Baña, Gyan Rhanzell Edwin

**CSIPT Final Project**

**Sales and Inventory Management System Documentation**

**Table of Contents**

1. **Overview**
2. **Data Models**
3. **API Endpoints**
4. **Serializers**
5. **ViewSets**
6. **Business Logic**
7. **API Client**
8. **State Management**
9. **Frontend Components**
10. **Overview**

The Sales and Inventory Management System is a full-stack application that helps food businesses manage their ingredients, recipes, production, and sales. The system consists of:

1. A Django-based REST API backend for data management
2. A React-based frontend for user interaction

The system allows for tracking inventory, planning production, recording sales, and generating financial reports.

1. **Data Models**

**Ingredient**

Represents raw materials used in recipes.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **name** | CharField | Unique ingredient name |
| **quantity** | FloatField | Available quantity |
| **unit** | CharField | Unit of measurement (e.g., kg, g, ml) |
| **min\_threshold** | FloatField | Minimum threshold for low stock warning |
| **cost\_per\_unit** | DecimalField | Cost per unit of the ingredient |

**Properties:**

* is\_low\_stock: Boolean indicating if quantity is at or below min\_threshold

**Recipe**

Represents a specific formula for a bakery item.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **name** | CharField | Unique recipe name |
| **instructions** | TextField | Preparation instructions |
| **preparation\_time** | IntegerField | Time in minutes to prepare |
| **ingredients** | ManyToManyField | Related ingredients through RecipeIngredient |
| **image** | ImageField | Optional recipe image |
| **prepared\_quantity** | FloatField | Current quantity prepared/available |

**Properties:**

* can\_make: Boolean indicating if recipe can be made with current ingredients
* max\_portions: Maximum portions that can be made with available ingredients
* cost: Total cost to make one unit of this recipe

**RecipeIngredient**

Links recipes and ingredients with specific quantities.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **recipe** | ForeignKey | Related recipe |
| **ingredient** | ForeignKey | Related ingredient |
| **quantity** | FloatField | Amount of ingredient needed |

**ProductionRecord**

Records when a recipe is prepared for sale.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **recipe** | ForeignKey | Recipe that was produced |
| **quantity** | FloatField | Quantity produced |
| **timestamp** | DateTimeField | When production occurred |
| **notes** | TextField | Optional production notes |

**Product**

Represents a menu item for sale.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **recipe** | ForeignKey | Recipe used to make the product |
| **name** | CharField | Product display name |
| **price** | DecimalField | Selling price |
| **is\_active** | BooleanField | Whether product is currently for sale |
| **created\_at** | DateTimeField | When product was added to menu |

**Properties:**

* cost: Production cost (from recipe)
* profit: Profit per unit (price - cost)
* profit\_margin: Profit as percentage of price
* prepared\_quantity: Current available quantity (from recipe)

**Sale**

Records individual sales transactions.

| **Field** | **Type** | **Description** |
| --- | --- | --- |
| **product** | ForeignKey | Product sold |
| **quantity** | IntegerField | Quantity sold |
| **unit\_price** | DecimalField | Price at time of sale |
| **timestamp** | DateTimeField | When sale occurred |

**Properties:**

* total\_price: Total price for this sale (quantity \* unit\_price)
* profit: Total profit for this sale (quantity \* unit\_profit)

1. **API Endpoints**

**Ingredients**

* GET /api/ingredients/ - List all ingredients
* POST /api/ingredients/ - Create new ingredient
* GET /api/ingredients/{id}/ - Get ingredient details
* PUT/PATCH /api/ingredients/{id}/ - Update ingredient
* DELETE /api/ingredients/{id}/ - Delete ingredient
* POST /api/ingredients/{id}/restock/ - Add to ingredient quantity

**Recipes**

* GET /api/recipes/ - List all recipes
* POST /api/recipes/ - Create new recipe
* GET /api/recipes/{id}/ - Get recipe details
* PUT/PATCH /api/recipes/{id}/ - Update recipe
* DELETE /api/recipes/{id}/ - Delete recipe
* POST /api/recipes/{id}/prepare/ - Prepare a recipe (consumes ingredients)

**Recipe Ingredients**

* GET /api/recipe-ingredients/ - List all recipe ingredients
* POST /api/recipe-ingredients/ - Create new recipe ingredient
* GET /api/recipe-ingredients/{id}/ - Get recipe ingredient details
* PUT/PATCH /api/recipe-ingredients/{id}/ - Update recipe ingredient
* DELETE /api/recipe-ingredients/{id}/ - Delete recipe ingredient

**Products**

* GET /api/products/ - List all products
* POST /api/products/ - Create new product
* GET /api/products/{id}/ - Get product details
* PUT/PATCH /api/products/{id}/ - Update product
* DELETE /api/products/{id}/ - Delete product

**Sales**

* GET /api/sales/ - List all sales
* POST /api/sales/ - Record a new sale
* GET /api/sales/{id}/ - Get sale details
* PUT/PATCH /api/sales/{id}/ - Update sale
* DELETE /api/sales/{id}/ - Delete sale
* GET /api/sales/report/ - Generate sales report
* GET /api/sales/dashboard/ - Get dashboard metrics

1. **Serializers**

**IngredientSerializer**

Handles ingredient data serialization and validation.

* Model: Ingredient
* Fields: id, name, quantity, unit, min\_threshold, cost\_per\_unit, is\_low\_stock
* Notes: Includes the calculated is\_low\_stock field to flag ingredients with quantity below threshold

**RecipeIngredientSerialize**r

Manages the relationship between recipes and their ingredients.

* Model: RecipeIngredient
* Fields: id, ingredient, ingredient\_name, ingredient\_unit, quantity
* Read-only fields: ingredient\_name, ingredient\_unit
* Notes: Provides ingredient details for better readability in API responses

**RecipeSerializer**

Handles complex recipe data with nested ingredients.

* Model: Recipe
* Fields: id, name, instructions, preparation\_time, image, ingredients\_detail, recipe\_ingredients, can\_make, max\_portions, cost, prepared\_quantity, cost\_per\_serving
* Read-only fields: ingredients\_detail, cost\_per\_serving, can\_make, max\_portions, cost, prepared\_quantity
* Write-only fields: recipe\_ingredients
* Special handling:
  + Manages nested creation and updates for recipe ingredients
  + Calculates recipe costs and availability based on current inventory
  + Provides ingredient details within recipe responses

**ProductionRecordSerializer**

Tracks recipe production events.

* Model: ProductionRecord
* Fields: id, recipe, recipe\_name, quantity, timestamp, notes
* Read-only fields: recipe\_name

**ProductSerializer**

Manages product data with profitability metrics.

* Model: Product
* Fields: id, name, recipe, recipe\_name, price, is\_active, cost, profit, profit\_margin, prepared\_quantity, created\_at
* Read-only fields: recipe\_name, cost, profit, profit\_margin, prepared\_quantity
* Special handling:
  + Enhances response with calculated profit metrics
  + Includes prepared quantity from associated recipe for inventory tracking

**SaleSerializer**

Handles sales transactions with extended profit information.

* Model: Sale
* Fields: id, product, product\_name, quantity, unit\_price, total\_price, profit, timestamp
* Read-only fields: product\_name, total\_price, profit, timestamp
* Validation: Ensures product is provided when creating a sale

1. **ViewSets**

**IngredientViewSet**

Manages ingredient CRUD operations and restocking.

* Model: Ingredient
* Serializer: IngredientSerializer
* Custom actions:
  + restock: POST endpoint to add specified amount to ingredient quantity
  + Validates that restocking amount is positive

**RecipeViewSet**

Handles recipe management and production operations.

* Model: Recipe
* Serializer: RecipeSerializer
* Custom actions:
  + prepare: POST endpoint to create production records and update inventory
  + Performs validation to ensure sufficient ingredients are available
  + Uses database transactions to ensure data consistency during preparation
  + Updates prepared quantity on successful production

**RecipeIngredientViewSet**

Basic CRUD operations for recipe ingredient relationships.

* Model: RecipeIngredient
* Serializer: RecipeIngredientSerializer

**ProductViewSet**

Manages bakery products for sale.

* Model: Product
* Serializer: ProductSerializer
* Notes: Enhanced queryset with logging for monitoring prepared quantities

**SaleViewSet**

Handles sales transactions and reporting.

* Model: Sale
* Serializer: SaleSerializer
* Custom actions:
  + report: GET endpoint generating sales and profit reports by time period
    - Supports daily, weekly, and monthly aggregation
    - Calculates revenue, cost, profit, and profit margin
    - Takes start and end date parameters for custom date ranges
  + dashboard: GET endpoint providing key metrics for current day and week
    - Calculates today's revenue, profit, margin, and transaction count
    - Provides weekly metrics for comparison
    - Includes daily chart data for the past 7 days with sales and profit metrics

1. **Business Logic**

**Inventory Management**

* When ingredients are below their minimum threshold (min\_threshold), they are flagged as low stock.
* Ingredients can be restocked using the restock endpoint.

**Production Process**

1. Check if there are enough ingredients to make the desired quantity of a recipe.
2. If possible, deduct ingredients from inventory.
3. Increase the prepared quantity of the recipe.
4. Create a production record.

**Sales Process**

1. Check if there's enough prepared quantity of the recipe to fulfill the sale.
2. If possible, deduct from the prepared quantity.
3. Record the sale with current pricing.

**Financial Tracking**

* Each recipe tracks its cost based on ingredient costs.
* Each product tracks its profit based on recipe cost and selling price.
* Sales reports aggregate financial data over time periods.

**Error Handling**

* Attempting to prepare recipes with insufficient ingredients returns a 400 error.
* Attempting to sell products with insufficient prepared quantity returns a 400 error.
* Invalid input formats return appropriate error responses.

**Data Consistency**

* Database transactions ensure that related operations succeed or fail together.
* Prepared quantities are tracked and updated when production or sales occur.

Frontend Integration

1. **API Client**

The frontend communicates with the backend through a centralized API client (api.js) built on Axios. This provides consistent access to all endpoints:

**// Example imports from api.js**

import \* as api from "../api/api";

**// Usage**

const ingredients = await api.getIngredients();

**Available API Functions**

| **Function** | **Description** | **Parameters** |
| --- | --- | --- |
| **Ingredients** |  |  |
| **getIngredients()** | Fetch all ingredients | None |
| **getIngredient(id)** | Fetch single ingredient | id: Ingredient ID |
| **createIngredient(data)** | Create new ingredient | data: Ingredient object |
| **updateIngredient(id, data)** | Update ingredient | id: Ingredient ID, data: Updated fields |
| **deleteIngredient(id)** | Delete ingredient | id: Ingredient ID |
| **restockIngredient(id, amount)** | Add stock to ingredient | id: Ingredient ID, amount: Quantity to add |
| **Recipes** |  |  |
| **getRecipes()** | Fetch all recipes | None |
| **getRecipe(id)** | Fetch single recipe | id: Recipe ID |
| **createRecipe(data)** | Create new recipe | data: Recipe object |
| **updateRecipe(id, data)** | Update recipe | id: Recipe ID, data: Updated fields |
| **deleteRecipe(id)** | Delete recipe | id: Recipe ID |
| **prepareRecipe(id, quantity, notes)** | Prepare a recipe | id: Recipe ID, quantity: Amount to make, notes: Production notes |
| **Products** |  |  |
| **getProducts()** | Fetch all products | None |
| **getProduct(id)** | Fetch single product | id: Product ID |
| **createProduct(data)** | Create new product | data: Product object |
| **updateProduct(id, data)** | Update product | id: Product ID, data: Updated fields |
| **deleteProduct(id)** | Delete product | id: Product ID |
| **Sales** |  |  |
| **getSales()** | Fetch all sales | None |
| **createSale(data)** | Record new sale | data: Sale object with product, quantity, unit\_price |
| **getSaleReport(period, startDate, endDate)** | Generate sales report | period: Aggregation period, startDate: Start date, endDate: End date |
| **getDashboardData()** | Fetch dashboard metrics | None |
| **Recipe Ingredients** |  |  |
| **getRecipeIngredients()** | Fetch all recipe ingredients | None |
| **getRecipeIngredient(id)** | Fetch single recipe ingredient | id: Recipe ingredient ID |
| **createRecipeIngredient(data)** | Create new recipe ingredient | data: Recipe ingredient object |
| **updateRecipeIngredient(id, data)** | Update recipe ingredient | id: Recipe ingredient ID, data: Updated fields |
| **deleteRecipeIngredient(id)** | Delete recipe ingredient | id: Recipe ingredient ID |

1. **State Management**

The application uses React Context API for global state management:

// Example usage of AppContext

import { useAppContext } from "../context/AppContext";

function MyComponent() {

const { ingredients, recipes, products, loading, error, refreshData } = useAppContext();

// Use state and functions from context

}

**AppContext Provider**

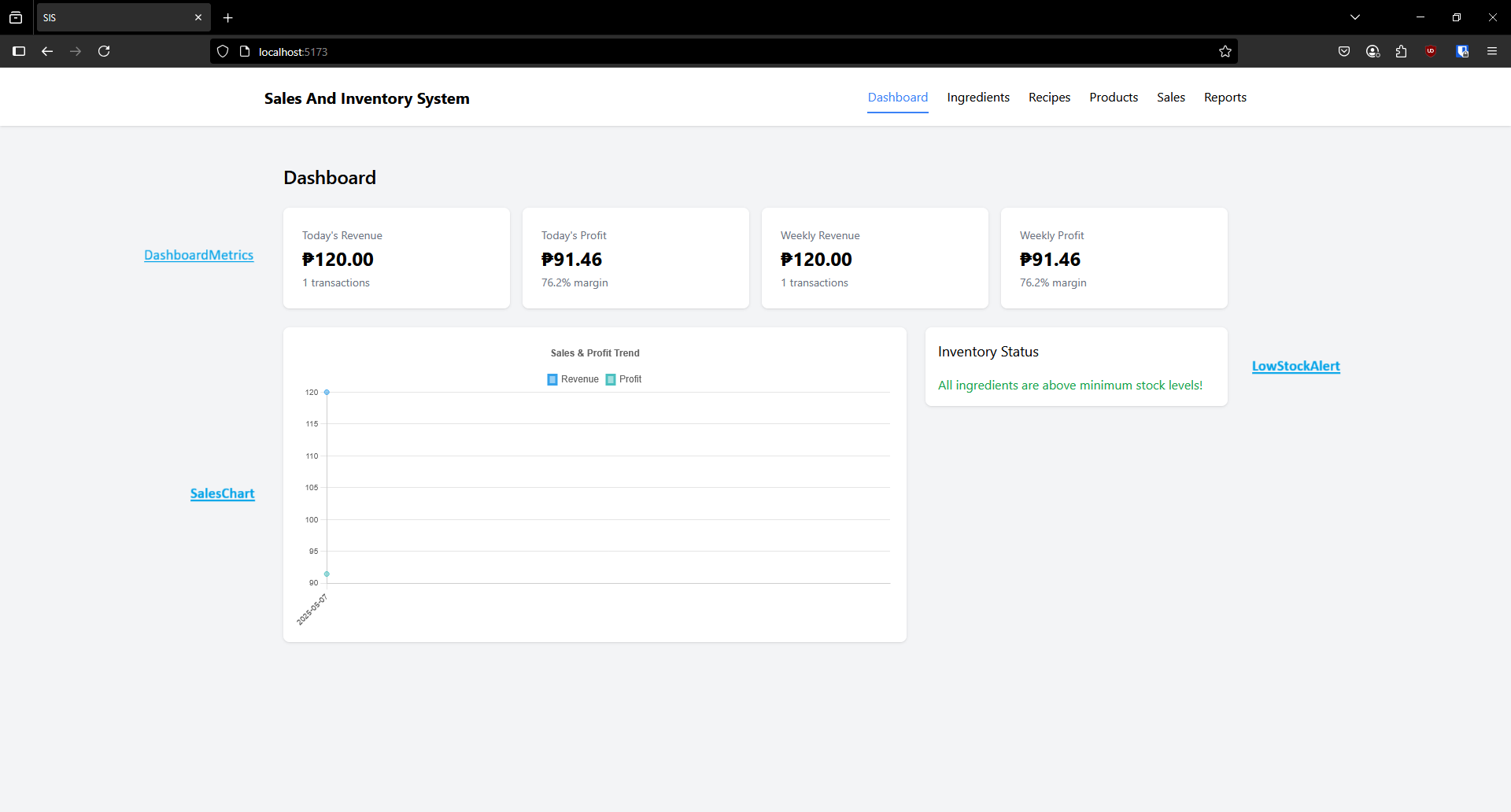
The AppProvider component manages centralized application state:

* State:
  + ingredients: Array of all ingredients
  + recipes: Array of all recipes
  + products: Array of all products
  + loading: Boolean indicating if data is being fetched
  + loaded: Boolean indicating if data has been successfully loaded
  + error: Error message if data fetching failed
* Functions:
  + refreshData(): Refetches all data from the API

The context automatically loads all essential data when the application initializes, making it available throughout the component tree.

1. **Frontend Components**

**Dashboard Components**

****

**DashboardMetrics**

This component displays key business metrics in card format including daily and weekly revenue, profit, transactions, and profit margins. It accepts dashboard data as a prop and formats the values using currency formatters, showing both primary values and secondary information like transaction counts and profit margins.

**LowStockAlert**

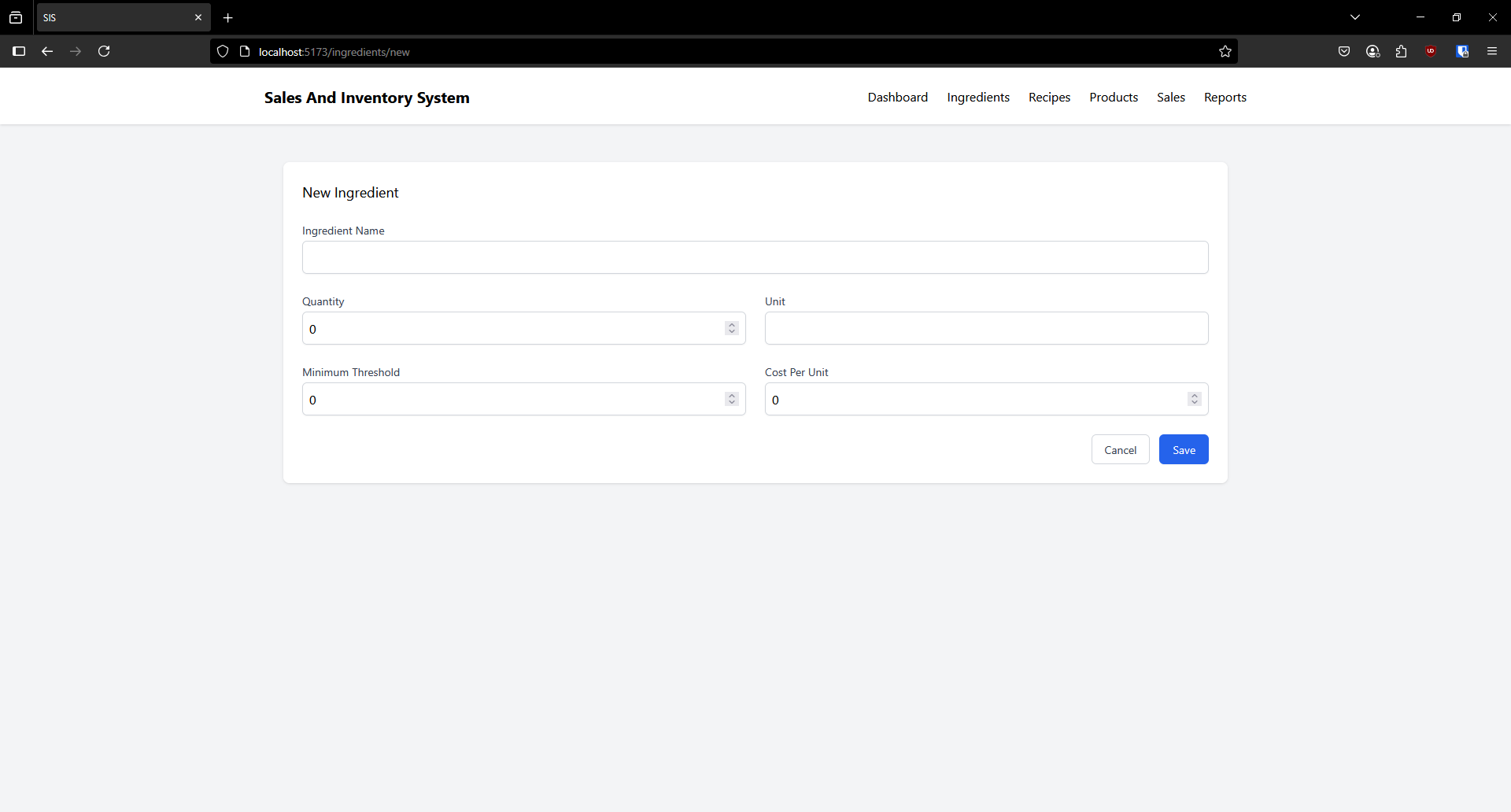
This component monitors inventory levels and displays a warning list of ingredients that are below their minimum threshold. It provides quick links to restock items, displaying quantity information and thresholds for each low-stock ingredient, with a green status message when all inventory is at adequate levels.

**SalesChart**

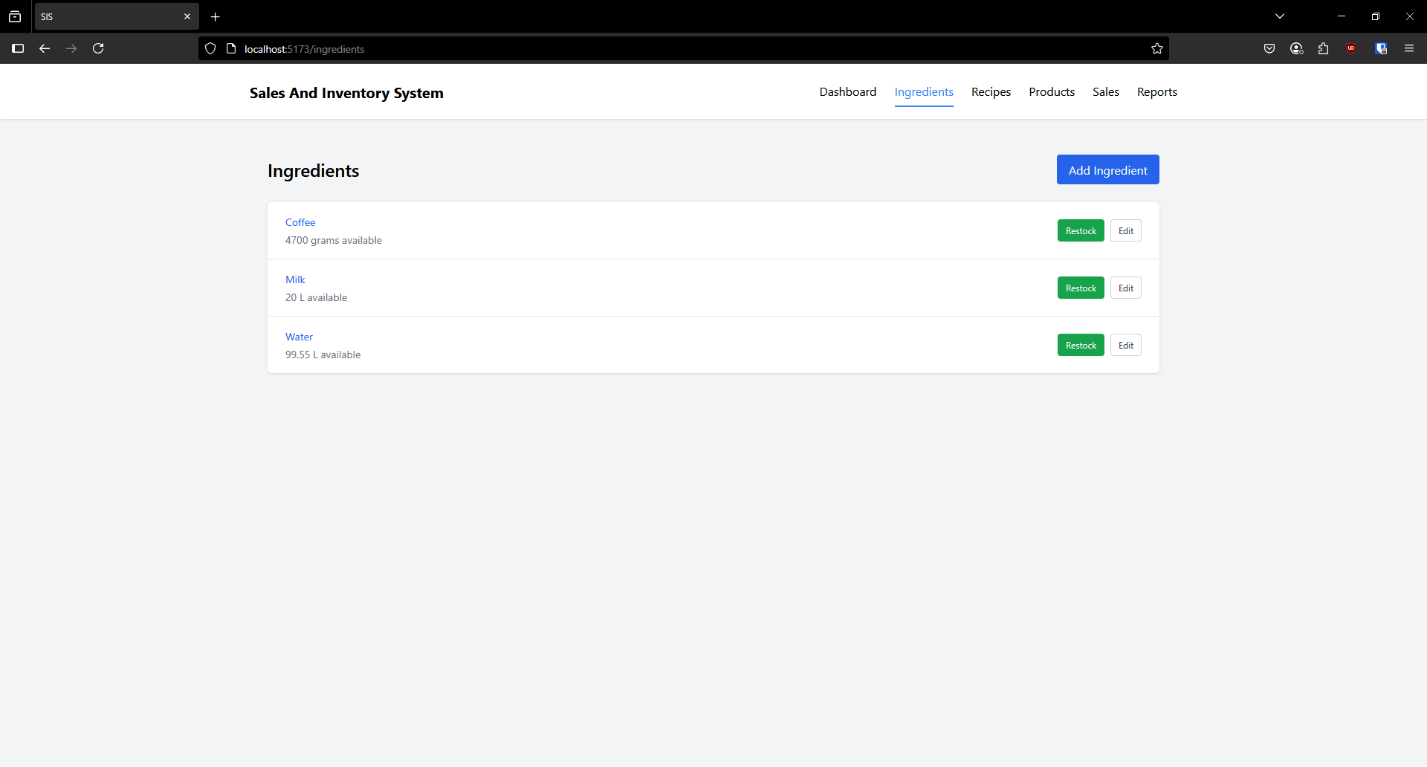
This component renders a line chart visualization showing revenue and profit trends over time. It uses Chart.js to display the data with configurable options for responsiveness, legends, and axis formatting. The chart adapts its display based on screen size for better mobile experience.

**Ingredient Components**

**IngredientForm**



This component provides a form interface for creating and editing ingredients in the inventory system. It includes fields for name, quantity, unit, minimum threshold, and cost per unit with form validation and error handling, supporting both create and update operations.

**IngredientList** 

This component displays all ingredients in the inventory with their current quantities and status. It includes functionality to restock ingredients directly from the list view with an inline form that appears when the restock button is clicked, along with edit options for each ingredient.

**Recipe Components**

**RecipeForm**

A screenshot of a computer

AI-generated content may be incorrect.

This component offers a form for creating and editing food recipes with dynamic ingredient management. Users can add or remove ingredients with specified quantities, set preparation times, and provide descriptions, with the form handling all validation and data formatting for server submission.

**RecipeList**

A screenshot of a computer

AI-generated content may be incorrect.

This component displays all available recipes with their details and preparation controls. It includes functionality to prepare recipes in specific quantities, showing how many portions can be made based on available ingredients, and provides warnings when ingredients are insufficient.

**Product Components**

**ProductForm**

A screenshot of a computer

AI-generated content may be incorrect.

This component provides a form for creating and editing food products with pricing analysis. It allows selection of recipes as the base for products and automatically calculates profit margins based on the selected recipe's cost and user-defined price, displaying this analysis in a visual format.

**ProductList**

A screenshot of a computer

AI-generated content may be incorrect.

This component displays all available products with their pricing details, profit margins, and active status. Each product shows its name, associated recipe, price, cost, profit, and profit margin percentage, with visual indicators for active/inactive status and quick edit links.

**Sales Components**

**PointOfSale**

**A screenshot of a computer

AI-generated content may be incorrect.**

This component serves as the primary sales interface for the application, allowing staff to select products for purchase and complete transactions. It displays available products with their quantities and prices, maintains a shopping cart for the current order, and processes sales by submitting them to the backend while tracking inventory levels.

**SalesHistory**

**A screenshot of a computer

AI-generated content may be incorrect.**

This component displays a chronological record of all completed sales transactions. It fetches and presents detailed information about each sale including date, product name, quantity, unit price, total price, and profit in a tabular format, providing a complete history of business transactions.

**SalesReport**

**A screenshot of a computer

AI-generated content may be incorrect.**

This component provides analytical insights into sales performance over customizable time periods. It allows users to generate reports with different time granularity (daily, weekly, monthly), visualizes the data using charts, and displays comprehensive metrics including transaction counts, revenue, costs, profits, and profit margins in both graphical and tabular formats.

**GitHub Repository:** https://github.com/kur0tsuki/cispt-final