Baña, Gyan Rhanzell Edwin

**CSIPT Final Project**

**Sales and Inventory Management System Documentation**

**Overview**

**The Sales and Inventory Management System is a full-stack application that helps bakeries 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.**

**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)**

**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**

**Special Endpoints**

**Restocking Ingredients**

**POST /api/ingredients/{id}/restock/**

**Request Body:**

**{**

**"amount": 10.5**

**}**

**Response: Updated ingredient data.**

**Preparing Recipes**

**POST /api/recipes/{id}/prepare/**

**Request Body:**

**{**

**"quantity": 5,**

**"notes": "Batch for afternoon rush"**

**}**

**Response:**

**{**

**"message": "Successfully produced 5 Croissant(s)",**

**"production": {**

**"id": 1,**

**"recipe": 3,**

**"recipe\_name": "Croissant",**

**"quantity": 5,**

**"timestamp": "2023-09-05T14:30:00Z",**

**"notes": "Batch for afternoon rush"**

**}**

**}**

**Sales Report**

**GET /api/sales/report/?start\_date=2023-09-01&end\_date=2023-09-30&period=day**

**Query Parameters:**

* **start\_date: Start date (YYYY-MM-DD)**
* **end\_date: End date (YYYY-MM-DD)**
* **period: Aggregation period (day, week, or month)**

**Response:**

**{**

**"data": [**

**{**

**"period": "2023-09-01",**

**"transactions": 25,**

**"total\_sales": 350.75,**

**"cost": 120.25,**

**"profit": 230.50,**

**"profit\_margin": 65.72**

**},**

**...**

**]**

**}**

**Dashboard**

**GET /api/sales/dashboard/**

**Response:**

**{**

**"today": {**

**"revenue": 450.25,**

**"profit": 300.10,**

**"profit\_margin": 66.65,**

**"transactions": 35**

**},**

**"week": {**

**"revenue": 3245.75,**

**"profit": 2156.80,**

**"profit\_margin": 66.45,**

**"transactions": 245**

**},**

**"chart\_data": [**

**{**

**"period": "2023-09-01",**

**"total\_sales": 425.50,**

**"transactions": 32,**

**"profit": 280.15**

**},**

**...**

**]**

**}**

**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**

**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** |

**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.**