**Project Report**

On

**[Stock-Up]**

**Inventory Management Website**

prepared as a part of the Full-Stack Development Course under IBM

Submitted by:

**Zeeshan Kadiwala**

Under Mentor/Supervisor:

**Mr. Jaivik Panchal**

**Submitted on:**

(18-7-2025)

**CERTIFICATE**

This is to certify that the project titled **“Stockup — Inventory & Sales Management System”** is an original work completed by **Zeeshan Kadiwala** under the guidance of  **Mr.Jaivik Panchal** at **IBM Internship** during Course of Full-Stack Development.

**Date: 18 / 7 / 2025**

Project Guide

Mr. Jaivik Panchal

**ACKNOWLEDGEMENT**

We would like to express our special thanks of gratitude to our faculty Professor Jaivik Panchal who gave us the golden opportunity to do this wonderful project under Full-Stack Development Course under IBM on topic of “Inventory Management Website [Stock-Up]” which also helped me in doing a lot of research and came to know about so many new things. I extend my sincere thanks to **IBM** for offering me the opportunity to be part of the **Full-Stack Development Internship**, which provided the platform to apply and deepen my understanding of modern software development practices. This experience has been instrumental in shaping both my technical skills and professional outlook, and I am genuinely grateful to all who were a part of this journey.

**TABLE OF CONTENTS**

|  |  |
| --- | --- |
| **TITLE OF THE PROJECT** |  |
| **CERTIFICATE** |  |
| **ACKNOWLEDGEMENT** | **I** |
| **TABLE OF CONTENTS** | **II** |
| **ABSTRACT** | **III** |
| **LIST OF FIGURES**  **OBJECTIVE**  **SYSTEM ARCHITECTURE**  **TECHNOLOGY STACK**  **MODULES/FEATURES**  **FRONT-END DEVELOPMENT**  **BACK-END DEVELOPMENT**  **DATABASE DESIGN**  **SECURITY MEASURES**  **LIMITATION**  **FUTURE ENCHANCEMENT**  **SCREENSHOT**  **CODE SNIPPETS**  **REFERENCE** | **IV**  **6**  **7**  **10**  **12**  **15**  **16**  **17**  **20**  **20**  **21**  **22**  **25**  **26** |

**ABSTRACT**

Stockup is a full-stack web application that centralizes inventory management, purchase tracking, and sales recording within a single platform. It supports multiple user roles—Admin, Owner, Employee, Supplier—each with a tailored interface and permissions.

To replace error-prone, manual stock-keeping processes in small-to-medium retail environments with an automated system that ensures accurate inventory counts, enforces accountability through activity logs, and accelerates transaction workflows.

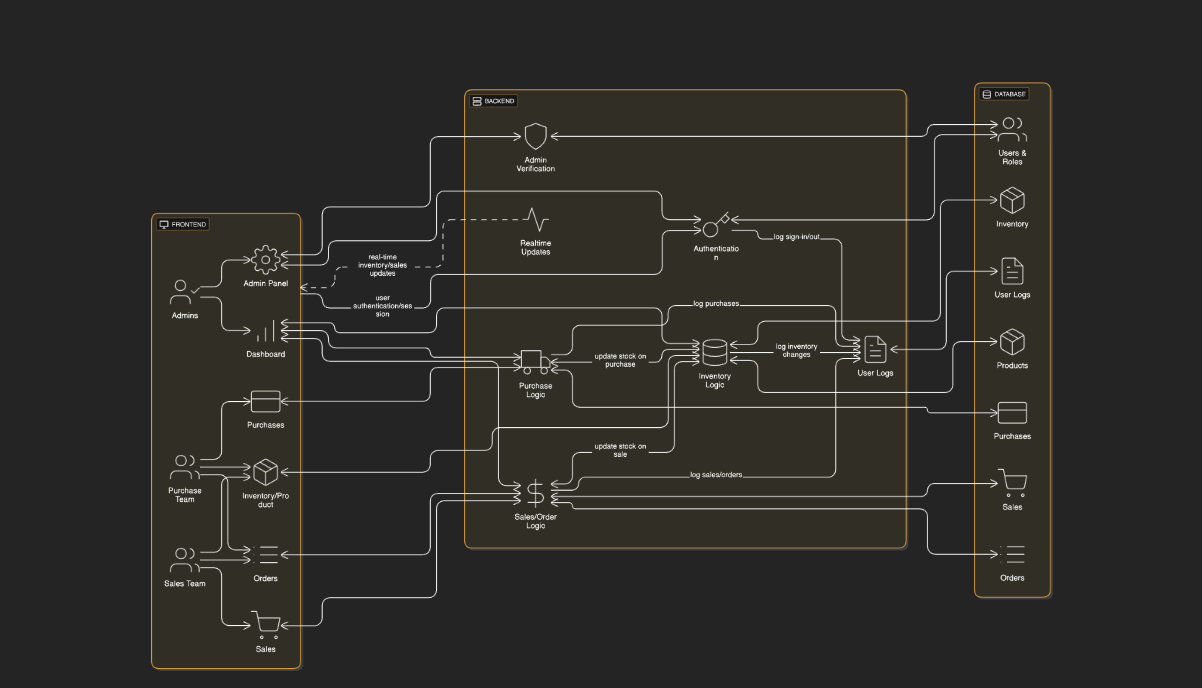
**Key features:**

* Secure user authentication with email-or admin-based account verification
* Role-based navigation and access control
* Purchase module to record stock acquisitions and update inventory
* Sales module with real-time stock validation, total calculations, and confirm-sale modal
* Activity logging of user login/logout times and “Last Online” display
* Admin panel for product CRUD, user management, and log inspection

**OBJECTIVE**

* **Automate and Streamline Inventory Workflows:** Replace manual spreadsheets and paper-based stock-keeping with a unified web application that handles product cataloging, stock additions (purchases), and stock decrements (sales) through intuitive forms and real-time validation.
* **Enforce Role-Based Access and Security:** Deliver a tailored experience for each user type—Admin, Owner, Employee, Supplier—so that only the required modules (e.g., purchasing, sales, order management, user verification) are visible and actionable. Ensure that unverified or unauthorized users cannot access protected endpoints.
* **Guarantee Data Accuracy and Consistency** Maintain up-to-the-second inventory counts by validating stock levels before each sale or purchase. Prevent overselling and automatically update product quantities in MongoDB to reflect real operations without manual reconciliation.
* **Provide Accountability Through Activity Logging** Capture every user’s login and logout timestamps, storing them in a dedicated log\_data collection. Surface “Last Online” information in the UI to enable audit trails and foster responsible system use.
* **Empower Administrators with Centralized Control** Offer an Admin Panel for full CRUD on products, oversight of user accounts (including account verification flows), and visibility into system logs. Equip administrators with the tools to manage data integrity, user permissions, and operational reporting.
* **Deliver Real-Time Business Insights** Generate and display daily and cumulative sales summaries immediately after each transaction. Enable business stakeholders to make data-driven decisions—such as identifying high-turnover items or monitoring revenue trends—directly from the dashboard.
* -**Prioritize User Experience and Reliability** :Craft a responsive, accessible front end using clean layouts, consistent iconography, and confirmation modals to reduce user errors. Implement client- and server-side validation to provide immediate feedback while safeguarding data integrity.

**System Architecture**

****

**Component Interaction & Technology Flow**

This section outlines how different components of the system interact with one another and the technologies used at each stage.

**Frontend (HTML, CSS, JavaScript)**

* Delivers static assets and user interface elements to the browser.
* Reads session-specific data from LocalStorage, including:
  + username
  + role
  + loginTime
* Sends HTTP requests with JSON payloads to the backend Flask API.
* Renders dynamic content and interfaces based on user role.

**Flask API (Python + PyMongo)**

* Provides RESTful API endpoints for:
  + Authentication
  + Product management (CRUD)
  + Purchases and sales
  + Log entries
  + User profile operations
* Processes incoming requests by:
  + Validating user input.
  + Enforcing role-based authorization.
  + Returning standardized JSON responses.
* Interfaces with the MongoDB database using PyMongo to perform operations such as:
  + find
  + insert
  + update

**MongoDB Database**

Stores persistent data across several logically separated collections:

| **Collection Name** | **Fields Stored** |
| --- | --- |
| users | credentials, profile info, role, verification status |
| products | barcode, name, price, quantity |
| purchases | barcode, quantity, price, date |
| sales | barcode, quantity, price, total, date |
| log\_data | username, login timestamp, logout timestamp |

* MongoDB handles queries over its native wire protocol.
* Flexible schemas and indexing improve performance for role-specific queries and history tracking.

**Data Flow Steps**

* **Page Load**
  + Browser requests HTML/CSS/JS
  + JS initializes UI, builds nav bar based on stored role
* **User Authentication**
  + Sign-up: POST /api/signup → MongoDB → returns success/error
  + Login: POST /api/login → verifies credentials & verified flag → returns user info
* **Role-Based Navigation**
  + JS filters a master list of links by the stored user role
  + Renders only authorized menu items (e.g., “Sales” for Employees, full Admin panel for Admin/Owner)
* **Purchase & Sales Workflows**
  + **Load Products**: GET /api/products → populate dropdown selectors
  + **Record Purchase**: POST /api/purchases → increase product stock in MongoDB → UI refresh
  + **Record Sale**:  
    • User selects product and quantity → client-side stock validation  
    • Confirmation modal appears → on confirm, POST /api/sales → MongoDB updates stock & logs sale  
    • Frontend re-fetches products and sales lists → updates tables and totals live
* **Activity Logging**
  + On logout, JS reads loginTime + current time → POST /api/logs → append { username, login, logout }
  + Logs page: GET /api/logs → displays sorted session records with “Last Online” calculation
* **Profile & Admin Panel**
* **Profile**: GET /api/users/{username} → shows user details
* **Admin Panel**:  
  • CRUD on products via /api/products endpoints  
  • User management: GET/PUT /api/users/{username} to verify accounts  
  • Log inspection: GET /api/logs for activity monitoring

**Technology Stack**

**Frontend**

The user interface is built with modern web standards to ensure compatibility, responsiveness, and ease of maintenance. By leveraging plain HTML5 and CSS3, we achieve a semantic document structure and flexible styling without the overhead of heavy frameworks. JavaScript (ES6+) drives dynamic behaviors—such as fetching data from the API, rendering tables and forms in real time, and managing client-side state via LocalStorage. FontAwesome is integrated for crisp, scalable icons that enhance usability and visual consistency across the application.

• HTML5  
• CSS3 (Flexbox, CSS Variables)  
• JavaScript (ES6+ features like async/await, modules)  
• FontAwesome (6.x icon set)

**Backend**

The server side is implemented in Python using the Flask microframework, chosen for its minimal setup, clear routing syntax, and strong community support. Flask’s modular design lets us define RESTful endpoints for authentication, product management, purchases, sales, logs, and profile operations. PyMongo serves as the direct bridge to MongoDB, allowing straightforward CRUD interactions without an ORM layer. This combination accelerates development and simplifies debugging, while retaining the flexibility to scale or migrate to more complex architectures in the future.

• Python 3.x  
• Flask (routing, request handling, middleware)  
• PyMongo (direct MongoDB driver)  
• Jinja2 (optional templating engine for server‐side views)

**Database**

A NoSQL database (MongoDB) underpins the project, selected for its schema-less document model, horizontal scalability, and JSON-like data interchange. Collections such as users, products, purchases, sales, and log\_data mirror the application’s domain model and evolve easily as requirements change. MongoDB Compass is used during development to visualize collections, inspect document structures, and run ad-hoc queries, facilitating rapid prototyping and data validation.

• MongoDB (Atlas or local deployment)  
• Collections: users, products, purchases, sales, log\_data  
• MongoDB Compass (GUI management)

**Tools & Utilities**

A suite of developer tools and services supports version control, testing, and collaboration. Visual Studio Code is the primary IDE, offering extensions for Python, JavaScript, and Docker. Git and GitHub maintain source history and enable issue tracking and pull-request workflows. Postman automates API testing and documents endpoints, while Chrome DevTools provides in-browser debugging for performance tuning and network inspection. Additional CLI utilities (e.g., npm for local servers, pipenv or venv for Python environments) ensure reproducible setups.

• Visual Studio Code (editor)  
• Git & GitHub (version control, collaboration)  
• Postman (API testing and documentation)  
• Chrome DevTools (frontend debugging)  
• Python virtual environments (venv/pipenv)

**Modules / Features**

Below is a breakdown of the Stockup project into its core modules. For each module, you’ll find a concise description, an optional screenshot placeholder, and the main technologies leveraged.

1. **Authentication Module**

* **Description**  
  Handles user registration, sign-in, and account verification. New accounts can be verified either via an email token or manually approved through the Admin Panel. Only verified users may log in.
* **Technologies Used**
* Frontend: HTML5, CSS3, JavaScript (fetch API)
* Backend: Flask routes (/api/signup, /api/login)
* Database: MongoDB (users collection) via PyMongo

**2. Role-Based Navigation Module**

* **Description**  
  Dynamically builds the navigation menu based on the logged-in user’s role (Admin, Owner, Employee, Supplier). Ensures that users only see links to pages they are authorized to access.
* **Technologies Used**
* JavaScript (DOM manipulation, LocalStorage)
* HTML templates with placeholder container

**3. Purchase Module**

* **Description**  
  Records new stock arrivals. Users select a product and enter quantity and unit price. The system updates the products collection, incrementing the quantity accordingly.
* **Technologies Used**
* Frontend: HTML form, JavaScript validation
* Backend: Flask POST /api/purchases
* Database: MongoDB (purchases collection), stock update in products

**4. Sales Module**

* **Description**  
  Captures sales transactions with real-time stock validation. Users choose a product, specify quantity, then confirm via a modal popup. Sale data is stored, and inventory is decremented. A live “Recent Sales” list and daily/all-time totals update instantly.
* **Technologies Used**
* Frontend: HTML/CSS, JavaScript (async/await), modal popups
* Backend: Flask POST /api/sales
* Database: MongoDB (sales collection), stock update in products

**5. Activity Logs Module**

* **Description**  
  Logs each user’s login and logout timestamps. A dedicated “User Logs” page fetches and displays records with a “Last Online” indicator (hours or minutes ago).
* **Technologies Used**
* Frontend: JavaScript fetch, table rendering
* Backend: Flask GET/POST /api/logs
* Database: MongoDB (log\_data collection)

**6. Profile Module**

* **Description**  
  Displays the logged-in user’s profile details (name, email, role, age, verification status). Includes a “Sign Out” button that logs the session end time before clearing LocalStorage.
* **Technologies Used**
* Frontend: JavaScript, HTML layout
* Backend: Flask GET /api/users/
* Database: MongoDB (users collection)

**7. Admin Panel Module**

* **Description**  
  Central dashboard for administrators and owners.
  + **User Management**: View new sign-ups, toggle verified status.
  + **Product CRUD**: Add, edit, or delete products.
  + **Log Inspection**: Browse all user sessions and export logs if needed.
* **Technologies Used**
* Frontend: HTML tables, JavaScript filters
* Backend: Flask routes for /api/users, /api/products
* Database: MongoDB (users, products, log\_data)

**8. Dashboard Module**

* **Description**  
  Landing page that aggregates key metrics at a glance:
  + Total products in stock
  + Today’s sales vs. all-time sales
  + Pending user verifications
  + Quick links to core modules
* **Screenshot (optional)**  
  Dashboard
* **Technologies Used**
* Frontend: JavaScript charts or summary cards
* Backend: Flask GET endpoints for aggregates
* Database: MongoDB aggregation queries

**Frontend Development**

**Framework Used**

* No JavaScript framework – built with vanilla ES6+ JavaScript.
* Core technologies: HTML5 for semantic structure, CSS3 (Flexbox, CSS Variables) for responsive styling, and FontAwesome for icons.

|  |
| --- |
| . **Reusable components:**   * bar dynamically populated via JS * Modal overlay for confirmation steps * Summary cards (today’s total, all-time total)   **UI/UX Design Strategy**   * Dark theme with high-contrast accents (green for actions, red for errors). * Icon-driven inputs and buttons to reinforce affordances. * Immediate validation feedback: disabled fields, inline error messages. * Confirmation modals to prevent accidental transactions. * Responsive breakpoints ensure panels stack on narrow screens.   **Sample Pages / Screenshots**   * Login / Signup page: simplified form with role selector * Dashboard view: summary cards + quick-access links * Sales module: product dropdown + total calculation + modal * Logs page: sortable table with “Last Online” column |

**Backend Development**

**Framework Used**

* Python 3.x with the Flask microframework for minimal routing overhead.
* PyMongo as the MongoDB driver—no ORM layer.

**API Structure**

* RESTful design: each resource (users, products, purchases, sales, logs) has dedicated endpoints.
* JSON over HTTP with standard status codes (200 OK, 201 Created, 400/401/404 errors).

**Routing and Controllers**

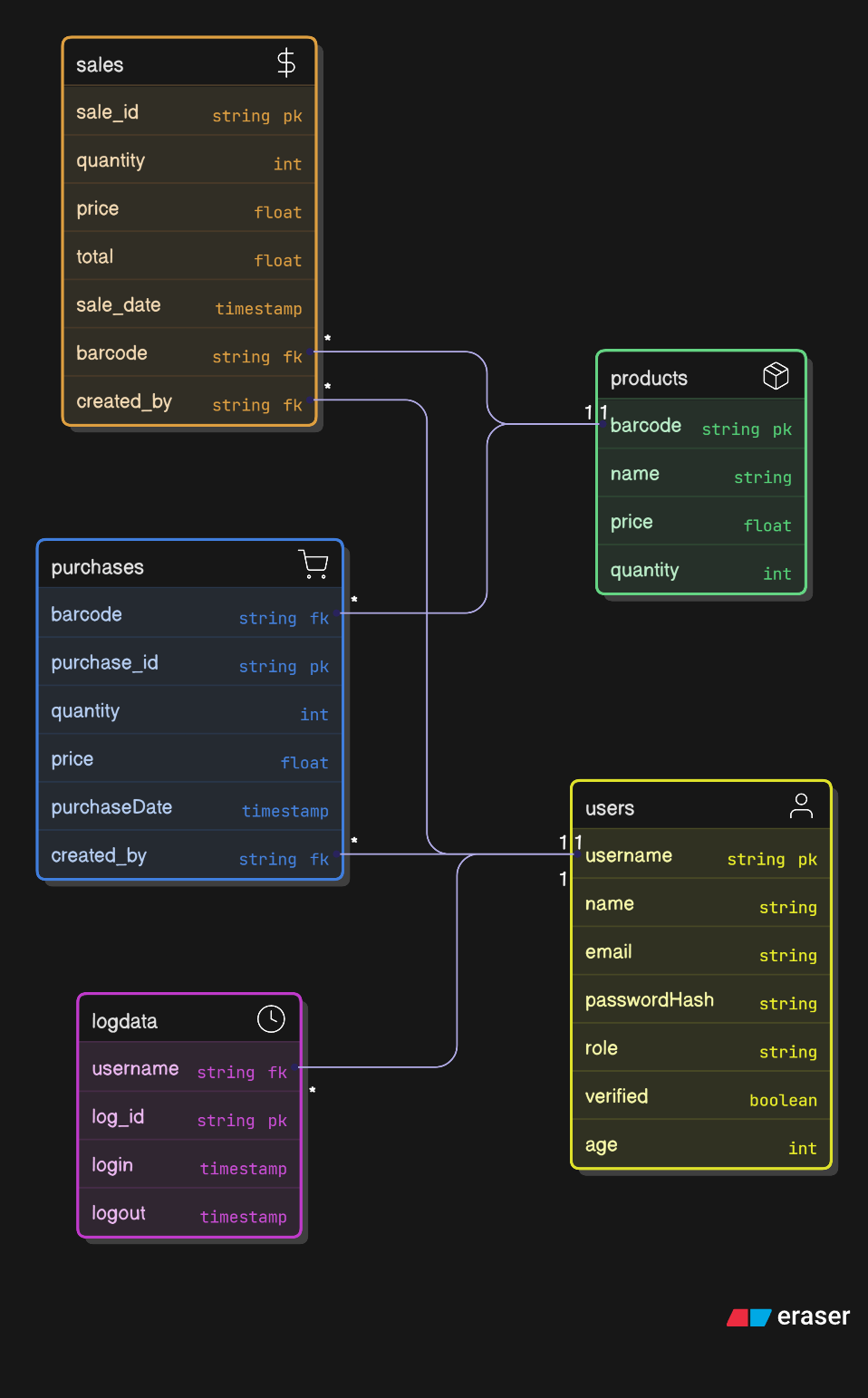
* /api/signup (POST): create new user, set verified=false.
* /api/login (POST): authenticate credentials and verified flag.
* /api/products (GET, POST, PUT, DELETE): product CRUD.
* /api/purchases (POST): record stock additions.
* /api/sales (GET, POST): record and list sales transactions.
* /api/logs (GET, POST): track login/logout events.
* /api/users/ (GET, PUT): fetch and update user profiles/verification.

**Authentication and Authorization**

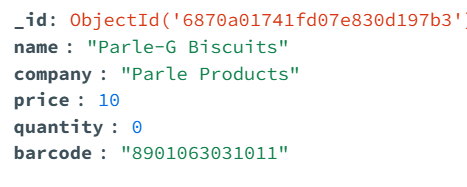
* Stateless JSON API: frontend stores username, role, and loginTime in localStorage**.**
* Each protected route checks role and verified status in request payload.

**Database Design**

* **Collections & Schemas**
  + User\_data: { username, name, role, email, password, verified, age }
  + Product\_data: { barcode, name, price, quantity }
  + Purchases\_data: { barcode, quantity, price, purchaseDate }
  + Sales\_data: { barcode, quantity, price, total, sale\_date }
  + log\_data: { username, login, logout }
* **ER Diagram**

****

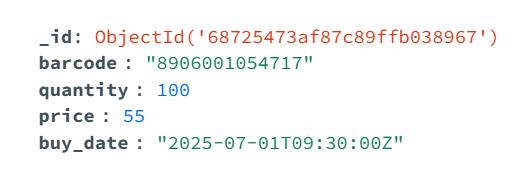
* **Sample Data**
* **Product\_data**

****

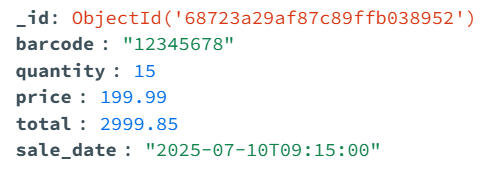
* **User\_data**

****

* **Purchase\_data**

****

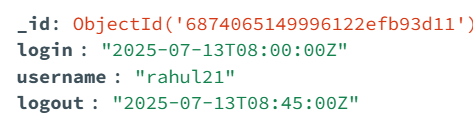
* **Sales\_data**

****

* **Order\_data**



* **Log\_data**



**Security Measures**

* Account Verification:
  + Email‐token flow or admin toggles verified flag
  + Unverified users cannot log in
* Input Validation & Sanitization: Both client-side and server-side checks

**Limitations**

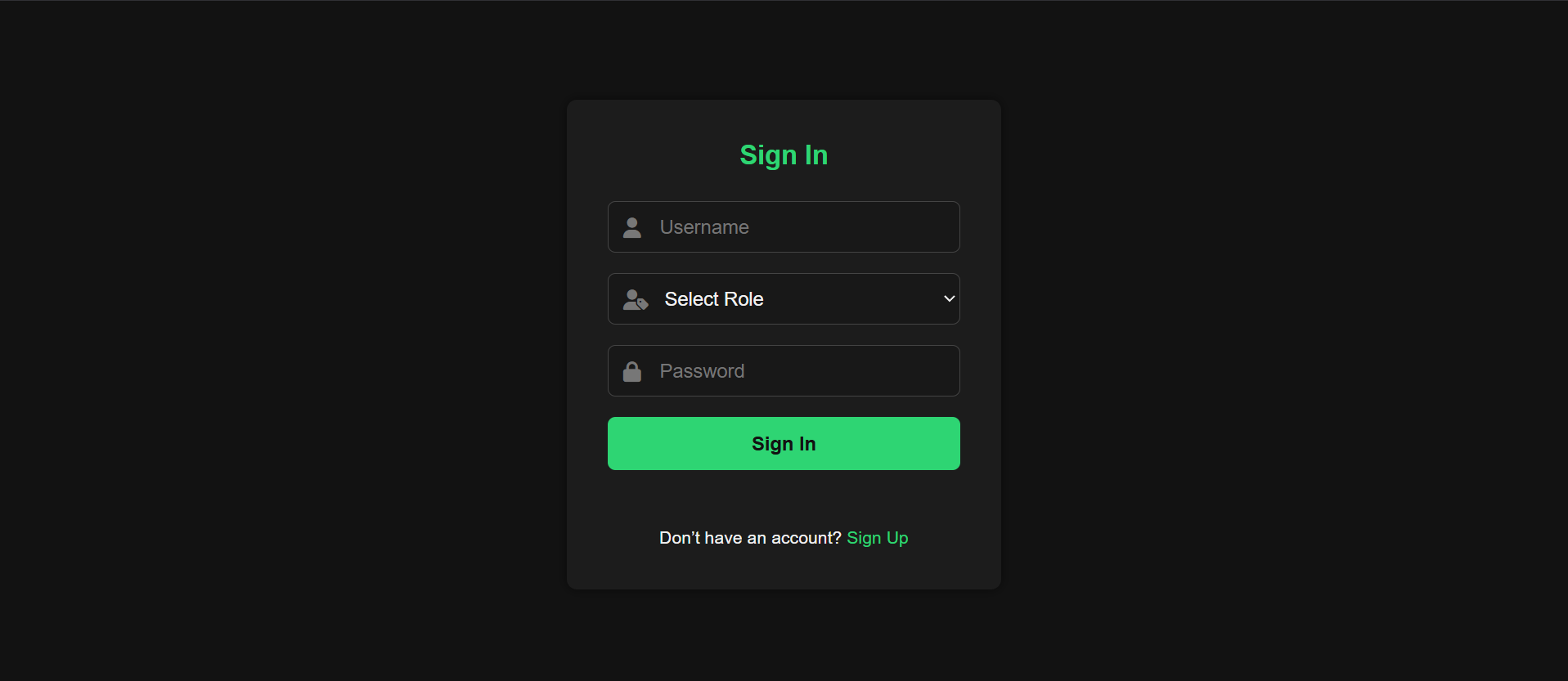
* **Account Verification Workflow**• Currently relies on manual toggling by an administrator; email-based token delivery is not fully integrated.
* **Data Volume & Pagination**  
  • Sales, purchase, and log tables display all records at once; no built-in pagination or lazy loading for large datasets.
* **Single-Tenant Design**• The system manages one business instance only; multi-tenant or branch support is not available.
* **UI Responsiveness**  
  • Layout optimized for desktop; mobile responsiveness and accessibility (WCAG) need further refinement.
* **Reporting & Export**  
  • No built-in CSV/PDF export or advanced reporting (charts/graphs) in the current version.
* **Security Hardening**  
  • Account lockout after repeated failed login attempts is not implemented.  
  • No CAPTCHA or two-factor authentication (2FA) options yet.

**Future Enhancements**

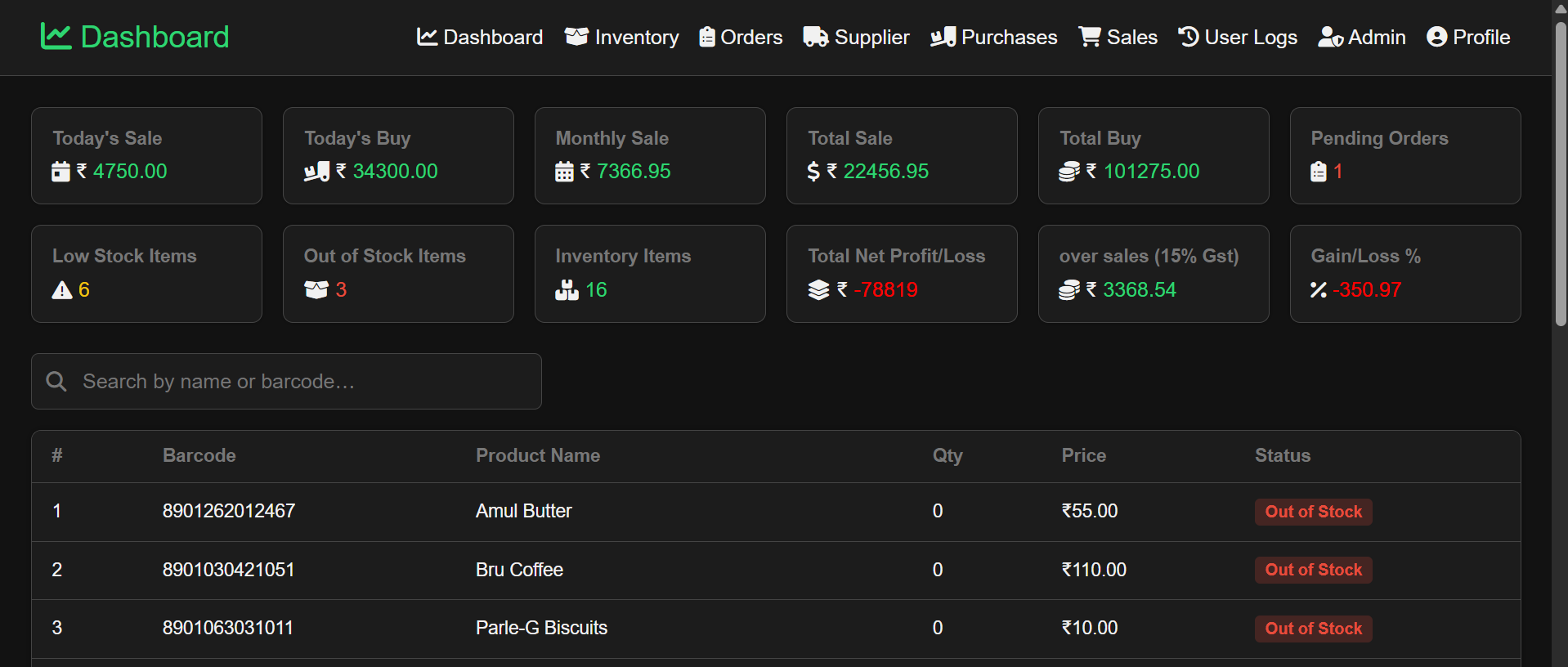
* **Robust Account Verification**  
  • Integrate email-or SMS-based token workflows for automated user verification.  
  • Add two-factor authentication (2FA) for high–security roles.
* **Data Management & Analytics**  
  • Implement server-side pagination, filtering, and search for large tables.  
  • Build interactive dashboards with Chart.js or D3.js for sales trends, stock alerts, and user activity.
* **Multi-Tenant & Role Extensions**  
  • Expand to support multiple branches or organizations in one instance.  
  • Create granular, customizable roles and permissions managed via the Admin Panel.
* **Performance & Scalability**  
  • Containerize services (Docker) and orchestrate with Kubernetes for horizontal scaling.  
  • Introduce a caching layer (Redis) for frequently accessed data (product lists, summaries).
* **DevOps & CI/CD**  
  • Set up automated testing pipelines (unit, integration, end-to-end) with tools like GitHub Actions.  
  • Automate deployments to staging/production environments and enable rollback strategies.
* **Reporting & Export Features**  
  • Add CSV/PDF export for sales, purchases, and activity logs.  
  • Schedule automated email reports or notifications for low-stock warnings and daily summaries.
* **Mobile & Offline Support**  
  • Develop a responsive mobile web UI or native mobile apps using React Native or Flutter.  
  • Implement offline data persistence and sync for on-the-go operations in low-connectivity areas.

**Screenshots**

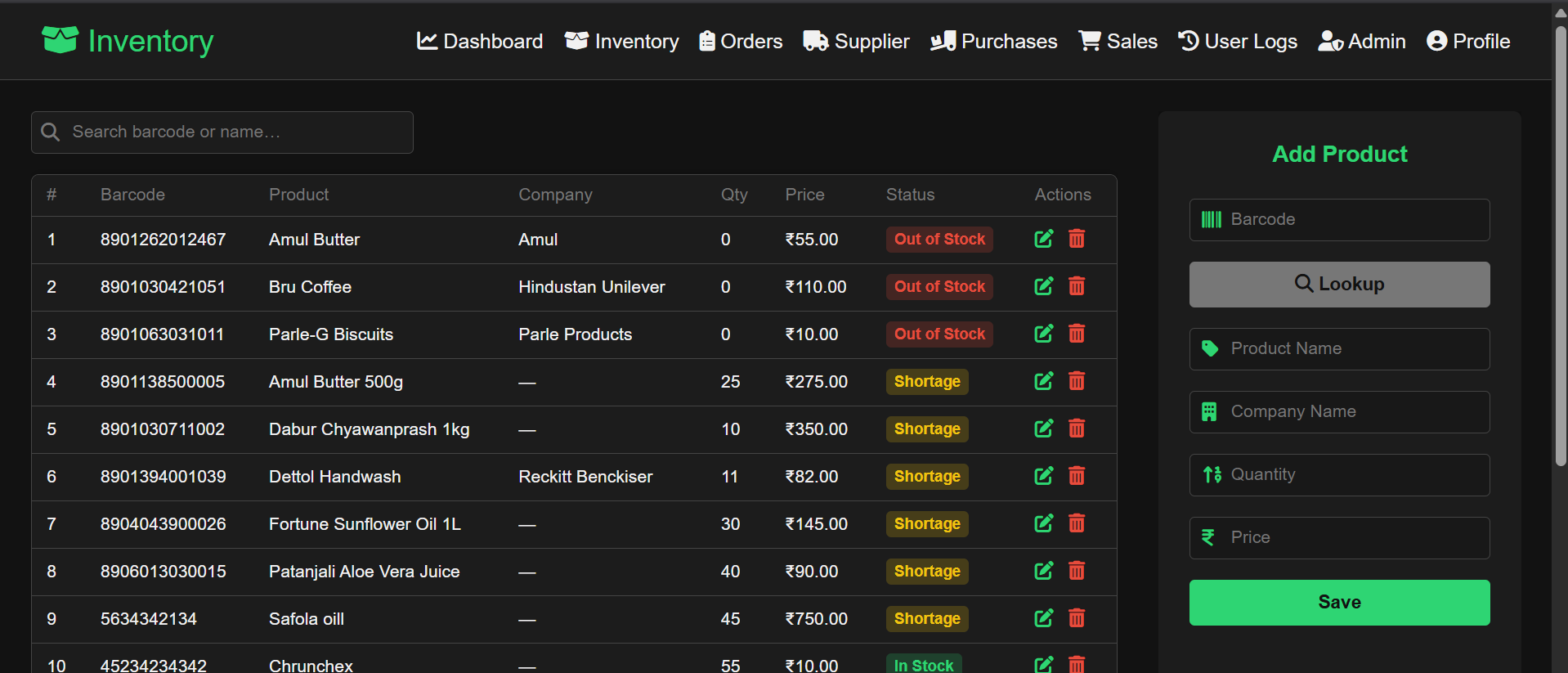
**Login page**

****

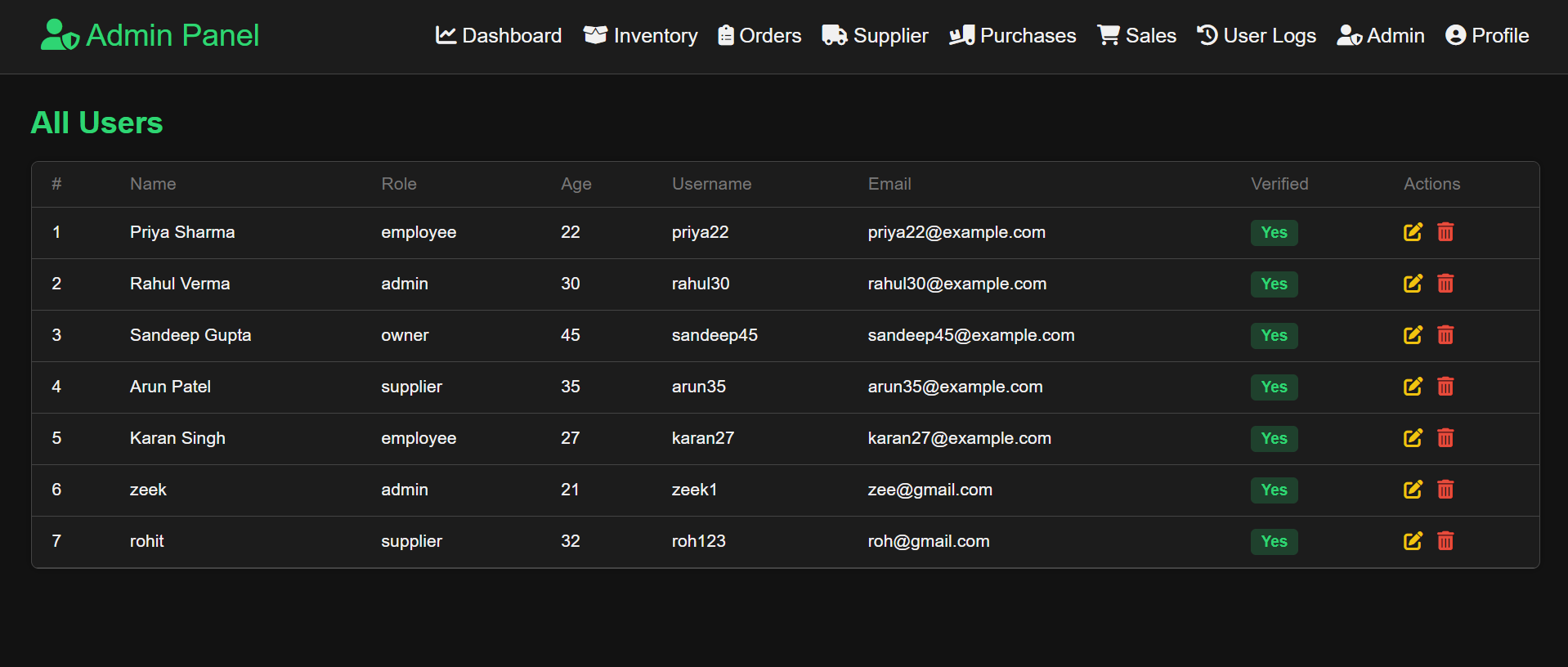
**Dashboard**

****

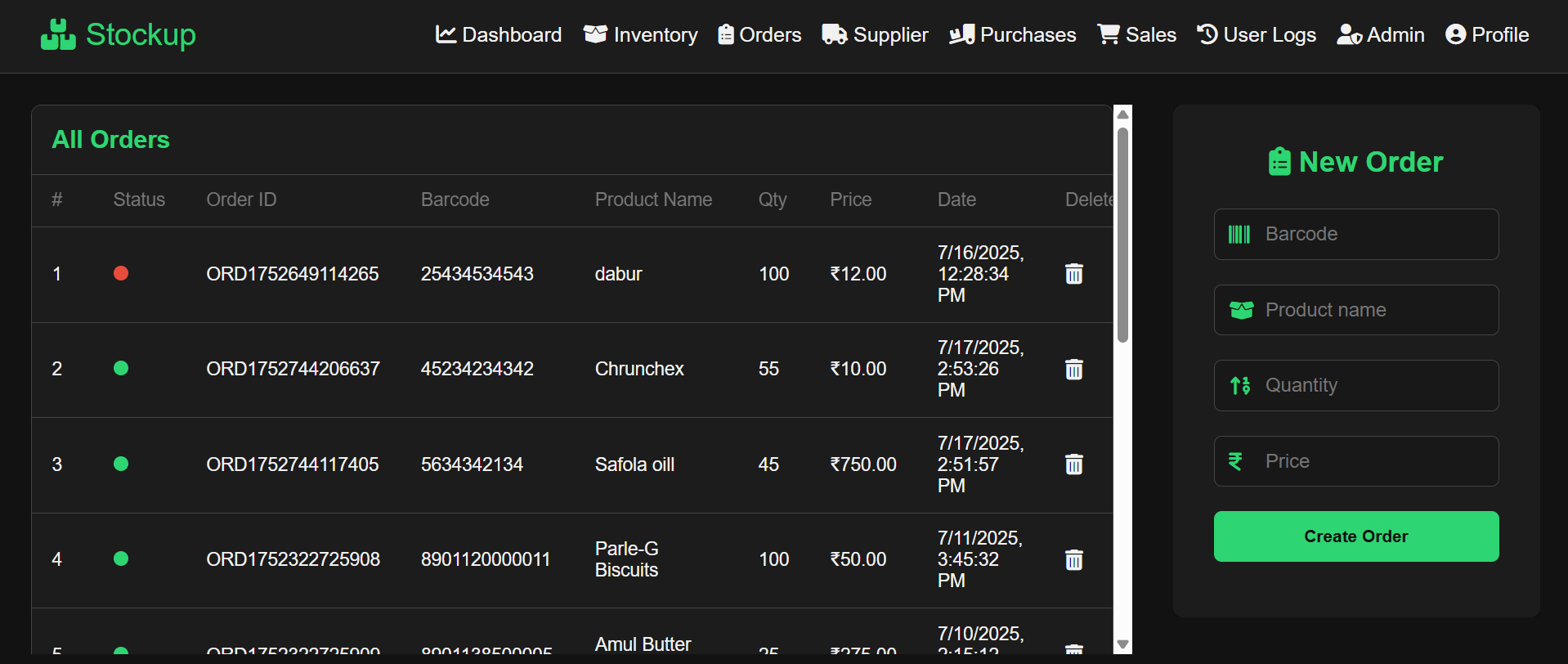
**CRUD operations**

****

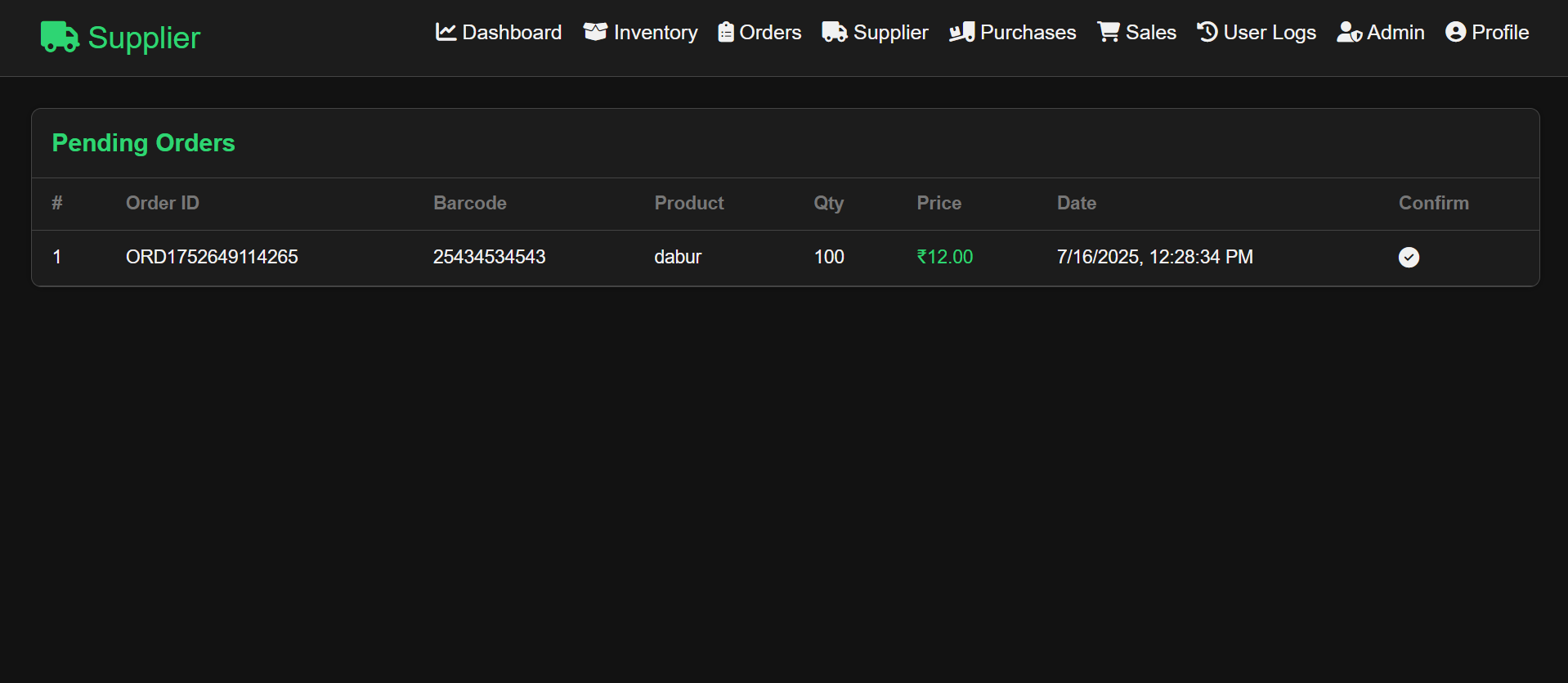
**• Admin panel etc.**

****

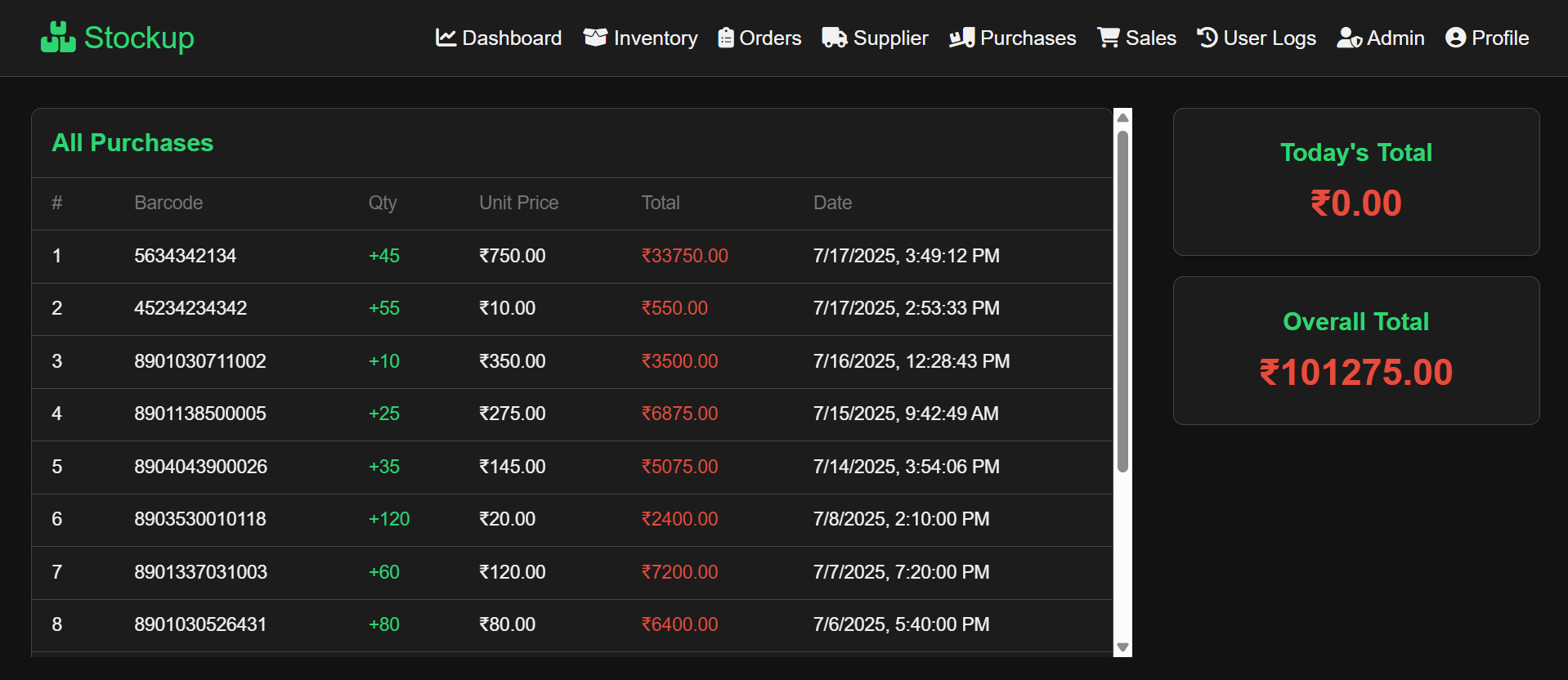
**Order page**

****

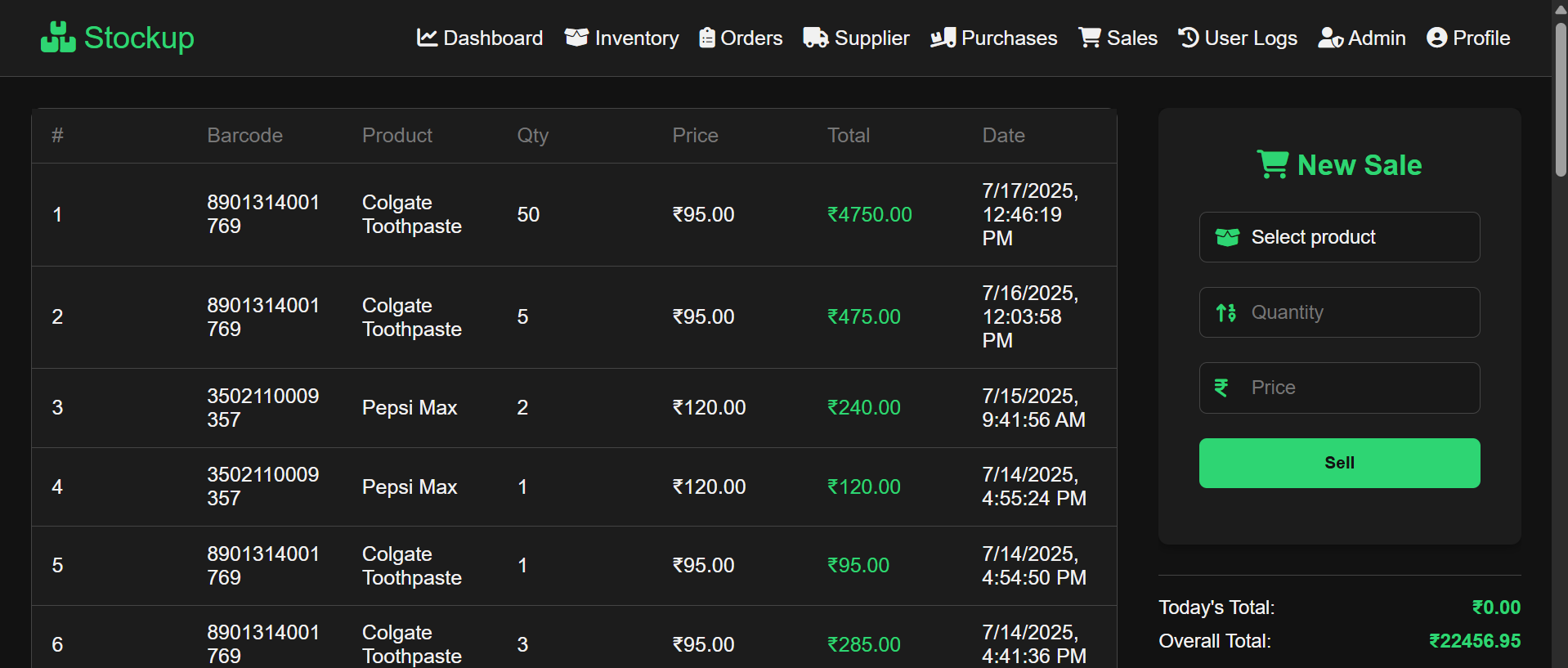
**Supplier page**

****

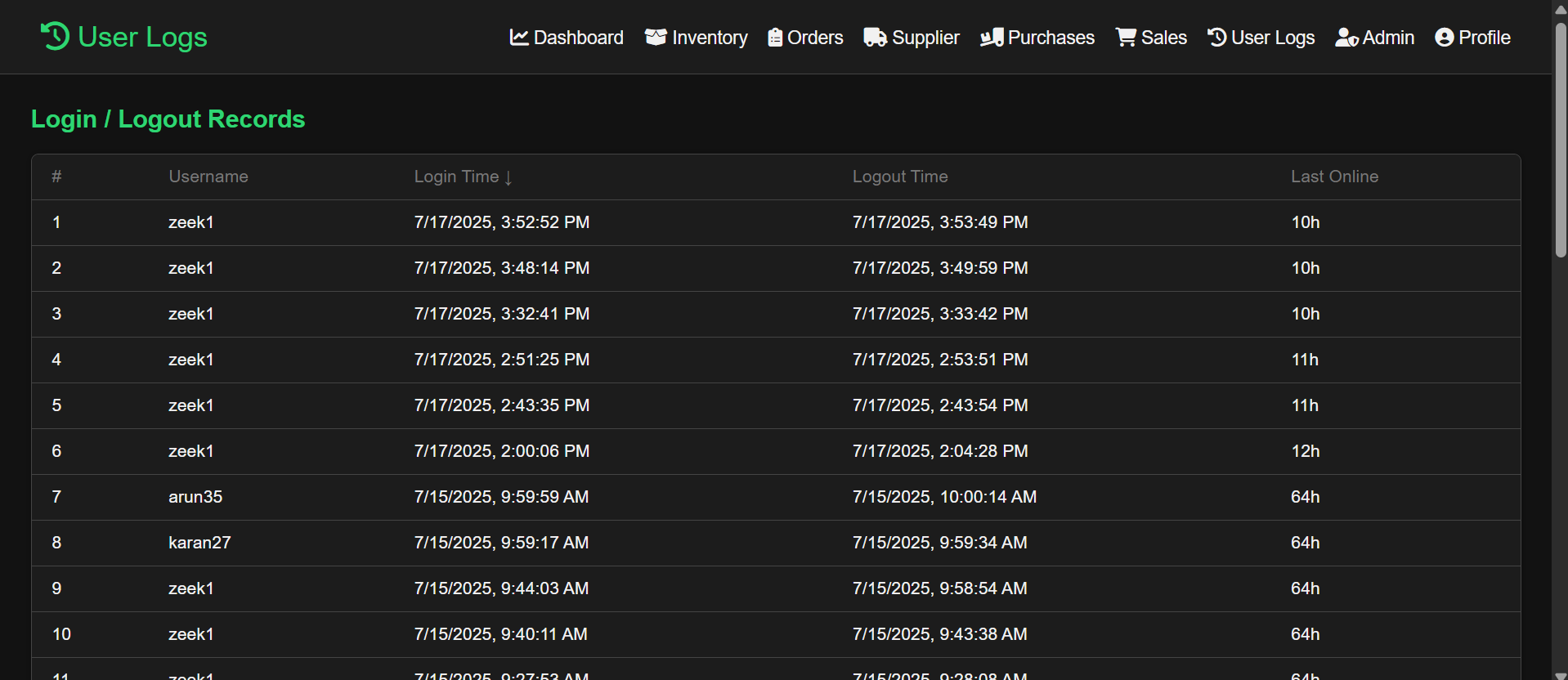
**Purchase page**

****

**Sales page**

****

**Logs page**

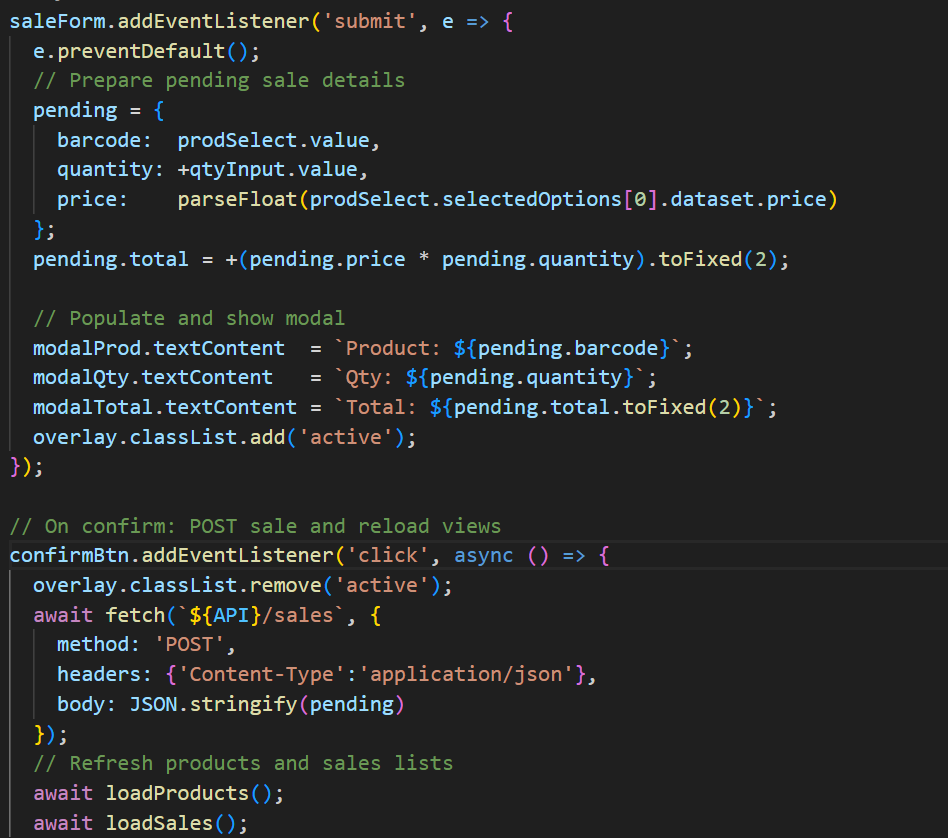
****

**Code Snippets (Key Functions)**

* **Flask Route: Record a Sale (/api/sales)**

****

* **JavaScript: Confirmation Modal & Dynamic List Refresh**

****

**GitHub repo link :** <https://github.com/Zeek20/Stock-up>

**REFERENCE**

* Miguel Grinberg. Flask Web Development: Developing Web Applications with Python. O’Reilly Media, 2018.
* MongoDB, Inc. MongoDB Manual. [https://docs.mongodb.com](https://docs.mongodb.com/)
* PyMongo Developers. PyMongo 3.x Documentation. [https://pymongo.readthedocs.io](https://pymongo.readthedocs.io/)
* Python Software Foundation. Python 3 Documentation. <https://docs.python.org/3/>
* bcrypt Developers. bcrypt Python Library. <https://pypi.org/project/bcrypt/>
* Mozilla Contributors. MDN Web Docs – HTML, CSS, JavaScript. [https://developer.mozilla.org](https://developer.mozilla.org/)
* FontAwesome. FontAwesome Icon Library. [https://fontawesome.com](https://fontawesome.com/)
* Postman, Inc. Postman API Platform. [https://www.postman.com](https://www.postman.com/)
* Open Food Facts. Open Food Facts API Documentation. <https://world.openfoodfacts.org/data>
* GitHub Docs. GitHub Guides & Workflows. [https://docs.github.com](https://docs.github.com/)