**Admin:**

* **FR-401: User Account Review and Management:** Admins can view, verify, or deactivate user accounts (customers and motorists).
* **FR-402: Motorist Document Verification:** Admins review uploaded documents for motorist account activation.
* **FR-403: Pricing Configuration:** Admins can set and update pricing rules for distance, time, and base delivery cost.
* **FR-404: Delivery Monitoring Dashboard:** Admins can monitor delivery activities and statuses in a centralized dashboard.
* **FR-405: Delivery and User Analytics:** Admins can review statistics and analytics on deliveries, users, and system usage.

**Super Admin:**

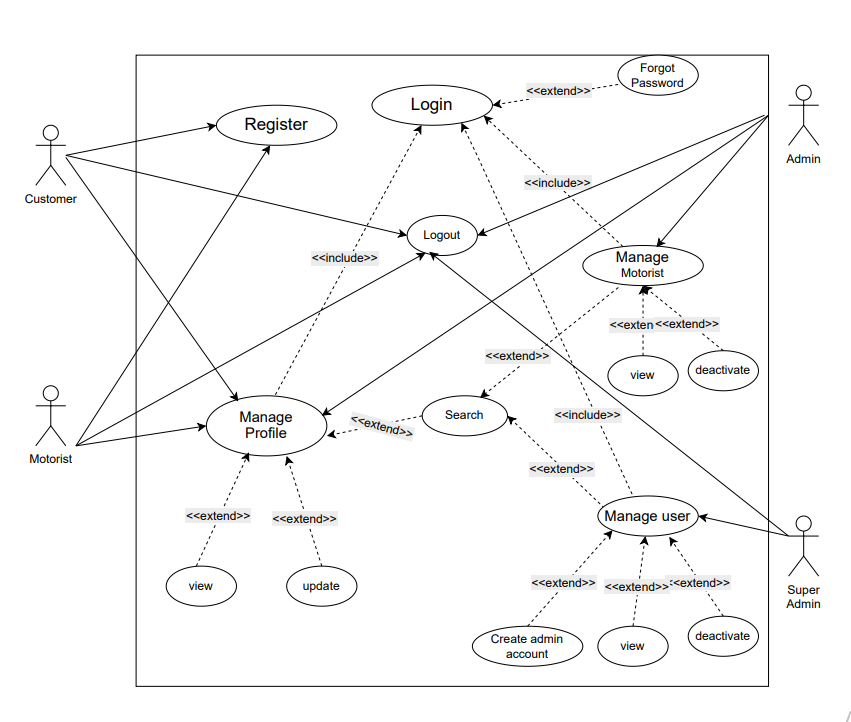
* **FR-501: Admin Account Creation:** Super Admins can create and manage Admin accounts with role-specific permissions.
* **FR-502: Global System Monitoring:** Super Admins can view platform-wide statistics, including delivery trends, user growth, and financial metrics.
* **FR-503: Initial Pricing Rule Setup:** Super Admins can define default pricing configurations for the platform.
* **FR-504: Platform-wide Audit Logs:** Super Admins can access logs of all user and admin activities for audit purposes.

**Non-Functional Requirements**

* **NFR-401: Concurrent User Support:** The system must support at least 100 concurrent users without performance degradation.
* **NFR-402: Multi-Device Responsiveness:** The platform should be fully responsive and provide optimal user experience on both mobile and desktop devices.
* **NFR-403: Real-Time GPS Tracking:** GPS location updates must occur every 15 seconds during active deliveries for accurate tracking.
* **NFR-404: Session Security:** Login tokens must automatically expire after 24 hours to enhance security and session control.
* **NFR-405: Password Security:** All user passwords must be securely hashed using the bcrypt algorithm before being stored in the database.
* **NFR-406: System Uptime:** The system should maintain a monthly uptime of at least 99.5% to ensure high availability.
* **NFR-407: Performance:** All pages within the system should load in under 2 seconds under normal network conditions.
* **NFR-408: Audit Logging:** The system must log key user and admin activities for accountability and audit purposes.

# 2.3 Use Case Packages

## The Delivery Platform system, now organized into the revised logical packages: User Management, Delivery Management, and Platform Management. This structured approach categorizes the system's functionalities, clearly separating concerns related to user interactions, core delivery operations, and administrative oversight, to provide a comprehensive and organized understanding of the platform's capabilities.

Figure 2.4 User management Package.

## 

Figure 2.4 Delivery management Package.

## 

Figure 2.4 platform management Package.

## Use case Description Table

Table 2:1 Create Account Use Case

|  |  |
| --- | --- |
| Use Case Name | Register User |
| Use Case Id | UC-101 |
| Actor | Customer, Motorist |
| Description | Allows a new user (Customer or Motorist) to register and create a new account on the platform. |
| Precondition |  The user does not have an existing account on the platform with the same phone number.   Motorists must have a valid license and vehicle details.   The user wants to register for the first time. |
| Basic course of action | 1. User visits the registration page (Customer or Motorist). 2. System displays a form: 3. **Customer**: First Name, Last Name, Phone, Password, FIDA 4. **Motorist**: First Name, Middle Name (optional), Last Name, Phone, Password, Gender (optional), Birthdate (optional), Address (optional), License Number, Vehicle Model, Plate Number, and file uploads. 5. User submits the form. 6. System validates required fields, checks phone and license number uniqueness, and formats. 7. If valid, password is hashed and account is created. 8. System returns a success message. 9. User is redirected to the login page. |
| Alternative course of action |  **A1. Missing/Invalid Fields**: System shows errors and returns to validation (Step 4).   **A2. Duplicate Phone/License**: System shows conflict error and returns to form submission (Step 3).   **A3. File Upload Fails (Motorist)**: System shows upload error and returns to form submission (Step 3). |
| Post Condition | The user has a new account on the platform and can log in to access features. |

Table 2:1 Login & Session Management

|  |  |
| --- | --- |
| Use Case Name | Login & Session Management |
| Use Case Id | UC-102 |
| Actor | Customer, Motorist |
| Description | Allows any registered user to log in securely and establishes a session with a JWT token. |
| Precondition |  The user has a registered account with a valid phone number and password.   The user wants to authenticate and start a session. |
| Basic course of action | 1. User opens the login page and enters phone number and password.  2. System validates input and checks user by phone number.  3. If user exists, system verifies password.  4. On success:   * Generates new session ID and logs out previous session if any. * Creates a JWT (valid for 30 days) with user info and session ID. * Sets JWT as a secure, HTTP-only cookie.   5. Returns user data (excluding password), access token, and motorist info if applicable.  6. User is now logged in and can access authorized features. |
| Alternative course of action |  **A1. Not POST Method**: Returns "Method not allowed."   **A2. Phone Not Found**: Returns "User not found."   **A3. Wrong Password**: Returns "Invalid password."   **A4. Invalid Token**: Returns "Not authenticated" or "Invalid token." |
| Post Condition | * The user is authenticated with a valid session token stored in an HTTP-only cookie. * The system tracks the user's session status. |

Table 2:1 Manage Profile

|  |  |
| --- | --- |
| Use Case Name | Manage Profile |
| Use Case Id | UC-103 |
| Actor | All Users |
| Description | Allows authenticated users to view and update their personal profile information, including contact details and password. |
| Precondition |  The user is authenticated via a valid JWT token.   The user has an existing account and valid session. |
| Basic course of action | 1. User opens the Profile section.  2. System verifies JWT and checks user role.  3. System fetches and displays current profile (e.g., name, phone, address).  4. User edits and submits updated info.  5. System validates and updates the database.  6. System confirms successful update.  **Extension – Change Password:**   * User selects "Change Password" and enters current + new password. * System verifies current password, validates and hashes the new one, updates it, and returns success. |
| Alternative course of action |  **A1. Invalid Method**: Returns “Method not allowed.”   **A2. Missing/Invalid Token**: Returns “Unauthorized.”   **A3. Unauthorized Role**: Returns “Forbidden.”   **A4. Profile Not Found**: Returns “User not found.”   **A5. Wrong Current Password**: Returns “Current password is incorrect.” |
| Post Condition |  The user’s profile information is updated successfully.   The user’s password is securely changed if applicable. |

Table 2:1 View Delivery History

|  |  |
| --- | --- |
| Use Case Name | View Delivery History |
| Use Case Id | UC-104 |
| Actor | **Customer Motorist** |
| Description | Allows customers and motorists to view their past delivery records. Each user sees deliveries relevant to their account, along with details such as status, distance, delivery time, locations, and associated entities. |
| Precondition |  The user is authenticated via a valid JWT token.   The user has an existing delivery history (DELIVERED or CANCELLED). |
| Basic course of action | **For Customer:**  1. Customer opens "Delivery History."  2. System verifies JWT and CUSTOMER role.  3. Retrieves paginated deliveries where phone matches and status is DELIVERED or CANCELLED.  4. Returns delivery details (status, locations, motorist info, package, date, pagination).  **For Motorist:**  1. Motorist opens "Delivery History."  2. System verifies JWT and matches motorist.userId.  3. Validates motoristId, page, and query params.  4. Retrieves paginated deliveries with distance, locations, status, fee, timestamps, and customer phone.  5. Returns deliveries with pagination. |
| Alternative course of action |  **A1. No Token**: 401 Unauthorized – No token provided.   **A2. Invalid Token**: 401 Unauthorized – Invalid token.   **A3. Unauthorized Role**: 403 Forbidden.   **A4. Motorist ID Mismatch**: 403 Forbidden – No permission.   **A5. User Not Found**: 404 Not Found.   **A6. Invalid Query Params**: 400 Bad Request with errors. |
| Post Condition |  The user receives a structured, paginated list of delivery records.   Each record includes location, delivery status, time, and other related details. |

Table 2:1 Create Account Use Case

|  |  |
| --- | --- |
| Use Case Name | Create Package Request |
| Use Case Id | UC-201 |
| Actor | Customer |
| Description | This use case allows a customer to initiate a delivery request by providing pickup and drop-off locations and linking it to a package with all necessary details. |
| Precondition |  Customer is authenticated via JWT.   Customer has an active account.   Location information is available and accurate. |
| Basic course of action | 1. Customer opens the delivery request form. 2. Enters pickup and drop-off addresses with GPS coordinates. 3. Fills in package description, category, weight, quantity, and special instructions. 4. Form is submitted 5. Backend validates the JWT and user role. 6. Backend parses and validates input . 7. Pickup and drop-off locations are saved to the database. 8. A new package is created in the database with PENDING status. 9. System finds nearby motorists. Via UC-204 10. Sends FCM push notifications with delivery details and pricing. 11. System responds with:     * Success message     * Created package data |
| Alternative course of action | **A1**: Invalid JWT or role  **A2**: Missing or invalid pickup/drop-off data  **A3**: Category not found in DB  **A4**: Internal DB error |
| Post Condition |  A new delivery request is stored and associated with the customer.   Notification is sent to nearby available motorists. |

Table 2:1 Create Account Use Case

|  |  |
| --- | --- |
| Use Case Name | Specify Package Details |
| Use Case Id | UC-202 |
| Actor | Customer |
| Description | Allows the customer to define what package they are sending, including the category, weight, quantity, and any special instructions for handling |
| Precondition |  Customer is authenticated and allowed to make a delivery.   The category exists in the system. |
| Basic course of action | 1. Customer opens the package input form. 2. Inputs:    * **Description** (required)    * **Category ID** (required, from predefined categories)    * **Weight** and **Quantity** (optional but recommended)    * **Special Instructions** (optional) 3. The data is validated using the zod schema. 4. The backend:    * Verifies the category exists    * Stores the package with the pickup and drop-off locations    * Links it to the authenticated customer |
| Alternative course of action |  **A1**: Missing or invalid required fields   **A2**: Category not found |
| Post Condition |  Package details are stored in the system.   Linked to the customer and ready for matching with a motorist. |

Table 2:1 Accept Delivery Request

|  |  |
| --- | --- |
| Use Case Name | Accept or Reject Delivery Request |
| Use Case Id | UC-302 |
| Actor | Motorist |
| Description | Allows a motorist to receive a delivery request, view its details, and either accept or reject the delivery assignment. Upon acceptance, the package status is updated, and the customer is notified via push notification |
| Precondition |  Motorist is authenticated via JWT token.   Package with the provided ID exists.   Motorist exists in the system.   Motorist has not accepted another package for the same delivery (optional constraint depending on implementation). |
| Basic course of action | 1. System notifies available motorists.  2. Motorist views delivery details.  3. Motorist accepts (POST: motoristId, packageId, status).  4. System validates token, motorist, package, and ownership.  5. Updates package status to "ASSIGNED" and links motoristId.  6. Sends FCM notification to customer with motorist info.  7. Returns: "Package assigned successfully." |
| Alternative course of action |  **A1. Missing/Invalid Token**: 401 Unauthorized   **A2. Package Not Found**: 404 Not Found   **A3. Already Assigned**: Conflict – update blocked   **A4. Unauthorized Motorist**: 403 Forbidden   **A5. Invalid Request Body**: 400 Bad Request   **A6. FCM Fails**: Logs error, assignment continues   **A7. Rejected by Motorist**: Status set to "REJECTED", next motorist notified |
| Post Condition | * Package is marked as assigned (status = "ASSIGNED") with motorist details updated. * Customer is notified of assignment. |

Table 2:1 Start Delivery

|  |  |
| --- | --- |
| Use Case Name | Start Delivery |
| Use Case Id | UC-303 |
| Actor | Motorist |
| Description | Allows a motorist to officially start the delivery process after receiving a package. The system verifies availability and online status, checks for active deliveries, assigns a delivery fee, and updates the delivery and motorist status accordingly. |
| Precondition | * Motorist is authenticated via a valid JWT token. * Motorist is marked as **online** and **available**. * No other active delivery ( status !== DELIVERED or CANCELLED) exists for the motorist. * System contains at least one valid pricing configuration. |
| Basic course of action | 1. Motorist initiates "start delivery."  2. Sends POST: motoristId, customerPhone, source info, startTime.  3. System authenticates via JWT.  4. Verifies motorist exists, is online, and not on another delivery.  5. Creates delivery with source location, fee, status "PENDING", and start time.  6. Sets motorist isAvailable to false.  7. Returns deliveryId, initialPrice, per km/min rate. |
| Alternative course of action |  **A1. Missing/Invalid Token**: 401 Unauthorized   **A2. Unauthorized Motorist**: 403 Forbidden   **A3. Motorist Offline**: 400 – Motorist is offline   **A4. Not Available**: 400 – Motorist is not available   **A5. Ongoing Delivery**: 400 – Active delivery exists   **A6. No Pricing Found**: 404 Not Found   **A7. Validation Errors**: 400 with details   **A8. Server Error**: 500 Internal Server Error |
| Post Condition |  Delivery created with status **"PENDING"**   Motorist marked **unavailable**   Pricing and start location saved   System ready for **live location updates** |

Table 2:1 Complete Delivery

|  |  |
| --- | --- |
| Use Case Name | Complete Delivery |
| Use Case Id | UC-304 |
| Actor | Motorist |
| Description | Enables a motorist to complete a delivery by submitting total distance, cost, and status. The system records the end location, updates the delivery, and marks the motorist as available again.. |
| Precondition |  Motorist is authenticated with a valid JWT token.   Delivery must exist and not be already marked as DELIVERED or CANCELLED.   Delivery must be assigned to the motorist.   Destination location details must be provided for status DELIVERED. |
| Basic course of action | 1. Motorist sends POST: deliveryId, distance, cost, status, endTime, destination info.  2. System validates input (Zod) and authenticates via JWT.  3. Fetches delivery and checks if it exists, has a motorist, and is active.  4. If status is "DELIVERED", saves end location and links it.  5. Updates delivery with distance, cost, end time, status, and location.  6. Sets motorist isAvailable to true if delivery is done.  7. Returns success with updated delivery details. |
| Alternative course of action |  **A1. Missing/Invalid Token**: 401 Unauthorized   **A2. Delivery Not Found**: 404 Not Found   **A3. No Motorist Assigned**: 400 – No motorist assigned   **A4. Already Finalized**: 400 – Delivery already completed/cancelled   **A5. Missing End Location Fields**: 400 – Validation failed   **A6. Server Error**: 500 Internal Server Error |
| Post Condition |  Delivery is marked as DELIVERED or other updated status.   End location is saved.   Motorist is marked as available if delivery is finished. |

Table 2:1 Update Delivery Status

|  |  |
| --- | --- |
| Use Case Name | Update Delivery Status |
| Use Case Id | UC-305 |
| Actor | Motorist |
| Description | Allows a motorist to update the delivery status. In this case, the motorist marks the delivery as CANCELLED, and the system updates the record and resets the motorist’s availability. |
| Precondition | * Motorist must be authenticated via a valid JWT token. * Delivery must exist and be assigned to the authenticated motorist. * Delivery must not already be marked as DELIVERED or CANCELLED. |
| Basic course of action | 1. Motorist sends PATCH with `deliveryId` and `status = "CANCELLED"`.  2. System authenticates via JWT and validates input (Zod).  3. Verifies delivery exists and motorist is assigned.  4. Updates delivery status to CANCELLED and sets motorist as available.  5. Returns confirmation with delivery details. |
| Alternative course of action | * **A1. No Token**: 401 Unauthorized – No token provided * **A2. Invalid Token**: 401 Unauthorized – Invalid token * **A3. Delivery Not Found**: 404 Not Found * **A4. Not Assigned to Motorist**: 400 Bad Request * **A5. Unauthorized Motorist**: 403 Forbidden * **A6. Validation Failed**: 400 Validation failed * **A7. Database Error**: 500 Internal Server Error |
| Post Condition |  Delivery status is updated to CANCELLED.   Motorist is marked as available for future assignments. |

Table 2:1 Create and Manage Admin Accounts

|  |  |
| --- | --- |
| Use Case Name | Create and Manage Admin Accounts |
| Use Case Id | UC-501 |
| Actor | Super admin |
| Description | Allows Super Admin to create and manage Admin accounts by registering new Admins, assigning roles, and setting or updating their permissions. |
| Precondition |  Super Admin is authenticated and authorized.   Admin roles and permissions schema exists in the system. |
| Basic course of action | 1. Super Admin opens Admin Management panel.  2. Clicks "Create New Admin."  3. Enters name, email, and selects or creates a role.  4. Assigns permissions via checkboxes or role-based settings.  5. Clicks Create — system validates, saves admin, and sends invite or temp password.  6. Super Admin can also view, update, deactivate, or reactivate Admin accounts. |
| Post Condition |  Admin account is created and available for login.   Permissions are enforced during login and access. |

# 2. 3 Sequence diagram

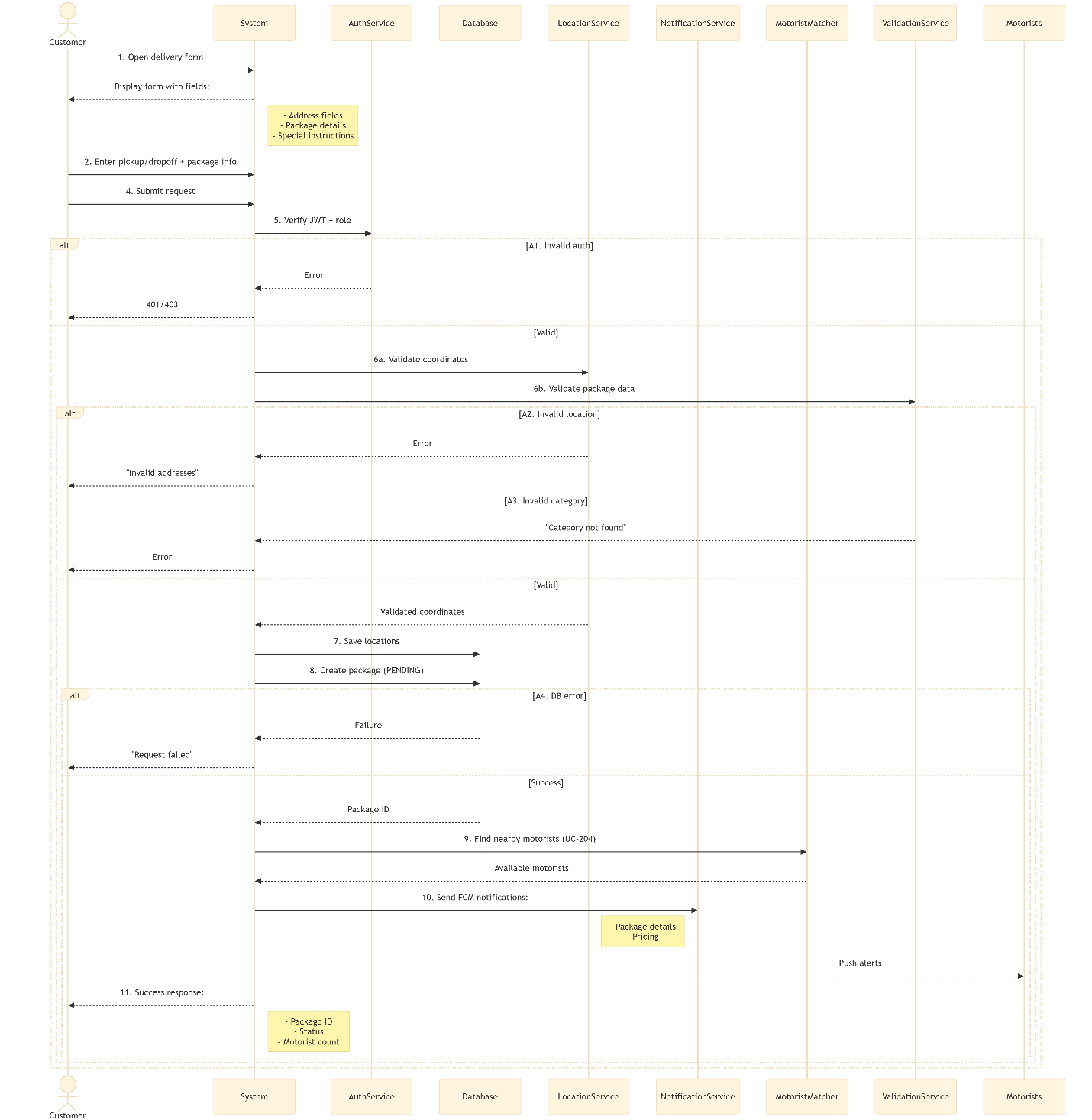


Figure 2:1

Figure 2:1

Figure 2:1

Figure 2:1

# Class diagram

# ****Software Architecture****

## ****1. Introduction****

The **Delivery Management System**, designed to support customers, motorists, administrators, and super administrators with secure, scalable, and real-time functionality.

The architecture follows a **layered approach**, ensuring separation of concerns, modularity, and ease of maintenance while leveraging modern technologies for optimal performance.

## ****2. Architectural Overview****

The system is structured into three primary layers:

1. **Presentation Layer (Frontend)** – User-facing interfaces.
2. **Application Layer (Backend Services)** – Business logic, API handling, and security.
3. **Data Layer (Database & Storage)** – Persistent data storage and retrieval.

The system adopts:

* **Client-Server Architecture** (Flutter ↔ Next.js API communication).
* **Service-Oriented Architecture (SOA)** (Modular backend services for different user roles).
* **3-Tier Architecture** (Clear separation between UI, logic, and data)

## ****Technology Stack****

|  |  |  |
| --- | --- | --- |
| **Layer** | **Technology** | **Purpose** |
| **Frontend** | Flutter | Mobile & Web app for customers, motorists, and admins. |
| **Backend API** | Next.js | RESTful API endpoints for business logic, authentication, and data processing. |
| **Database** | PostgreSQL | Relational database for structured data |
| **Auth/Security** | JWT | Secure user authentication and session management. |
| **Notifications** | Firebase | Real-time alerts for delivery updates and system notifications. |

**Key Architectural Components**

**Client-Server Architecture**

* Flutter client interacts with Next.js backend via HTTP/HTTPS APIs.
* Decouples UI from business logic for scalability and independent development.

**Service-Oriented Architecture (SOA)**

* Backend split into independent services:
  + **User Service:** registration, login, profile, auth
  + **Delivery Service:** request handling, status updates, tracking
  + **Payment Service:** payment processing, transaction management
  + **Notification Service:** real-time notifications via FCM
  + **Admin Service:** user reviews, document verification, pricing, analytics

**3-Tier Architecture**

1. **Presentation Layer (Flutter):** UI rendering, user input, API calls, multi-device support.
2. **Application Layer (Next.js):** Business logic, JWT auth & RBAC, audit logging, API security (validation, rate limiting, protection).
3. **Data Layer (PostgreSQL):** Stores users, deliveries, payments, configs; passwords hashed with bcrypt.

# Database Schema

The database schema for the Delivery Management System describes each major table (model) and its attributes (columns), including their data types and purpose, as well as how different tables are connected through relationships.

## 1. User Management Data

**Table: User**

Represents all users of the system (Super Admins, Admins, Motorists). Customers are managed separately.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the user. |
| firstName | String | User's first name. |
| middleName | String? | User's middle name (optional). |
| lastName | String? | User's last name (optional). |
| phone | String? | User's phone number (optional). |
| password | String? | Hashed password for security (optional, if external auth). |
| gender | String? | User's gender (optional). |
| birthdate | DateTime? | User's date of birth (optional). |
| address | String? | User's physical address (optional). |
| role | Enum (Role)? | User's role: SUPERADMIN, ADMIN, MOTORIST. Defaults to MOTORIST. |
| profile | String? | Reference to a user's profile image or specific profile data |
| isLoggedIn | Boolean? | Indicates if the user is currently logged in. Defaults to false. |
| sessionId | String? | Identifier for the current login session (optional). |
| createdAt | DateTime? | Timestamp when the user account was created. |
| updatedAt | DateTime? | Timestamp of the last update to the user's data. |
| status | Boolean? | Account active status (e.g., active/deactivated). Defaults to true. |

* **Relationships:**
  + Has many Motorist records (if the user is a Motorist).
  + Has many Price records (if the user sets pricing, e.g., Admin).
  + Has many Notification records.

**Table: Customer**

Represents individual customers who use the delivery service.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the customer. |
| firstName | String | Customer's first name. |
| lastName | String | Customer's last name. |
| phone | String? | Customer's primary phone number (optional). |
| phonenumber | String? | Another phone number field, perhaps for specific contacts . |
| password | String | Hashed password for customer login. |
| address | String? | Customer's primary address (optional). |
| fcmToken | String? | Firebase Cloud Messaging token for push notifications |
| createdAt | DateTime | Timestamp when the customer account was created. |
| updatedAt | DateTime | Timestamp of the last update to customer data. |

* **Relationships:**
  + Has many Delivery records (deliveries initiated by this customer).
  + Has many Package records (packages created by this customer).
  + Has many Notification records.

**Table: Motorist**

Stores specific details about a motorist, linked to a general User account.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the motorist profile. |
| userId | String? | Foreign key linking to the User table. |
| licenseNumber | String? | Motorist's driving license number (optional). |
| vehicleModel | String | Model of the motorist's vehicle. |
| vehiclePlateNumber | String | Vehicle's license plate number. |
| Librephoto | String | File path or URL to the vehicle's registration photo. |
| driversLicencephotoFront | String | File path or URL to the front of the driver's license. |
| businessPermit | String | File path or URL to the motorist's business permit. |
| isAvailable | Boolean | Indicates if the motorist is available for new assignments. Defaults to true. |
| isOnline | Boolean | Indicates if the motorist app is currently online. Defaults to false. |
| currentLocationId | String? | Foreign key linking to the Location table for real-time tracking (optional). |
| fcmToken | String? | Firebase Cloud Messaging token for push notifications (optional). |
| createdAt | DateTime | Timestamp when the motorist record was created. |
| updatedAt | DateTime | Timestamp of the last update to motorist data. |

* **Relationships:**
  + Belongs to one User.
  + Has many Delivery records (deliveries performed by this motorist).
  + Has many Package records (packages assigned to this motorist).
  + Belongs to one Location (for currentLocation).

## 2. Delivery & Tracking Data

Table: Package

Details about individual packages being sent through the system.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the package. |
| description | String | Description of the package contents. |
| categoryId | String? | Foreign key linking to the PackageCategory table (optional). |
| weight | Float? | Weight of the package in a specified unit (optional). |
| quantity | Int? | Number of items in the package (optional). |
| specialInstructions | String? | Any special handling instructions for the package (optional). |
| status | Enum | Current status of the package. Defaults to PENDING. |
| createdAt | DateTime | Timestamp when the package record was created. |
| updatedAt | DateTime | Timestamp of the last update to package data. |
| customerId | String? | Foreign key linking to the Customer who sent the package. |
| motoristId | String? | Foreign key linking to the Motorist assigned to the package. |
| deliveryId | String? | Foreign key linking to the Delivery it belongs to. |
| pickupLocationId | String? | Foreign key linking to the Location for pickup. |
| dropoffLocationId | String? | Foreign key linking to the Location for drop-off. |

* **Relationships:**
  + Belongs to one PackageCategory (optional).
  + Belongs to one Customer.
  + Belongs to one Motorist.
  + Belongs to one Delivery.
  + Belongs to one Location (for pickup).
  + Belongs to one Location (for drop-off).

**Table: PackageCategory**

Defines categories for packages (e.g., "Documents", "Fragile").

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the package category. |
| name | String? | Name of the category (e.g., "Documents"). |
| description | String? | A brief description of the category (optional). |
| createdAt | DateTime | Timestamp when the category was created. |
| updatedAt | DateTime | Timestamp of the last update to the category. |
| isActive | Boolean | Indicates if the category is currently active/selectable. Defaults to true. |

* **Relationships:**
  + Has many Package records.

**Table: Location**

Stores geographical location data, used for pickups, drop-offs, and motorist current locations.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the location. |
| name | String | A friendly name for the location (e.g., "Home", "Office"). |
| address | String | Full street address of the location. |
| latitude | Float | Geographical latitude coordinate. |
| longitude | Float | Geographical longitude coordinate. |
| createdAt | DateTime | Timestamp when the location record was created. |
| updatedAt | DateTime | Timestamp of the last update to location data. |

* **Relationships:**
  + Used as currentLocation by many Motorist records.
  + Used as startLocation by many Delivery records.
  + Used as endLocation by many Delivery records.
  + Used as pickupLocation by many Package records.
  + Used as dropoffLocation by many Package records.

**Table: Delivery**

Represents a single delivery instance, linking customers, motorists, and packages.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the delivery. |
| motoristId | String? | Foreign key linking to the Motorist assigned to this delivery (optional). |
| customerPhone | String? | Phone number of the customer associated with this delivery (optional). |
| startLocationId | String? | Foreign key linking to the Location of pickup. |
| endLocationId | String? | Foreign key linking to the Location of drop-off. |
| status | Enum | Current status of the delivery. Defaults to PENDING. |
| distance | Float? | Calculated distance of the delivery in kilometers/miles (optional). |
| fee | Float? | Final calculated fee for the delivery (optional). |
| startTime | DateTime? | Actual start time of the delivery (when picked up) (optional). |
| endTime | DateTime? | Actual end time of the delivery (when delivered) (optional). |
| createdAt | DateTime | Timestamp when the delivery record was created. |
| updatedAt | DateTime | Timestamp of the last update to delivery data. |

* **Relationships:**
  + Belongs to one Motorist.
  + Belongs to one Location (for startLocation).
  + Belongs to one Location (for endLocation).
  + Has many Package records.
  + Has many Customer records (likely through an implicit link via package or direct customer association).

## 3. Platform Management & Utilities Data

**Table: Price**

Defines pricing rules for deliveries, potentially set by Admin/Super Admin users.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the price rule. |
| userId | String | Foreign key linking to the User (Admin/Super Admin) who created/modified this rule. |
| Price | Float | Base price for a delivery. Defaults to 100. |
| perkilometer | Float | Additional cost per kilometer. Defaults to 1.2. |
| perminute | Float | Additional cost per minute. Defaults to 0.5. |
| isActive | Boolean | Indicates if this pricing rule is currently active. Defaults to false. |
| createdAt | DateTime | Timestamp when the price rule was created. |
| updatedAt | DateTime | Timestamp of the last update to the price rule. |
| isActiveDate | DateTime? | Date from which this price rule becomes active. Defaults to now(). |

* **Relationships:**
  + Belongs to one User.

**Table: Notification**

Stores records of notifications sent to users.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the notification. |
| userId | String? | Foreign key linking to the User who received the notification |
| customerId | String? | Foreign key linking to the Customer who received the notification |
| title | String | Title of the notification. |
| body | String | Main content of the notification. |
| data | Json? | Optional JSON payload for extra notification data. |
| isRead | Boolean | Indicates if the notification has been read by the user. Defaults to false. |
| createdAt | DateTime | Timestamp when the notification was created/sent. |

* **Relationships:**
  + Belongs to one User (optional).
  + Belongs to one Customer (optional).

**Table: Faq**

Stores frequently asked questions and their answers.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the FAQ entry. |
| question | String | The question text. |
| answer | String | The answer text for the question. |
| createdAt | DateTime | Timestamp when the FAQ entry was created. |
| updatedAt | DateTime | Timestamp of the last update to the FAQ entry. |

**Table: PrivacyTerm**

Stores privacy policies and terms of service documents.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the policy/term document. |
| title | String | Title of the policy or term (e.g., "Privacy Policy"). |
| content | String | Full text content of the policy or term. |
| type | String | Type of document (e.g., "Privacy", "Terms"). |
| createdAt | DateTime | Timestamp when the document was created. |
| updatedAt | DateTime | Timestamp of the last update to the document. |

**Table: About**

Stores content for the "About Us" page or section.

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Data Type** | **Description** |
| id | String | Unique identifier for the "About Us" content. |
| title | String | Title of the "About Us" section. |
| content | String | Full text content for the "About Us" page. |
| createdAt | DateTime | Timestamp when the content was created. |
| updatedAt | DateTime | Timestamp of the last update to the content. |

## 4. Enums (Enumerated Types)

Defines the distinct roles a User can have in the system.

* SUPERADMIN: Highest level of administrative access.
* ADMIN: Standard administrative access.
* MOTORIST: User role for delivery drivers.

**Enum: DeliveryStatus**

Defines the possible states for a Delivery.

* PENDING: Delivery request has been created, awaiting assignment.
* ASSIGNED: A motorist has accepted the delivery.
* PICKED\_UP: Motorist has picked up the package.
* IN\_TRANSIT: Package is on its way to the destination.
* DELIVERED: Package has been successfully delivered.
* CANCELLED: Delivery has been cancelled.

**Enum: PackageStatus**

Defines the possible states for a Package.

* PENDING: Package details created, awaiting assignment to a delivery.
* ASSIGNED: Package is assigned to an active delivery.
* PICKED\_UP: Package has been picked up by the motorist.
* IN\_TRANSIT: Package is currently being transported.
* DELIVERED: Package has been successfully delivered.
* CANCELLED: Package delivery has been cancelled.