Software Requirements Specification

for

LAUNDRY MANAGEMENT FOR HOSTEL SERVICES

Version 1.0 approved

Prepared by SAHANA K

2303717610422112

<date created>

Table of Contents

Revision History

Name	Date	Reason For Changes	Version

1.Introduction

1.1 Purpose

The purpose of this project is to develop a comprehensive Laundry Management System for automating laundry service requests, tracking, and delivery processes. This application is designed to allow users to place laundry requests, track their status in real time, and receive notifications upon completion. The system aims to eliminate manual record-keeping, streamline operations, and provide a reliable and efficient laundry service experience for both users and administrators.

1.2 Document Conventions

This Software Requirements Specification (SRS) document adheres to standardized formatting and writing conventions to ensure clear communication of requirements and maintain consistency throughout the document. The conventions adopted in this document are as follows:

- **Bold Text**: Used to emphasize section titles, key feature names, and important module identifiers for easy navigation and quick reference.
- *Italic Text*: Applied for the first appearance of specific roles, terms, or definitions introduced in the system, such as *Customer*, *Laundry Manager*, and *Employee*.
- Numbered Headings: Organized in a hierarchical structure (e.g., 1, 1.1, 1.2) to systematically arrange topics and subtopics for logical flow and clarity.
- Bullet Points (•): Employed to list system features, user benefits, functional and non-functional requirements, ensuring information is grouped and easy to scan.
- Equal Priority: All listed requirements are considered equally important unless explicitly stated otherwise to highlight criticality.
- Formatting Style: The document is formatted in **Times New Roman**, font size **12 pt**, with **1.15 line spacing** for readability and professional presentation

1.3 Intended Audience and Reading Suggestions

This document is intended to serve as a comprehensive reference for various individuals involved in the lifecycle of the Laundry Management System, from development to deployment and usage. It provides a clear overview of the system's functional and non-functional requirements, aiding different roles in understanding their interaction with the system.

- **End Users (Customers and Employees)**
  To understand how to interact with the system, including user registration, order placement, order tracking, and receiving notifications through a simple and intuitive interface.
- **Software Developers**
  To gain insights into system architecture, data flow, API endpoints, database design, and technical specifications required for efficient development and integration.

- **Project Managers**
  To monitor project progress, manage timelines, allocate resources effectively, and ensure the project aligns with specified requirements.
- **Quality Assurance (QA) and Testers**
  To design test cases and validation strategies based on the functional and non-functional specifications, ensuring the system behaves as expected and is robust.

1.4 Product Scope

The Laundry Management System is a comprehensive web-based solution designed to streamline and automate the management of laundry services for customers. The system provides an easy-to-use interface where customers can register, place laundry orders, track order status, and make payments online. The management staff can oversee customer orders, update the status of orders, generate invoices, and handle billing efficiently.

Key functionalities covered in the project scope include:

- Customer Registration and Authentication
- Online Order Booking (Drop-off & Pickup Scheduling)
- Real-time Order Status Tracking
- Automatic Invoice Generation and Payment Management
- Notification System for Order Updates
- Admin Dashboard for Managing Orders, Pricing, and Reports
- Feedback Collection for Continuous Service Improvement

1.5 References

1. Sebastian Ramirez – *FastAPI for Beginners*: *Building Modern Web APIs with Python*
Luciano Ramalho – *Fluent Python*, O'Reilly Media
(For advanced Python concepts used in backend logic, file handling, etc.)

2.Mohammed Anas, Syed Imran Ali, Syed Ahmad Tirmizi –
"Smart Laundry Service System Using Web Technology", International Journal for Research in Applied Science & Engineering Technology (IJRASET), 2021.
Link to Paper

2. Overall Description

2.1 Product Perspective

This product is being developed to reduce the burden of laundry people of the burden of keeping track of the clothes being given and taken and the bills of the customers and their payments. The purpose is to finish off all these works in less time so that the laundry work can be done quickly and also more new orders can be taken as the old ones are given away. This drives more business for the laundry people and the customers can also quickly receive their clothes.

2.2 Product Functions

The application allows the admin view the statistical analysis of the clothes with most order, customers with maximum order, section of orders which has got most revenue. The admin can manage orders by constantly updating status of the order based on which an email is automatically generated and sent to the email id of the customers. A daily report containing no. of pending, delivered orders, no of clients registered and revenue generated is shown in the form of a table which gives the daily expenditure. Any problem regarding clothes or any service problem can be conveyed to the customer through email through the contact customer section. The admin or the manager can print the invoice of the customer and hand it over to the employee when he is assigned a delivery.

2.3 User Classes and Characteristics

The different users of this laundry management system are laundry managers, laundry employees and the customers. The admin can manage orders by constantly updating status of the order based on which an email is automatically generated and sent to the email id of the customers. The admin or the manager can print the invoice of the customer and hand it over to the employee when he is assigned a delivery. The customer on the other hand has a separate login credentials where can view the orders he has booked and the status of those orders. He can book orders online once he logs in. He can update his profile once he logs in which includes his password. This gives flexibility to the customer to change passwords. The laundry employees can update the customer orders into the system.

2.4 Operating Environment

The Laundry Management System is designed to function in a modern client–server architecture, accessible via web browsers and designed for efficient interaction between customers, employees, and admins.

- **Hardware Platform**
  - **Server**: Minimum requirements include a quad-core processor, 8 GB RAM, and 250 GB storage. Recommended specifications are an octa-core processor, 16 GB RAM, and 500 GB+ storage.
  - **Client Devices**: Desktops and laptops (4 GB RAM minimum), smartphones, and tablets (2 GB RAM minimum).
- **Operating System**
  - **Server**: Linux (Ubuntu 20.04 or later) or Windows Server 2019+.
  - **Client**: Windows 10/11, macOS, Android 9+, iOS 13+.
- **Software Components**
  - **Backend**: Python 3.9+ with FastAPI framework.
  - **Database**: PostgreSQL 13+.
  - **Web Server**: Uvicorn/Gunicorn for serving APIs.
  - **Frontend**: HTML, CSS, JavaScript, and React.js framework.
  - **Reporting Tools**: jsPDF + AutoTable for PDF/CSV export.
  - **Visualization**: Chart.js for interactive graphical reports.
  - **Development & Testing**: VS Code, Postman, Git/GitHub.
- **Coexistence with Other Applications**
  - Supports future API integration with hostel management systems or financial systems.
  - Compatible with modern web browsers for seamless operation.

2.5 Design and Implementation Constraints

The following constraints limit the design and development of the Laundry Management System:

- **Technology Constraints**
  - Backend implementation strictly in Python using FastAPI framework.
  - PostgreSQL is mandatory for relational data storage.
  - Deployment relies on Uvicorn/Gunicorn as the ASGI application server.
- **Hardware Limitations**
  - Performance depends on a minimum of 8 GB RAM and a quad-core processor.
  - Low-end client devices may experience slower load times for heavy data or media.
- **Integration Constraints**
  - The system must provide RESTful APIs for easy extensibility and integration with future systems (e.g., hostel management or billing systems).
  - Must support cross-browser compatibility (Chrome, Firefox, Edge).

2.6 User Documentation

The following documentation will be delivered with the Laundry Management System:

- **User Manual (PDF/Digital)**: Step-by-step instructions for customers, employees, and admins to perform tasks such as booking orders, updating order status, managing pricing, and generating reports.

- **Online Help**: Context-sensitive FAQ and help sections accessible directly from the web dashboard.
- **API Documentation**: Automatically generated Swagger/OpenAPI documentation for developers and integrators to understand system endpoints.
- **Tutorials**: Short video guides and slide-based walkthroughs for onboarding customers and admins quickly.

2.7 Assumptions and Dependencies

2.5.1 Assumptions

2.5.1.1 Basic Computer Knowledge:
It is assumed that both users and administrators have a basic understanding of how to operate web applications, including navigating dashboards, submitting forms, and viewing records.

2.5.1.2 Controlled Environment:
It is assumed that the application will be used in a stable network environment, such as institutional or hostel Wi-Fi, to ensure consistent data flow between frontend and backend

2.5.2 Dependencies

2.5.2.1 Web Browser:
The system requires a modern, standards-compliant web browser (e.g., Chrome, Firefox) for users and admins to interact with the frontend UI effectively.

2.5.2.2 Backend Framework and Libraries:
The application relies on FastAPI for backend development and PostgreSQL for database management. These must be correctly installed and configured for the system to operate.

3. External Interface Requirements

3.1 User Interfaces

3.1.1 Laundry Management Dashboard:
The Laundry Management System includes a user-friendly, web-based dashboard that provides both users and administrators with an interactive interface to manage laundry requests and view status updates in real-time. The dashboard is built using standard web technologies (HTML, CSS, JavaScript) and is integrated with the FastAPI backend.
This dashboard includes:

3.1.1.1 Laundry Request Panel:
Allows users to submit laundry requests by selecting item types, quantities, and pickup slots.

3.1.1.2 Order Tracking Section:
Displays real-time status of submitted requests (e.g., Collected, Processing, Ready, Delivered) using UUID tracking.

3.1.1.3 Admin Controls:
Enables administrators to view and update order statuses, manage users, set pricing, and access feedback.

3.1.1.4 Report and Feedback Viewer:
Provides access to laundry order history and user feedback, along with options to export reports in CSV or PDF format.

3.2 Hardware Interfaces

- **Mouse and Keyboard:**
  Standard input devices such as a mouse and keyboard are used by users and administrators to interact with the system. These are essential for submitting laundry requests, navigating the dashboard, updating statuses, and managing reports.
- **Display Monitor:**
  A display monitor is required to view the Laundry Management System's web-based dashboard. It displays user interfaces for request submission, order tracking, and administrative tools, enabling efficient and real-time management of laundry operations.

3.3 Software Interfaces

- **Database Storage:**
  All user data, laundry requests, pricing, and feedback are stored securely in a **PostgreSQL database**. This ensures efficient data storage, retrieval, and management across the application with support for relational queries and integrity constraints.
- **FastAPI Framework:**
  The backend of the system is developed using **FastAPI**, a modern web framework that supports asynchronous operations and rapid API development. It handles request routing, database interaction, and business logic.
- **Frontend Interface (HTML/CSS/JavaScript):**
  The user interface is built using **HTML**, **CSS**, and **JavaScript**, providing responsive web pages for both users and administrators to interact with the system via a browser.

3.4 Communications Interfaces

System communication requirements:

- **Protocols & Standards:**
  - HTTP/HTTPS for API communication.
  - SMTP API for email notifications.
- **Message Formatting:** JSON for API data exchange.
- **Security & Encryption:** HTTPS (TLS 1.2+), encrypted sensitive data.
- **Data Transfer Requirements:**
  - Minimum bandwidth: 2 Mbps.
  - Sync mechanism for intermittent connectivity.

4. System Features

4.1. Description and Priority

Allows students, laundry staff, and admin/warden to register and log in securely.
Priority: High (without authentication, system access cannot be controlled).
Priority Ratings:

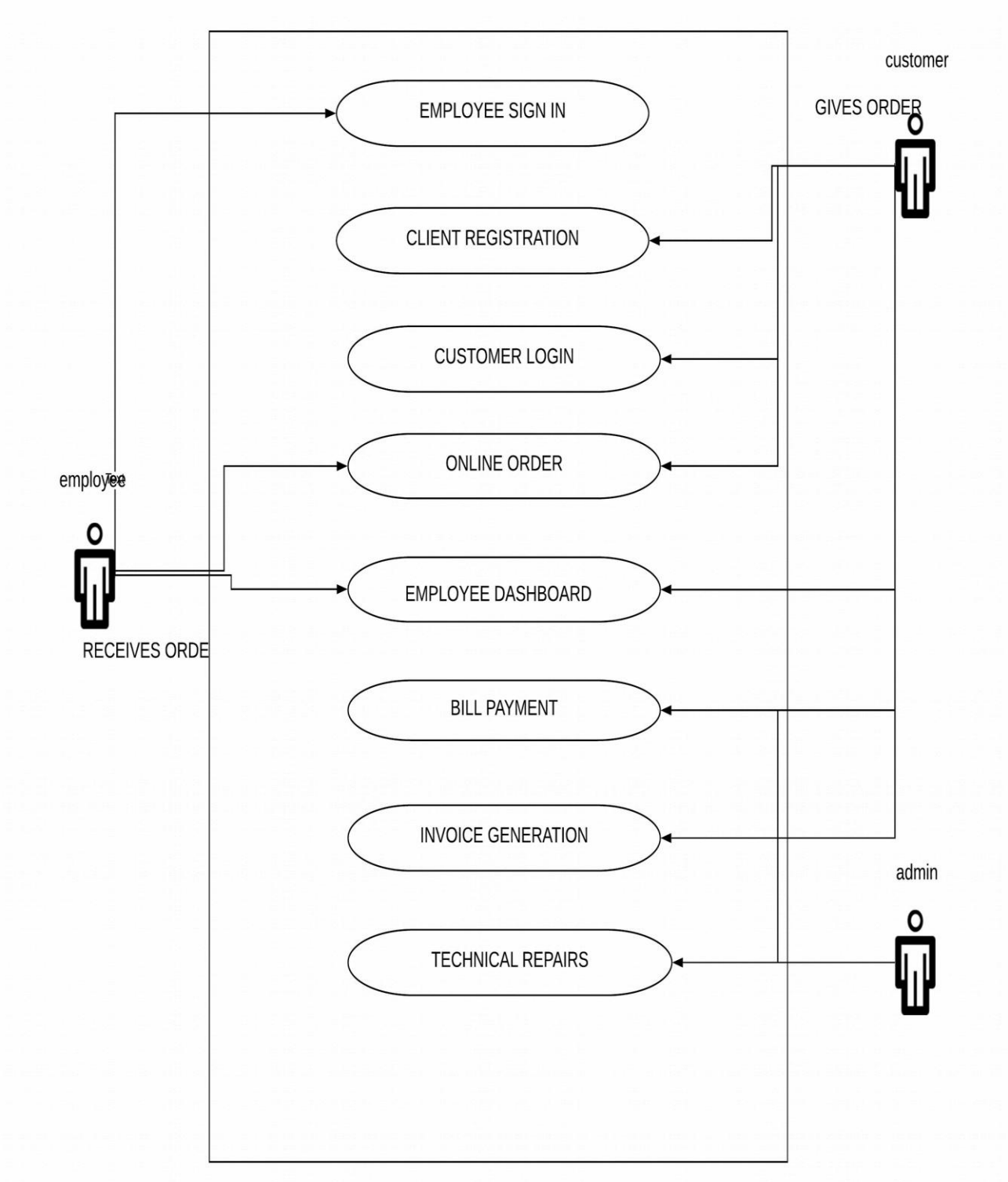
Benefit: 9
Penalty: 9
Cost: 5
Risk: 6

4.2 Stimulus/Response Sequences

- **Stimulus:** A new customer registers via the application.
  Response:
  The system validates the user information, ensures no duplication, creates a user profile, and assigns the "Customer" role.
- **Stimulus:** A registered customer logs in to place laundry service requests.
  Response:
  The system validates credentials and redirects the customer to their personalized dashboard to book, track, or manage orders.
- **Stimulus:** An employee logs in to view assigned tasks.
  Response:
  The system verifies credentials and shows the list of assigned pickup and delivery orders.
- **Stimulus:** Admin modifies or deletes a user or employee profile.
  Response:
  The system validates the request, updates the database accordingly, and confirms the successful operation.

4.3 Functional Requirements

USE CASE DIAGRAM:



Use Case 1: Place Laundry Order

Actor: Customer

Precondition: Customer is registered and logged in

Trigger: Customer wants to place a laundry service request

Main Flow:

1. Customer logs in to the system
2. Customer selects the clothes and desired service (e.g., wash, dry-clean, iron)
3. Customer schedules a pickup slot
4. System checks and confirms availability
5. Customer confirms the laundry order
6. System assigns an employee for pickup
7. Clothes are picked up from customer
8. Clothes are received by staff and processed
9. Processed clothes are packed
10. Clothes are delivered to the customer
11. System sends delivery confirmation
12. Order is marked completed

Exceptions:

- Pickup staff unavailable → Notify and reschedule
- Item damaged → Log complaint, escalate to admin
- Payment failed → Notify and hold the order

Use Case 2: Employee Process Order

Actor: Employee

Precondition: Employee is logged in

Trigger: New order assigned

Main Flow:

1. Employee signs in and views the dashboard
2. Sees assigned orders
3. Checks clothes and logs special cases
4. Proceeds with washing, drying, ironing
5. Updates the status for each step
6. Packs the clothes
7. Marks the order "Ready for Delivery"

Use Case 3: Make Payment

Actor: Customer

Precondition: Order has been processed

Trigger: Customer is ready to pay

Main Flow:

1. Customer opens invoice
2. Selects payment method (Card/UPI/Wallet)

3. System securely processes payment
4. Confirmation and receipt sent to customer
5. Order marked as "Paid"

Exception:

- Payment failed → Retry or choose another method

Use Case 4: Track Order

Actor: Customer
Precondition: Order placed
Trigger: Customer wants to check status
Main Flow:

1. Customer logs in and navigates to "My Orders"
2. System shows current order stage (Picked → Processing → Ready → Delivered)
3. Estimated delivery time is shown

Use Case 5: Generate Invoice

Actor: Admin
Precondition: Admin is logged in
Trigger: Order is marked completed and ready for billing
Main Flow:

1. Admin accesses order summary
2. System auto-generates invoice
3. Invoice sent to customer and saved in system

Use Case 6: View Reports

Actor: Admin
Precondition: Admin logged in
Trigger: Admin wants analytics
Main Flow:

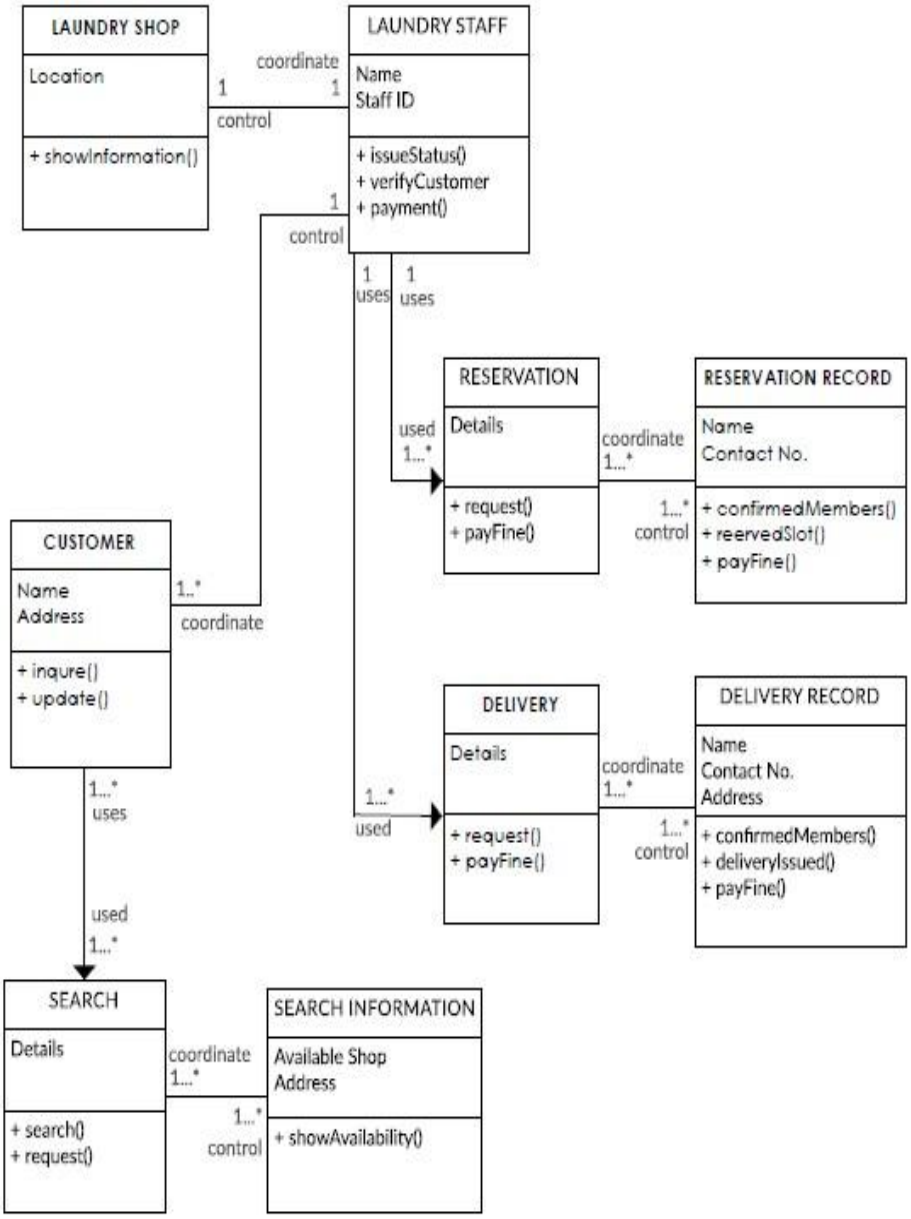
1. Admin selects report type (Orders, Revenue, etc.)
2. System fetches and shows/export report
3. Used for internal analysis and decisions

Use Case 7: Handle Technical Repairs

Actor: Admin
Precondition: Complaint received or system fault
Trigger: Maintenance or complaint reported
Main Flow:

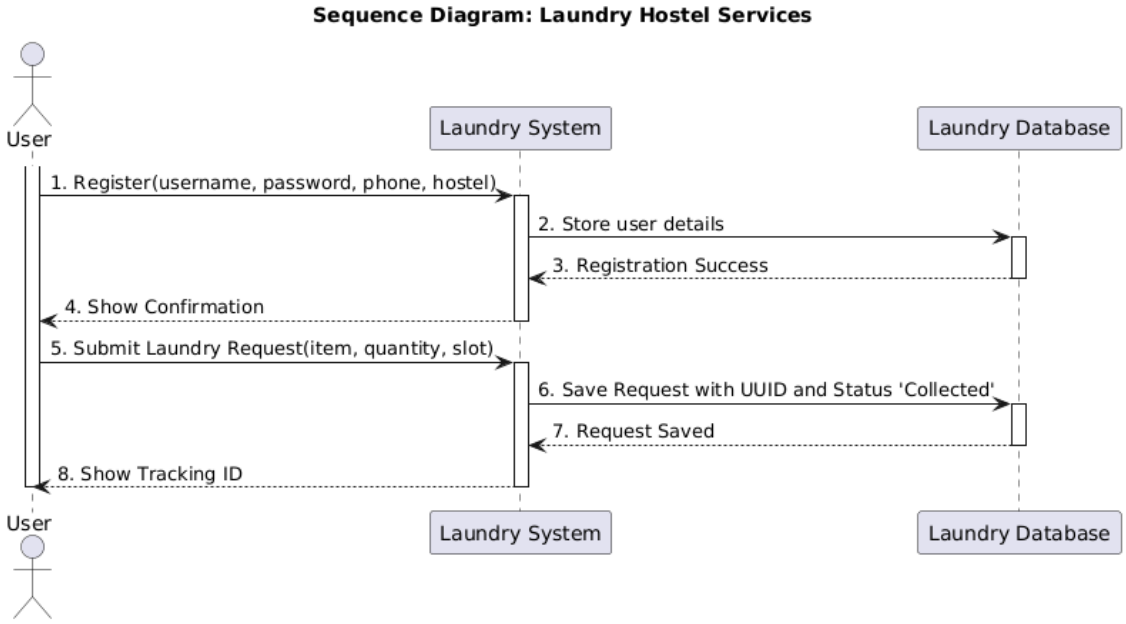
1. Admin checks reported issues
2. Assigns technical repair tasks
3. Repairs logged and resolved

CLASS DIAGRAM:

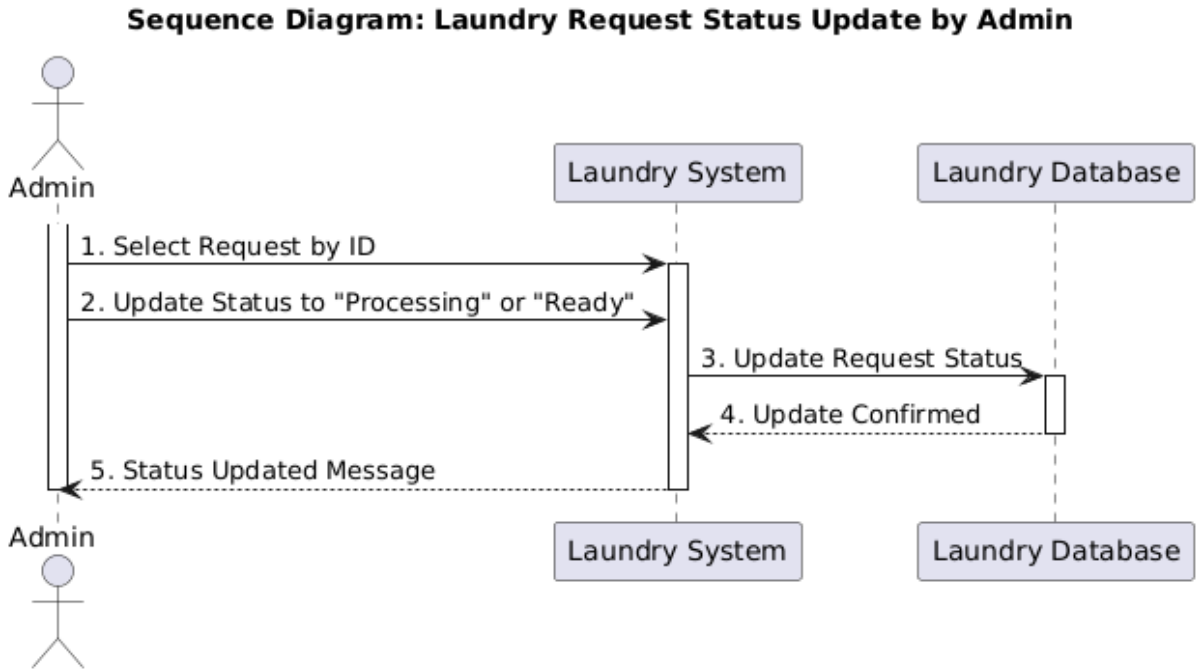


SEQUENCE DIAGRAM:

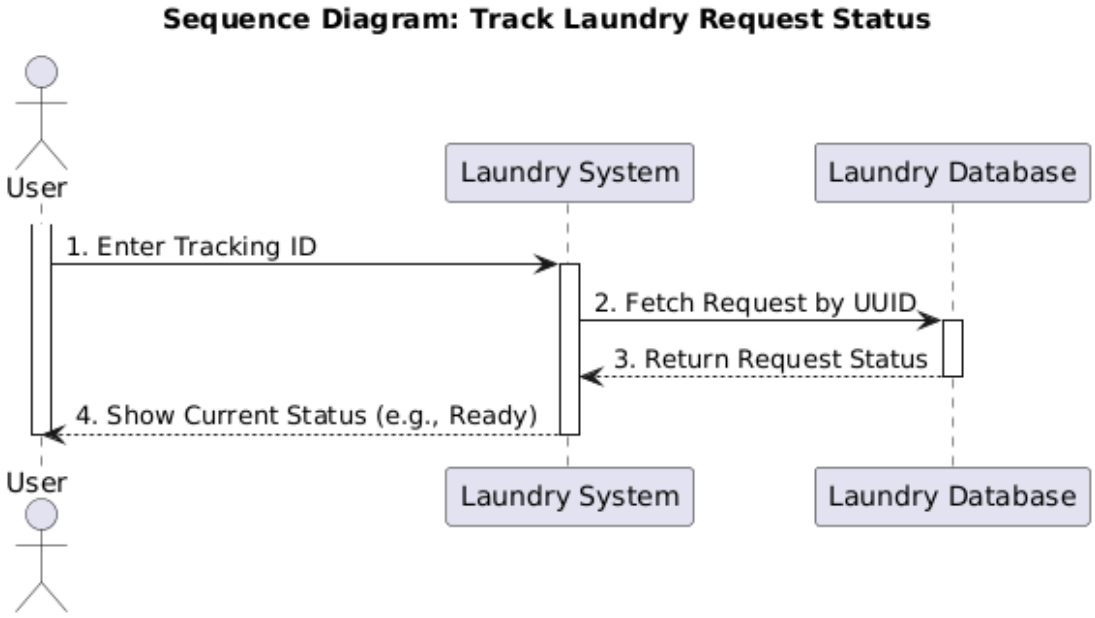
User Registers and Submits Laundry Request:



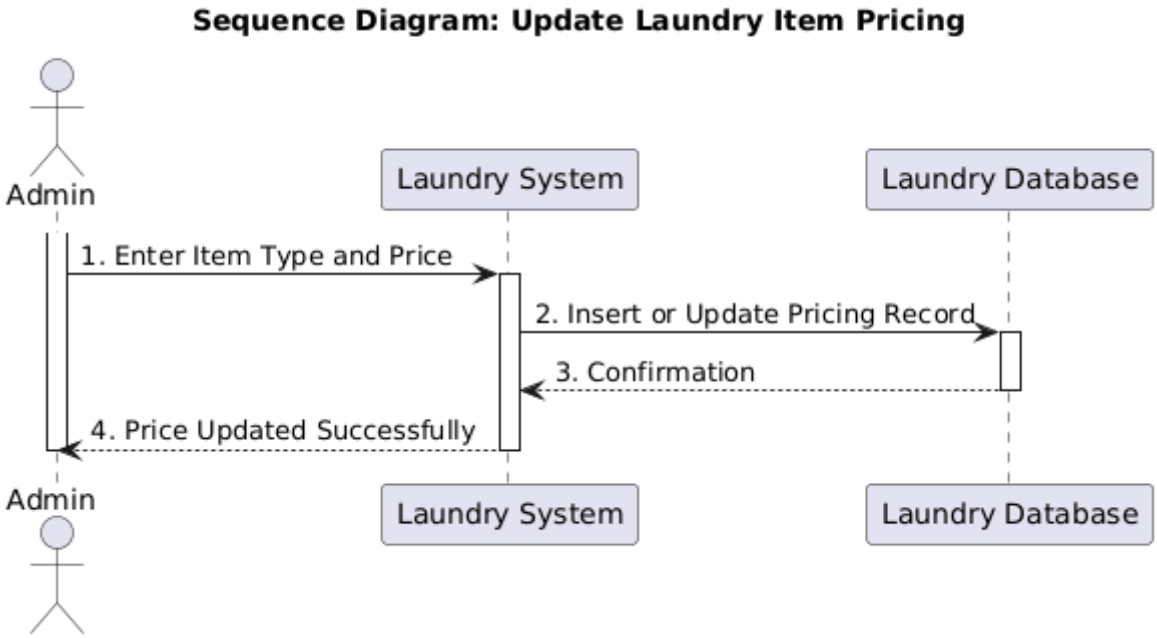
Admin updates and Laundry Request:



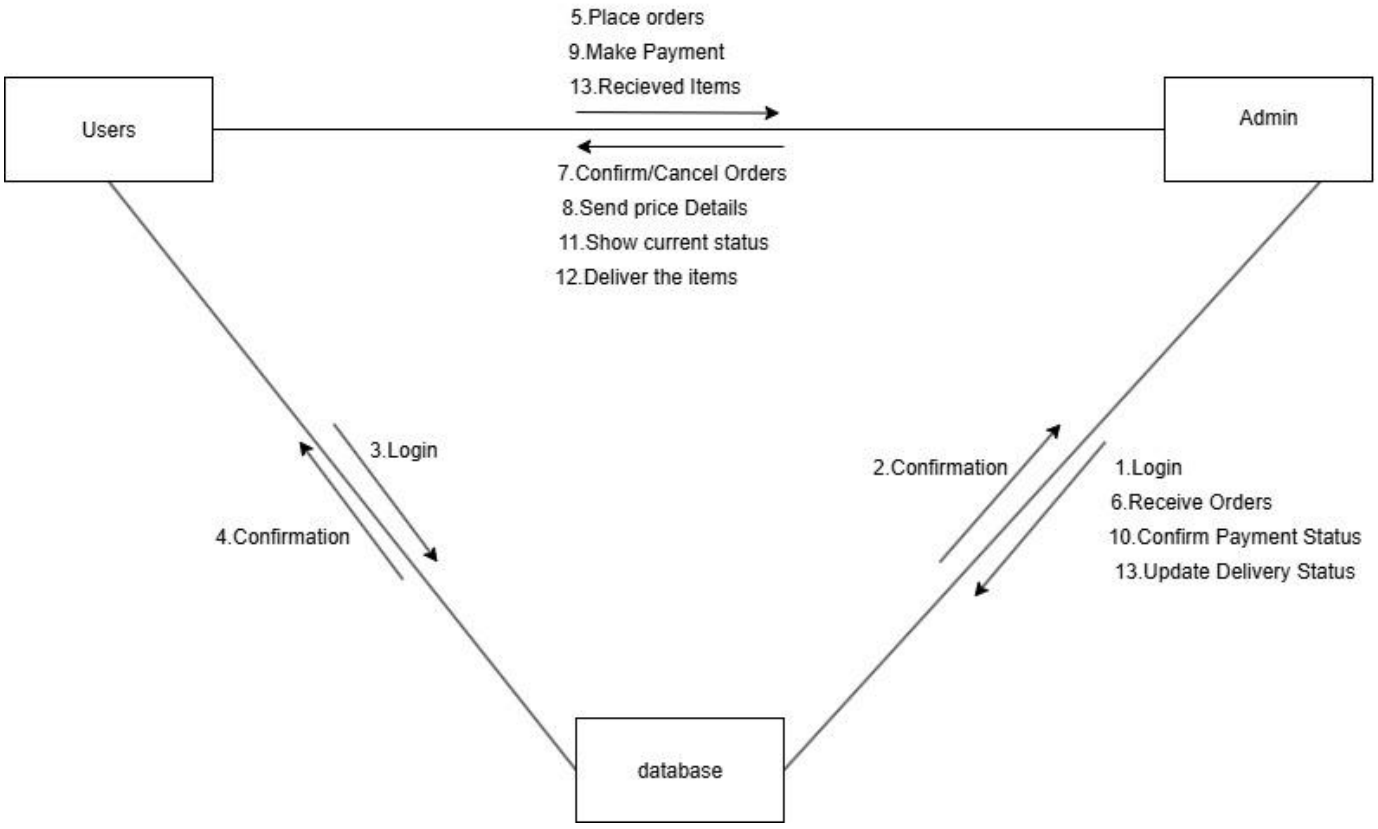
User Tracks Laundry Order Using UUID:



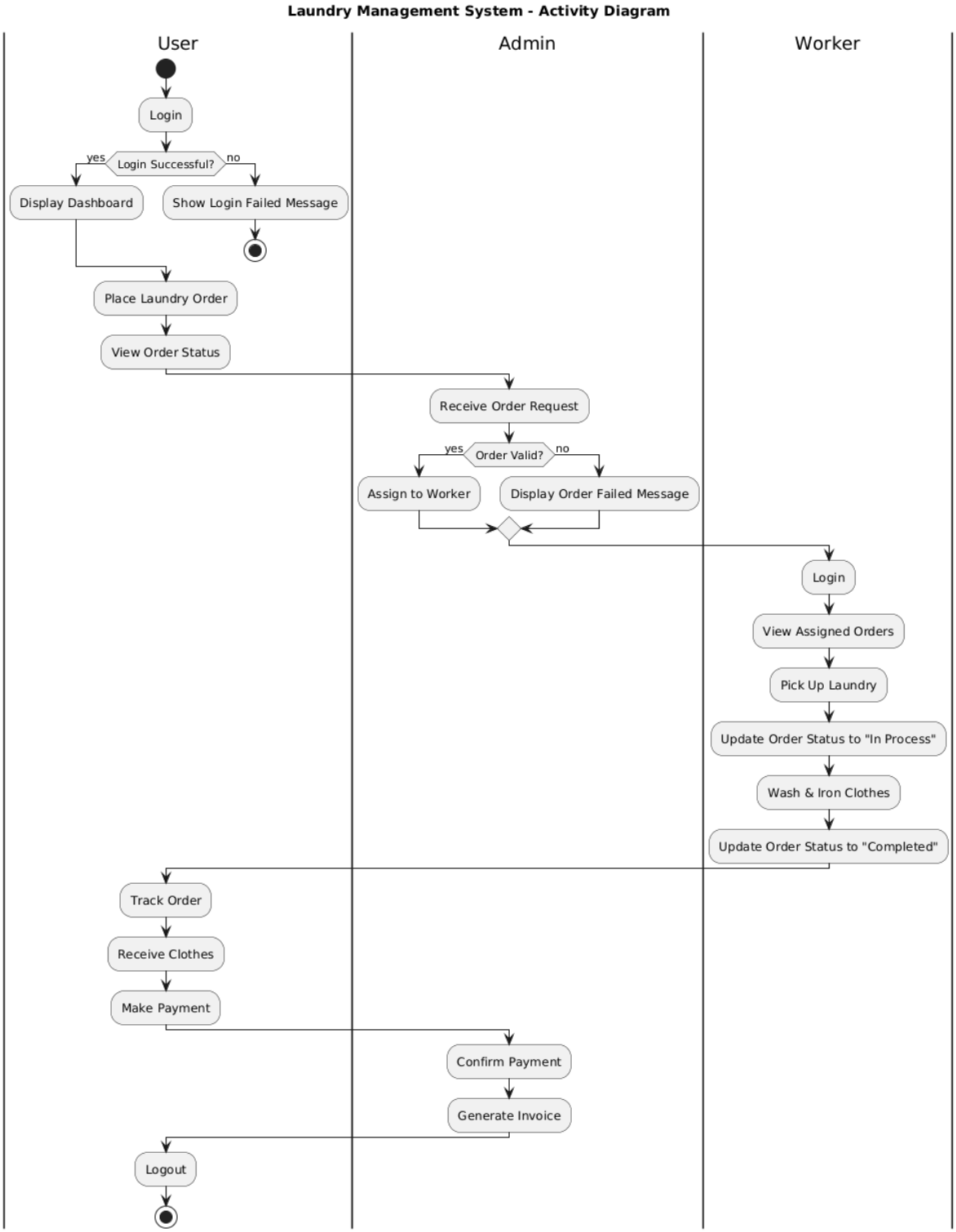
Admin Sets or Update pricing:



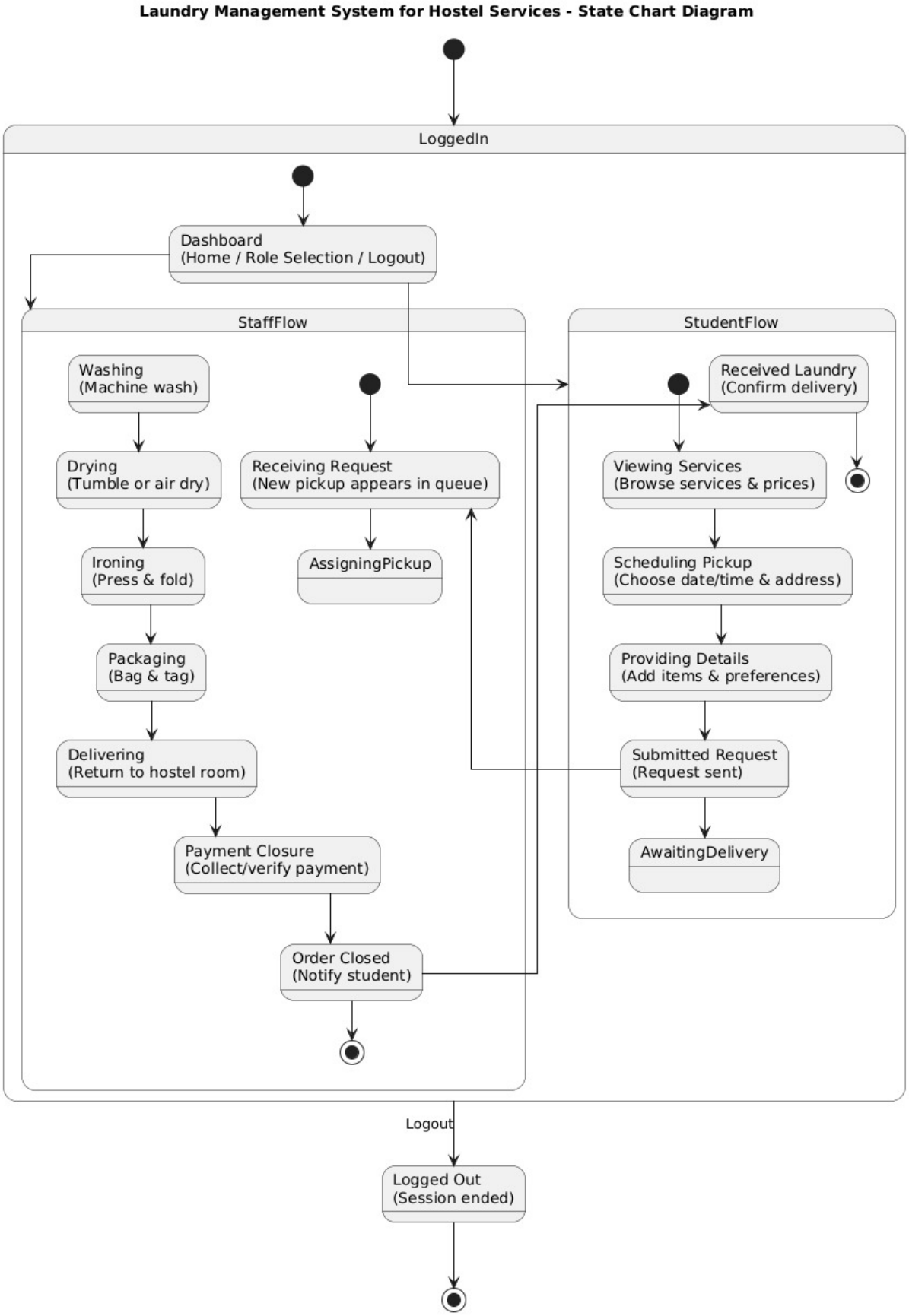
COLLABORATION DIAGRAM:



ACTIVITY DIAGRAM:



STATE DIAGRAM:



5 Other Nonfunctional Requirements

5.1 Performance Requirements

- Response Time:
  - API responses < 2 seconds.
  - Receipt generation < 10 seconds.
- Scalability: Support up to 2000 concurrent users.
- Availability: 99.5% uptime during hostel working hours (8 AM – 8 PM).
- Real-Time Notifications: Delivered within 5 seconds.

5.2 Safety Requirements

- Prevent accidental deletion of user and order data.
- File uploads restricted to safe formats (PNG, JPG, PDF).
- Data validation to prevent invalid records.

5.3 Security Requirements

- User authentication and role-based access.
- Data encryption during transmission and storage.
- Passwords stored as salted hashes.
- All actions logged with tamper-proof mechanisms.
- GDPR-like privacy compliance.

5.4 Software Quality Attributes

- Usability: Simple, minimal training.
- Reliability: Handles 2000 concurrent users.
- Maintainability: Modular code with standard practices.
- Portability: Works on major browsers and mobile devices.
- Interoperability: REST APIs for third-party integrations.
- Testability: Testable via Postman.
- Robustness: Handles invalid input gracefully.

5.5 Business Rules

- Students/Residents can request laundry pickup, track orders, and make payments.
- Laundry Staff can update order status (e.g., “Picked Up,” “In Progress,” “Delivered”).
- Warden/Admin can view all student orders, generate reports, and manage staff assignments.
- Laundry pickup is allowed only between 7 AM – 9 AM.
- Delivery of washed clothes must be completed within 48 hours of pickup.
- Each student is allowed a maximum of two laundry requests per week.
- One order can contain up to 10 kg of clothes; any extra will be charged additionally.

6 Other Requirements

Database Requirements

- PostgreSQL with normalized schema (3NF).
- Daily backups & disaster recovery.

Internationalization Requirements

- English support; future plan for additional languages.

Legal Requirements

- Compliance with local data protection laws.
- Copyright adherence.

Reuse Objectives

- Reusable modules for authentication, order processing.

Documentation Requirements

- User manual, API docs (Swagger), tutorials.

Future Enhancements

- Mobile app integration (React Native).
- AI-driven order recommendations.

Appendix A: Glossary

- Resident/Student – A hostel occupant using laundry services.
- Warden/Admin – Hostel authority managing operations and monitoring laundry services.
- Laundry Staff – Personnel responsible for handling, washing, and delivering clothes.
- Order – A student's request for laundry service (pickup, washing, delivery).
- Dormant Account – An account with no laundry activity for 60 days.
- TBD – To Be Determined; requirements or details pending finalization.

Appendix B: Analysis Models

Entity-Relationship Diagram (ERD)

Entities: Student, Order, Payment, LaundryStaff, Admin

Relationships:

Student places Order

Staff processes Order

Order has Payment

Admin monitors Staff and Orders

State-Transition Diagram for Order

States: Requested → Picked Up → In Progress → Ready for Delivery → Delivered → Closed

Data Flow Diagram (DFD)

Level 0: Student → Laundry Management System → Laundry Staff/Admin → Delivery

Appendix C: To Be Determined List

- Finalization of pricing model (per kg / per item).
- Confirmation of payment gateway provider.
- Decision on supported regional languages.
- Approval of maximum storage capacity for the database.
- Hostel's refund and grievance policy details.

SOURCE CODE:

FRONTEND:

index.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Call Me Donkey - Laundry App</title>
     <link
href="https://fonts.googleapis.com/css2?family=Pacifico&family=Quicksand:wght@400;600&family=Playfair+Display&display
=swap" rel="stylesheet">
  <style>
    * {
      margin: 0;
      padding: 0;
      box-sizing: border-box;
    }
    body {
      font-family: 'Quicksand', sans-serif;
      background: url('https://thumbs.dreamstime.com/b/cool-neon-party-donkey-headphones-sunglasses-generative-ai-listening-
music-pop-art-style-colors-267684376.jpg') no-repeat center center fixed;
      background-size: cover;
      color: #ffffff;
      min-height: 100vh;
      display: flex;
      justify-content: center;
      align-items: center;
      animation: fadeIn 2s ease-in-out;
    }
    .overlay {
      background: rgba(0, 0, 0, 0.7);
      padding: 40px 50px;
      border-radius: 20px;
      text-align: center;
      max-width: 700px;
      width: 90%;
      box-shadow: 0 8px 30px rgba(0, 0, 0, 0.4);
    }
    h1 {
      font-family: 'Pacifico', cursive;
      font-size: 60px;
      color: #00f5ff;
      margin-bottom: 15px;
    }
    h2 {
      font-size: 26px;
      margin-bottom: 30px;
      color: #f0f0f0;
      font-weight: 600;
    }
    a {
      display: inline-block;
      margin: 10px;
      padding: 12px 24px;
      background: rgba(255, 255, 255, 0.2);
      border: 2px solid #ffffff;
      border-radius: 10px;
      text-decoration: none;
      color: #fff;
      font-weight: bold;
```

```
    font-size: 16px;
    transition: 0.3s ease;
   }
   a:hover {
    background: #1abc9c;
    color: #000;
   }
   .quote {
    font-size: 22px;
    font-style: italic;
    color: #ffdede;
    margin-top: 35px;
    font-family: 'Playfair Display', serif;
    max-width: 600px;
    margin-left: auto;
    margin-right: auto;
    animation: slideUp 1.2s ease-out;
   }
   @keyframes fadeIn {
    from { opacity: 0; }
    to   { opacity: 1; }
   }
   @keyframes slideUp {
    from { opacity: 0; transform: translateY(30px); }
    to { opacity: 1; transform: translateY(0); }
   }
   @media (max-width: 600px) {
    h1 { font-size: 40px; }
    h2 { font-size: 20px; }
    a  { font-size: 14px; padding: 10px 20px; }
    .quote { font-size: 18px; }
    .overlay { padding: 30px 20px; }
   }
  </style>
</head>
<body>
  <div class="overlay">
   <div class="container">
    <h1>Call Me Donkey</h1>
    <h2>Welcome to Laundry Pickup & Delivery</h2>
    <a href="/login.html">Login</a>
    <a href="/register.html">Register</a>
    <a href="/orders.html">View Orders</a>
    <a href="/admin.html">Admin Analytics Board</a>
    <a href="/track.html">Track Orders</a>
    <p class="quote">“Clean clothes, clean mind, clean spirit.”</p>
   </div>
  </div>
  <script>
   const params = new URLSearchParams(window.location.search);
   if (params.get("logout") === "true") {
    const msg = document.createElement("p");
    msg.textContent = "You have been logged out successfully.";
    msg.style.color = "lightgreen";
    msg.style.textAlign = "center";
    msg.style.marginBottom = "10px";
    document.querySelector(".container").prepend(msg);
   }
  </script>
</body>
</html>
```

login.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>Laundry App - Login</title>
  <link href="https://fonts.googleapis.com/css2?family=Quicksand:wght@400;600&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="/static/css/styles.css">
  <style>
    body.login {
      margin: 0;
      padding: 0;
      font-family: 'Quicksand', sans-serif;
      background: url('https://images.unsplash.com/photo-1675329585872-
c247d01f7668?q=80&w=1170&auto=format&fit=crop&ixlib=rb-
4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D') no-repeat center center fixed;
      background-size: cover;
      display: flex;
      justify-content: center;
      align-items: center;
      height: 100vh;
    }
    .container {
      background-color: rgba(255, 255, 255, 0.95);
      padding: 35px 40px;
      border-radius: 16px;
      box-shadow: 0 8px 20px rgba(0, 0, 0, 0.2);
      max-width: 400px;
      width: 90%;
    }
    h2 {
      font-size: 2rem;
      text-align: center;
      color: #1e3d59;
      margin-bottom: 20px;
      font-weight: 600;
    }
    form {
      margin-top: 20px;
    }
    .form-group {
      margin-bottom: 18px;
      position: relative;
    }
    input[type="text"],
    input[type="password"] {
      width: 100%;
      padding: 12px 14px;
      border: 1px solid #ccc;
      border-radius: 8px;
      background-color: #f7f7f7;
      font-size: 15px;
      font-family: 'Quicksand', sans-serif;
      transition: 0.3s ease;
    }
    input:focus {
      border-color: #00a8cc;
      outline: none;
      background-color: #fff;
    }
    .toggle-password {
```

```
    position: absolute;
    right: 10px;
    top: 10px;
    background: none;
    border: none;
    color: #00a8cc;
    cursor: pointer;
    font-size: 16px;
   }
   .remember-me {
    font-size: 14px;
    color: #555;
    margin-bottom: 15px;
   }
   button[type="submit"] {
    width: 100%;
    padding: 13px;
    font-size: 16px;
    background-color: #00a8cc;
    color: white;
    border: none;
    border-radius: 10px;
    font-weight: bold;
    font-family: 'Quicksand', sans-serif;
    cursor: pointer;
    transition: background-color 0.3s ease;
   }
   button[type="submit"]:hover {
    background-color: #008fb3;
   }
   p {
    text-align: center;
    margin-top: 20px;
    color: #333;
   }
   p a {
    color: #00a8cc;
    text-decoration: none;
    font-weight: bold;
   }
   p a:hover {
    text-decoration: underline;
   }
   @media (max-width: 600px) {
    .container {
     padding: 25px 20px;
    }
    h2 {
     font-size: 1.6rem;
    }
   }
  </style>
</head>
<body class="login">
  <div class="container">
   <h2>Login</h2>
   <form id="loginForm">
    <div class="form-group">
     <input type="text" id="username" placeholder="Username" required autofocus>
    </div>
    <div class="form-group">
```

```
    <input type="password" id="password" placeholder="Password" required>
    <button type="button" class="toggle-password" onclick="togglePassword()">👁</button>
   </div>
   <label class="remember-me">
    <input type="checkbox" id="rememberMe"> Remember Me
   </label>
   <button type="submit">Login</button>
  </form>
  <p>Don't have an account? <a href="/register.html">Register here</a></p>
 </div>
 <script>
  function togglePassword() {
   const pwd = document.getElementById('password');
   pwd.type = pwd.type === 'password' ? 'text' : 'password';
  }
  window.onload = () => {
   const savedUsername = localStorage.getItem('savedUsername');
   const savedPassword = localStorage.getItem('savedPassword');
   const remember = localStorage.getItem('rememberMe') === 'true';
   if (remember && savedUsername && savedPassword) {
    document.getElementById('username').value = savedUsername;
    document.getElementById('password').value = savedPassword;
    document.getElementById('rememberMe').checked = true;
   }
  };
  document.getElementById('loginForm')?.addEventListener('submit', async (e) => {
   e.preventDefault();
   const username = document.getElementById('username').value.trim();
   const password = document.getElementById('password').value.trim();
   const remember = document.getElementById('rememberMe').checked;
   if (!username || !password) {
    alert('Please enter both username and password.');
    return;
   }
   try {
    const response = await fetch('http://localhost:8000/login', {
     method: 'POST',
     headers: { 'Content-Type': 'application/json' },
     body: JSON.stringify({ username, password })
    });
    if (!response.ok) {
     alert("Login failed: Invalid credentials.");
     return;
    }
    const data = await response.json();
    if (data.user_id) {
     localStorage.setItem('user_id', data.user_id);
     localStorage.setItem('is_admin', data.role === 'admin' ? 'true' : 'false');
     localStorage.setItem('role', data.role);
     localStorage.setItem('user', JSON.stringify(data));
     if (remember) {
      localStorage.setItem('savedUsername', username);
      localStorage.setItem('savedPassword', password);
      localStorage.setItem('rememberMe', 'true');
     } else {
      localStorage.removeItem('savedUsername');
      localStorage.removeItem('savedPassword');
      localStorage.setItem('rememberMe', 'false');
     }
     window.location.href = data.role === 'admin' ? '/admin.html' : '/dashboard.html';
    } else {
```

```
        alert("Login failed: Unexpected error.");
       }
      } catch (error) {
       console.error("Login error:", error);
       alert("An error occurred during login.");
      }
    });
  </script>
</body>
</html>
```

dashboard.html

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <title>User Dashboard</title>
  <link href="https://fonts.googleapis.com/css2?family=Quicksand:wght@400;600&display=swap" rel="stylesheet">
  <link rel="stylesheet" href="/static/css/styles.css">
  <style>
    body {
      margin: 0;
      font-family: 'Quicksand', sans-serif;
      background: url('https://images.unsplash.com/photo-1470309864661-
68328b2cd0a5?q=80&w=1170&auto=format&fit=crop&ixlib=rb-
4.1.0&ixid=M3wxMjA3fDB8MHxwaG90by1wYWdlfHx8fGVufDB8fHx8fA%3D%3D') no-repeat center center fixed;
      background-size: cover;
      display: flex;
    }
    .sidebar {
      width: 240px;
      background: rgba(44, 62, 80, 0.95);
      color: white;
      height: 100vh;
      padding-top: 20px;
      position: fixed;
      left: 0;
      top: 0;
      overflow-y: auto;
      box-shadow: 2px 0 10px rgba(0, 0, 0, 0.2);
    }
    .sidebar h2 {
      text-align: center;
      font-size: 24px;
      margin-bottom: 30px;
    }
    .sidebar ul {
      list-style: none;
      padding: 0;
    }
    .sidebar ul li {
      padding: 15px 20px;
      border-bottom: 1px solid rgba(255, 255, 255, 0.1);
      cursor: pointer;
      transition: background 0.3s;
    }
    .sidebar ul li:hover {
      background: #1abc9c;
    }
    .main {
      margin-left: 240px;
      padding: 30px;
```

```
  width: calc(100% - 240px);
  overflow-x: hidden;
  background-color: rgba(255, 255, 255, 0.92);
  min-height: 100vh;
}

h2 {
  font-weight: 600;
  font-size: 28px;
  color: #2c3e50;
  margin-bottom: 10px;
}
h3 {
  font-weight: 500;
  margin-bottom: 15px;
  color: #34495e;
}
.cards {
  display: grid;
  grid-template-columns: repeat(auto-fit, minmax(180px, 1fr));
  gap: 20px;
  margin-bottom: 30px;
}
.card {
  padding: 20px;
  color: white;
  border-radius: 12px;
  text-align: center;
  font-weight: 600;
  box-shadow: 0 4px 12px rgba(0, 0, 0, 0.15);
}
.card.total { background: #3498db; }
.card.new { background: #f1c40f; color: #333; }
.card.accept { background: #27ae60; }
.card.inprocess { background: #2980b9; }
.card.finish { background: #e74c3c; }
.card.users { background: #2ecc71; }
.section {
  background: white;
  padding: 25px;
  border-radius: 15px;
  box-shadow: 0 4px 10px rgba(0, 0, 0, 0.08);
  margin-bottom: 30px;
}
input, select, button {
  width: 100%;
  padding: 12px;
  margin: 8px 0;
  border: 1px solid #ccc;
  border-radius: 8px;
  font-family: 'Quicksand', sans-serif;
  font-size: 15px;
}
button {
  background: #3498db;
  color: white;
  font-weight: bold;
  border: none;
  cursor: pointer;
  transition: background 0.3s;
}
```

```
button:hover {
  background: #2980b9;
}
#logout {
  background: #e74c3c;
  margin-top: 20px;
}
#logout:hover {
  background: #c0392b;
}
#notificationBox {
  display: none;
  background: #dff0d8;
  color: #3c763d;
  padding: 12px;
  border-radius: 10px;
  margin-bottom: 10px;
  font-size: 14px;
}
ul#requestList li {
  background: #f8f9fa;
  padding: 12px;
  margin-bottom: 10px;
  border-radius: 8px;
  font-size: 14px;
  box-shadow: 0 1px 3px rgba(0, 0, 0, 0.08);
}
a {
  color: #3498db;
  text-decoration: none;
  text-align: center;
  display: block;
  margin-top: 20px;
  font-weight: bold;
}
a:hover {
  text-decoration: underline;
}
table {
  width: 100%;
  border-collapse: collapse;
  margin-top: 10px;
}
table th, table td {
  border: 1px solid #ddd;
  padding: 10px;
  text-align: left;
  font-size: 14px;
}
table th {
  background-color: #f9f9f9;
}
@media (max-width: 768px) {
  .sidebar {
    position: relative;
    width: 100%;
    height: auto;
  }
  .main {
    margin-left: 0;
    width: 100%;
```

```
      padding: 20px;
    }
    .cards {
      grid-template-columns: 1fr;
    }
  }
 </style>
</head>
<body>
 <div class="sidebar">
  <h2>Laundry App</h2>
  <ul>
   <li>Dashboard</li>
   <li>Submit Request</li>
   <li>Order History</li>
   <li>Notifications</li>
  </ul>
 </div>
 <div class="main">
  <h2>Welcome to Your Dashboard</h2>
  <h3 id="userPlan">Your Plan: Loading...</h3>
  <div id="notificationBox">✅ Loading notification...</div>
  <div class="cards">
   <div class="card total">Total Requests: 0</div>
   <div class="card new">New Requests: 0</div>
   <div class="card accept">Accepted: 0</div>
   <div class="card inprocess">In Process: 0</div>
   <div class="card finish">Completed: 0</div>
   <div class="card users">Registered Users: 0</div>
  </div>
  <div class="section">
   <h3>Laundry Price Chart (Per Unit)</h3>
   <table>
    <tr><th>Item</th><th>Price (₹)</th></tr>
    <tr><td>Top Wear</td><td>₹14</td></tr>
    <tr><td>Bottom Wear</td><td>₹22</td></tr>
    <tr><td>Woolen Cloth</td><td>₹20</td></tr>
   </table>
  </div>
  <div class="section">
   <h3>Submit New Laundry Request</h3>
   <form id="requestForm">
    <input type="text" id="item_type" placeholder="Item Type (e.g. Shirt)" required>
    <input type="number" id="quantity" placeholder="Quantity" required>
    <input type="file" id="item_image" accept="image/*">
    <input type="text" id="item_note" placeholder="Note (e.g. Dry clean only)">
    <input type="text" id="hostel_name" placeholder="Hostel Name (e.g., A Block)" required>
    <input type="text" id="room_number" placeholder="Room Number (e.g., F1-18)" required>
    <select id="pickup_time_slot" required>
     <option value="">Select Pickup Time Slot</option>
     <option value="8:00 AM - 10:00 AM">8:00 AM - 10:00 AM</option>
     <option value="5:00 PM - 7:00 PM">5:00 PM - 7:00 PM</option>
    </select>
    <select id="payment_method" required>
     <option value="">Select Payment Method</option>
     <option value="UPI">UPI</option>
     <option value="Card">Card</option>
     <option value="Cash">Cash</option>
     <option value="Wallet">Wallet</option>
    </select>
    <button type="button" onclick="calculatePrice()">Calculate Price</button>
```

```
      <p id="estimatedPrice"></p>
      <button type="submit">💸 Pay & Submit Request</button>
    </form>
   </div>
   <div class="section">
    <h3>Your Laundry Requests</h3>
    <ul id="requestList"></ul>
   </div>
   <button id="logout">Logout</button>
   <a href="/index.html">← Back to Home</a>
  </div>
  <script src="/static/js/dashboard.js"></script>
</body>
</html>
```

dashboard.js

```
const BASE_URL = 'http://127.0.0.1:8000';
window.onload = () => {
  const userId = localStorage.getItem('user_id');
  const user = JSON.parse(localStorage.getItem('user') || '{}');
  const userPlanDisplay = document.getElementById('userPlan');
  const notificationBox = document.getElementById('notificationBox');
  if (!userId || !user?.plan) {
    alert("Session expired. Please login again.");
    window.location.href = "/login.html";
    return;
  }
  userPlanDisplay.textContent = `Your Plan: ${user.plan}`;
  document.getElementById("logout").addEventListener("click", () => {
    localStorage.clear();
    window.location.href = "/index.html";
  });
  document.getElementById('requestForm')?.addEventListener('submit', async (e) => {
    e.preventDefault();
    const item_type = document.getElementById('item_type').value.trim();
    const quantity = parseInt(document.getElementById('quantity').value.trim());
    const note = document.getElementById('item_note').value.trim();
    const fileInput = document.getElementById('item_image');
    const paymentMethod = document.getElementById('payment_method').value;
    const hostelName = document.getElementById('hostel_name').value.trim();
    const roomNumber = document.getElementById('room_number').value.trim();
    const pickupTimeSlot = document.getElementById('pickup_time_slot').value;
    if (!item_type || isNaN(quantity) || quantity <= 0) {
      alert('Enter valid item type and quantity.');
      return;
    }
    if (!paymentMethod || !hostelName || !roomNumber || !pickupTimeSlot) {
      alert('Please fill all required fields.');
      return;
    }
    const formData = new FormData();
    formData.append('item_type', item_type);
    formData.append('quantity', quantity);
    formData.append('user_id', userId);
    formData.append('payment_method', paymentMethod);
    formData.append('hostel_name', hostelName);
    formData.append('room_number', roomNumber);
    formData.append('pickup_time_slot', pickupTimeSlot);
    if (note) formData.append('note', note);
    if (fileInput?.files.length > 0) {
      formData.append('image', fileInput.files[0]);
    }
```

```
    try {
      const res = await fetch(`${BASE_URL}/upload-request`, {
        method: 'POST',
        body: formData
      });
      const data = await res.json();
      if (res.ok) {
        alert(`Request submitted! Order ID: ${data.order_id}`);
        document.getElementById('requestForm').reset();
        document.getElementById('estimatedPrice').textContent = '';
        fetchRequests();
      } else {
        alert(data.detail || 'Request submission failed.');
      }
    } catch (err) {
      console.error('Request failed:', err);
      alert('Something went wrong while submitting.');
    }
  });
  fetchRequests();
  fetchNotifications();  // ✅ Call notifications on load
};
async function fetchRequests() {
  const userId = localStorage.getItem('user_id');
  try {
    const res = await fetch(`${BASE_URL}/order-history/${userId}`);
    const data = await res.json();
    const list = document.getElementById('requestList');
    list.innerHTML = '';
    if (!Array.isArray(data) || data.length === 0) {
      list.innerHTML = '<li>No requests found.</li>';
      return;
    }
    data.forEach(req => {
      const li = document.createElement('li');
      li.innerHTML = `
        <strong>Order ID:</strong> <code>${req.id}</code><br>
        <strong>${req.item_type}</strong> - ${req.quantity} pcs
        <br>Status: <strong>${req.status}</strong>
        ${req.payment_method ? `<br><strong>Payment:</strong> ${req.payment_method}` : ''}
        ${req.hostel_name ? `<br><strong>Hostel:</strong> ${req.hostel_name}` : ''}
        ${req.room_number ? `<br><strong>Room No:</strong> ${req.room_number}` : ''}
        ${req.pickup_time_slot ? `<br><strong>Pickup Slot:</strong> ${req.pickup_time_slot}` : ''}
        ${req.image_url ? `<br><img src="${BASE_URL}/${req.image_url}" width="100">` : ''}
        ${req.note ? `<br><em>Note: ${req.note}</em>` : ''}
        <hr>
      `;
      list.appendChild(li);
    });
  } catch (error) {
    console.error('Failed to fetch requests:', error);
    alert('Error fetching requests.');
  }
}
async function calculatePrice() {
  const item = document.getElementById('item_type').value.trim();
  const qty = parseInt(document.getElementById('quantity').value.trim());
  if (!item || isNaN(qty) || qty <= 0) {
    alert('Enter valid item and quantity.');
    return;
  }
```

```
 try {
   const res = await fetch(`${BASE_URL}/calculate-price?item_type=${encodeURIComponent(item)}&quantity=${qty}`);
   const data = await res.json();
   if (res.ok) {
     document.getElementById('estimatedPrice').textContent = `Estimated Cost: ₹${data.estimated_price}`;
   } else {
     document.getElementById('estimatedPrice').textContent = 'Pricing not found.';
   }
 } catch (err) {
   console.error('Calculation failed:', err);
   document.getElementById('estimatedPrice').textContent = 'Error calculating price.';
 }
}
async function fetchNotifications() {
 const userId = localStorage.getItem('user_id');
 const notificationBox = document.getElementById('notificationBox');
 try {
   const res = await fetch(`${BASE_URL}/user/notifications/${userId}`);
   const notifications = await res.json();
   if (res.ok && Array.isArray(notifications) && notifications.length > 0) {
     const message = notifications
       .filter(n => n.status === 'Completed')
       .map(n => `✅ Your Order #${n.order_id} is Completed!`)
       .join('<br>');
     if (message) {
       notificationBox.innerHTML = message;
       notificationBox.style.display = 'block';
     } else {
       notificationBox.style.display = 'none';
     }
   } else {
     notificationBox.style.display = 'none';
   }
 } catch (err) {
   console.error('Notification fetch failed:', err);
   notificationBox.style.display = 'none';
 }
}
```

BACKEND:

database.py

```
from sqlalchemy import create_engine
from sqlalchemy.ext.declarative import declarative_base
from sqlalchemy.orm import sessionmaker
from dotenv import load_dotenv, find_dotenv
import os
load_dotenv(find_dotenv())
DATABASE_URL = os.getenv("DATABASE_URL")
print(f"Loaded DATABASE_URL: {DATABASE_URL}")
if DATABASE_URL is None:
    raise ValueError("DATABASE_URL not found in environment")
engine = create_engine(DATABASE_URL)
SessionLocal = sessionmaker(autocommit=False, autoflush=False, bind=engine)
Base = declarative_base()
def get_db():
    db = SessionLocal()
    try:
        yield db
    finally:
        db.close()
```

main.py

```
from fastapi import FastAPI, Depends, HTTPException, UploadFile, File, Form
from fastapi.middleware.cors import CORSMiddleware
from fastapi.staticfiles import StaticFiles
from fastapi.responses import FileResponse, HTMLResponse, Response
from sqlalchemy.orm import Session
from sqlalchemy import extract, func
from datetime import datetime
import uuid, shutil, os
import models, schemas
from uuid import uuid4
import pytz
from database import engine, get_db
from typing import List
from reportlab.pdfgen import canvas
models.Base.metadata.create_all(bind=engine)
app = FastAPI(debug=True)
# CORS settings
app.add_middleware(
    CORSMiddleware,
    allow_origins=["*"],
    allow_credentials=True,
    allow_methods=["*"],
    allow_headers=["*"],
)
BASE_DIR = os.path.dirname(os.path.abspath(__file__))
FRONTEND_DIR = os.path.abspath(os.path.join(BASE_DIR, "../frontend"))
ASSETS_DIR = os.path.join(FRONTEND_DIR, "assets")
UPLOADS_DIR = os.path.join(BASE_DIR, "uploads")
RECEIPTS_DIR = os.path.join(BASE_DIR, "receipts")
os.makedirs(UPLOADS_DIR, exist_ok=True)
os.makedirs(RECEIPTS_DIR, exist_ok=True)
app.mount("/static", StaticFiles(directory=ASSETS_DIR), name="static")
app.mount("/uploads", StaticFiles(directory=UPLOADS_DIR), name="uploads")
app.mount("/receipts", StaticFiles(directory=RECEIPTS_DIR), name="receipts")
@app.get("/favicon.ico")
def favicon():
    return Response(content="", media_type="image/x-icon")
@app.get("/", response_class=HTMLResponse)
def serve_root():
    return FileResponse(os.path.join(FRONTEND_DIR, "index.html"))
@app.get("/home", response_class=HTMLResponse)
def serve_home():
    return HTMLResponse("""
    <h2>Laundry App Pages</h2>
    <ul>
        <li><a href="/register.html">Register</a></li>
        <li><a href="/login.html">Login</a></li>
        <li><a href="/dashboard.html">Dashboard</a></li>
        <li><a href="/admin.html">Admin Panel</a></li>
        <li><a href="/track.html">Track Orders</a></li>
        <li><a href="/orders.html">Orders</a></li>
    </ul>
    """)
@app.get("/index.html")
def serve_index():
    return FileResponse(os.path.join(FRONTEND_DIR, "index.html"))
@app.get("/register.html")
def serve_register():
    return FileResponse(os.path.join(FRONTEND_DIR, "register.html"))
@app.get("/login.html")
def serve_login():
```

```
    return FileResponse(os.path.join(FRONTEND_DIR, "login.html"))
@app.get("/dashboard.html")
def serve_dashboard():
    return FileResponse(os.path.join(FRONTEND_DIR, "dashboard.html"))
@app.get("/admin.html")
def serve_admin():
    return FileResponse(os.path.join(FRONTEND_DIR, "admin.html"))
@app.get("/track.html")
def serve_track():
    return FileResponse(os.path.join(FRONTEND_DIR, "track.html"))
@app.get("/orders.html")
def serve_orders():
    return FileResponse(os.path.join(FRONTEND_DIR, "orders.html"))
notifications = []
def generate_receipt(order, path):
    c = canvas.Canvas(path)
    c.drawString(100, 750, f"Receipt for Order ID: {order.id}")
    c.drawString(100, 730, f"Item: {order.item_type}")
    c.drawString(100, 710, f"Quantity: {order.quantity}")
    c.drawString(100, 690, f"Hostel: {order.hostel_name or 'N/A'}")
    c.drawString(100, 670, f"Room No: {order.room_number or 'N/A'}")
    c.drawString(100, 650, f"Pickup Slot: {order.pickup_time_slot or 'N/A'}")
    c.drawString(100, 630, f"Total Price: ₹{order.total_price}")
    c.drawString(100, 610, f"Status: {order.status}")
    c.drawString(100, 590, f"Date: {order.date_created.strftime('%Y-%m-%d %H:%M')}")
    c.save()
@app.get("/download-receipt/{order_id}")
def download_receipt(order_id: str, db: Session = Depends(get_db)):
    order = db.query(models.Order).filter(models.Order.id == order_id).first()
    if not order:
        raise HTTPException(status_code=404, detail="Order not found")
    receipt_filename = f"receipt_{order.id}.pdf"
    receipt_path = os.path.join(RECEIPTS_DIR, receipt_filename)
    if not os.path.exists(receipt_path):
        generate_receipt(order, receipt_path)
   return FileResponse(path=receipt_path, filename=receipt_filename, media_type='application/pdf')
@app.get("/order/{order_id}", response_model=schemas.OrderOut)
def get_order_by_id(order_id: str, db: Session = Depends(get_db)):
    order = db.query(models.Order).filter(models.Order.id == order_id).first()
    if not order:
        raise HTTPException(status_code=404, detail="Order not found")
    return schemas.OrderOut.from_orm(order)
@app.post("/register")
def register(user: schemas.UserCreate, db: Session = Depends(get_db)):
    existing_user = db.query(models.User).filter(models.User.username == user.username).first()
    if existing_user:
        raise HTTPException(status_code=400, detail="Username already exists")
    new_user = models.User(
        id=str(uuid.uuid4()), username=user.username,
        password=user.password, email=user.email,
        phone=user.phone, plan=user.plan, role="user"
    )
    db.add(new_user)
    db.commit()
    return {"message": "User registered successfully"}
@app.post("/login")
def login(user: schemas.UserLogin, db: Session = Depends(get_db)):
    db_user = db.query(models.User).filter(
        models.User.username == user.username,
        models.User.password == user.password
    ).first()
```

```
    if not db_user:
        raise HTTPException(status_code=401, detail="Invalid credentials")
    return {
        "message": "Login successful",
        "user_id": db_user.id,
        "role": db_user.role,
        "plan": db_user.plan
    }
@app.post("/upload-request")
def upload_request(
    user_id: str = Form(...),
    item_type: str = Form(...),
    quantity: int = Form(...),
    note: str = Form(None),
    payment_method: str = Form(None),
    hostel_name: str = Form(None),
    room_number: str = Form(None),
    pickup_time_slot: str = Form(None),
    image: UploadFile = File(None),
    db: Session = Depends(get_db)
):
    image_url = None
    if image:
        file_path = os.path.join("uploads", f"{uuid.uuid4()}_{image.filename}")
        full_path = os.path.join(BASE_DIR, file_path)
        with open(full_path, "wb") as buffer:
            shutil.copyfileobj(image.file, buffer)
        image_url = file_path
    new_request = models.LaundryRequest(
       id=str(uuid.uuid4()), user_id=user_id,
        item_type=item_type, quantity=quantity,
        note=note, payment_method=payment_method,
        image_url=image_url, status="Pending",
        created_at=datetime.utcnow()
    )
    db.add(new_request)
    price_entry = db.query(models.Pricing).filter(models.Pricing.item_type == item_type).first()
    unit_price = price_entry.price if price_entry else 50
    total_price = unit_price * quantity
    new_order = models.Order(
        id=str(uuid.uuid4()), item_type=item_type,
        quantity=quantity, status="Pending", user_id=user_id,
        hostel_name=hostel_name, room_number=room_number,
        pickup_time_slot=pickup_time_slot, total_price=total_price,
        date_created=datetime.utcnow()
    )
    db.add(new_order)
    db.commit()
  notifications.append(f"New request for {item_type} by user {user_id}")
    return {"message": "Laundry request and order submitted", "request_id": new_request.id, "order_id": new_order.id}
@app.get("/order-history/{user_id}")
def order_history(user_id: str, month: int = None, year: int = None, db: Session = Depends(get_db)):
    query = db.query(models.Order).filter(models.Order.user_id == user_id)
    if month or year:
        ist = pytz.timezone("Asia/Kolkata")
        now = datetime.now(ist)
        if not year: year = now.year
        if not month: month = now.month
        query = query.filter(
            extract("month", models.Order.date_created) == month,
            extract("year", models.Order.date_created) == year
```

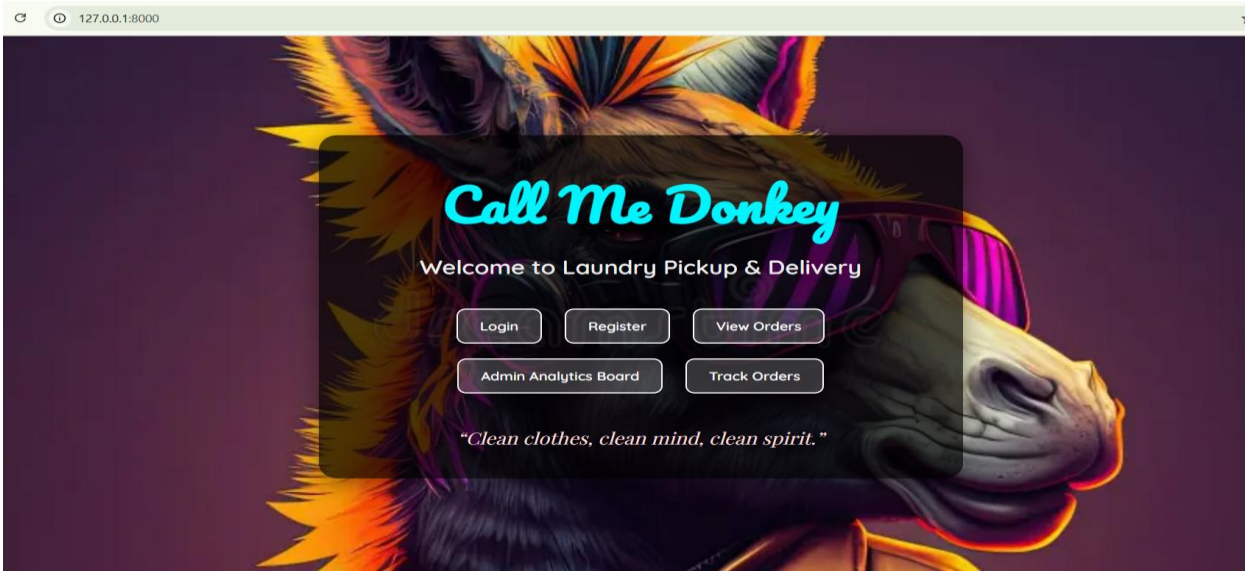
```
        )
    return query.all()
@app.get("/all-orders", response_model=List[schemas.OrderOut])
def get_all_orders(db: Session = Depends(get_db)):
    orders = db.query(models.Order).all()
    return [schemas.OrderOut.from_orm(order) for order in orders]

@app.get("/pricing")
def get_all_pricing(db: Session = Depends(get_db)):
    prices = db.query(models.Pricing).all()
    return {p.item_type: p.price for p in prices}
@app.post("/pricing")
def set_pricing(pricing: schemas.PricingUpdate, db: Session = Depends(get_db)):
    existing = db.query(models.Pricing).filter(models.Pricing.item_type == pricing.item_type).first()
    if existing:
        existing.price = pricing.price
    else:
        new_price = models.Pricing(item_type=pricing.item_type, price=pricing.price)
        db.add(new_price)
    db.commit()
    return {"message": "Pricing updated"}
@app.get("/calculate-price")
def calculate_price(item_type: str, quantity: int, db: Session = Depends(get_db)):
    entry = db.query(models.Pricing).filter(models.Pricing.item_type == item_type).first()
    price = entry.price if entry else 50
    return {"estimated_price": price * quantity}
@app.get("/notifications")
def get_notifications():
    return notifications[-10:]
@app.get("/user/notifications/{user_id}")
def user_notifications(user_id: str, db: Session = Depends(get_db)):
    orders = db.query(models.Order).filter(models.Order.user_id == user_id, models.Order.status == "Completed").all()
    return [{"order_id": o.id, "status": o.status} for o in orders]
@app.get("/admin/summary")
def get_admin_summary(db: Session = Depends(get_db)):
    total_users = db.query(func.count(models.User.id)).scalar()
    completed_orders = db.query(func.count(models.Order.id)).filter(models.Order.status == "Completed").scalar()
    total_revenue = db.query(func.sum(models.Order.total_price)).scalar() or 0
    most_washed = db.query(models.Order.item_type, func.count(models.Order.item_type).label("count"))\
        .group_by(models.Order.item_type).order_by(func.count(models.Order.item_type).desc()).first()
    return {
        "total_users": total_users,
        "completed_orders": completed_orders,
        "total_revenue": total_revenue,
        "most_washed_item": most_washed[0] if most_washed else "N/A"
    }
@app.get("/create-admin")
def create_admin(db: Session = Depends(get_db)):
    existing = db.query(models.User).filter(models.User.username == "admin").first()
    if existing:
        return {"message": "Admin already exists"}
    admin = models.User(
        id=str(uuid4()), username="admin", password="admin123",
        email="admin@example.com", phone="9999999999",
        plan="premium", role="admin"
    )
    db.add(admin)
    db.commit()
    return {"message": "✅ Admin created: username=admin, password=admin123"}
@app.get("/admin/orders-per-month")
def orders_per_month(db: Session = Depends(get_db)):
```

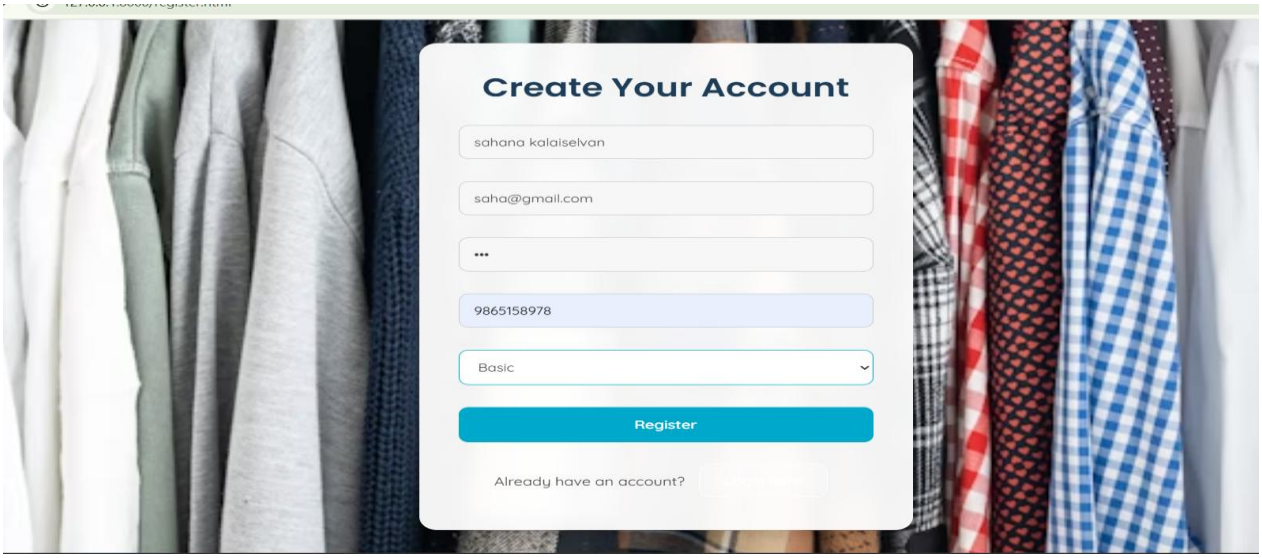
```
    results = db.query(
        extract("month", models.Order.date_created).label("month"),
        func.count(models.Order.id).label("count")
    ).group_by(extract("month", models.Order.date_created)).all()
    return [{"month": int(month), "count": count} for month, count in results]
@app.put("/admin/update-status/{order_id}")
def mark_order_completed(order_id: str, db: Session = Depends(get_db)):
    order = db.query(models.Order).filter(models.Order.id == order_id).first()
    if not order:
        raise HTTPException(status_code=404, detail="Order not found")
    if order.status != "Pending":
        raise HTTPException(status_code=400, detail="Only pending orders can be marked as completed")
    order.status = "Completed"
    db.commit()
    notifications.append(f"✅ Order {order.id} completed for user {order.user_id}")
    return {"message": "Order marked as completed and user notified"}
@app.put("/cancel-order/{order_id}")
def cancel_order(order_id: str, db: Session = Depends(get_db)):
    order = db.query(models.Order).filter(models.Order.id == order_id).first()
    if not order:
        raise HTTPException(status_code=404, detail="Order not found")
    if order.status != "Pending":
        raise HTTPException(status_code=400, detail="Only pending orders can be cancelled")
    order.status = "Cancelled"
    db.commit()
    @app.post("/feedback/{order_id}")
def submit_feedback(order_id: str, feedback: str = Form(...), db: Session = Depends(get_db)):
    order = db.query(models.Order).filter(models.Order.id == order_id).first()
    if not order:
        raise HTTPException(status_code=404, detail="Order not found")
    order.feedback = feedback
    db.commit()
    return {"message": "Feedback submitted successfully"}
```

DEMONSTRATION:

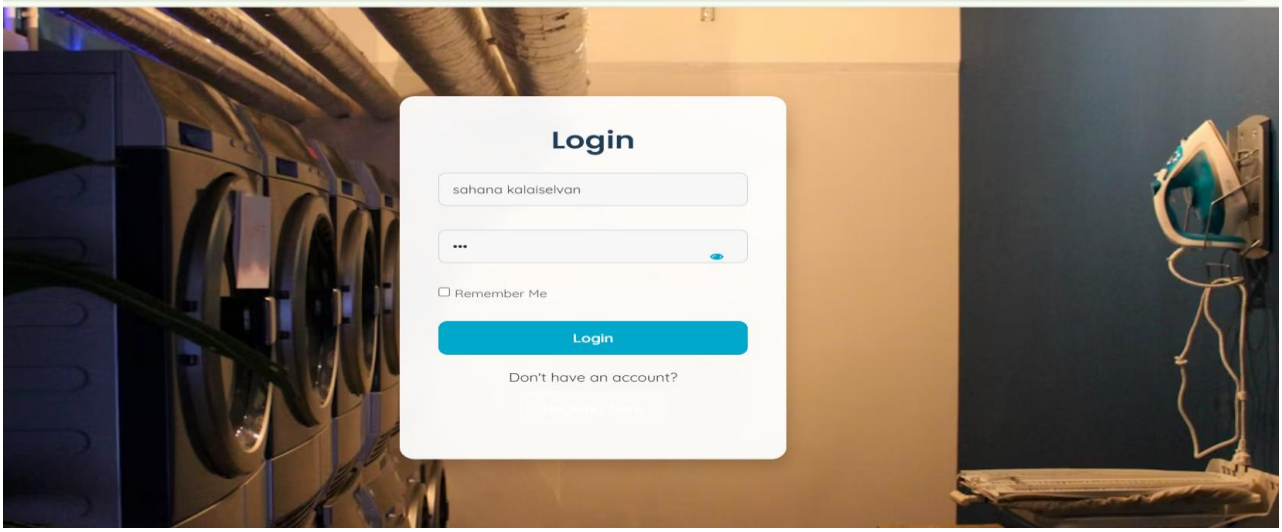
HOME PAGE:



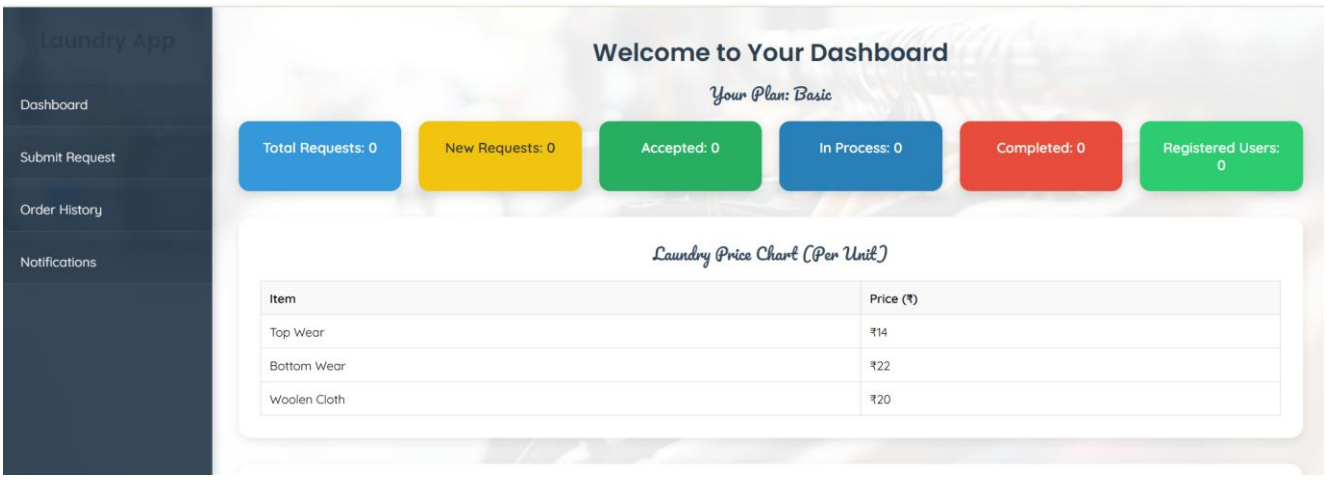
REGISTRATION PAGE:

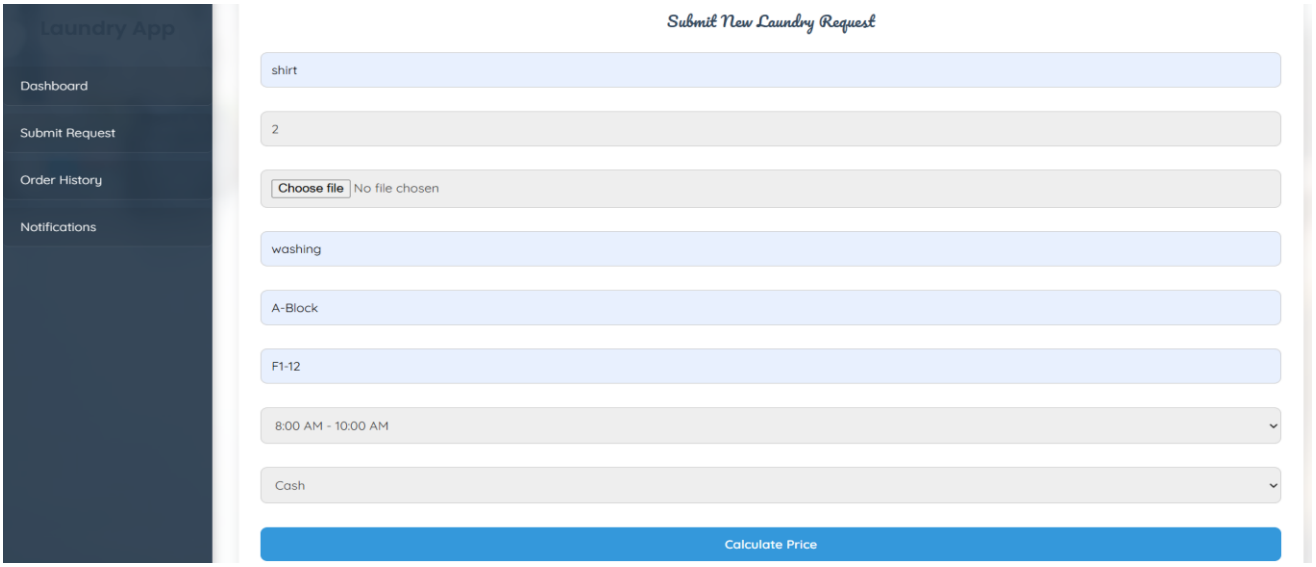


LOGIN PAGE:

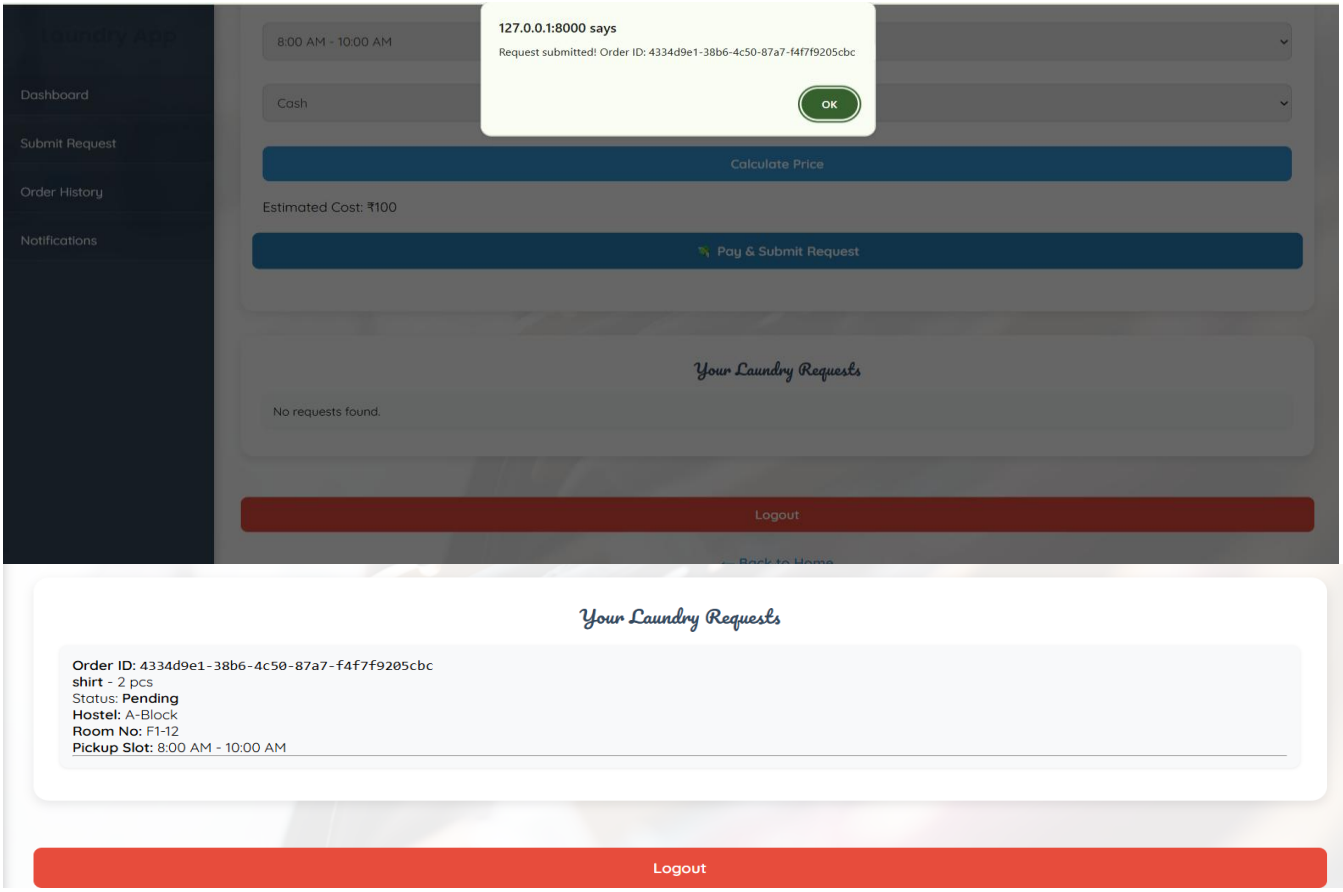


DASHBOARD PAGE:

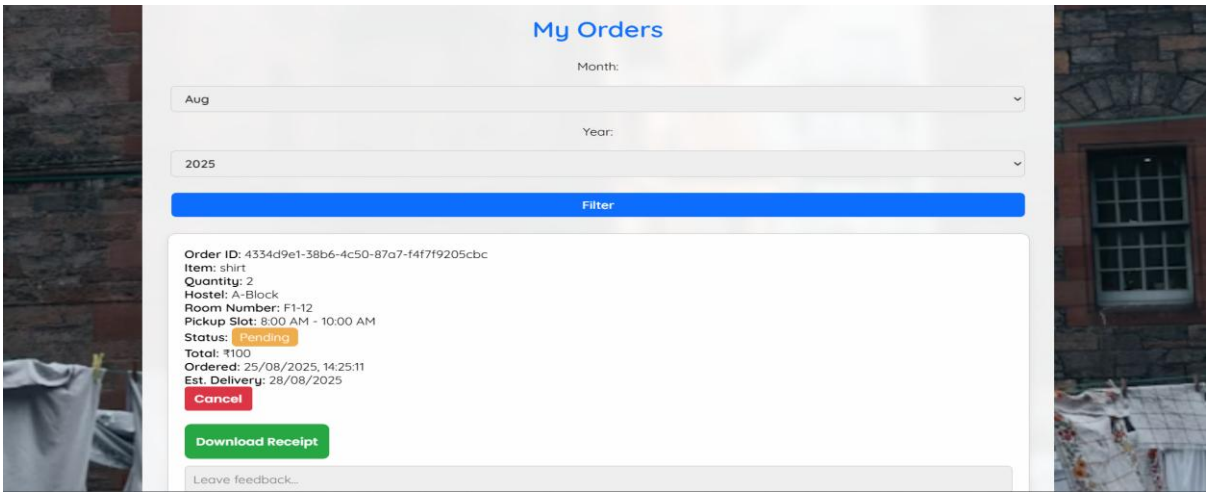




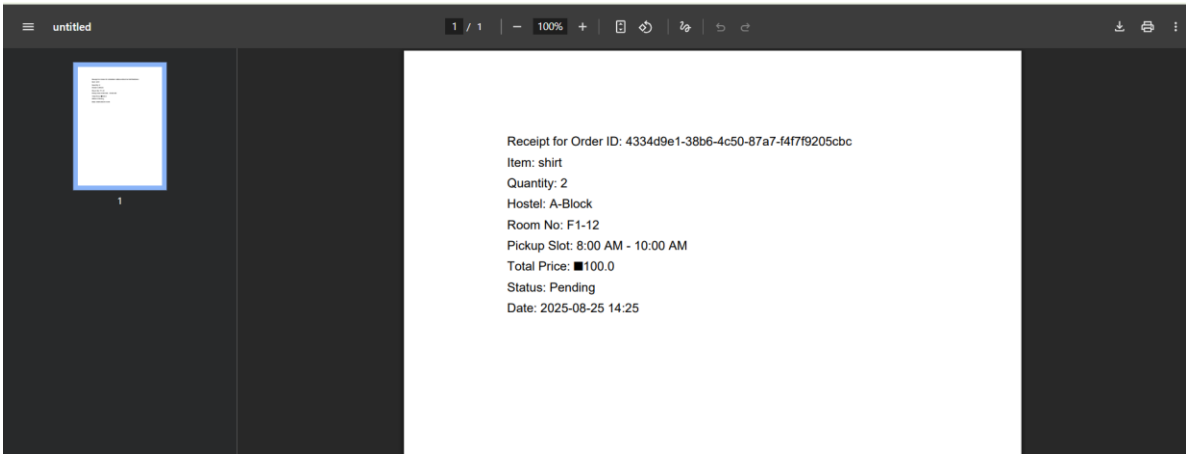
UUID FOR TRACKING:



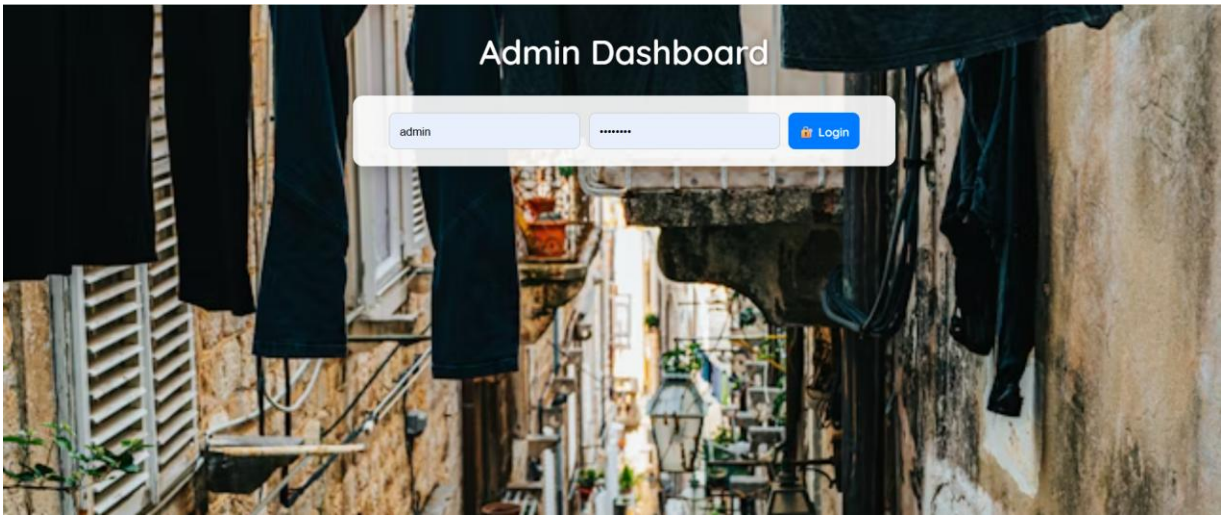
ORDERS PAGE:



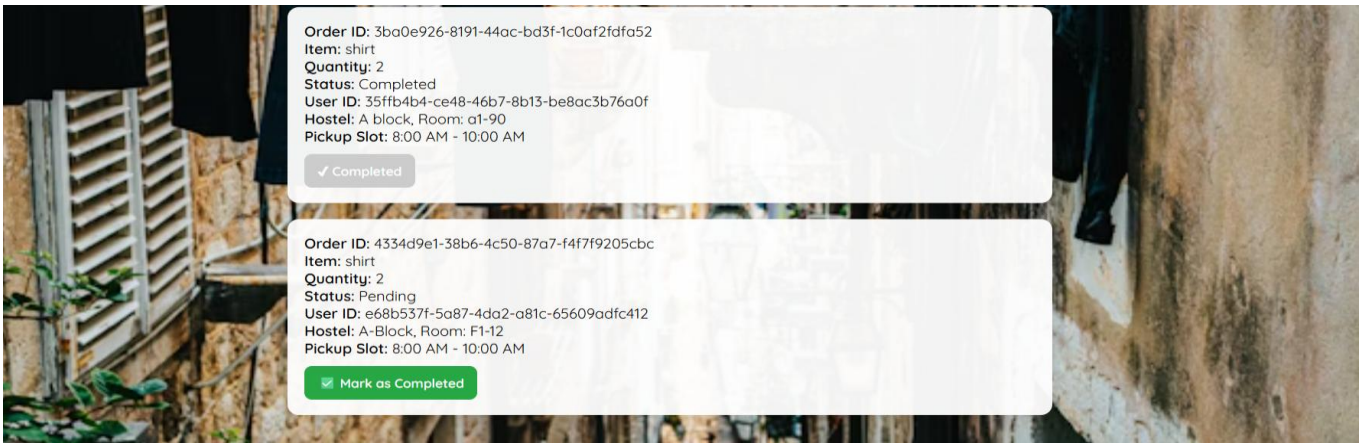
GENERATION OF PDF RECEIPTS:



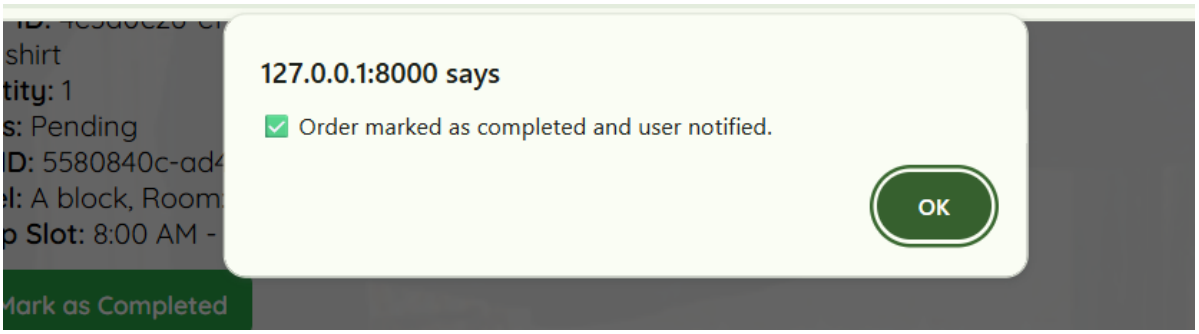
ADMIN LOGIN:



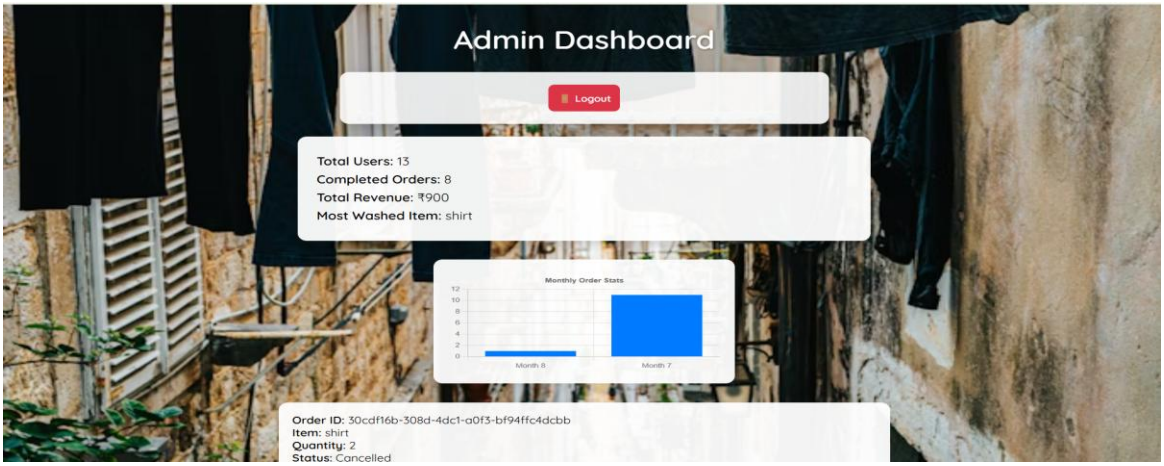
ADMIN DASBOARD:



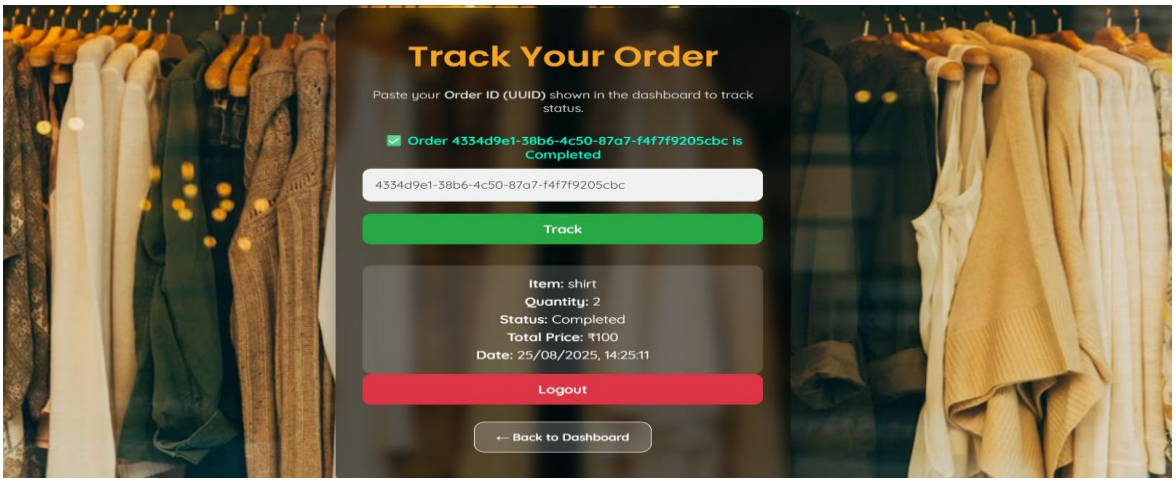
NOTIFICATION:



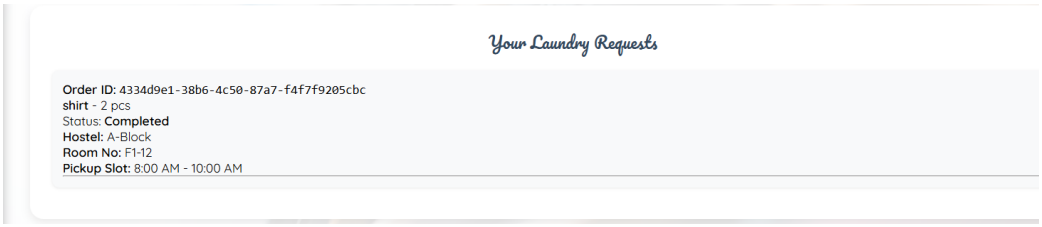
GENERATION OF CHART:



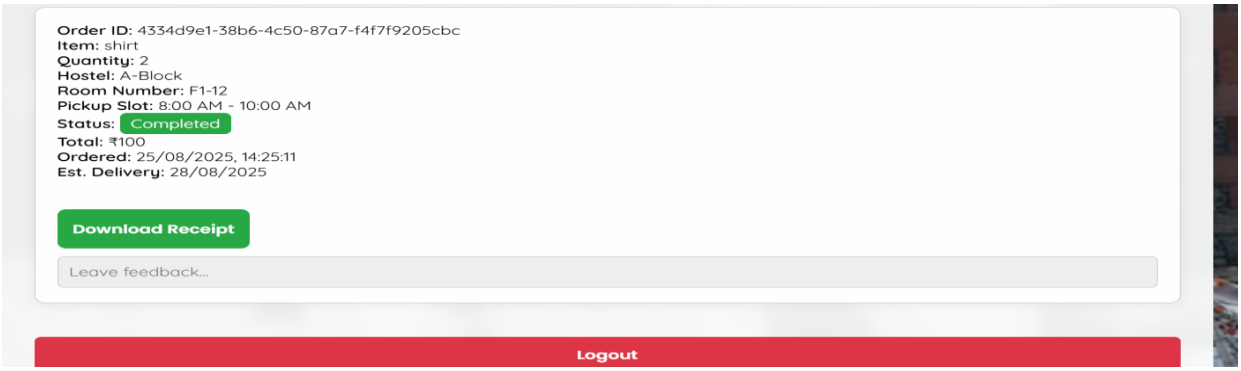
ORDER TRACKING:



REAL-TIME NOTIFICATION:



ORDER HISTORY*:*



FEEDBACK SUBMISSION: