**CLEAN WASH**

**a**

**Laundry Management System**

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfilment of the requirements to award the degree of

**Bachelor of Technology**

In

**Computer Science and Engineering**

**School of Engineering and Sciences**

*Submitted by*

Gamidi Bala Satwick – AP22110010214

Chelikani Aakash – AP22110010225

Mopuru Tharun – AP22110010230

Potugari Venkata Ravi Teja – AP22110010256



Under the Guidance of

**Dr** **Singarapu Pavan Kumar,**

**Assistant professor, Dept. Of CSE**

**SRM University–AP**

**Neerukonda, Mangalagiri, Guntur**

**Andhra Pradesh – 522 240**

**April 2025**

**CERTIFICATE**

Date: 23-04-2025

This is to certify that the work present in this Project entitled “**Clean Wash a Laundry Management System**” has been carried out by Gamidi Bala Satwick, chelikani Aakash, Mopuru Tharun, Potugari Ravi Teja under my supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology in School of Engineering and Sciences.

Supervisor

(Signature)

Dr. **Singarapu Pavan Kumar,**

Assistant Professor,

Department of CSE

**Acknowledgements**

We would like to sincerely acknowledge the invaluable guidance and unwavering support of our mentor and advisor, Dr. Singarapu Pavan Kumar. Your wisdom, experience, and thoughtful feedback were instrumental in shaping our ideas and steering the project in the right direction.

I would also like to extend my heartfelt gratitude to my team members, whose dedication, collaboration, and shared insights played a pivotal role in the success of this project. Your teamwork and commitment truly made a difference.

Furthermore, we are deeply thankful to our university for providing us with the necessary resources, infrastructure, and opportunities that enabled us to carry out this project successfully.

**Abstract**

The Clean Wash Flow System is a comprehensive, full-stack web application designed to digitize and optimize the on-campus laundry service workflow for students and staff at SRM University. Its primary objective is to replace paper-based requests and fragmented tracking methods with a unified, role-based platform that covers every step—from order submission through washing, drying, and ready-for-pickup notifications.

Students access a secure Student Dashboard to place new laundry orders (specifying pickup location, item count, and special instructions), view real-time status updates as their orders progress through stages (Received → Washing → Drying → Ready), and receive alerts when their laundry is complete. Staff members use a dedicated Operations Dashboard to claim incoming jobs, update statuses via a streamlined form, and communicate order readiness instantly. All interactions occur in a single page React interface, eliminating full-page reloads and minimizing user friction.

Under the hood, the system leverages Supabase as a Backend-as-a-Service—providing a hosted PostgreSQL database, built-in JWT-based authentication, and WebSocket-powered realtime subscriptions for live status updates. The frontend is built with React 18 and TypeScript, bundled via Vite for lightning-fast hot reloading, and styled with Tailwind CSS alongside Radix UI primitives for a consistent, accessible design. Deployment on Vercel ensures global CDN delivery and seamless CI/CD.

By centralizing laundry orders, enforcing secure role-based access, and delivering instant notifications, the Clean Wash Flow System greatly reduces manual errors, shortens turnaround times, and enhances transparency. Its scalable, serverless architecture and modern tech stack lay a robust foundation for future enhancements such as QR-code-based check-in/check-out, push notifications, and data-driven analytics ultimately elevating the campus laundry experience for all stakeholders.

**TABLE OF CONTENTS**

| **S.NO** | **CONTENT** | **PAGE NO.** |
| --- | --- | --- |
|  | ABSTRACT | IV |
|  | LIST OF FIGURES | VII |
| 1. | INTRODUCTION | 8 |
| 2. | SYSTEM REQUIREMENTS | 11 |
| 3. | SYSTEM DESIGN | 14 |
| 3.1 | CONTEXT LEVEL DIAGRAM (LEVEL 0 DFD) | 14 |
| 3.2 | LEVEL 1 DFD | 16 |
| 3.3 | DATA DICTIONARY | 18 |
| 4. | FUNCTION ORIENTED DIAGRAMS | 20 |
| 4.1 | ENTITY RELATIONSHIP (ER) DIAGRAM | 20 |
| 5. | USER’S VIEW ANALYSIS | 23 |
| 5.1 | USE CASE DIAGRAM | 23 |
| 5.2 | USE CASE SCENARIOS | 24 |
| 6. | STRUCTURAL VIEW DIAGRAMS | 26 |
| 6.1 | CLASS DIAGRAM | 26 |
| 6.2 | OBJECT DIAGRAM | 28 |
| 6.3 | PACKAGE DIAGRAM | 30 |
| 7. | BEHAVIORAL VIEW DIAGRAM | 32 |
| 7.1 | SEQUENCE DIAGRAM | 33 |
| 7.2 | COLLABORATION DIAGRAM | 34 |
| 8. | IMPLEMENTATION | 37 |
| 8.1 | TECHNOLOGIES USED (MERN STACK, JWT, NODEMAILER) | 37 |
| 8.2 | STUDENT DASHBOARD FUNCTIONALITIES | 38 |
| 8.3 | RECRUITER DASHBOARD FUNCTIONALITIES | 41 |
| 8.4 | EMAIL NOTIFICATION SYSTEM | 43 |
| 9. | RESULT AND DISCUSSION | 45 |
| 10. | CONCLUSION & FUTURE WORK | 52 |
| 11. | REFERENCES | 54 |

LIST OF FIGURES

1. Figure-1: DFD Level-0
2. Figure-2: DFD Level-1
3. Figure-3: ER Diagram
4. Figure-4: Use case Diagram
5. Figure-5: Use scenario Diagram
6. Figure-6: Class Diagram
7. Figure-7: Object Diagram
8. Figure-8: Package Diagram
9. Figure-9: Sequence Diagram
10. Figure-10: Collaboration Diagram

LIST OF TABLES

* + - 1. Table-1: Data Dictionary

**INTRODUCTION**

The rapid evolution of digital technologies has reshaped how campus services are delivered—but many student‐run laundry workflows remain stuck in paper forms and ad-hoc communication. The Clean Wash Flow System is a modern, web-based solution that digitizes and streamlines every step of the on-campus laundry process. Built as a single-page React 18+TypeScript application (bundled with Vite and styled via Tailwind CSS/Radix UI), it offers secure, role-based interfaces: students submit orders, track real-time status updates (Accepted → Processing → Completed), and receive instant notifications, while operations staff claim, update, and manage jobs from a dedicated dashboard. Under the hood, Supabase provides a hosted PostgreSQL database, JWT authentication, and WebSocket-powered realtime subscriptions, eliminating the need for a custom server. By centralizing order placement, realtime tracking, and notifications, this system reduces manual errors, accelerates turnaround, and delivers full transparency for both students and service staff.

**1.1. Background of the Study**

In the current campus laundry workflow, many institutions still rely on paper request slips, phone calls, and scattered spreadsheets to manage student orders. Staff track jobs on whiteboards or in loose physical logs, leading to missed requests, lost notes, and constant back-and-forth to clarify details. Without a unified system, it’s difficult to know which orders are pending, who’s responsible, or how long each batch has been in process—and there’s no easy way to drill into historical data for performance or cost analysis.

The Clean Wash Flow System remedies these pain points by providing a fully integrated, end-to-end laundry management platform. Students authenticate with their university credentials, submit pickup location, item counts, and special instructions via a web form, then watch their order progress in real time (Accepted → Processing → Completed). Operations staff see a live queue of new jobs, can claim and update orders with a single click, and trigger instant in-app. Because every action is timestamped in a centralized PostgreSQL database, administrators gain visibility into turnaround times, peak demand periods, and staff utilization—transforming a once-fragmented process into a transparent, data-driven service.

**1.2. Objectives of the Project**

**The primary objectives of the Clean Wash Flow System are:**

* **To develop an interactive web portal** using React 18 + TypeScript, Vite, Tailwind CSS and Radix UI, backed by Supabase (PostgreSQL + Auth + Realtime), that serves both students and operations staff.
* **To enable seamless order submission and tracking**, allowing students to place laundry requests (specifying pickup location, item count, special instructions) and monitor each order’s progress through stages (Received → Washing → Drying → Ready).
* **To automate status updates and notifications**, delivering instant in-app alerts and optional email notifications whenever an order advances or is ready for pickup.
* **To provide secure authentication and role-based access control**, ensuring only authenticated students can submit orders and only authorized staff can view and update job statuses.
* **To maintain a centralized database** for all laundry orders, user profiles, status histories, and timestamps, enabling reliable record-keeping and auditability.
* **To ensure real-time visibility**, leveraging Supabase’s WebSocket-powered subscriptions so both students and staff see live updates without manual refreshes.

**1.3. Scope of the Project**

* **Student Accounts:** Students register with their university email, set up their profile (contact details and preferred pickup locations), and review a history of all past laundry orders.
* **Staff Dashboard:** Operations personnel log in to see a live queue of new requests, claim jobs, and advance each order through its stages with a single click.
* **Status Tracking:** Each laundry order displays clear, timestamped progress markers (Accepted → Processing → Completed) that update in real time.
* **Notifications:** Students receive immediate in-app alerts—whenever their order status changes or is ready for pickup.
* **Authentication:** Secure, JWT-based login ensures only verified students can place orders and only authorized staff can access operational tools.
* **Database:** A single Supabase-hosted PostgreSQL instance stores all user profiles, orders, status histories, and notification records for consistent, reliable data access.

**1.4. Problem Statement**

Managing campus laundry services involves multiple stakeholders students placing orders, operations staff processing them, and administrators overseeing workflows. The traditional approach paper request slips, phone calls, whiteboard logs, and scattered spreadsheets introduces several inefficiencies:

* **Order Visibility Gaps:** Hard to know which requests are pending, in process, or completed.
* **No Real-Time Updates:** Students receive no instant notification when their laundry moves between stages.
* **High Administrative Overhead:** Staff juggle manual logs, leading to missed or duplicate orders.
* **Limited Scalability:** As order volume grows, paper-based tracking and ad-hoc communication buckle under the load.

These shortcomings cause lost slips, pickup delays, and frustration for everyone involved. The Clean Wash Flow System addresses them by providing a centralized, automated platform built with modern web technologies and a scalable serverless backend.

**2. SYSTEM REQUIREMENTS**

To ensure smooth operation and deployment of the **Clean Wash**, both hardware and software requirements need to be specified clearly.

**2.1. Hardware Requirements**

| **Component** | **Minimum Requirement** |
| --- | --- |
| **Processor** | **Intel i5 (8th Gen) or equivalent** |
| **RAM** | **8 GB (Recommended: 16 GB)** |
| **Storage** | **100 GB SSD** |
| **Display** | **14″–15.6″, 1080p resolution** |
| **Internet** | **Stable broadband (≥ 10 Mbps)** |
| **Development Server** | **Local dev machine as above** |
| **Deployment Environment** | **Serverless (Vercel/Netlify/Supabase) or on-prem with ≥ 4 vCPUs, 8 GB RAM** |

These specifications are sufficient for development, testing, and small-scale deployment. For production-level scalability, cloud deployment with autoscaling support is preferred.

**2.2. Software Requirements**

| **Software / Tool** | **Purpose** |
| --- | --- |
| Operating System | Windows 10/11, macOS 12+, or Ubuntu 20.04+ |
| Code Editor | Visual Studio Code (or any IDE with TypeScript & React support) |
| Web Browser | Chrome, Firefox, or Edge (latest stable release) |
| Runtime & Bundler | Node.js v18+ & Vite (for local dev server and hot reload) |
| Package Manager | npm or Yarn |
| Frontend Framework | React.js + TypeScript |
| Styling | Tailwind CSS & Radix UI |
| Backend-as-a-Service | Supabase (PostgreSQL, Auth, Realtime) |
| CLI & Management | Supabase CLI (for local emulation and migrations) |
| Version Control | Git + GitHub (or GitLab) |
| API Testing / Database | Postman or Supabase Studio |
| Deployment | Vercel, Netlify, or Supabase hosting |

**2.3. Technologies Used Overview**

This project uses a modern React + Supabase stack in place of the traditional MERN components. Here’s a brief overview:

* **Supabase**:  
  A Backend-as-a-Service providing:
  + **PostgreSQL** for relational data (orders, users, status logs)
  + **Auth** for JWT-based sign-up/login and role-based access
  + **Realtime** subscriptions for live order updates
* **React.js**:

A component-driven UI library (v18) in TypeScript, powering the single-page application and dynamic dashboards for students and staff.

* **Vite:**  
  A lightning-fast build tool and dev server that supports ES modules, hot-module replacement, and TypeScript out of the box.
* **Tailwind CSS & Radix UI**  
  Utility-first styling (Tailwind) combined with accessible, headless UI primitives (Radix) to deliver a consistent, responsive design without writing custom CSS components.
* **TypeScript:**  
  Static typing across both frontend and integration code, ensuring early error detection and improved maintainability.
* **Vercel(orsimilar):**  
  Serverless deployment for the frontend, with CI/CD, global CDN, and zero-config SSL while Supabase handles the backend hosting and scaling.

**Additional Libraries/Tools:**

**• Supabase Auth & Notifications –** Supabase Auth issues and manages JWTs under the hood for secure log-in/sign-up, and its built-in SMTP settings can be used to send email alerts on status changes**.  
• @supabase/supabase-js –** The single client library for all CRUD, auth, and realtime websocket calls—so there’s no need for Axios or a separate HTTP client.  
**• Tailwind CSS & Radix UI –** Utility-first styling (Tailwind) combined with headless, accessible primitives (Radix) replaces both Material UI and custom CSS components. **• PostgreSQL via Supabase** – All data modeling is done directly in Postgres tables; there’s no ODM like Mongoose or manual schema syncing.  
**• Vite + TypeScript –** Fast ES-module bundling and hot reload, with static typing across the entire codebase.

**3. SYSTEM DESIGN**

System design is a crucial phase in software development that lays out the blueprint of the Clean Wash Flow System, defining its structure, data flows, and component interactions. In this project, it ensures that all stakeholders—students placing orders, operations staff processing them, and administrators overseeing workflows—are accounted for in a modular, scalable, and secure architecture.

This section presents:

* **Context-Level Diagram**: Shows the system as a single process interacting with external entities (students, staff, notification services).
* **Level 1 Data Flow Diagram**: Breaks down the core processes (order submission, status updates, notification dispatch) and the data stores they read from/write to.
* **Data Dictionary**: Defines each data element (e.g. Order, User, StatusLog, Notification) and its attributes.
* **System Architecture Diagram**: Illustrates how the React/Vite front end, Supabase backend (PostgreSQL, Auth, Realtime), and deployment platform (Vercel) fit together.

Together, these views map exactly how a laundry request travels through the platform—from initial form submission to real-time status updates—and clarify the roles of each software component in that journey.

**3.1. Context Level Diagram (Level 0 DFD)**

**Purpose:** Represents the Clean Wash Flow System as a single process interacting with its external entities.

**External Entities:**

* **Student:**
  + Registers or logs in via their university email
  + Places laundry orders (item counts, pickup locations, special instructions)
  + Views real-time status updates and receives notifications
* **Worker:**
  + Logs in with staff credentials
  + Views the queue of incoming orders
  + Claims jobs and advances each order’s status (Accepted → Processing → Completed)

**Process (Laundry Management System)**

* Takes input from students, workers.
* Provides outputs such as item listings, order confirmations, status updates, delivery notifications.

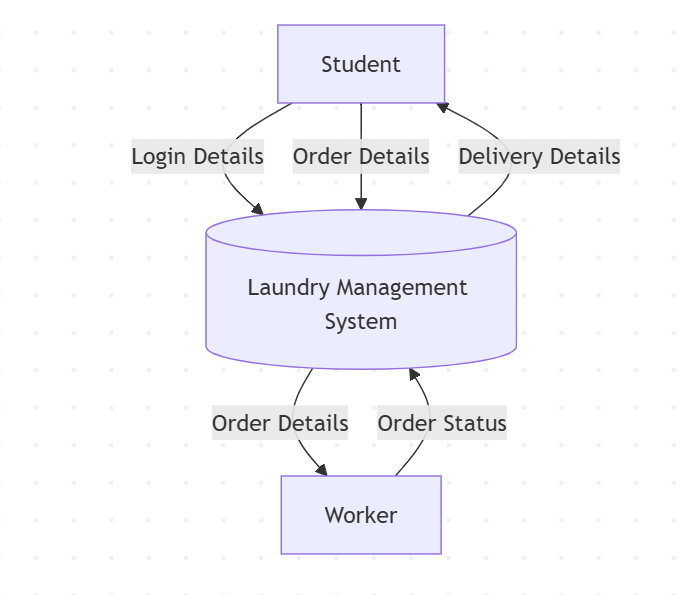


Figure-1: DFD Level-0

**3.2. Level 1 DFD:**

**Purpose:  
The Level 1 DFD breaks the Clean Wash Flow System into its key sub-processes and shows how data moves between external entities, processes, and data stores.**

**Main Components in Level 1 DFD:**

1. **External Entities (Rectangles):**
   * **Students:** Authenticate, submit laundry orders, view status updates.
   * **Workers:** Retrieve pending orders, claim jobs, advance statuses**.**
2. **Processes (Circles/Rounded Rectangles):**
   * **User Registration & Profile Management**
     + **Stores student and staff credentials and profile details in the Users & Roles DB.**
     + **Allows updates to contact info, pickup preferences, and staff roles.**
   * **Order Submission**
     + **Captures new laundry requests (item counts, pickup location, instructions).**
     + **Writes each new order to the Orders DB with an initial “Pending” status.**
   * **Order Processing & Status Update**
     + **Operations staff fetch “Received” orders, claim them, and advance statuses (Accepted→ Processed → Completed).**
     + **Updates are written back to the Orders DB, maintaining a full timestamped history.**
   * **Notification Dispatch**
     + **Logs every status change in the Notifications Log data store.**
     + **Sends real-time in-app alerts to students on each status update.**
3. **Data Stores (Open-Ended Rectangles or Parallel Lines):**

**Laundry Orders Database:**

* **Profiles Store (**Stores student and worker account/profile details)
* **Clothing Items Store (**Stores catalog of available laundry items**)**
* **Orders Store (**Stores all laundry order records with status and timestamps**)**

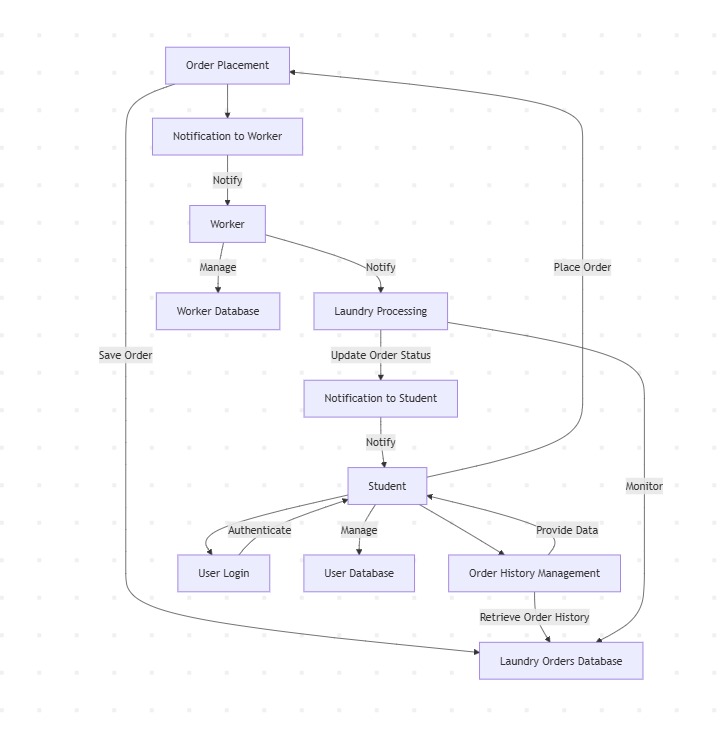


Figure-2: DFD Level-1

**3.3. Data Dictionary**

The data dictionary provides a detailed description of all data elements and data stores used in the system. It helps in understanding the structure and semantics of data flowing through the system. Key entries include:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Table Name | Field Name | Data Type | Description | Constraints |
| profiles | id | |  | | --- | | UUID | | |  |  |  | | --- | --- | --- | | |  | | --- | |  |  |  | | --- | | Unique identifier (matches auth.users.id) | | | |  | | --- | | Primary Key, Required | |
| profiles | full\_name | TEXT | User’s full name | Required |
| profiles | email | TEXT | User’s email address | Required |
| profiles | gender | TEXT | User’s gender | Required |
| profiles | role | TEXT | “student” or “worker” | Required |
| profiles | registration\_number | TEXT | Student’s registration number | Nullable |
| profiles | worker\_id | TEXT | Auto-generated worker ID | Nullable |
| profiles | hostel | TEXT | |  | | --- | |  |  |  | | --- | | Student’s hostel block | | Nullable |
| profiles | floor | TEXT | Student’s floor | Nullable |
| profiles | Assigned\_hotel | TEXT | |  | | --- | |  |  |  | | --- | | Worker’s assigned hostel block | | Nullable |
| profiles | Created\_at | TIMESTAMPTZ | Profile creation timestamp | Required |
| profiles | washes\_left | INT4 | Remaining wash allowance (students only) | Required |
| profiles | total\_washes | INT4 | Total wash allowance (students only) | Required |
| orders | id | UUID | Unique identifier for the order | Primary Key, Required |
| orders | student\_id | UUID | References profiles.id (student who ordered) | Foreign Key, Required |
| orders | worker\_id | UUID | References profiles.id (assigned worker) | Foreign Key, Nullable |
| orders | status | Order\_status | Current order status (pending → completed) | Required |
| orders | total\_price | Numeric | Sum of all item prices | Required |
| orders | pickup\_date | TIMESTAMPTZ | Scheduled pickup date/time | Required |
| orders | delivery\_date | TIMESTAMPTZ | Scheduled delivery date/time | Nullable |
| orders | notes | TEXT | Optional student notes | Nullable |
| orders | Created\_At | TIMESTAMPTZ | timestamp when the order was created. | Required |
| orders | Updated\_at | TIMESTAMPTZ | Timestamp when the order was last updated | Required |
| orders | floor | TEXT | |  | | --- | |  |  |  | | --- | | Pickup floor specified by student | | Required |
| orders | block | TEXT | Hostel block used for routing the order | Required |
| order\_items | id | UUID | Unique identifier for the order-item record | Primary Key, Required |
| order\_items | order\_id | UUID | References orders.id (parent order) | Foreign key,Required |
| order\_items | clothing\_item\_id | UUID | References clothing\_items.id (selected item) | Foreign key,Required |
| order\_items | quantity | INT4 | |  | | --- | |  |  |  | | --- | | Number of units ordered | | Required |
| order\_items | price | NUMERIC | Total price for this line (unit price × quantity) | Required |
| clothing\_items | id | UUID | Unique identifier for clothing item | Primary Key, Required |
| clothing\_items | name | TEXT | Name of the clothing item | Required |
| clothing\_items | gender | Gender\_type | Applicable gender for the item | Required |
| clothing\_items | price | NUMERIC | Price per unit | Required |
| clothing\_items | description | TEXT | |  | | --- | |  |  |  | | --- | | Optional description of the item | | Nullable |

Table-1: Data Dictionary

**4. FUNCTION ORIENTED DIAGRAMS**

The functional view of the system helps in analyzing how CleanWash behaves in terms of user input, data processing, and output generation. Since our Data Flow Diagrams were detailed under System Design (Sections 3.1 & 3.2), this section will instead focus on the Entity-Relationship (ER) Diagram, which captures the structure and relationships of core data entities in CleanWash.

**4.1. ENTITY RELATIONSHIP (ER) DIAGRAM**

The ER diagram is a high-level data model that describes the main entities in the CleanWash laundry system and how they relate. It’s essential for designing an efficient, consistent database schema.

**Key Entities:**  
• **Profile**: Stores user account details (full\_name, email, gender, role, registration\_number or worker\_id, hostel, floor, etc.) tied to auth.users.id.  
• **ClothingItem**: Catalog of available laundry items (id, name, gender, price, description).  
• **Order**: Records each laundry request (id, student\_id, worker\_id, status, total\_price, pickup\_date, delivery\_date, floor, block, notes, timestamps).  
• **OrderItem**: Line-item entries linking Orders to ClothingItems (id, order\_id, clothing\_item\_id, quantity, price).

**Relationship Overview:**

• **One-to-Many**: A single Profile (role=student) can place many Orders.  
• **One-to-Many:** Each Order can contain many OrderItems.  
• **Many-to-One**: Each OrderItem references one ClothingItem.  
• **One-to-Many**: A single Profile (role=worker) can be assigned to many Orders.

**Benefits:**

* Provides visual clarity on entity cardinalities and foreign keys.
* Guides Supabase table and constraint definitions.
* Forms the foundation for API endpoint design and React state modeling.
* Ensures data consistency, enforces business rules (status flows, role-based access), and simplifies future schema extensions.



Figure-3: ER Diagram

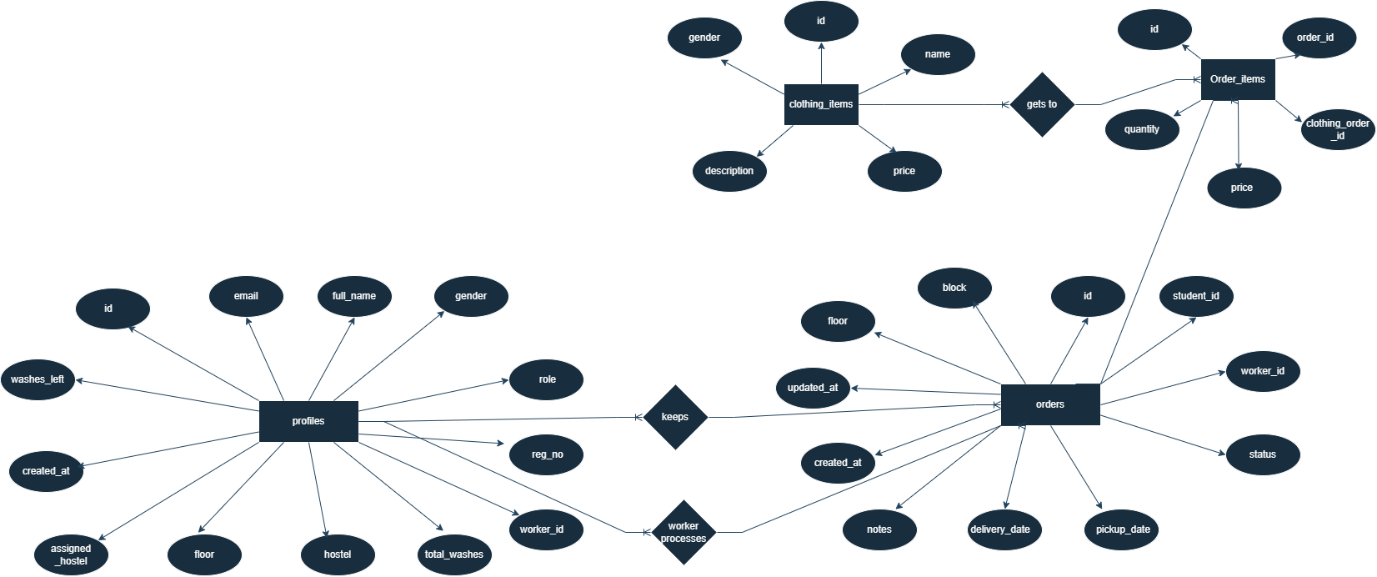


Figure-3.1: ER Diagram

**5. USER’S VIEW ANALYSIS**

**User view analysis** plays a key role in understanding how different users engage with the system. It centers on identifying various user roles, their interactions, and how the system responds to those actions. This perspective is essential for highlighting system functionalities from the user's point of view, ensuring the interface and interactions are user-friendly and comprehensive.

User analysis is typically divided into two main components:

* **Use Case Diagrams**
* **Use Case Scenarios**

These tools are fundamental for gathering requirements and mapping out the functional flow of the system.

**5.1 USE CASE DIAGRAM**

A Use Case Diagram is a UML behavioral diagram that shows how the three CleanWash actors (Students, Workers, and Admins) interact with the system to accomplish laundry-related tasks. Each use case represents a discrete function the system offers in response to an actor’s action.

**✦ Key Elements:**

* **Actors**: The users who interact with the system (e.g., Student, Worker).
* **Use Cases**: The services the system provides (e.g., Accept and process order, Update Order Status).
* **System Boundary**: Represents the scope of the system.
* **Relationships**: Connect actors and use cases via communication paths.

**✦ Purpose:**

* To capture all system functions from each actor’s perspective
* To visualize high-level functionality and define system scope
* To guide developers and testers in validating user requirements

**✦ Benefits:**

* Enhances communication with stakeholders.
* Serves as a blueprint for design and testing phases.
* Helps in identifying all possible user interactions.

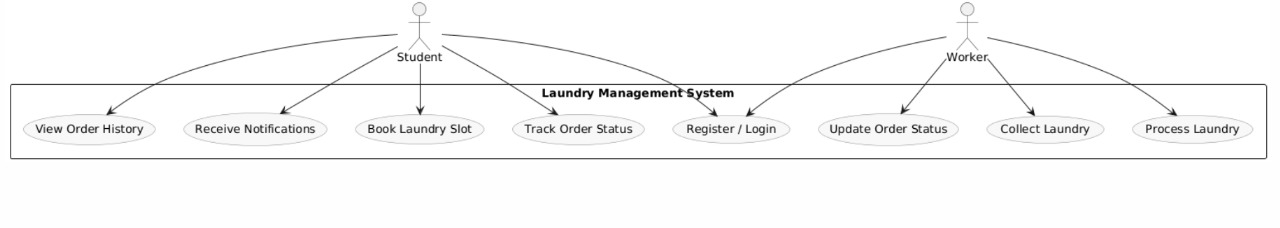


Figure-4: Use case Diagram

**5.2 USE CASE SCENARIOS**

A Use Case Scenario provides a detailed narrative of how an actor interacts with the Clean Wash Flow System for a specific feature. It typically includes:

* **Use Case Name:** Title of the action (e.g., “Submit Laundry Order”).
* **Primary Actor:** The user performing the action.
* **Preconditions:** What must be true before starting.
* **Postconditions:** What the system guarantees afterward.
* **Main Flow:** Step-by-step path under normal conditions.
* **Alternate Flows:** Variations or error paths that might occur.

**✦ Example – "Place an Order":**

* **Use Case Name: Submit Laundry Order**
* **Primary Actor:** Student
* **Preconditions:**

1. Student is authenticated (logged in).
2. Student profile includes at least one valid pickup location**.**

* **Main Flow:**

1. Student enters item count, selects pickup location, and adds any special instructions.
2. Student clicks Submit Order.
3. System validates input and creates a new order record with status = “Pending.”
4. System displays a confirmation screen showing the order ID .

* **Postcondition**:

1. The new order exists in the Orders DB and is visible on the Student Dashboard.
2. Student receives an in-app confirmation

.

**✦ Importance:**

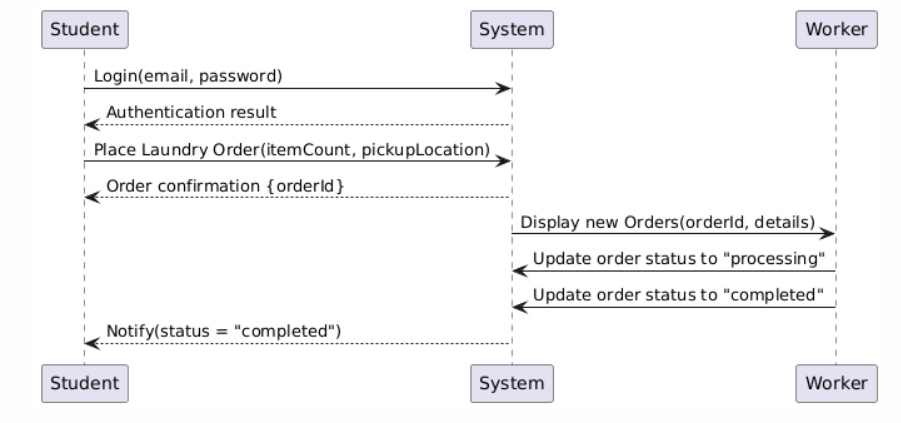
* Clarify functional requirements for developers.
* Provide a basis for writing targeted test cases.
* Ensure all edge cases and error conditions are considered.
* Ensures all possible interactions and exceptions are covered. 

Figure-5: Use scenario Diagram

**6. STRUCTURAL VIEW DIAGRAMS**

Structural view diagrams in software engineering depict a system’s static architecture, illustrating how classes, objects, and packages relate and connect. These diagrams play a crucial role in object-oriented analysis and design by revealing the system’s organizational structure and component relationships prior to implementation.

**6.1 CLASS DIAGRAM**

A **Class Diagram** is a type of UML (Unified Modeling Language) diagram that shows the **classes in a system**, along with their **attributes**, **methods**, and **relationships** (like inheritance, association, aggregation, etc.).

**✦ Key Components:**

**• Class:** Blueprint for objects (e.g. Profile, Order). **• Attributes:** Data fields of a class (e.g. pickup\_date, total\_price). **• Methods:** Operations or behaviors (e.g. placeOrder(), updateStatus()). **• Relationships:  
– Association (“uses-a”):** Profile → Order (a student places orders)  
**– Composition (“has-a”):** Order → OrderItem (an order is composed of line items) **– Generalization (“is-a”):** Student and Worker as specializations of Profile

**✦ Purpose:**

* Helps model the object-oriented structure of the system.
* Provides a visual reference for developers to understand data and behavior encapsulation.
* Useful for identifying key classes and their responsibilities.

**✦ Example:**

* Student, Worker, Order, ClothingItem could be primary classes.
* Relationships like:
  + A Student **Places** Order
  + Order **contains** ClothingItem

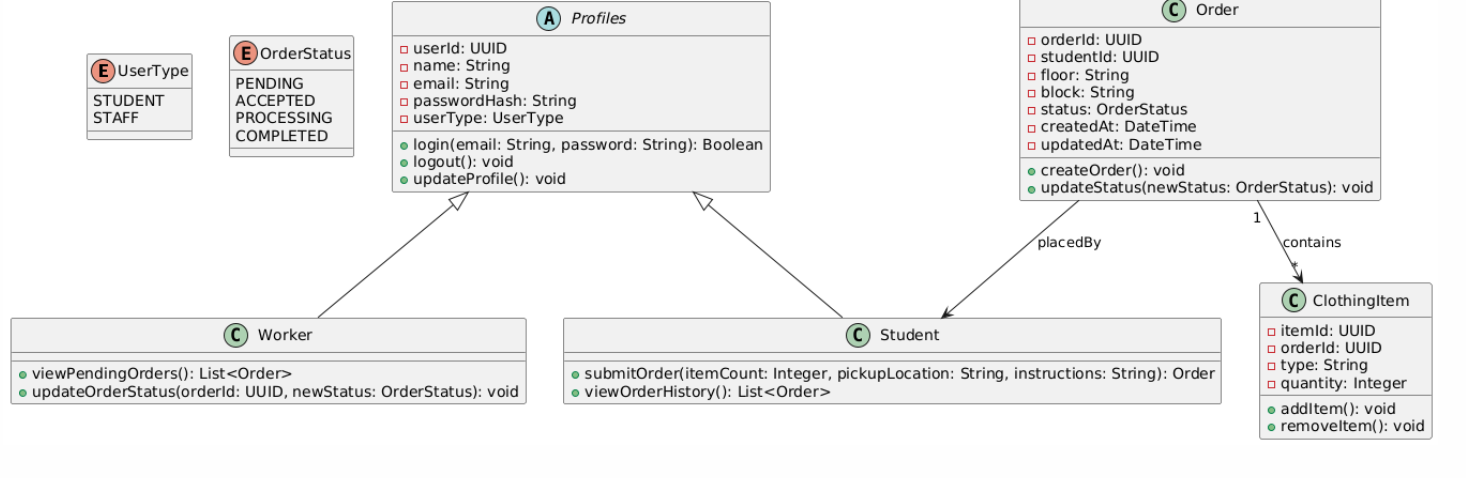


Figure-6: Class Diagram

**6.2 OBJECT DIAGRAM**

An Object Diagram captures the state of a system at a specific moment by depicting concrete instances of classes and how they’re connected.

✦ **Key Components:**

* **Objects:** Named instances showing current attribute values.
* **Links:** Associations between these instances, reflecting relationships in the class model.

✦ **Purpose:**

* Provides a real-world snapshot of how classes interact at runtime.
* Aids in debugging by making object structures and values explicit.
* Clarifies the current state of relationships and data in a given scenario.

**✦ Example:**

* Object instances like student1: Student, Order123:Order.
* Relationships showing how student1 has placed an Order123.

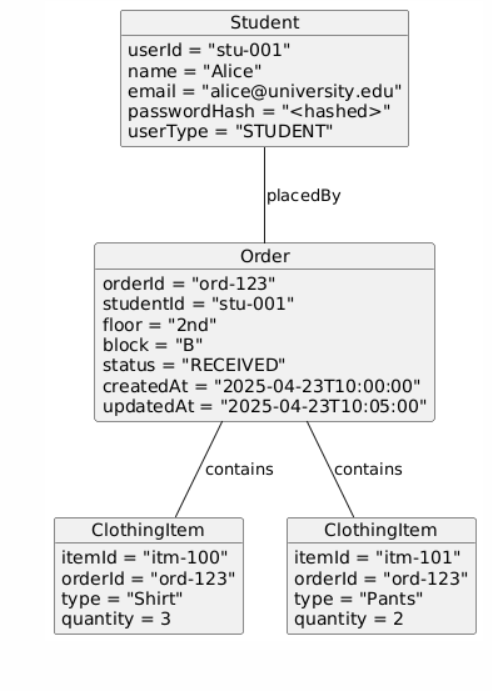


Figure-7: Object Diagram

**6.3 PACKAGE DIAGRAM**

**Package Diagram:** A UML structural diagram that groups related classes, interfaces, and subsystems into named packages. It visually organizes large systems into manageable modules.

**• Why Use Package Diagrams?**

* Break down complex architectures into coherent, logical chunks.
* Reveal dependencies between modules so you know which packages rely on others.
* Improve maintainability by showing clear separation of concerns.

**• Purpose:**

* Organize and simplify intricate class diagrams by collecting related elements into folders.
* Serve as a roadmap for both design and implementation phases.
* Facilitate team collaboration by clarifying module boundaries and interfaces.

**• Key Notation:**

* Package: Shown as a folder‐shaped rectangle with a small tab—its name appears on the tab or inside.
* Dependency: A dashed arrow from one package to another signifies that changes in the target package may affect the source.

By laying out high-level groupings and their interdependencies, package diagrams provide a top-down view that makes large-scale systems easier to understand and evolve**.**

**Main Components:**

* **Package 📦** → Represents a group of related classes, interfaces, or subsystems.
* **Class/Interface 📄** → The actual elements inside a package (e.g., Student, Application).
* **Relationships:**
  + **Dependency (..>)** → One package relies on another (e.g., UserManagement depends on SystemServices).
  + **Containment (+--)** → A package contains specific elements.
  + **Association (--)** → Shows a general connection between packages.

**Example Use Cases:**

* Software architecture design
* Layered system modeling
* Large-scale enterprise applications

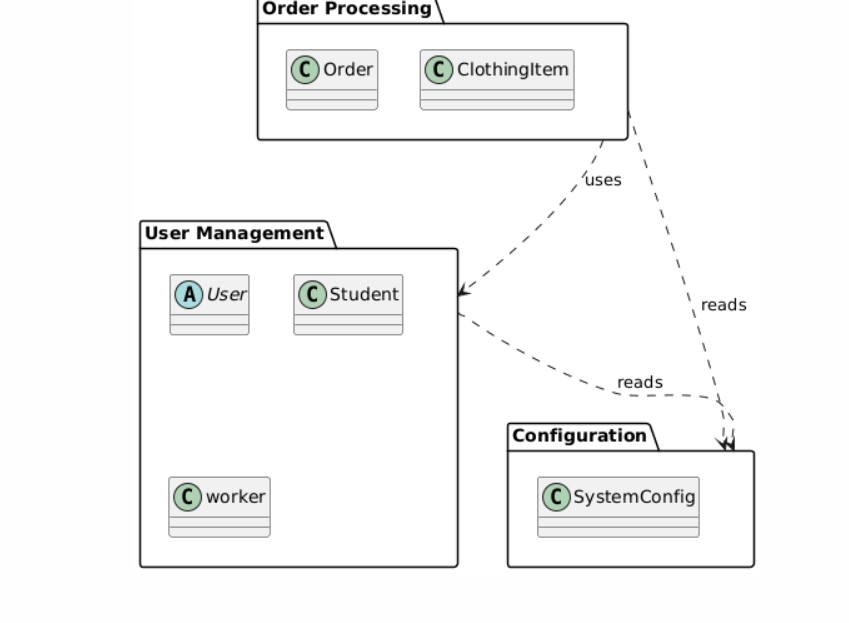


Figure-8: Package Diagram

**7. BEHAVIORAL VIEW DIAGRAM**

Behavioral view diagrams in UML capture a system’s runtime dynamics, showing how components interact and how control flows through various operations. They’re crucial for modeling the system’s reaction to events and the sequence of activities.

Behavioral diagrams allow you to:

* Illustrate the system’s responses to external stimuli and internal triggers.
* Show the message exchanges and collaborations between objects or components.
* Map out control logic, decision points, and lifelines over time.

In this section, we’ll include:

* **Sequence Diagrams** – Depicting the chronological order of interactions among objects.
* **Activity Diagrams** – Charting workflows, branches, and concurrent activities within a process.

**7.1 SEQUENCE DIAGRAM**

**A Sequence Diagram** is an interaction diagram that depicts how objects and actors exchange messages over time to accomplish a specific scenario in your system.

**Purpose:**

* Model the flow of control and message exchanges between components (e.g., UI, backend, workers).
* Capture how external actors (students, staff) interact with the system and with each other.

**Key Components:**

* **Actors** – External users of the system (e.g., Student).
* **Objects** – Instances of the classes (e.g., Order, Worker).
* **Lifelines** – Vertical dashed lines showing an object’s existence during the scenario.
* **Messages** – Horizontal arrows representing method calls or data exchanges.
* **Activation Bars** – Thin rectangles on lifelines indicating when an object is processing.

**Use Case Example (Laundry Order):**

1. **Student** → **System**: Login(email, password)
2. **System** → **Student**: Authentication result
3. **Student** → **System**: PlaceOrder(itemCount, pickupLocation, instructions)
4. **System** → **Orders DB**: INSERT new order (status = “Received”)
5. **System** → **Operations Staff**: NotifyNewOrder(orderId, details)
6. **Worker** → **System**: UpdateOrderStatus(orderId, “Processing”)
7. **Worker** → **System**: UpdateOrderStatus(orderId, “Completed”)
8. **System** → **Student**: Notify(status = “Completed”)

This diagram makes clear the step-by-step interaction among the student, the core system components, and the operations staff during a typical laundry request lifecycle.

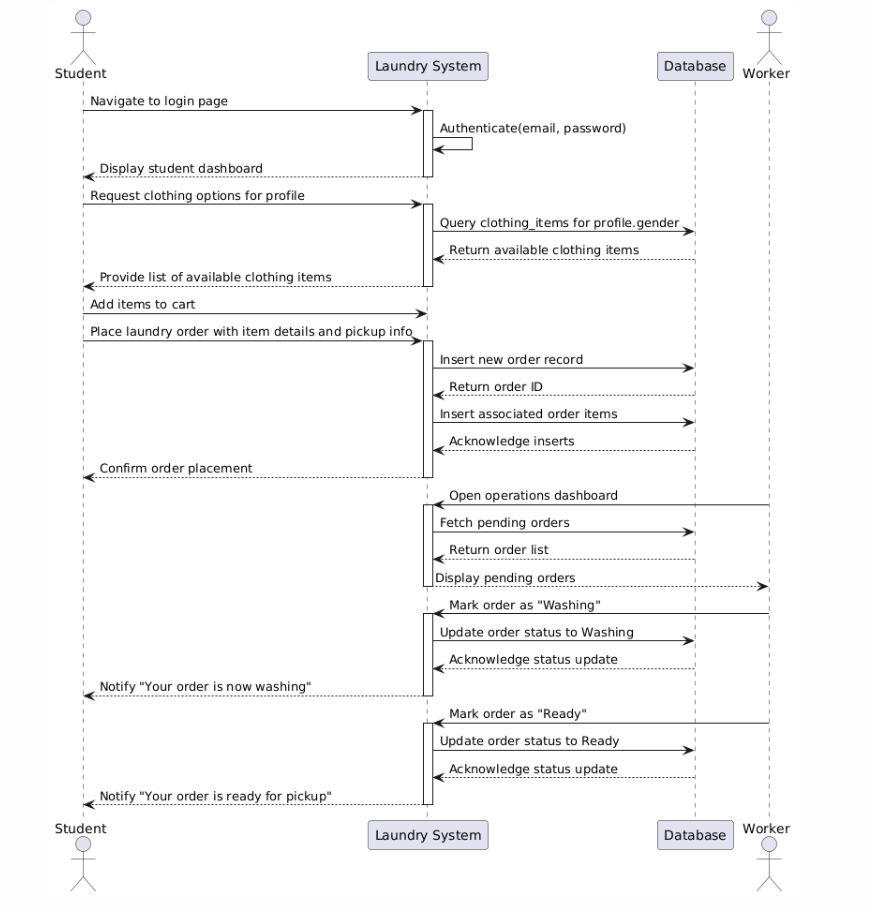


Figure-9: Sequence Diagram

**7.1 COLLABORATION DIAGRAM**

A Collaboration Diagram (or Communication Diagram) is a UML behavioral view that emphasizes the structural relationships between objects while showing the messages they exchange to carry out a particular scenario. Rather than focusing on time ordering (like a sequence diagram), it highlights which objects collaborate and how they’re linked.

**In the context of the Clean Wash Flow System, a collaboration diagram helps to:**

* **Show how Students, the Laundry System core, Workers, and the Database interact** to fulfill an order—from login to pickup notification.
* **Illustrate message exchanges**: e.g. Student → System (“placeOrder”), System → DB (“createOrderRecord”), System → Worker (“newOrderAssigned”), Worker → System (“updateOrderStatus”), System → Student (“notifyStatusChange”).
* **Reveal object relationships**: which service components (AuthService, OrderService, ItemService, NotificationService) are connected and what messages flow along those links.

**Key elements for your Laundry System collaboration diagram:**

* **Actors & Objects:**
  + student:Student
  + system: LaundrySystem (your React+Supabase backend)
  + db:Database (PostgreSQL via Supabase)
  + worker:Worker
* **Services (as collaborating objects):**
  + **AuthService** – handles “login”/“logout” calls
  + **OrderService** – manages “placeOrder,” “fetchOrders,” “updateStatus”
  + **ItemService** – retrieves and associates ClothingItem objects with orders
  + **NotificationService** – pushes in-app toasts on status changes
* **Typical message flow for “Submit Laundry Order”:**

1. student → AuthService: login(email, password)
2. student → OrderService: placeOrder(orderDetails)
3. OrderService → db: INSERT orders → returns orderId
4. ItemService → db: INSERT order\_items
5. OrderService → NotificationService: newOrderCreated(orderId)
6. NotificationService → student: notify(“order placed.”)

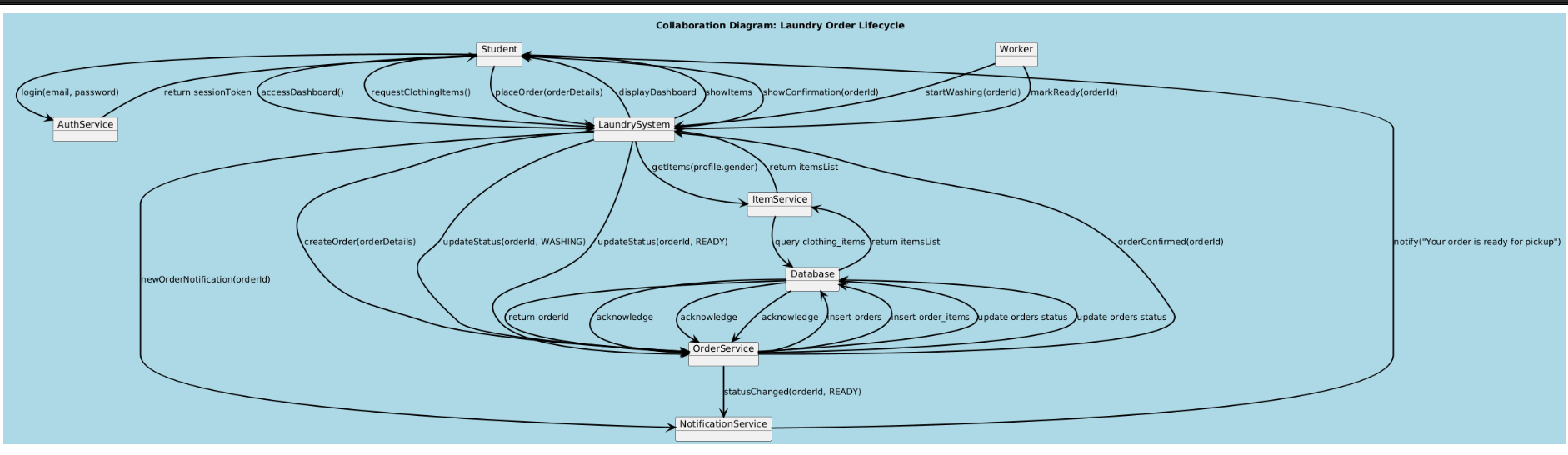


Figure-10: Collaboration Diagram

**8. IMPLEMENTATION**

The implementation phase focuses on turning the CleanWash design into a working system. For CleanWash, this involves building a responsive front end with React (Vite, TypeScript, Tailwind CSS) and a managed back end on Supabase (PostgreSQL, Auth, Realtime), integrating user authentication via Supabase Auth, in-app notifications via Realtime subscriptions, and deploying as a Progressive Web App.  
This section details the technology stack and explains how each component—React for UI, Supabase for data and auth, and Tailwind for styling—ensures a secure, interactive, and scalable laundry management solution.

**8.1 TECHNOLOGIES USED (React + Vite, Supabase Auth & Realtime, Tailwind CSS)**

🔹 **Front-end: React + Vite + TypeScript:**

* **React 18** for building a component-driven, single-page interface (Student and Staff dashboards).
* **Vite** for super-fast hot-module replacement and ES module development.
* **TypeScript** to catch errors early and improve maintainability.

🔹 **Styling: Tailwind CSS & Radix UI:**

* **Tailwind CSS** for utility-first, responsive styling without writing custom CSS classes.
* **Radix UI primitives** (via @radix-ui/react-\* and shadcn/ui) to assemble accessible, headless components (modals, dropdowns, toasts).

🔹 **Backend-as-a-Service: Supabase**

* **PostgreSQL:**
  + Hosted relational database for all data (orders, users, items, status logs).
* **Auth:**
  + JWT-based sign-up/sign-in out-of-the-box, with email verification and row-level security policies.
* **Realtime:**
  + WebSocket subscriptions drive live status updates and in-app toasts without a custom server.

🔹 **Data Integration: @supabase/supabase-js:**

* Single client library for all CRUD, auth, and realtime operations in src/integrations/supabase/client.ts.

🔹 **Notifications:**

* **In-app toasts** powered by Supabase Realtime channels and Radix/UI to instantly alert students when their laundry status changes.

🔹 **Deployment & Hosting:**

* **Vercel** (or Netlify) for front-end hosting with automatic CI/CD, global CDN, and zero-config SSL.
* **Supabase** handles scaling, backups, and security for the database and auth layer—no custom server required.

*By combining a modern React/Vite front-end with Supabase’s fully managed backend, the Clean Wash Flow System delivers a secure, real-time laundry management experience with minimal operational overhead.*

**8.2 STUDENT DASHBOARD FUNCTIONALITIES**The Student Dashboard is the single‐page hub where authenticated students manage everything related to their laundry requests browsing services, placing orders, tracking status, and receiving real-time updates.

**8.2.1 LOGIN & AUTHENTICATION**

**Overview:**Students sign in with their university email and password via Supabase Auth (JWT under the hood).

**Key Features:**

* Secure Sign-In/Sign-Out: Uses supabase.auth.signInWithPassword and signOut.
* Session Management: JWT tokens maintained by Supabase; protected routes guard the dashboard.
* Access Control: Only authenticated users can view the dashboard or place orders.

**Benefits:**

* Ensures only verified students can access laundry services.
* Leverages Supabase’s built-in security and row-level policies.

**8.2.2 ITEM BROWSING & CART MANAGEMENT**

**Overview:**Students browse available clothing-item types (e.g. shirts, pants) tailored to their profile, add them to a “laundry basket,” and review selections before placing an order.

**Key Features:**

* **Filtered Listing:** Automatically fetches clothing\_items for the student’s gender.
* **Loading States:** Shows a spinner while items load.
* **Add to Cart:** Clickable cards let students add/remove items; a live cart count is shown.
* **Client-Side Cart:** Maintains selected items locally until checkout.

**Benefits:**

* Smooth, responsive browsing without page reloads.
* Clear, visual feedback on cart contents and item availability**.**

**8.2.3 PLACING & TRACKING ORDERS**

**Overview:**Students finalize their laundry requests by specifying pickup date, floor, block, and notes, then submit an order that’s stored in two tables (orders and order\_items). They can view all past and current orders with real-time status

**Key Features:**

* **Order Form:** Validates non-empty cart; collects pickup details.
* **Atomic Inserts:**
  1. Creates an orders record (status = pending).
  2. Inserts each cart item into order\_items.
* **Order History Tab:** Lists every order with status badges (pending, processing, completed, cancelled).
* **Cancel Order:** Students can cancel pending orders, updating status to cancelled.
* **Status Color-Coding:** Badges adapt color based on status.

**Benefits:**

* One-click order placement with clear success/error feedback via toasts.
* Full order history and ability to cancel before processing begins.

**8.2.4 REAL-TIME NOTIFICATIONS**

**Overview:**  
Students receive instant in-app toasts when an order’s status changes to completed.

**Key Features:**

* **Realtime Subscription:** Uses supabase.channel('public:orders') to listen for any CRUD on that student’s orders.
* **Toast Alerts:** Pops up notifications only when payload.new.status === 'completed'.
* **Notification Popover:** A bell icon opens a dropdown of recent notifications (with timestamp and message).

**Benefits:**

* Eliminates manual refreshes—students immediately know when laundry is ready.
* Keeps a short-lived history of alerts for reference

**8.3 WORKER DASHBOARD FUNCTIONALITIES**

The Worker Dashboard is the operations control panel for laundry staff. It provides a real-time view of incoming orders, tools to advance each order through its workflow, and quick access to order history for audit and inquiry.

**8.3.1 VIEWING & FILTERING ORDERS**

* **Overview:**Staff can switch between active orders (pending → processing → ready) and the full order history.
* **Key Features:**
* Tabs: “Active Orders” vs. “All Orders” for easy toggling.
* Realtime Updates: Subscribes to Supabase’s public:orders channel so new orders or status changes appear instantly—no manual refresh needed.
* Order Cards: Each card shows student name, pickup info (floor, block, date), item summary, and current status badge.
* Benefits:
* Ensures staff always see the latest queue.
* Helps prioritize and balance workloads.
  + 1. **UPDATING ORDER STATUS**

**Overview:**  
 Workers advance orders through each stage of the laundry lifecycle with a simple inline form.

**Key Features:**

* **Status Dropdown:** Choose from pending, processing, completed, or cancelled.
* **Optional Notes:** Add comments (e.g., “Delicate wash performed”) saved with the update.
* **Submit & Cancel Buttons:** Apply or abort status changes without leaving the list.
* **Toast Feedback:** Instant success or error messages via Radix toasts.

**Benefits:**

Minimizes clicks to move orders forward.

Captures context and exceptions for transparency.

**8.3.3 ORDER HISTORY & SEARCH**

**Overview:**Staff can review past orders—completed or cancelled—to handle questions or run simple audits.

**Key Features:**

* Color-Coded Badges: Quickly scan orders by status.
* Search & Filters: By student name, date range, or status.
* Cancel Functionality: Ability to cancel a pending order if needed.

**Benefits:**

* Simplifies follow-up on special requests or disputes.
* Provides a clear audit trail of all order actions.
* By combining real-time order feeds, inline status updates, and rich filtering, the Worker Dashboard gives laundry staff a fast, efficient, and transparent way to manage every step of the laundry process.

**8.4 NOTIFICATION SYSTEM**The Clean Wash Flow System features a real-time, in-app notification mechanism powered by Supabase Realtime channels and Radix-UI toasts, with optional email alerts via Supabase’s built-in SMTP integration.

**8.4.1 IN-APP NOTIFICATIONS**

**Overview:**  
Students see live toast messages in their dashboard whenever their laundry order moves through key stages.

**Triggers:**

* Order Created: When a new order is inserted → “Order Placed.”
* Status = completed: When staff marks it ready → “Your order is ready for pickup.”

**Implementation Details:**

* In StudentDashboard.tsx, the app calls

ts

CopyEdit

supabase

.channel('public:orders')

.on(

{ event: '\*', schema: 'public', table: 'orders', filter: `student\_id=eq.${user.id}` },

payload => { /\* show toast for payload.new.status \*/ }

)

.subscribe();

* Toasts use the use-toast hook from Radix-UI for consistent styling and placement.
* A bell-icon popover logs recent notifications (title, message, timestamp).

**Benefits:**

* Instant Feedback: No page reloads—students know status changes immediately.
* User Engagement: Visual cues keep students informed and reduce “Where’s my laundry?” queries.
* Audit Trail: In-app history of recent alerts for quick reference.

**9. RESULT AND DISCUSSION**

This section highlights the final outcomes of the Clean Wash Laundry Management System, showcasing how various modules perform in real-time conditions. It provides insights into the user interface, functionalities, and system robustness through screenshots, testing, and evaluation of expected vs actual outcomes.

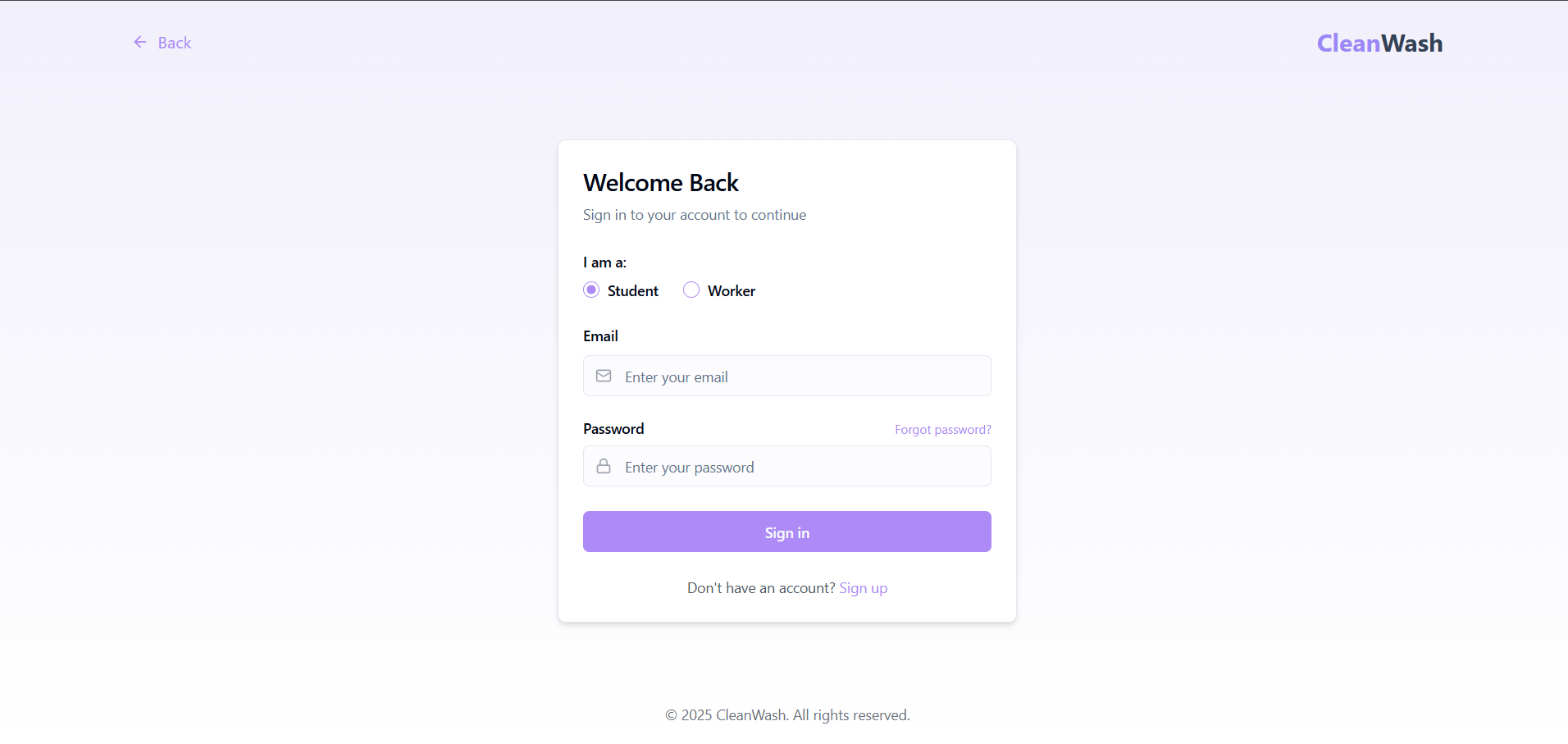
**9.1 SCREENSHOTS OF APPLICATION**

Screenshots provide a visual demonstration of the core functionalities and user interfaces of the application. They serve as evidence of the working product and help in validating the user experience. Below are key areas captured in the screenshots:

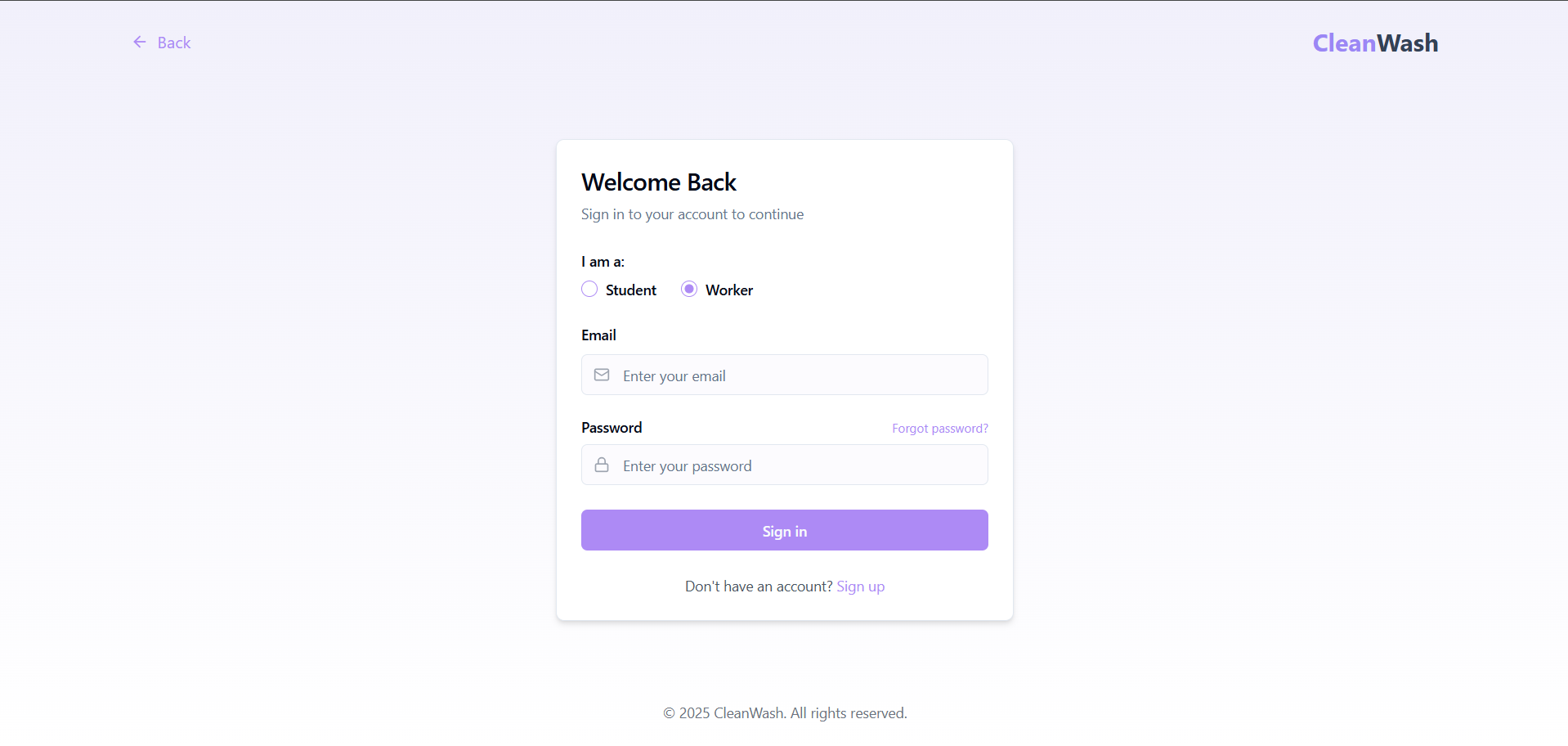
* **Login Pages**
  + Student and Worker login interfaces showing authentication functionality.

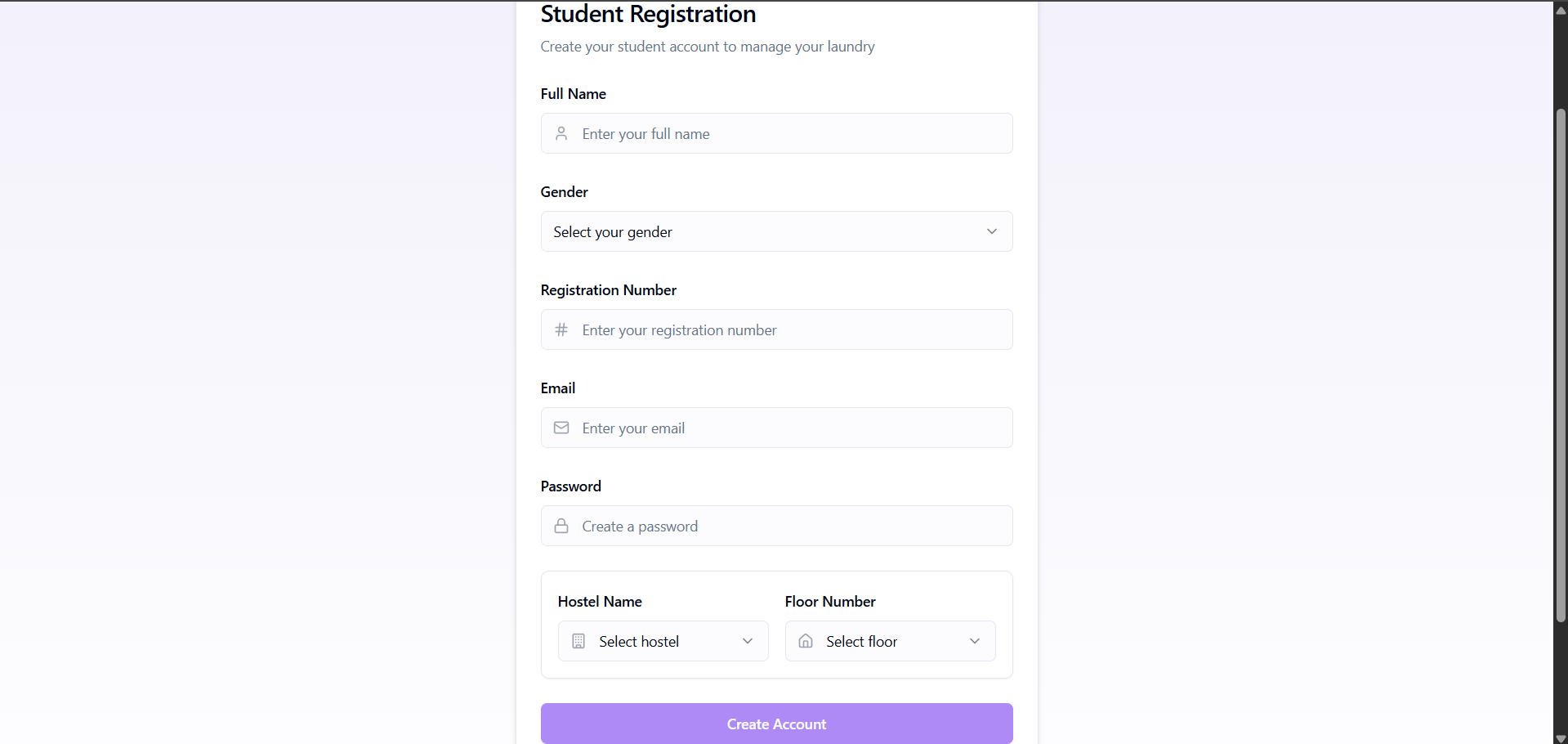


Student Login Page:

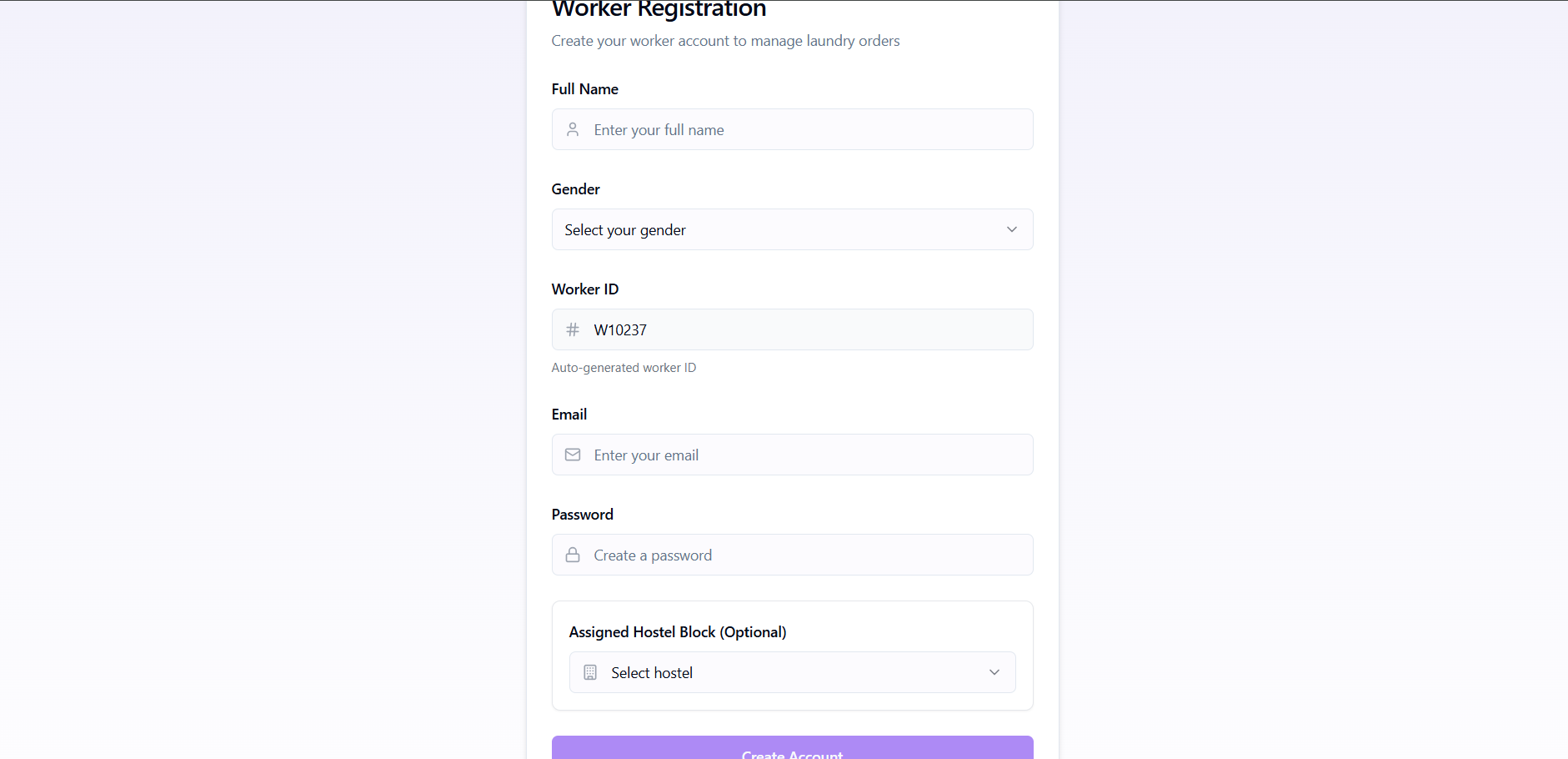


Worker Login Page:

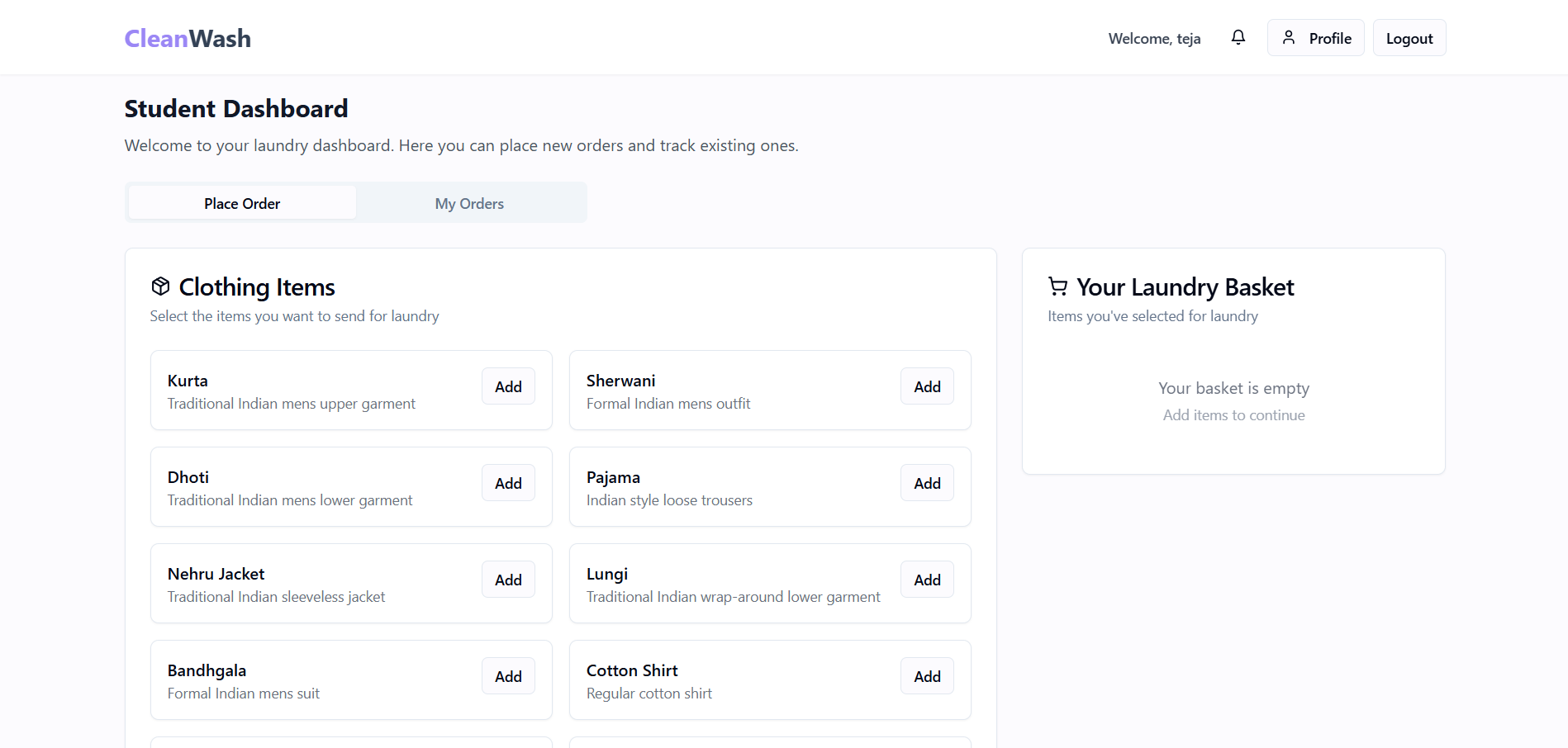
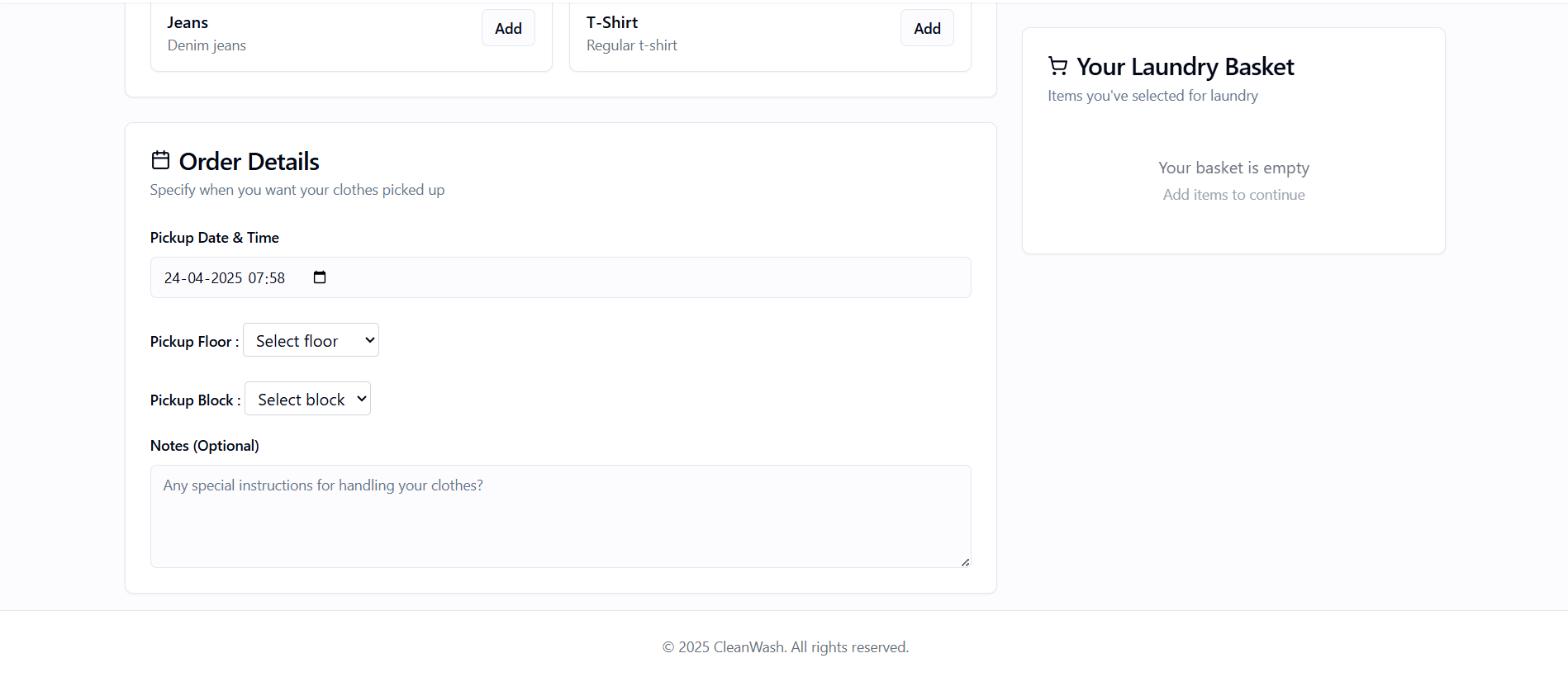
 Student Registration Page:



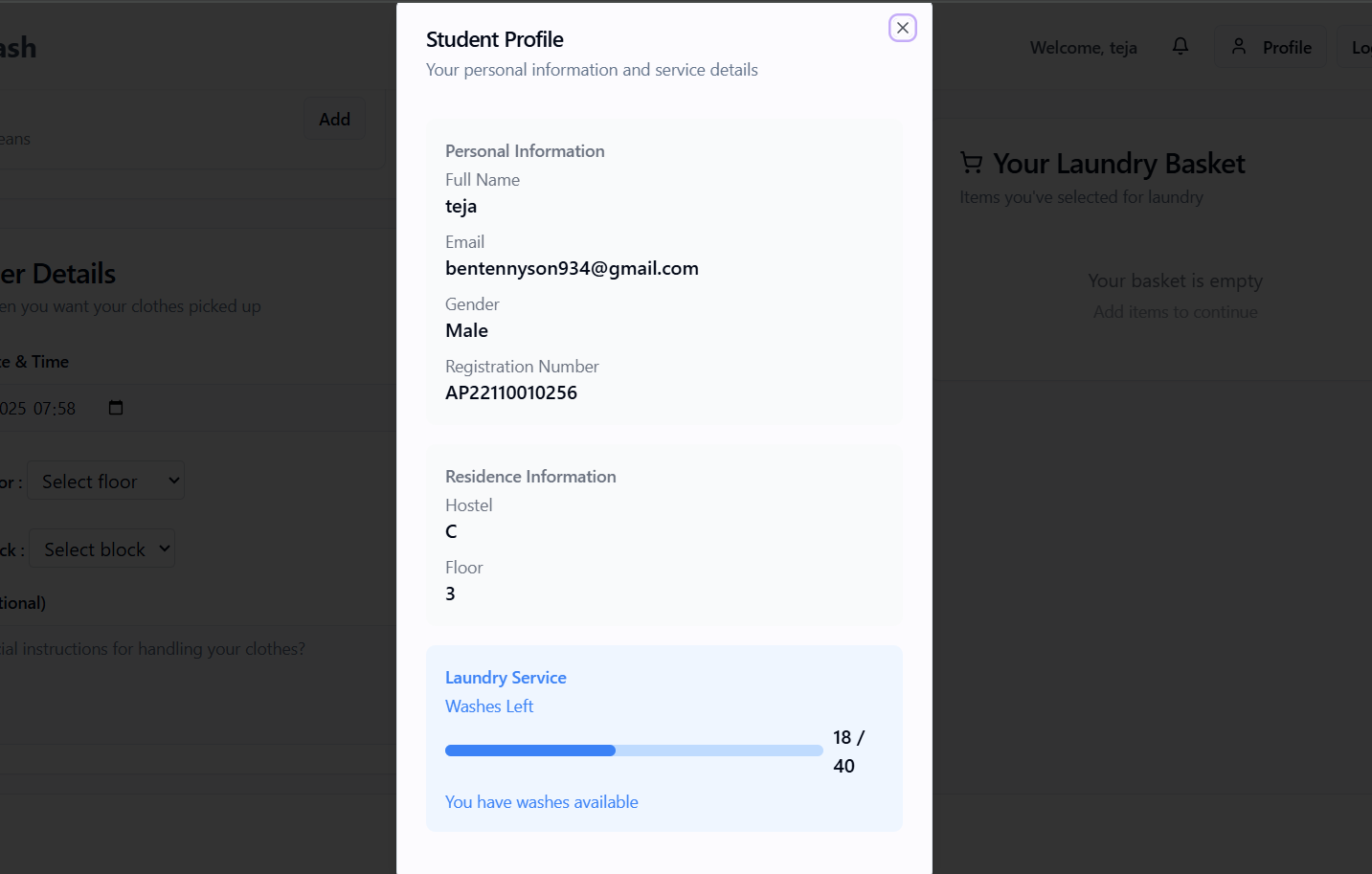
Worker Registration Page:



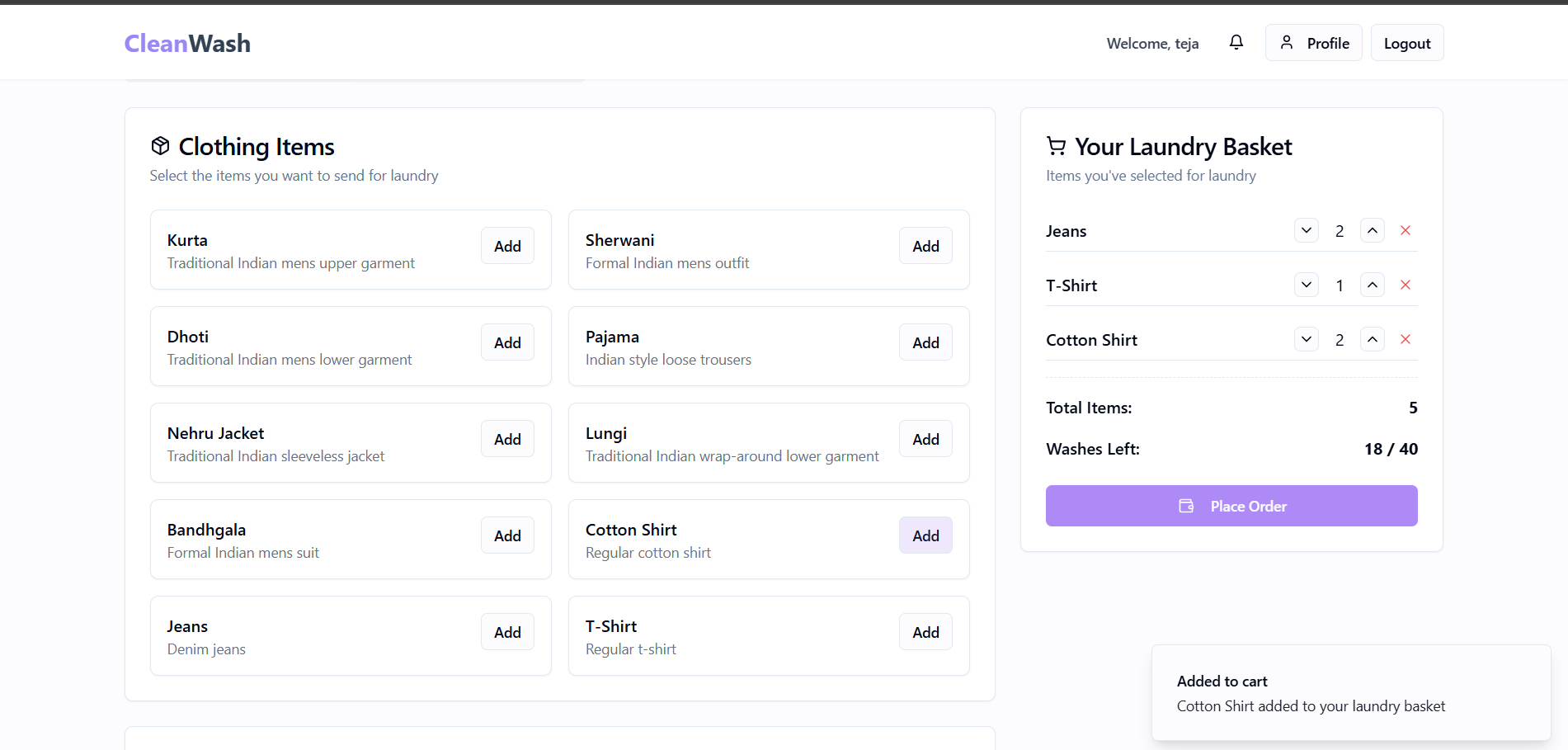
* **Student Dashboard:**

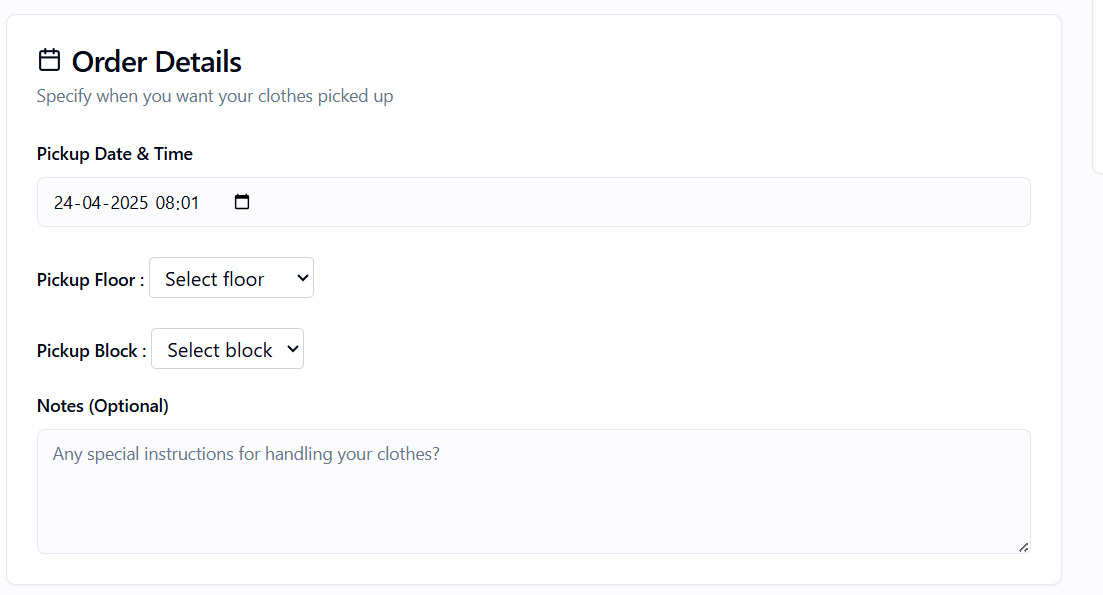
Profile overview:



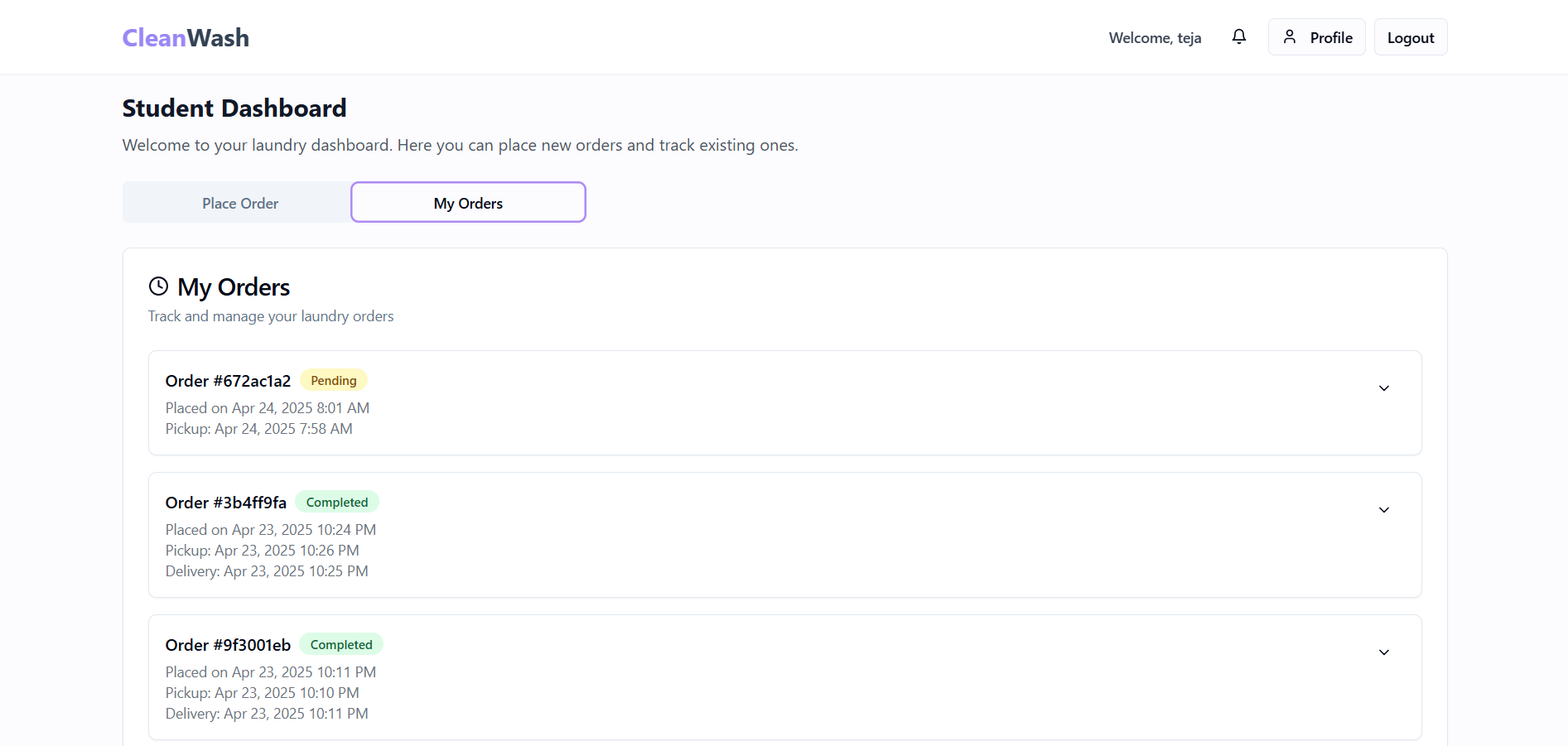
Adding items to Basket:



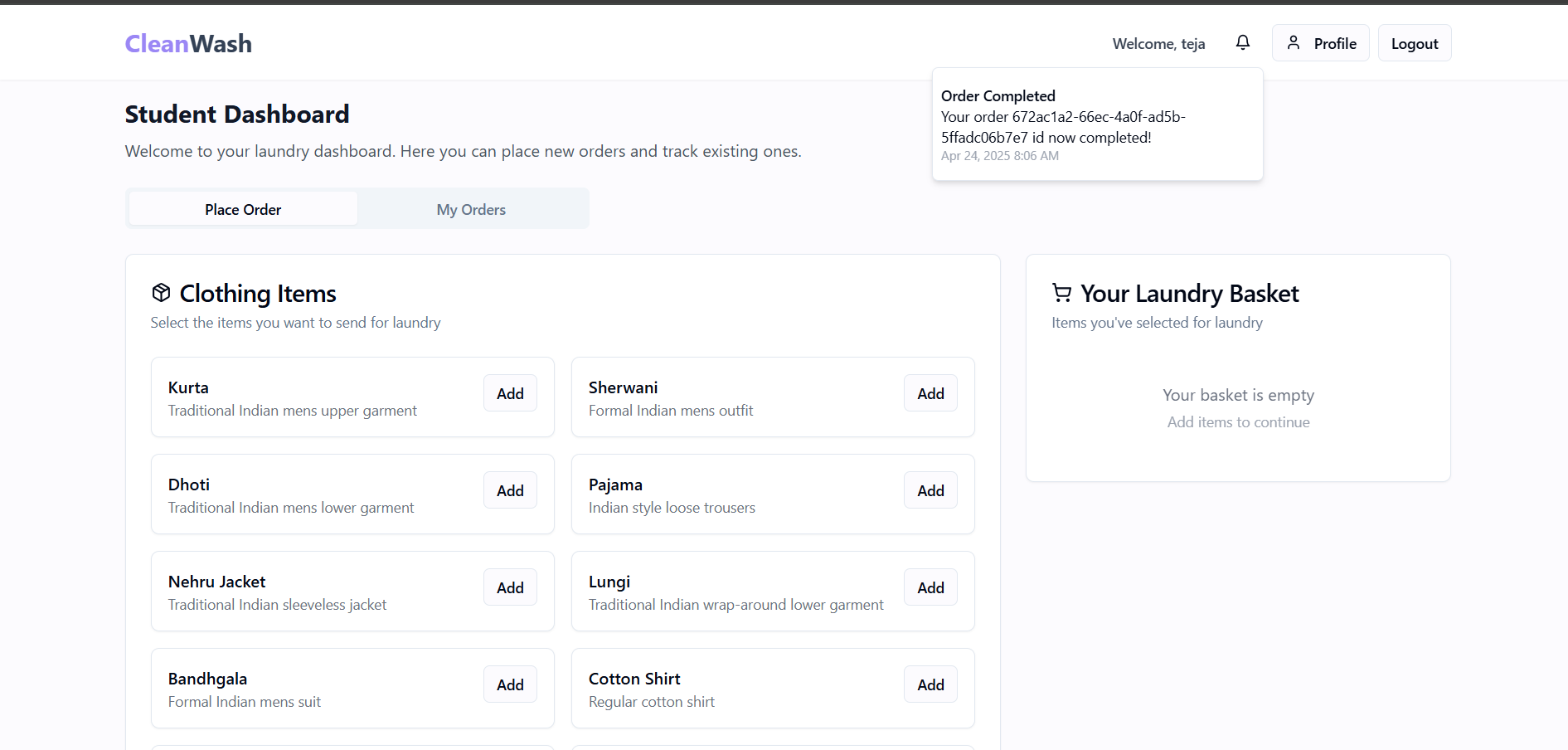
Select time and place to order:

****

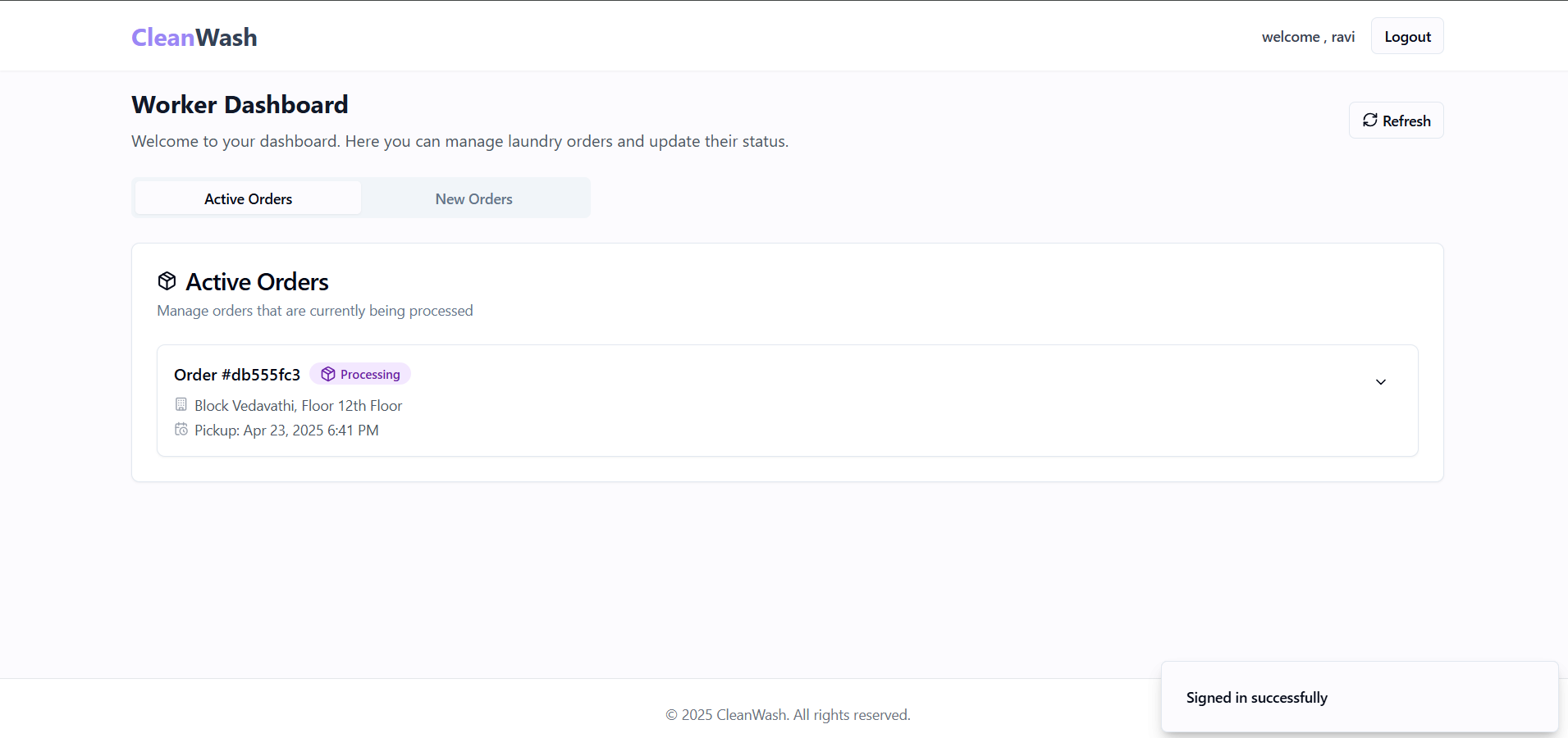
After placing Order:



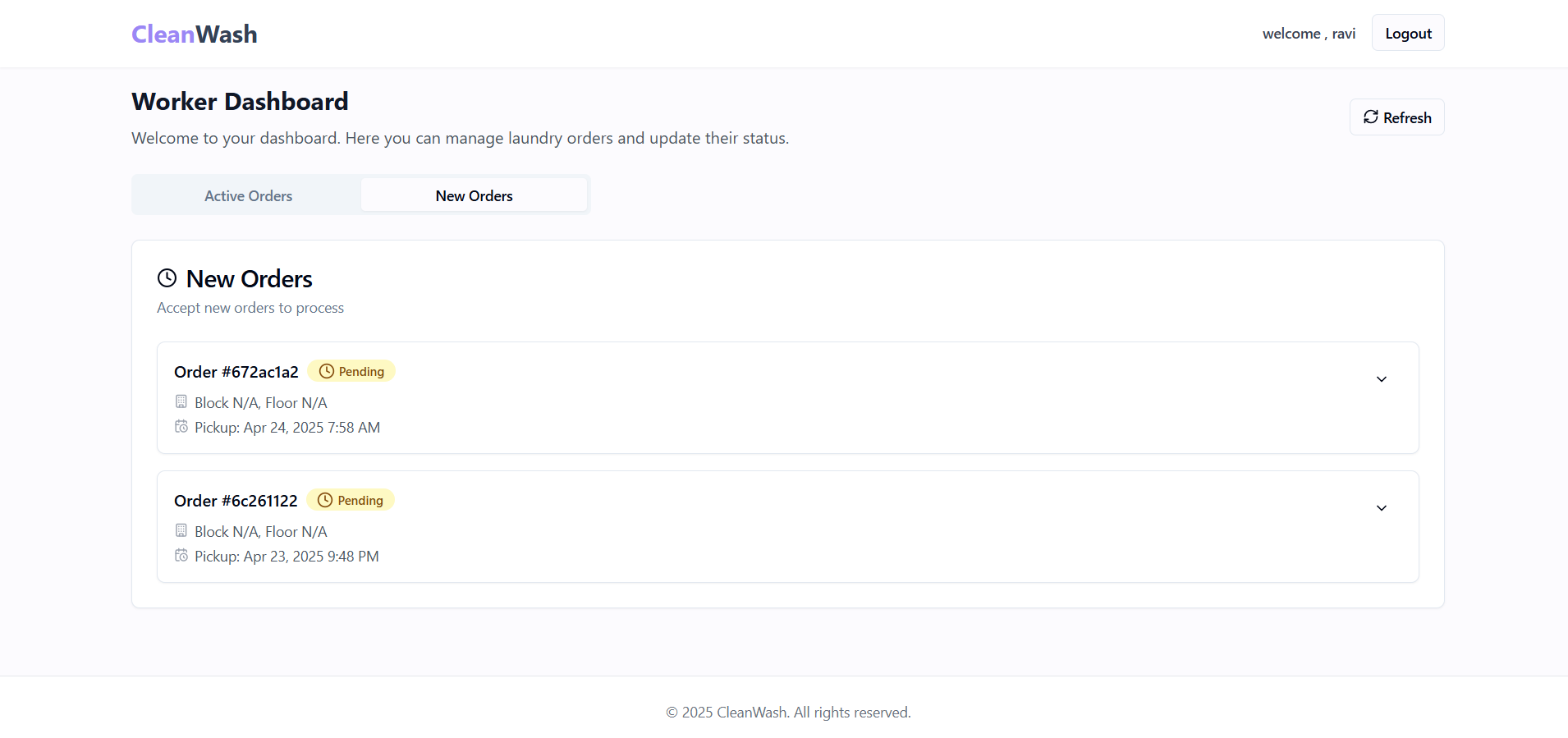
Notifications:



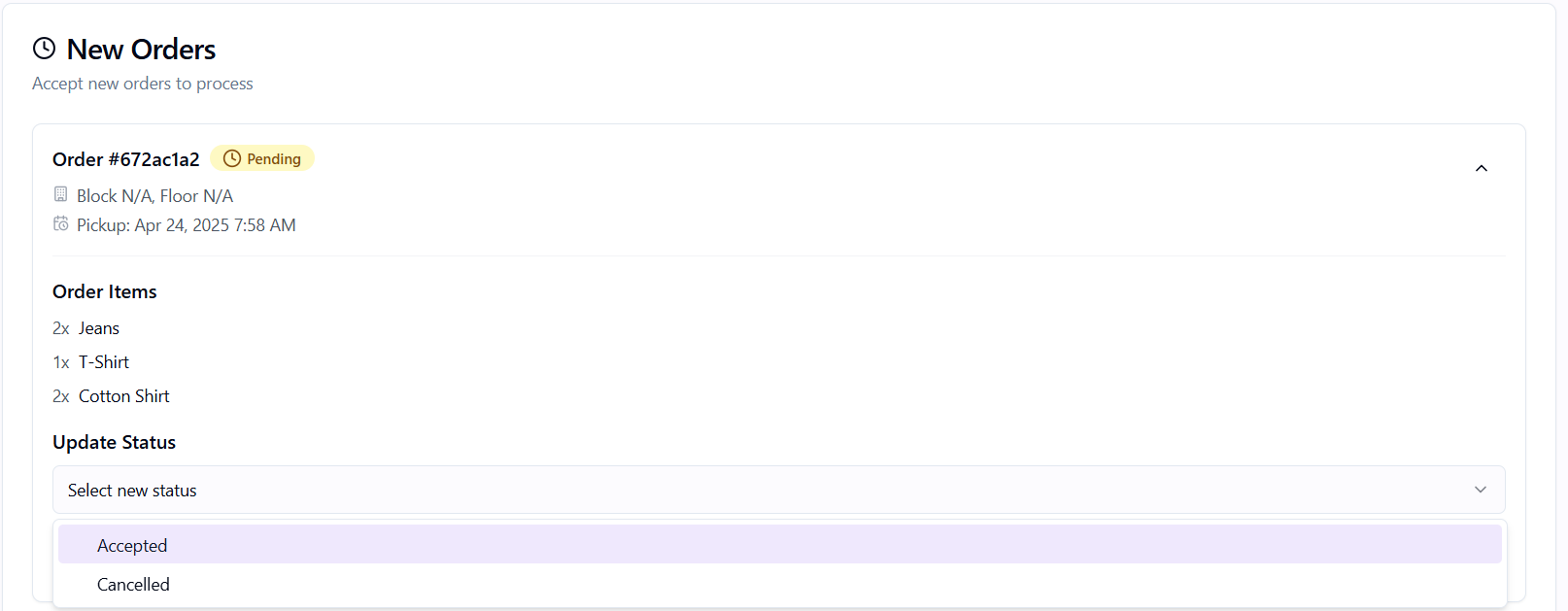
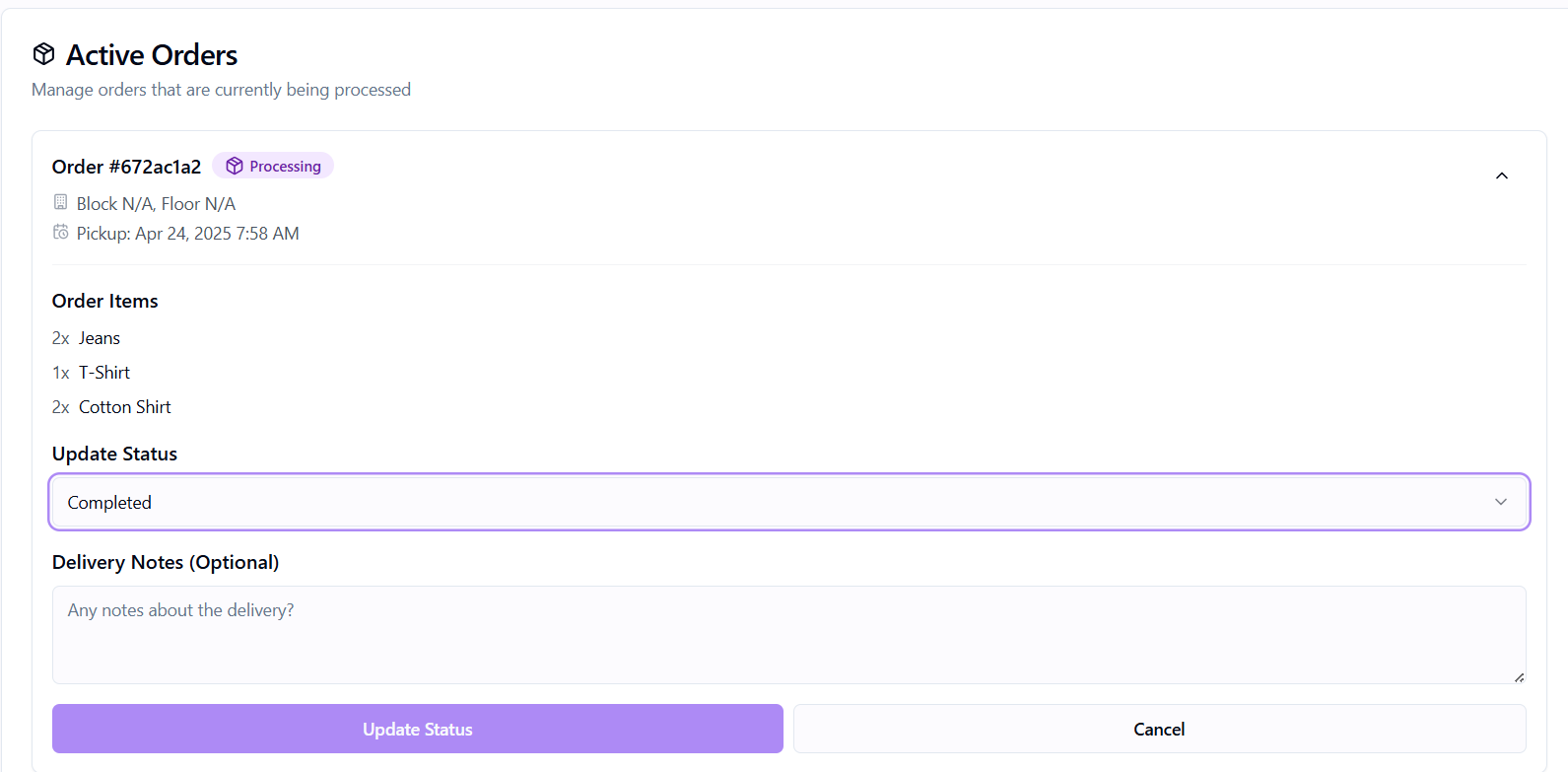
Worker Dashboard:



New Orders:



Updating Status like (Accepted -> Processing -> Completed):

**10. CONCLUSION & FUTURE WORK**

**10.1 CONCLUSION**  
The Clean Wash Flow System transforms campus laundry services into a fully digital, end-to-end workflow. By leveraging React 18 + TypeScript with Vite, styled via Tailwind CSS and Radix UI, and backed by Supabase (PostgreSQL + Auth + Realtime), it delivers:

* **Secure, role-based access** for students and operations staff via JWT-poweredSupabase Auth.
* **Intuitive Student Dashboard** for browsing clothing items, adding to cart, and placing orders with pickup details.
* **Real-time order tracking**: live status badges (Pending → Washing → Ready) and in-app toast notifications driven by Supabase Realtime subscriptions.
* **Worker Dashboard** for staff to view pending jobs, update statuses inline, and manage order history with color-coded badges.
* **Centralized data store** in a serverless, managed PostgreSQL instance—eliminating paper logs, whiteboards, and lost slips.
* **Seamless notifications** so students know instantly when their laundry is being washed or ready for pickup.

By automating manual workflows, improving transparency, and cutting turnaround times, Clean Wash Flow System meets its objectives of a robust, user-friendly, and easily scalable laundry management platform.

**10.2 FUTURE ENHANCEMENTS**

1. **Admin Analytics Dashboard**  
   – Super-admin view to monitor order volumes, peak hours, and staff performance.
2. **QR-Code Bag Tagging**  
   – Print QR codes for laundry bags to enable one-scan check-in/check-out and reduce misplacement.
3. **Push Notifications**  
   – Browser/mobile push alerts (via FCM or OneSignal) for status updates even when the web app is closed.
4. **Advanced Reporting & Insights**  
   – Data visualizations for average turnaround times, service bottlenecks, and usage trends.
5. **Mobile App**  
   – Cross-platform React Native client for placing and tracking orders on the go.
6. **Offline Support**  
   – Local caching of orders and status logs so students can submit requests in low-connectivity zones.
7. **Payment Integration**  
   – Campus-card or digital wallet payments to enable pre-paid laundry credits and automated billing.
8. **AI-Driven Scheduling**  
   – Predict busy periods and auto-assign jobs to available staff to balance workloads.
9. **Dynamic Service Options**  
   – Offer express or specialty services (e.g., dry cleaning) with tiered pricing in the UI.
10. **Feedback & Rating Module**  
    – Allow students to rate service quality, helping staff improve and maintain accountability.

These enhancements will further streamline operations, elevate user satisfaction, and prepare the system for campus-wide scaling.

**11. REFERENCES**The development of the Clean Wash Flow System was informed by established software-engineering practices, modern web-development frameworks, and UML modeling tools. Key references include:

**1. Books & Research Papers**

* Sommerville, I. (2015). *Software Engineering* (10th ed.). Pearson Education.
* Pressman, R. S., & Maxim, B. R. (2014). *Software Engineering: A Practitioner’s Approach*. McGraw-Hill Education.
* Gamma, E., Helm, R., Johnson, R., & Vlissides, J. (1994). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.

**2. Web Resources & Documentation**

* React 18 Documentation – Component patterns, hooks, and concurrent features.
* Vite Official Guide – Fast development server and build configuration.
* TypeScript Handbook – Static typing best practices.
* Tailwind CSS Docs – Utility-first styling conventions.
* Radix UI Repository – Accessible headless UI primitives (accordion, dialog, toast).
* Supabase Documentation – Auth, Realtime, and PostgreSQL reference.
* @supabase/supabase-js – Client-side API usage.
* PostgreSQL Manual – SQL syntax and database design.
* MDN Web Docs – JavaScript, HTML, and CSS reference.

**3. UML & Design Tools**

* PlantUML – Text-based UML diagram generation (class, sequence, collaboration, package, object diagrams).
* Lucidchart – Collaborative drag-and-drop modeling.
* Draw.io (diagrams.net) – Free diagramming for architecture and DFDs.
* Visual Paradigm – UML modeling and ER-diagram support.