

Capstone II Project Documentation

Enzo Gomes, Michael Lopez, Gabriela Hernandez, Aiden Beaton, Alexander Soto

1. Executive Summary

Pantry Sync is a full-stack web application designed to improve the management of campus food pantry operations. The system provides tools for inventory tracking, food ordering, pickup scheduling, and role-based dashboards for different users.

The platform addresses inefficiencies caused by manual systems such as spreadsheets and informal coordination methods. By centralizing inventory management, order workflows, and operational analytics, Pantry Sync improves visibility and efficiency across pantry operations.

The system follows a client-server architecture, where a React frontend communicates with a Node.js Express backend. Data is stored in a PostgreSQL database, and Supabase is used for authentication through JWT-based access control. This architecture enables secure, scalable, and maintainable application behavior.

2. Problem & Requirements

2.1 Problem Description

Campus food pantries often rely on manual processes to manage inventory and coordinate food distribution. These processes can lead to inaccurate inventory tracking, inefficient workflows, and limited visibility into pantry operations.

Pantry Sync addresses these challenges by providing a structured digital system that supports real-time inventory tracking, order management, and improved coordination between users and staff.

2.2 Stakeholders

- **Students (Users):** Place food orders and schedule pickups
- **Volunteers (Team):** Manage inventory and assist pantry operations
- **Administrators:** Manage users, monitor activity, and analyze system data

2.3 Functional Requirements

- User authentication and registration
- Role-based access control
- Inventory management (create, update, delete items)
- Food order creation and tracking
- Pickup scheduling
- Alerts for low-stock and expiring items
- Analytics and reporting dashboards

2.4 Non-Functional Requirements

- Secure authentication using JWT
- Reliable and consistent data storage
- Scalable architecture
- User-friendly interface

3. System Overview & Architecture

3.1 Architecture Description

Pantry Sync follows a three-tier client–server architecture:

- **Frontend:** React with TypeScript, Vite, and TailwindCSS
- **Backend:** Node.js with Express REST API
- **Database:** PostgreSQL accessed via Drizzle ORM
- **Authentication:** Supabase (JWT-based)

3.2 Data Flow

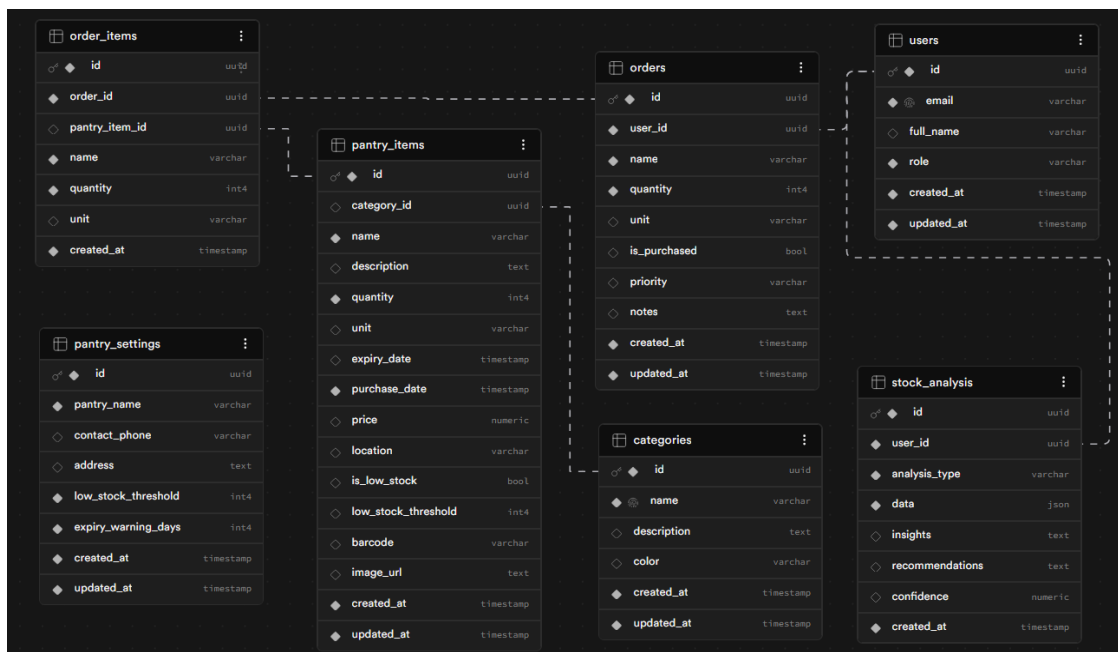
1. User logs in through Supabase authentication
2. Frontend sends API requests with JWT token
3. Backend validates token and processes the request
4. Database queries are executed
5. Results are returned to the frontend

3.3 Tech Stack

Layer	Technology	Its Purpose
Frontend	React, TypeScript, Vite, TailwindCSS	UI and client interaction
Backend	Node.js, Express	API and business logic
Database	PostgreSQL, Supabase	Data storage
ORM	Drizzle ORM	Simpler database queries
Authentication	Supabase	User authentication

4. System Design

4.1 Database Schema



Pantry Sync uses a relational database structure to manage application data. The main tables include:

- **users**: Stores user information and roles
- **pantry_items**: Tracks inventory items and quantities
- **orders**: Stores food orders placed by users
- **order_items**: Links inventory items to orders
- **categories**: Organizes inventory items
- **pantry_settings**: Stores system configuration
- **stock_analysis**: Supports future analytics functionality

4.2 Entity Relationships

- Users can place multiple orders
- Orders can contain multiple pantry items
- Pantry items belong to categories
- Orders are linked to users and inventory items

This structure ensures consistent and organized data flow across the system.

5. Implementation Details

5.1 Frontend Implementation

The frontend is built as a React single-page application using React Router for navigation. It provides dashboards tailored to different user roles and communicates with backend APIs using authenticated requests.

5.2 Backend Implementation

The backend is implemented using Express and provides RESTful API endpoints for managing users, inventory, orders, analytics, and settings.

Role-based access control is enforced through middleware to restrict access to specific routes.

5.3 API Design

The application uses REST APIs structured around resources such as:

```
/api/auth for authentication  
/api/pantry for inventory and orders  
/api/user for user management  
/api/settings for configuration
```

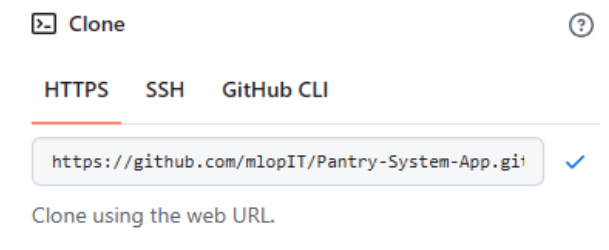
5.4 Key Design Decisions & Trade-offs

- Use of Supabase for authentication instead of building custom auth
- REST architecture for simplicity and maintainability
- Separation of frontend and backend for scalability
- Reuse of certain database fields for simplified schema design

6. Setup and Execution Instructions

Steps to Run the System

6.1. Clone the repository



```
https://github.com/mlopIT/Pantry-System-App.git
```

6.2. Install dependencies

cd backend && cd frontend:

```
npm install
```

6.3. Set up .env environment for backend && frontend

.env backend:

```
SUPABASE_URL=https:*****  
  
# Database Configuration pooler  
DATABASE_URL=postgresql:*****  
  
# JWT Secret Supabase  
JWT_SECRET=*****
```

.env frontend:

```
VITE_SUPABASE_URL=https:*****  
VITE_SUPABASE_ANON_KEY=*****  
VITE_API_URL=http:*****
```

6.4. Run application

cd backend && cd frontend:

```
npm run dev
```

6.5. Access application on localhost

```
Frontend: http://localhost:3000  
Backend: http://localhost:5000
```

7. Security, and Privacy/Readability Measures Taken

- JWT-based authentication using Supabase
- Role-based authorization middleware
- Secure API access via Bearer tokens
- Basic input validation on critical routes
- Error handling with fallback responses

8. Testing & Evaluation

The system was tested through manual end-to-end workflows, including:

- User authentication
- Inventory management operations
- Order creation and tracking
- Pickup scheduling
- Role-based access validation
- Alerts and analytics features

No automated testing framework is currently implemented.

9. Results

Pantry Sync demonstrates a fully functional system capable of managing pantry operations efficiently.

The system provides:

- Centralized inventory management
- Structured order workflows
- Role-based dashboards
- Automated alerts
- Analytics for operational insights

10. Limitations & Future Work

Limitations

- No automated testing
- No CI/CD pipeline
- Some database tables not fully utilized
- No multi-tenant support
- Missing truly real-time updates

Future Work

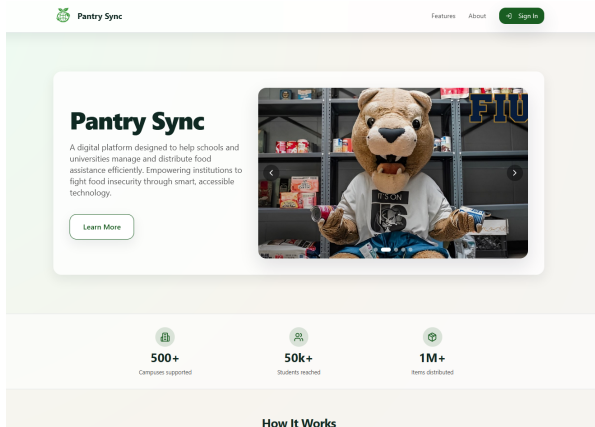
- Implement automated testing
- Add CI/CD pipeline
- Expand analytics capabilities
- Support multiple pantry locations
- Add real-time system updates
- Integrate AI-based analytics

11. Lessons Learned

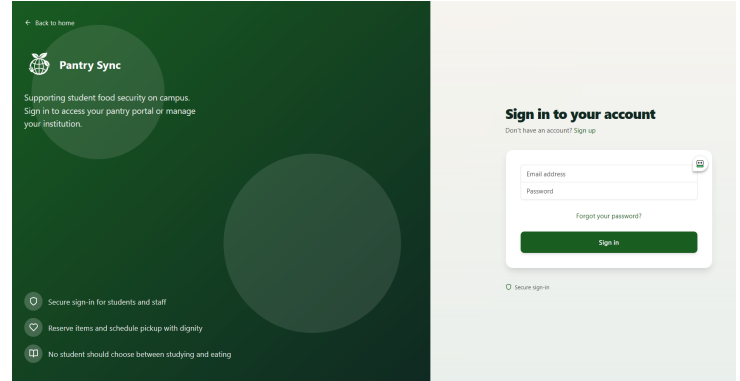
- Importance of clear database design
- Challenges of integrating authentication with authorization
- Need for consistent role-based access control
- Benefits of modular API design
- Importance of testing across user roles

12. System Main Screens

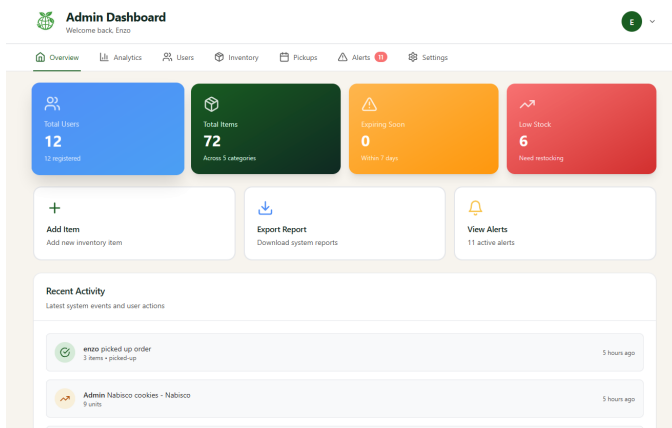
System landing page (loads first):



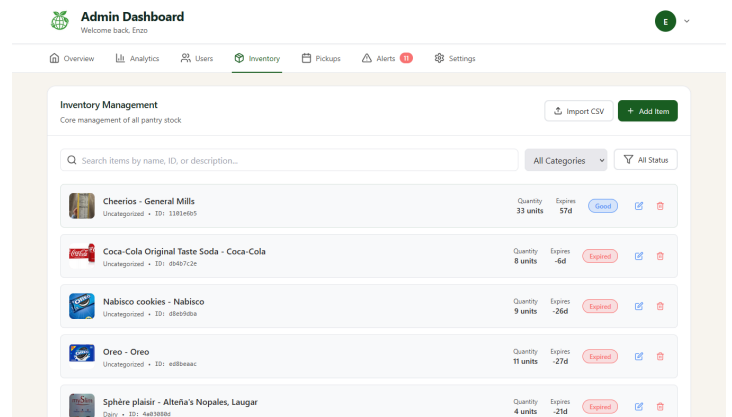
Log in page:



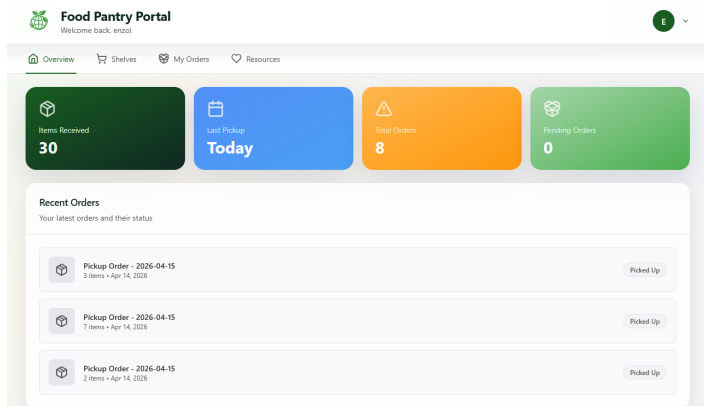
Admin dashboard:



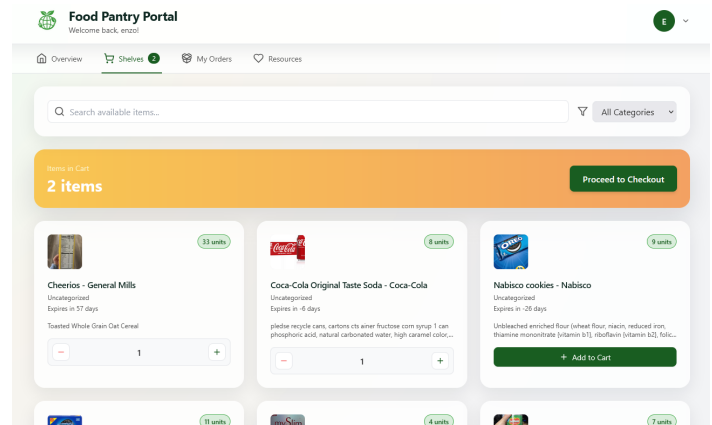
Pantry Inventory:



User Dashboard:



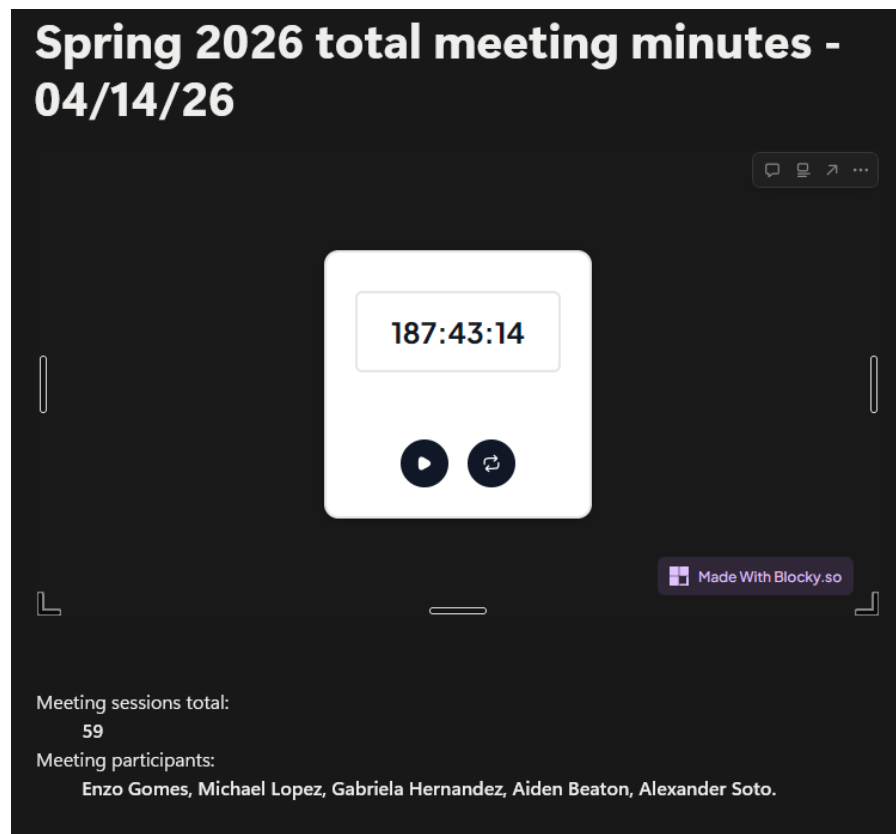
Order page:



14. Daily Scrum Summary

Daily Scrum meetings were conducted throughout the project to track progress, identify blockers, and coordinate tasks. A total of **59** meetings took place **Mon-Fri** since January, with an estimated average session being **3 hours**.

These meetings included updates on completed work, upcoming tasks, and issues encountered during development. And were logged into the same document totaling **187:43:14h**.



15. Additional Supporting Materials

- Poster presentation
- System demonstration videos
- Installation and setup instructions
- Future work and limitations documentation

16. Documentation used

Supabase Documentation - <https://supabase.com/docs>

React Documentation - <https://react.dev/>

Node.js Documentation - <https://nodejs.org/docs>

Express Framework - <https://expressjs.com/>

PostgreSQL Documentation - <https://www.postgresql.org/docs>

Drizzle ORM - <https://orm.drizzle.team/>

JWT Specification - <https://datatracker.ietf.org/doc/html/rfc7519>

Vite Documentation - <https://vitejs.dev/>