SmartStock System Design Document

EECS 3311 — Sprint 2

Group: Project JUME (Group 9)

Date: March 26, 2025

Table of Contents

---

1. Introduction

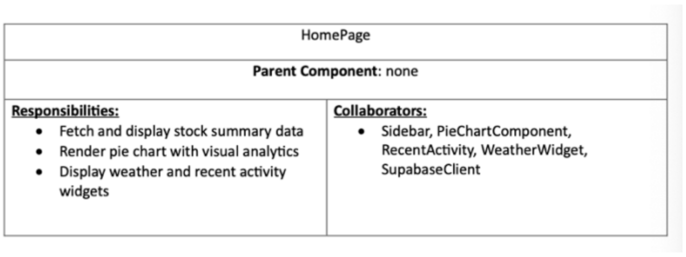
SmartStock is a warehouse inventory and stock management system developed using Next.js (React), Tailwind CSS, Supabase (PostgreSQL), and Recharts. The goal is to provide a modern, responsive UI for users to manage products, view orders, track recent activity, and analyze inventory insights through visual dashboards.

The platform is designed for warehouse staff, administrators, and managers who need a fast, visual, and user-friendly way to keep track of real-time inventory operations. SmartStock offers support for viewing order statuses, managing product entries, tracking changes made to the system, and viewing graphical summaries of fulfillment trends and inventory levels. The user interface is accessible across different devices and adapts to role-based interactions.

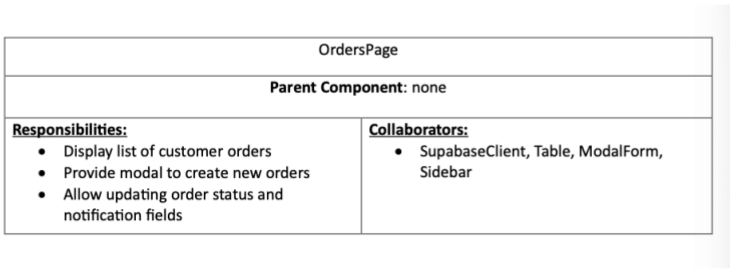
This Sprint focused on completing core UI and backend functionalities including order management, dynamic dashboard integration with Supabase, real-time updates, and visual analytics.

2. CRC Cards

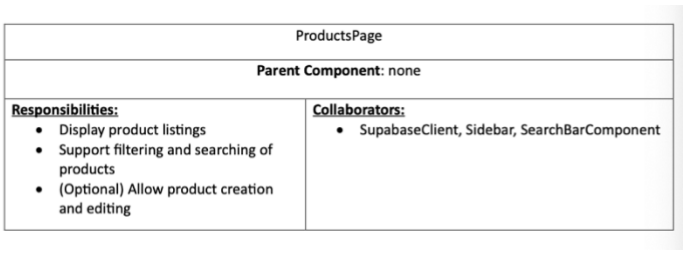
2.1 Frontend Components



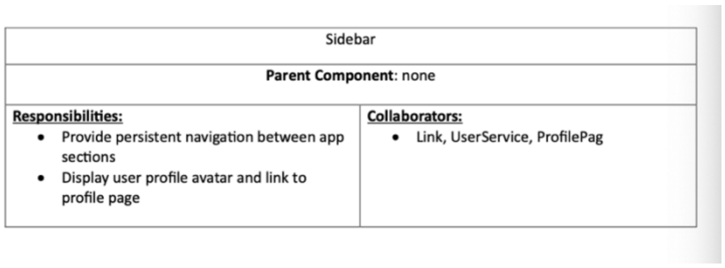
HomePage	
Parent Component: none	
Responsibilities: • Fetch and display stock summary data • Render pie chart with visual analytics • Display weather and recent activity widgets	**Collaborators:** • Sidebar, PieChartComponent, RecentActivity, WeatherWidget, SupabaseClient



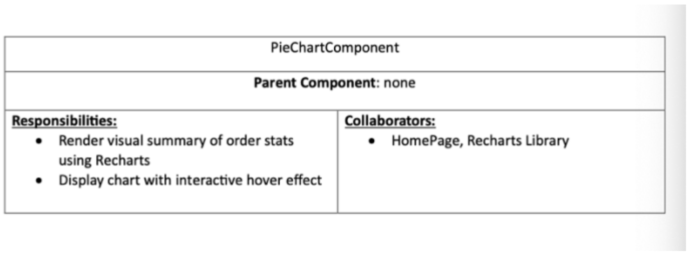
OrdersPage	
Parent Component: none	
Responsibilities: • Display list of customer orders • Provide modal to create new orders • Allow updating order status and notification fields	**Collaborators:** • SupabaseClient, Table, ModalForm, Sidebar



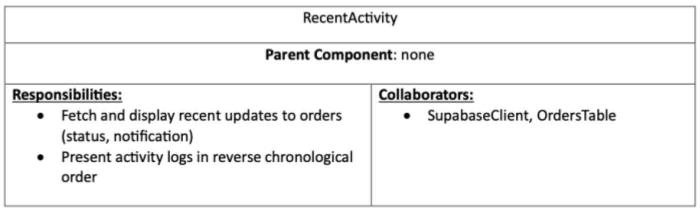
ProductsPage	
Parent Component: none	
Responsibilities: • Display product listings • Support filtering and searching of products • (Optional) Allow product creation and editing	**Collaborators:** • SupabaseClient, Sidebar, SearchBarComponent



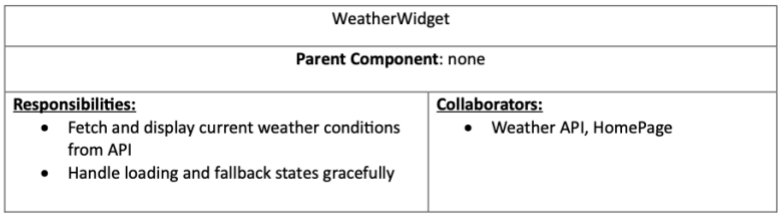
Sidebar	
Parent Component: none	
Responsibilities: • Provide persistent navigation between app sections • Display user profile avatar and link to profile page	**Collaborators:** • Link, UserService, ProfilePag



PieChartComponent	
Parent Component: none	
Responsibilities: • Render visual summary of order stats using Recharts • Display chart with interactive hover effect	**Collaborators:** • HomePage, Recharts Library



RecentActivity	
Parent Component: none	
Responsibilities: • Fetch and display recent updates to orders (status, notification) • Present activity logs in reverse chronological order	**Collaborators:** • SupabaseClient, OrdersTable



WeatherWidget	
Parent Component: none	
Responsibilities: • Fetch and display current weather conditions from API • Handle loading and fallback states gracefully	**Collaborators:** • Weather API, HomePage

2.2 Backend/Service Layer

SupabaseClient	
Parent Component: none	
Responsibilities: • Interface with Supabase for all data fetching and mutations • Handle authentication and table queries	**Collaborators:** • HomePage, OrdersPage, RecentActivity, ProductsPage, UserService

OrderService	
Parent Component: none	
Responsibilities: • Insert and update orders in the orders table • Retrieve order data based on filters or sort parameters	**Collaborators:** • SupabaseClient, OrdersPage, RecentActivity

ProductService	
Parent Component: none	
Responsibilities: • Fetch product records from the products table • (Optional) Create or update product entries	**Collaborators:** • SupabaseClient, ProductsPage

UserService	
Parent Component: none	
Responsibilities: • Retrieve user information including name and avatar • Handle profile logic and role validation	**Collaborators:** • Supabase Auth, Sidebar, ProfilePage

Database (Main connection)	
Parent Class: Supabase/PostgreSQL **Subclasses:** users, orgs, org_members, customers, products, orders, product_restock_notifications	
Responsibilities: • Store and manage structured data (notifications, users, products, orders, customers, etc.) • Provide centralized access for all CRUD operations • Enforce security via RLS (Row-Level Security) policies • Enable support functions (e.g. moddatetime, UUID generation, inventory triggers) • Enable scheduled jobs and helper functions via extensions	**Collaborators:** • Supabase Client SDK: Executes CRUD and auth logic • Supabase Auth: Triggers user sync into `users` table • Triggers & Policies: Handle data integrity and access control

Users	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Store user profile data synced from Supabase Auth • Provide personal info like name, email, avatar • Enforce RLS policies for per-user access control	**Collaborators:** • auth.users: Supabase auth table that triggers insert • public.handle_new_user(): Function syncing auth metadata • policy.users_*: RLS functions/policies

orgs	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Represent organizations with address, contact, and financial data • Define org-wide data separation (for multitenancy) • Provide org identifiers for other tables (e.g., orders, products)	**Collaborators:** • org_members: Links users to orgs and defines roles • policy.orgs_*: RLS functions for view, update, delete

orgs_members	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Link users to orgs with roles (`admin`, `manager`, `staff`) • Enforce permissions based on roles (RLS policies) • Track membership history (with timestamps)	**Collaborators:** • users: Member identity • orgs: Organization reference • policy.org_members_*: Access control rules

customers	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Represent customers of an org (contact + address info) • Serve as a link in the org–customer–order–product chain • Support role-based RLS access per organization	**Collaborators:** • orgs: Parent org • orders: Reference customers in transactions • policy.customers_*: Insert/update/delete rules

products	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Store product metadata (name, price, stock, etc.) • Monitor inventory levels and trigger notifications • Track ownership (org/customer), category, SKU	**Collaborators:** • customers: Product recipient • product_restock_notifications: Trigger on low inventory • policy.products_*: Access and role-based controls

orders	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Record customer purchases with associated products • Store totals, fulfillment status, and notification status • Support full CRUD and RLS per org & role	**Collaborators:** • products: Product reference • customers: Customer placing the order • policy.orders_*: Access control logic

Product_restock_notifications	
Parent Class: SQL Table **Subclasses:** none	
Responsibilities: • Store alerts when product stock drops below a threshold • Avoid duplicate notifications for the same product • Allow only authorized org members to view/delete alerts	**Collaborators:** • products: Triggers insert via handle_product_restock_notification • policy.product_restock_notifications_*: Access control logic

3. Software Architecture

The SmartStock system follows a **client-server MVC architecture**:

• **Frontend (Client):** Built with Next.js and Tailwind. It handles UI rendering, page routing, and component logic.

• **Backend (Supabase):** Acts as a database layer (PostgreSQL) and provides API endpoints for authentication and data operations.

• **Charting Engine:** Recharts (client-side) is used for rendering visual data analytics.

Data Flow Diagram:

1. User logs in via Supabase Auth.

2. Sidebar loads navigation and profile data.

3. HomePage fetches summary stats and renders UI.

4. OrdersPage connects to Supabase to fetch and manipulate order data.

5. Real-time updates (like status change) appear in RecentActivity.

6. WeatherWidget pulls data from external API.

4. Error Handling Strategy

Frontend (Client-Side)

• **Error Boundaries**: React-based error boundaries are used to catch rendering exceptions in isolated components, preventing complete UI crashes.

• **Toast Notifications & Alerts**: All asynchronous operations (e.g., form submissions, data fetches) provide instant visual feedback via toasts, ensuring users are aware of both successful and failed actions.

• **Form Validation**: Client-side form inputs are validated using controlled components with dynamic error messages before API requests are made, reducing unnecessary backend load.

• **Fallback UI Rendering**: Conditional rendering strategies display appropriate placeholders or error cards in components such as charts, activity feeds, and weather widgets if data fails to load or is malformed.

- **Retry Logic**: Lightweight retry mechanisms are applied for transient fetch failures, particularly for third-party APIs like the weather service.

Backend (Supabase & API Layer)

- **Supabase Response Guarding**: All database operations check for .error fields in returned objects. These errors are logged to the console and surfaced to the UI in a user-friendly format.

- **Role-based Access Control Errors**: Backend queries are scoped to authenticated users. Unauthorized access attempts return detailed access errors with appropriate logging.

- **Data Schema Enforcement**: Supabase schema constraints, such as non-null checks, enum validations, and foreign key relations, ensure that malformed data never enters the system.

- **API Status Monitoring (Optional)**: Integration with services like UptimeRobot or Supabase logs ensures uptime and alerts for backend failures (planned future enhancement).

- **Rate Limiting and Abuse Handling (Planned)**: To prevent spam or malicious usage, the system is designed to later implement rate limiting on critical endpoints.

---

5. Conclusion

The second sprint of SmartStock marks a significant milestone in the development of a robust, responsive, and data-driven inventory management system tailored for warehouse and operations environments. With the successful integration of Supabase for real-time data handling, the application now supports dynamic order management, comprehensive product listings, and visually engaging dashboards.

Key components such as the recent activity log, weather widget, and order fulfillment analytics not only enhance operational visibility but also lay the groundwork for future intelligent insights and predictive inventory planning. The modular architecture, built on modern web technologies, ensures the system remains scalable and adaptable as new features are introduced.

By focusing on user-centric design, real-time responsiveness, and clean visual presentation, SmartStock continues to evolve into a professional-grade platform that balances usability with powerful backend capabilities. The foundation set in Sprint 2 positions the team to confidently move forward with advanced features such as role-based dashboards, detailed reporting, and notification systems in future sprints.

Prepared by: Team JUME — Usman, Max, Jay, Erfan