Inventory Management System (StockEz)
Presented to

K.R. Mangalam University

K.R. MANGALAM UNIVERSITY

Bachelors In Computer Application

with Specialisation in

Artificial Intelligence and Data Science

SUBMITTED BY : G218

- Dhruv Gupta (2401201015)
- Mehul Srivastava (2401201005)
- Kshitij Marwah (2401201021)

SCHOOL OF ENGINEERING & TECHNOLOGY

K. R. MANGALAM UNIVERSITY

SOHNA, HARYANA 122103 , INDIA

Introduction

A **StockEz** is a modern, full-stack Inventory Management System designed to streamline stock control, ordering, and vendor communication for small-to-medium businesses. With a clean, responsive UI built in vanilla HTML, CSS, and JavaScript, StockEz lets administrators and warehouse staff:

- Authenticate and verify via secure signup/login flows, including email validation and a one-time verification code.
- Manage Products & Categories, with dynamic stock updates and real-time history logging of every change (quantity adjustments, price updates, category moves).
- Handle Customers & Vendors in one place, validating contact details on create/edit and maintaining supplier assignments for each product.
- Create & Edit Orders, enforcing minimum-stock checks, automatic stock refunds on order edits/deletions, and recalculated order values.
- Define Low-Stock Alerts, so that when a product's quantity falls below a set threshold, StockEz automatically emails the responsible vendor requesting replenishment.
- Track Full Audit Trails, storing all inventory mutations in a "product_history" table, timestamped in your local timezone (Asia/Kolkata).

On the frontend, HTML provides structure, CSS handles styling (including a dark/light theme toggle), and JavaScript drives interactivity—modals for forms, sidebar expand/collapse with dynamic logos, and seamless single-page transitions for signup/login.

The backend is powered by Node.js and Express, backed by PostgreSQL. On startup, StockEz automatically creates any missing tables (users, products, orders, vendors, alerts, etc.) with the correct foreign-key constraints and cascade rules. Secure password hashing (bcrypt), email delivery via Gmail SMTP (nodemailer), and external email-validation using MailboxLayer ensure both reliability and data integrity.
Together, these components make StockEz an end-to-end solution for real-time stock management, order processing, vendor coordination, and full audit-trail visibility—all in a lightweight, easy-to-deploy package.

Objectives:-

1. **Centralize Inventory Control**
   - Provide a unified platform for managing products, categories, customers, and vendors, eliminating fragmented spreadsheets and manual record-keeping.
2. **Ensure Data Accuracy & Auditability**
   - Automatically log every change—stock adjustments, price updates, category moves, order creations/edits/deletions—in a timezone-aware history table, enabling full traceability and accountability.
3. **Automate Order Processing**
   - Enforce real-time stock validations on order placement and edits; dynamically update product quantities; and recalculate order values to prevent overselling and maintain accurate inventory levels.
4. **Streamline Vendor & Customer Management**
   - Validate email addresses at creation and edit time via a mail-validation API; support secure vendor and customer data handling; and enable swift communications for restocking alerts.
5. **Implement Low-Stock Alerting**
   - Allow administrators to set per-product reorder thresholds; automatically detect when stock falls below the threshold and send templated, SMTP-delivered emails to the assigned vendor requesting replenishment.
6. **Secure Authentication & Verification**
   - Leverage bcrypt-hashed passwords, email-based OTP verification during signup, and sessionStorage for user sessions to protect access and maintain data privacy.
7. **Deliver a Responsive, Intuitive UI**
   - Utilize a single-page, CSS-animated login/signup flow, light/dark theme toggle, collapsible sidebar with dynamic logos, and modal dialogs for all CRUD operations to ensure a smooth user experience on desktop and mobile.
8. **Facilitate Future Scalability**
   - Architect the backend with Express + PostgreSQL auto-migrations, modular API routes, and clear separation of concerns, so new features (e.g., reporting dashboards, multi-warehouse support, mobile apps) can be added with minimal rework.
9. **Enable Easy Deployment & Maintenance**
   - Provide a comprehensive README with setup scripts, environment-based configuration (dotenv), and automatic table creation on startup, ensuring that new developers and deployments can get StockEz running locally or in the cloud in minutes.

Technology Used:-

Backend :

Component	Technology
Runtime	**Node.js v18+**
Framework	**Express.js**
Database	**PostgreSQL v12+**
ORM	**pg(PostgreSQL client)**
Authentication	**bcrypt, sessionStorage**
Email	**Nodemailer, MailboxLayer API**
Validation	**Reges, MailboxLayer**

Frontend :

Component	Technology
Markup	**HTML5**
Styling	**CSS**
Scripting	**Javascript**
State Management	**sessionStorage**
Charts	**Chart.js**

DevOps :

Service	Purpose
Render.com	Backend Hosting
Github Pages	**Frontend Deployment**

System Requirements

System Components:-

- **Harware:**
  - 1 GB RAM Minimum
  - 100 MB Disk Space
- **Software:**
  - Node.js v18+
  - Express.js
  - Web Brower (Brave, Chome Edge, MS Edge)
  - Code Editor (VS Code)
- **Services:**
  - MailBoxLayer API Key
  - Gmail SMTP Credentials

Features Implemented:-

Feature	Description
User Authentication and Verification	Secure signup/login with bcrypt-hashed passwords; email OTP verification on signup before account activation; sessionStorage to manage user sessions.
Dynamic Login/Signup UI	Single HTML file “pages” for login & signup with smooth CSS fade transitions; form reset on panel switch; password-show toggles with context-sensitive eye icons.
Dark/Light Theme Toggle	Persistent (sessionStorage) theme switcher affecting sidebar, modals, tables, forms, and controls across all pages.
Collapsible Sidebar with Dynamic Logos	Sidebar expands/collapses on toggle; displays two distinct logos based on state; consistent icons and labels for navigation links.
Product History Tracking	Automatic logging of every product change—creation, updates, stock adjustments (order creation/edit/delete)—into a product_history table with timestamps (Asia/Kolkata) and a human-readable history UI (not raw JSON).
Category Management	CRUD for categories with name, description, and associated products; responsive table and modal form UI supporting dark/light themes.
Customer Management	CRUD for customers with name, validated email (via MailboxLayer API) and phone number entry; modals for add/edit; history not tracked.
Vendor Management	CRUD for vendors with name, validated email & phone (10-digit) on create/edit; ability to assign supply areas; styled vendor list & modals adhering to theme.
Order Processing	Create/Edit/Delete orders in a modal workflow: select customer, date, add multiple products & quantities; real-time stock checks; dynamic total calculation; prevents overselling.

Feature	Description
Order Editing Logic	Editing refunds previous stock, then deducts new quantities; updates order date/customer/order value atomically; avoids creating duplicate orders.
Low-Stock Alerts	Administrators set reorder thresholds; to record when stock dips below, logging the alert in an alerts table.
API-First Backend	Express.js + PostgreSQL with auto-migrations on startup; secure requireUser middleware (x-user-id header); RESTful endpoints for all resources; CORS enabled for cross-origin frontends.
Environment-Driven Configuration	.env support for database credentials, API keys, email SMTP settings; no secrets in source; automatic table creation if missing.

Flow of the Application:-

1. Homepage loads with a dashboard interface including a collapsible sidebar, welcome message, and a theme toggle.
2. User can:

- 🔍 View overall stock statistics (e.g., total products, categories, vendors, customers).
- 📦 Add, edit, or delete products via modal forms with validation and live updates.
- 🗂️ Manage categories, vendors, and customers through structured interfaces with CRUD operations.
- 🧾 Create or modify orders, choosing customers and adding multiple products with real-time stock deduction and total calculation.
- 📜 Track product history, viewing all changes (like updates and stock movements) in a readable format.
- ⚠️ Receive low-stock alerts, and if threshold is crossed, a replenishment email is sent to the vendor.
- 🌓 Toggle between dark and light themes for better accessibility and comfort.
- 🔐 Log in or sign up securely with email verification (via OTP) and encrypted password storage.

Deployment:-

To run the project locally and understand its deployment, follow the steps below:

1. 🔧 Pre-requisites
   - Node.js (v16+ recommended) and npm installed
   - PostgreSQL database (either locally or a hosted service like render
   - A github account (for hosting the frontend via github pages)
2. Clone the Repository

```
git clone https://github.com/BeastBoom/Dhruv_Gupta_BCA-A_Inventory-Management-System.git
```

3. Backend Setup (Express + PostgreSQL)
   - Install Dependencies

     ```
     npm install
     ```

   - Configure Environment
     - Create a file named .env in the backend/ directory with:

       ```
       PG_HOST=<your-db-host>
       PG_PORT=<your-db-port>          # e.g. 5432
       PG_USER=<your-db-username>
       PG_PASSWORD=<your-db-password>
       PG_DATABASE=<your-db-name>
       EMAIL_VALIDATION_API_KEY=<mailboxlayer-or-other-api-key>
       EMAIL_SERVICE=gmail             # or your chosen SMTP provider
       EMAIL_USER=<your-email-address>
       EMAIL_PASS=<your-email-password-or-app-password>
       ```

   - Initialize data base & start server

     ```
     npm run start
     ```

4. Frontend Setup (Vanilla HTML/CSS/JS)
    - In a second terminal, from the project root:

```
cd frontend
npm install http-server --global
http-server . -p 8080
```

    - Open your browser to

```
http://localhost:8080/index.html
```

5. Deploying to render (backend)
    - Push your backend code to GIthub
    - In your render dashboard, click New → Web Service.
    - Connect to your github repo and select the backend/ directory
    - Set the environment to Node, Build Command to npm install, and Start Command to npm run start
    - In Advanced → Environment, add all the same environment variables as your local .env
    - Click Create Web Service.
6. Deploying to Github Pages (Frontend)
    - In your frontend/ directory, add a CNAME file if you have a custom domain, otherwise skip.
    - Commit and push to the main branch
    - In Github Repository Settings → Pages, Set:
        - Source : Main branch → /frontend folder
        - Theme : (Optional)

Working Process:-

1. **User Authentication**
   - Visitor lands on the StockEz login/signup page.
   - They can toggle between Sign In and Sign Up panels.
   - New users sign up with email & password (validated via API) and then verify their email before first login.
   - Returning users log in; successful authentication stores their user ID in session storage and redirects them to the dashboard.
2. **Dashboard & Navigation**
   - After login, users land on the Overview dashboard showing key metrics (total products, low-stock alerts, recent orders).
   - The sidebar (expandable/collapsible) provides navigation to Products, Categories, Orders, Customers, Vendors, and Alerts pages.
3. **Managing Products**
   - On Products page, users can Add, Edit, or Delete products via modal forms.
   - Each action logs history (quantity changes, name/category edits, order impacts) with a timestamp.
   - Low-stock products automatically generate alerts (email to assigned vendor if configured).
4. **Categories & Vendors**
   - On Categories, users group products and view associated items.
   - On Vendors, users add/edit vendors (with email validation) and assign them to specific products or categories.
5. **Orders Workflow**
   - On Orders, users create new orders by selecting customer, date, and product items.
   - The system checks stock availability in real time, updates inventory, and logs product history.
   - Editing an order refunds previous quantities before applying updates; deleting an order also restores stock.

6. Customers Management
    - On Customers, users add/edit customer records (phone & email validated), view their past orders, or delete them.
7. Alerts System
    - Whenever a product's inventory drops below its reorder threshold, the system raises an alert:
        i. Logs an entry in the alerts table.
        ii. Sends an automated email to the assigned vendor(s).
    - On the Alerts page, users can view, acknowledge, or resolve these alerts.
8. Product History & Audit
    - Every change—product edits, orders, deletions—appears in the History modal for that product (with change type, details, and accurate local timestamp).
9. Theming
    - Users can toggle between Light and Dark themes; all forms, tables, and modals respect the chosen theme.

Limitation:-

- **Email Verification Dependency**:
    - Relies on third-party validation APIs (MailBoxLayer), which may impose rate limits or occasional downtime.
- **Simple Authentication**:
    - While we hash passwords and verify email, there is no multi-factor authentication or role-based access control beyond a single "user" role.
- **No File or Image Uploads:**
    - Product and vendor images must be hosted elsewhere; the app only stores URLs.
- **Static Frontend Hosting**:
    - GitHub Pages serves the UI; any frontend changes require a redeploy.
- **Limited Reporting & Analytics**:
    - No built-in dashboards for historical sales, trends, or performance beyond basic overview metrics.
- **Single-Tenant Data Mode**l:
    - All data is scoped per user ID, but there's no multi-organization support.

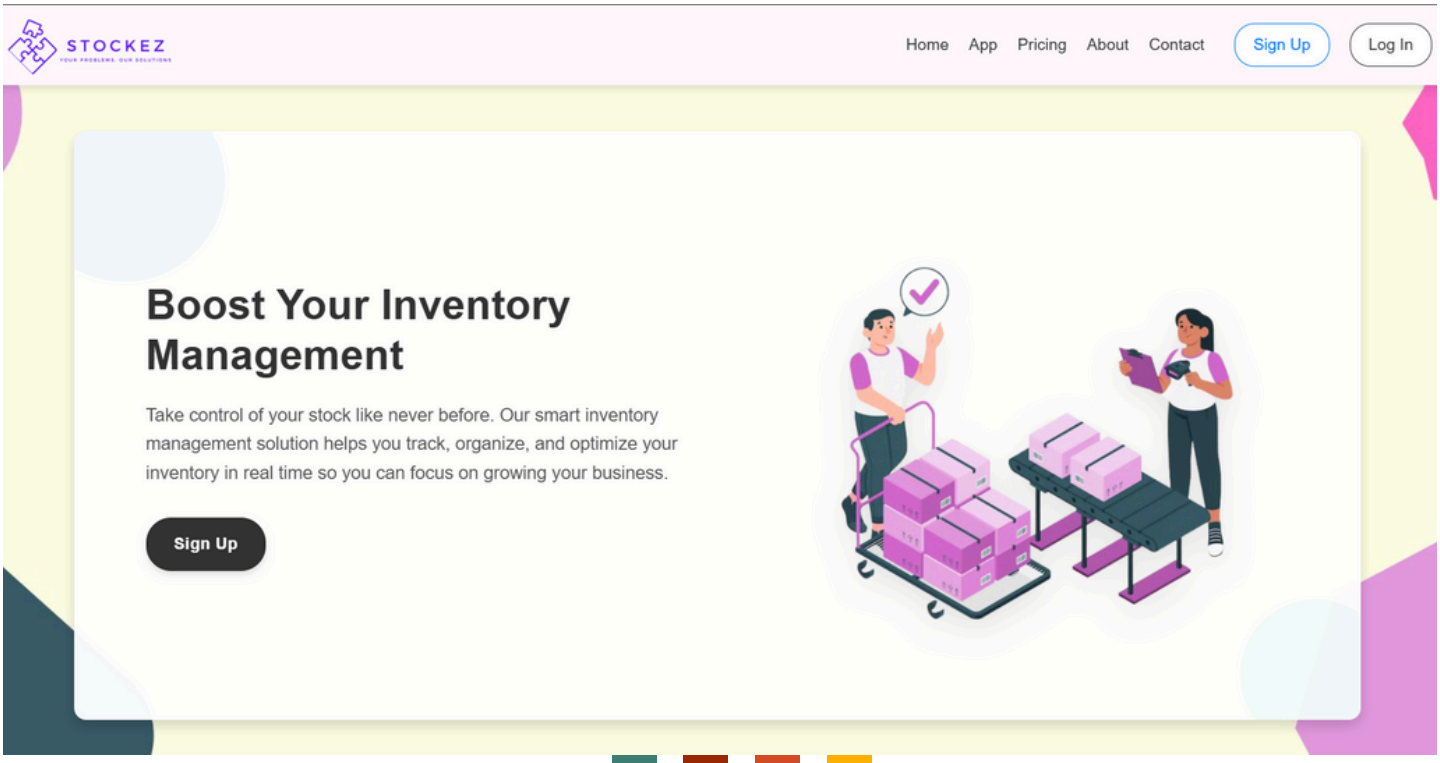
- **No Offline Support**:
  - The application requires an active internet connection; there is no caching or PWA capability.
- **Basic Error Handling:**
  - Most API errors are surfaced as alerts; there is no centralized logging or retry mechanism in the frontend.
- **Scalability Constraints**:
  - Designed for small to mid-sized inventories; may require optimization (connection pooling, pagination, queueing) for enterprise-level volumes.

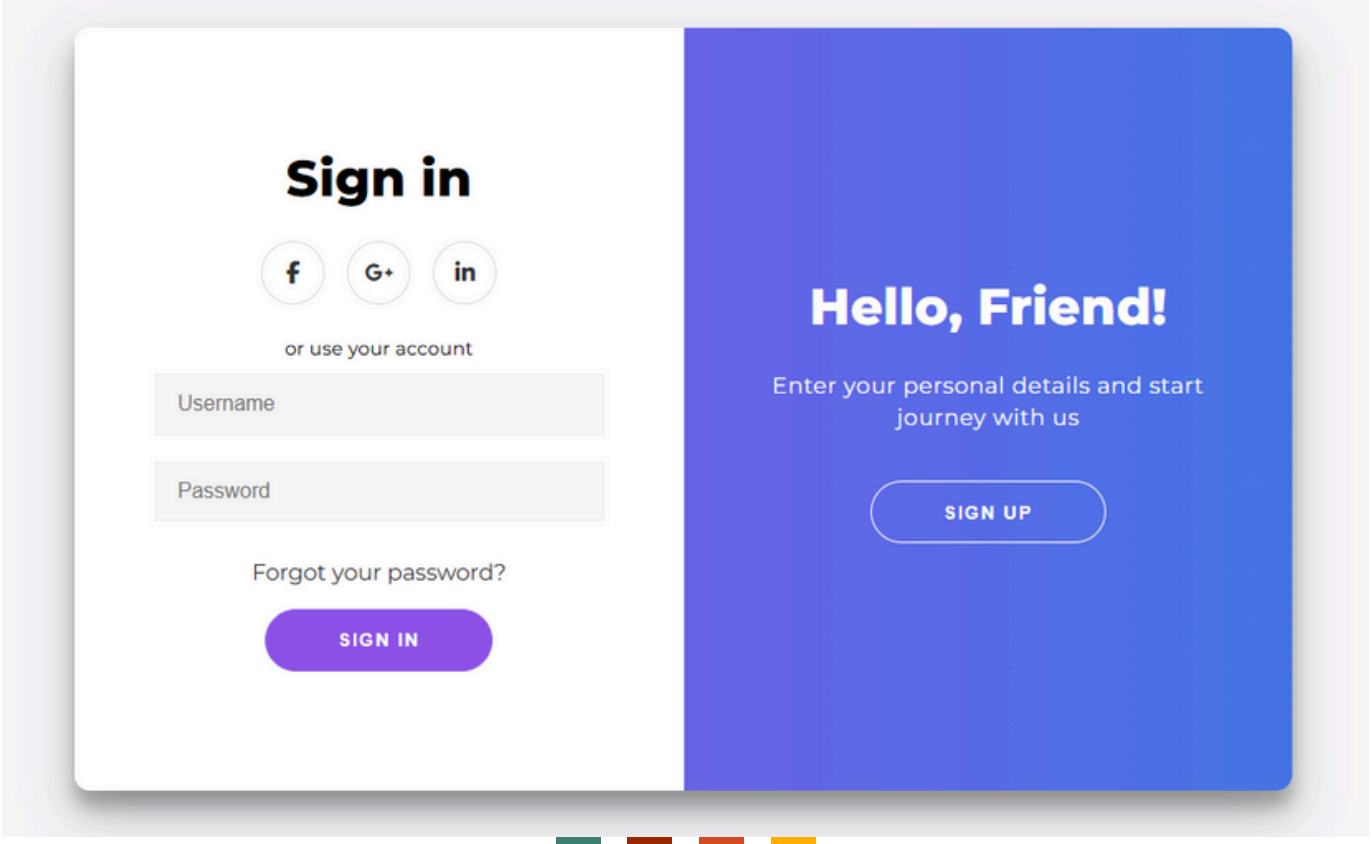
Future Improvement:-

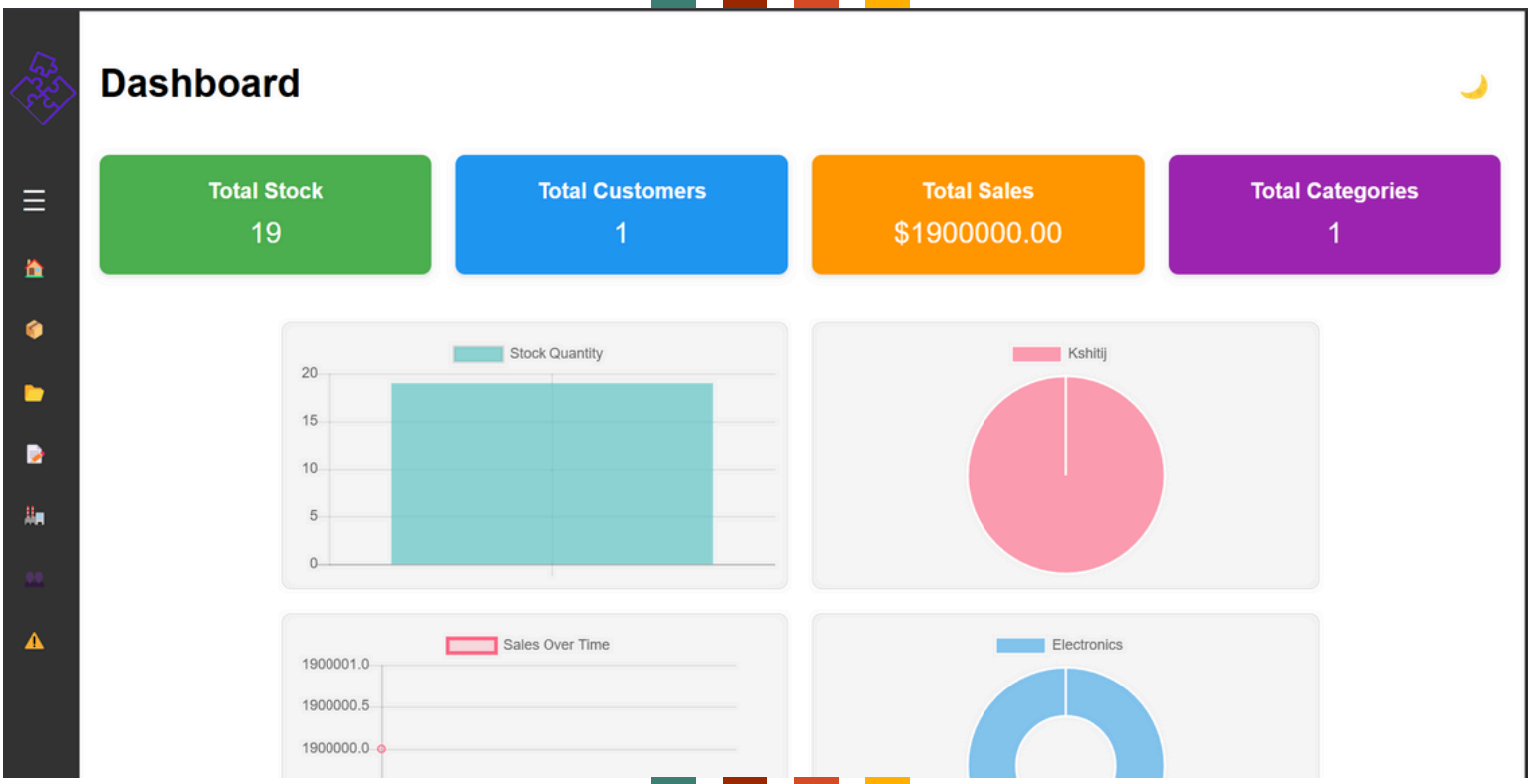
- **Scalable Database Integration**
  - Migrate from PostgreSQL on Render to a more robust clustered setup (e.g., Amazon RDS Multi-AZ) or add Redis caching to handle high-throughput inventory operations.
- **Advanced Authentication & Authorization**
  - Implement JWT-based login with refresh tokens, role-based access control (Admin, Manager, Vendor), and optional OAuth2/SAML single sign-on for enterprise environments.
- **Product Image & Document Uploads**
  - Allow users to upload product photos, vendor contracts, and invoices directly; integrate with cloud storage (AWS S3, Google Cloud Storage) and generate secure, time-limited URLs.
- **Order & Stock Reporting Dashboard**
  - Create interactive charts and tables summarizing sales trends, low-stock alerts, vendor lead times, and order fulfillment rates, leveraging Recharts or Chart.js on the frontend.
- **Notifications & Alerts**
  - Add configurable email and SMS notifications for critical events (low stock, order shipments, vendor responses) using services like SendGrid, Twilio, or Mailgun.

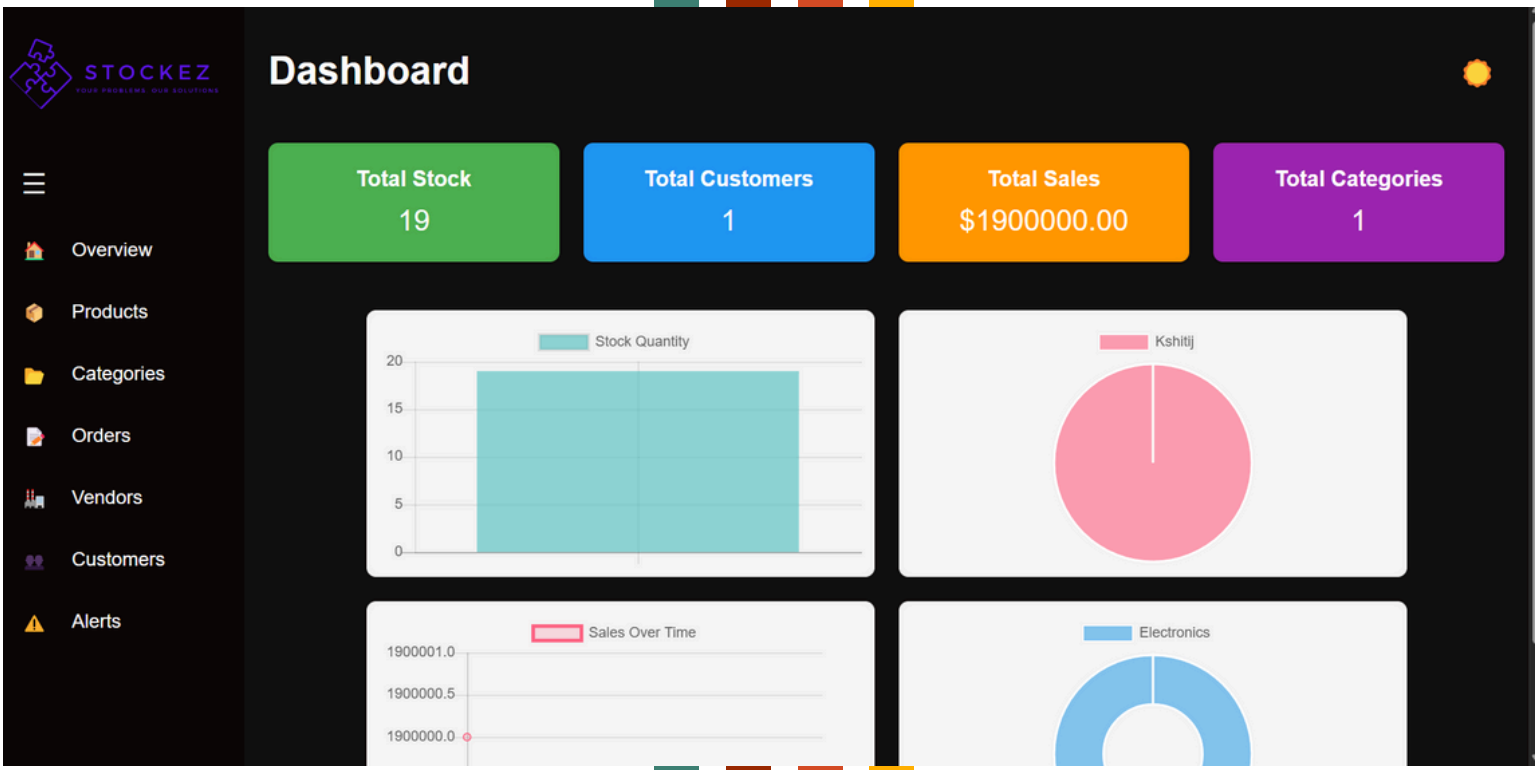
- **Batch Import/Export**
  - Enable CSV/XLSX import and export of products, customers, orders, and vendor lists to support bulk data operations and integrations with ERP systems.
- **Audit Trails & Activity Logs**
  - Record all CRUD actions with timestamps, users, and before/after values; surface these logs in an admin UI for compliance and troubleshooting.
- **Offline & Mobile Support**
  - Build a Progressive Web App (PWA) with local caching and background synchronization so warehouse staff can continue working during intermittent connectivity.
- **API Versioning & Documentation**
  - Adopt semantic versioning for your REST API, add OpenAPI/Swagger documentation, and publish a developer portal for external integrations.
- **Plugin/Module Architecture**
  - Refactor the codebase into modular services (products, orders, vendors, alerts) and consider migrating to a microservices or serverless architecture for future scalability and maintainability.

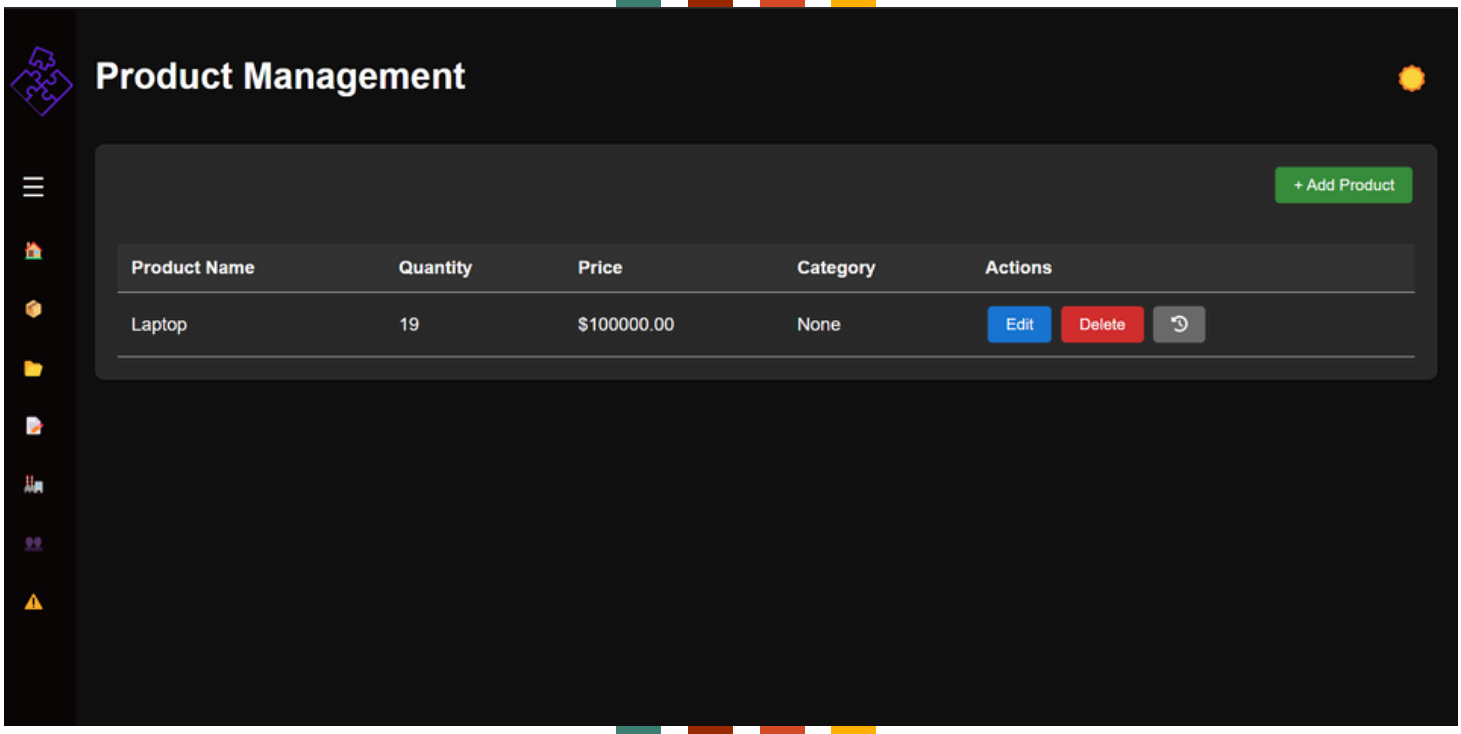
Screenshots:

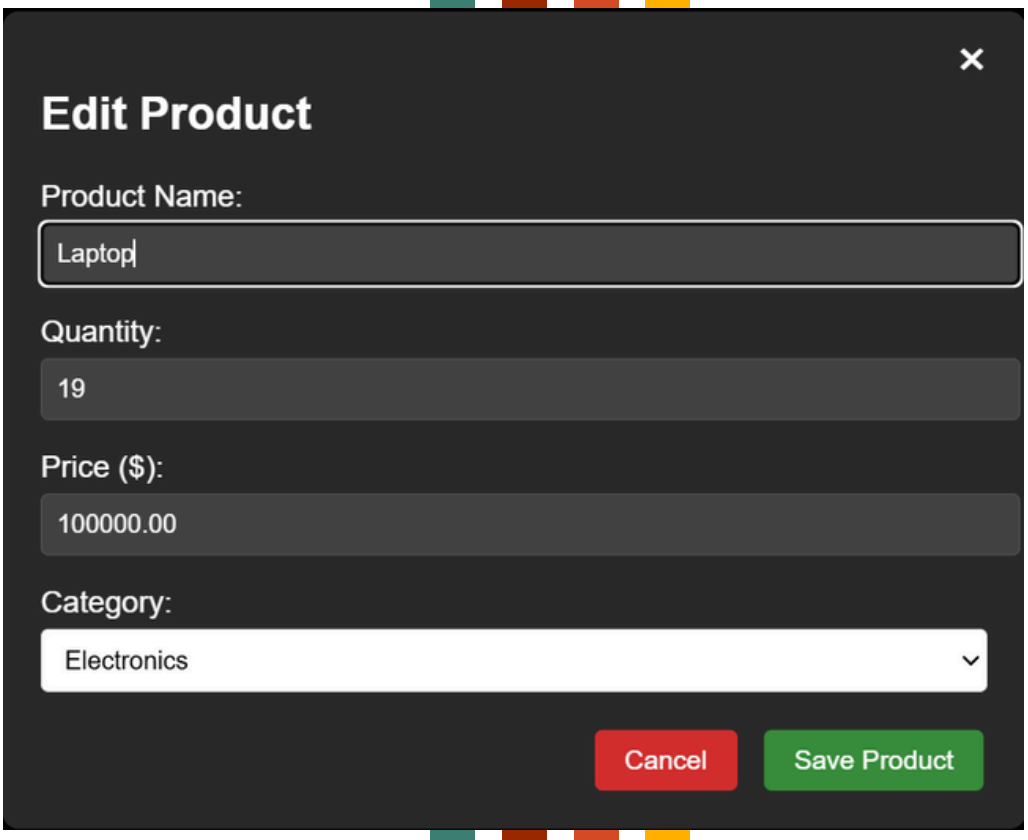





Dashboard
Total Stock
19
Total Customers
1
Total Sales
$1900000.00
Total Categories
1
Stock Quantity
20
15
10
5
0
Kshitij
Sales Over Time
1900001.0
1900000.5
1900000.0
Electronics


STOCKEZ
Dashboard
Total Stock
19
Total Customers
1
Total Sales
$1900000.00
Total Categories
1
Overview
Products
Categories
Orders
Vendors
Customers
Alerts
Stock Quantity
20
15
10
5
0
Kshitij
Sales Over Time
1900001.0
1900000.5
1900000.0
Electronics


Product Management
+ Add Product
Product Name
Quantity
Price
Category
Actions
Laptop
19
$100000.00
None
Edit
Delete


Edit Product
Product Name:
Laptop
Quantity:
19
Price ($):
100000.00
Category:
Electronics
Cancel
Save Product

Product History

5/1/2025, 10:56:43 PM

Type: Product Updated

Details: **name:** Laptop
quantity: 19
price: 100000
category_id: 1

4/30/2025, 4:06:26 AM

Type: Order Deleted - Refunded

Details: Refunded quantity 14 for deleted order 11

4/30/2025, 3:35:56 AM

Type: Product Updated

Details: **name:** A115

Conclusion:

StockEz (“Inventory Management System”) delivers a comprehensive, user-friendly platform for managing products, orders, customers, categories, and vendors in a single, responsive web application. Built on a vanilla JavaScript front end and a Node.js/Express + PostgreSQL back end, StockEz implements:

- Real-time Inventory Control: Automatic stock updates on order creation, editing, and deletion, with low-stock alerts to trigger vendor replenishment.
- Product & Order History: Detailed audit trails for product changes and order edits, stored in a dedicated history table.
- Customer & Vendor Management: Creation and editing workflows with API-driven email validation to ensure correct contact information.
- Dark/Light Themes & Responsive Layout: A sidebar that toggles between collapsed/expanded modes, adaptive forms, modals, and table designs that maintain usability across devices.
- Security & Multi-User Support: Per-user data isolation via x-user-id headers, bcrypt-hashed passwords, and signup/login flows with email verification codes.
- Extensible Architecture: Modular API endpoints, consistent SQL table creation on startup, and clear separation of concerns ready for future microservices or serverless migrations.

While StockEz currently uses PostgreSQL on Render and CORS-enabled REST APIs, it lays the groundwork for future enhancements such as:

- Advanced role-based authentication (JWT/OAuth2)
- Cloud file storage for product media
- Interactive reporting dashboards
- Notification integrations (email/SMS)
- Bulk import/export and audit dashboards

Overall, StockEz transforms manual inventory and order workflows into a streamlined digital system, fostering accuracy, traceability, and collaboration among administrators, warehouse staff, and vendors—setting the stage for scalable growth and enterprise-level deployment.