CEBU INSTITUTE OF TECHNOLOGY
UNIVERSITY

IT342-[Section]
System Integration and Architecture

System Design Document (SDD)

Project Title: InStock

Prepared By: Badinas, Gregory Ivan Onyx Montibon

Version: 0.3

Date: 03/01/2026
Status: Draft

REVISION HISTORY TABLE

Version	Date	Author	Changes Made	Status
0.1	02/18/2026	Gregory Ivan Onyx M. Badinas	Initial draft	Draft
0.2	02/26/2026	Gregory Ivan Onyx M. Badinas	Added API specifications	Draft
0.3	03/01/2026	Gregory Ivan Onyx M. Badinas	Added wireframe screenshots	Draft
0.4		Gregory Ivan Onyx M. Badinas		
0.5		Gregory Ivan Onyx M. Badinas		
0.6		Gregory Ivan Onyx M. Badinas		
1		Gregory Ivan Onyx M. Badinas		

TABLE OF CONTENTS

Contents

EXECUTIVE SUMMARY

1.1 Project Overview

InStock is an application aimed at reducing food waste by enabling users to select ingredients they currently have in stock, and dynamically suggesting recipes that match those selected ingredients. The system includes a Spring Boot backend API, a React web application, and an Android mobile application, all integrated to provide a seamless, multi-platform experience. To ensure robust functionality, the system leverages a third-party public API for extensive recipe data and incorporates secure modern authentication standards.

1.2 Objectives

1. Develop a fully functional recipe suggestion MVP with secure authentication, ingredient list management, a recipe catalog, and intelligent recipe suggestion capabilities.
2. Implement a professional three-tier architecture utilizing Spring Boot (backend), React (web), and Android (mobile).
3. Create secure RESTful APIs for communication between all system components, utilizing JWT for session management.
4. Integrate mandatory system requirements including Google OAuth for social login, SMTP for email notifications, file uploading for user/recipe images, and external API consumption for recipe generation.
5. Deploy all system components to production-ready environments.

1.3 Scope

Included Features:

- User registration and authentication (Email/Password & Google OAuth 2.0).
- Ingredient listing with search functionality.
- "Add" to ingredients in-stock list (User's digital pantry).
- Persistent Allergy & Dietary Filtering (User preferences saved to database).
- Recipe suggestions based on selected ingredients and filtered by allergy constraints.

- "Add to favorites" functionality for recipes (Full CRUD).
- File upload capability (e.g., user profile pictures or custom ingredient images).
- Email notifications via SMTP (Welcome emails and system notifications).
- Role-Based Access Control (Admin vs. Regular User).
- Relational database implementation (minimum 5 tables).
- Responsive Web app and native Mobile app.

Excluded Features:

- Nutritional information of recipes.
- AI generation features.
- Real-time recipe creation.
- "Create your own recipe" manual functionality.

1.0 INTRODUCTION

1.1 Purpose

This document serves as the comprehensive design specification for InStock. It provides detailed requirements, architectural decisions, API contracts, database design, and an implementation roadmap to guide development and ensure all components integrate seamlessly according to IT342 standards.

2.0 FUNCTIONAL REQUIREMENTS SPECIFICATION

2.1 Project Overview

Project Name: InStock

Domain: Cooking / Food Waste Reduction

Primary Users:

1. General Public

Problem Statement: Individuals often struggle to figure out what meals to cook using the ingredients they already have at home, leading to unnecessary grocery purchases and high rates of household food waste.

Solution: A cross-platform digital pantry and recipe suggestion tool that cross-references a user's available ingredients with an external recipe database to provide immediate, actionable meal ideas.

2.2 Core User Journeys

Journey 1: First-time User Recipe Discovery

1. User visits the web application or downloads the mobile app.
2. User registers an account (via standard email or Google OAuth).
3. System sends a welcome email via SMTP.
4. User sets their dietary preferences (e.g., selects "Peanuts" and "Shellfish" from the Allergy Filter). System saves these preferences.
5. User navigates to the "My Pantry" section and searches/adds ingredients they currently own. (Ingredients containing selected allergens are hidden or flagged).
6. User clicks "Suggest Recipes".
7. The system calls an external public API, matching the ingredients while excluding the saved allergens, and displays suggested recipes.
8. User saves a favored recipe to their "Favorites" list.

Journey 2: Returning Customer Management

1. User logs in with existing credentials.
2. User uploads a profile picture (File Upload feature).
3. User removes depleted ingredients from their stock list.
4. User generates a new recipe based on the updated stock.
5. User views their saved "Favorites" list for future reference.

Journey 3: Administrator System Management

1. Admin logs in with administrative credentials (triggering RBAC validation).
2. Admin navigates to the dashboard.
3. Admin manages the master ingredient list (Adds new baseline ingredients or deletes obsolete ones).
4. Admin views system usage statistics (e.g., total users, total favorites saved).

2.3 Feature List (MoSCoW)

MUST HAVE

1. User authentication (Register, Login, Logout, Google OAuth, JWT).
2. Role-Based Access Control (Admin vs Regular User).

3. Ingredient catalog and personal stock management (CRUD).
4. Recipe suggestion engine (consuming a public external API like Spoonacular).
5. Favorites list management.
6. Email notifications (SMTP).
7. File upload functionality (stored on server, linked to DB).
8. Persistent Allergy/Dietary Filtering (Saved to User Profile).

SHOULD HAVE

1. Ingredient categories and filtering.
2. Basic input validation feedback on frontend and backend.
3. Responsive design for all screen sizes.

COULD HAVE

1. Basic user dashboard showing most frequently used ingredients.
2. Password reset flow.
3. Nutritional info of recipes.

WON'T HAVE

1. AI features.
2. Real-time recipe creation.
3. Payment gateway integration (Not applicable to this domain).

2.4 Detailed Feature Specifications

Feature: User Authentication & Security

- **Screens:** Registration, Login.
- **Fields:** Email, Password, Full Name.
- **Validation:** Email format, password strength (BCrypt hashing), uniqueness.
- **API Endpoints:** POST /api/auth/register, POST /api/auth/login, GET /api/auth/me, POST /api/auth/oauth/google.
- **Security:** JWT tokens generated internally (even after OAuth), protected routes.

Feature: Allergy & Dietary Management

- **Screens:** Recipe List (Filter Modal), Ingredient List (Filter Modal), User Settings.

- **Display:** Checkbox list of common allergens (e.g., Dairy, Peanuts, Gluten, Shellfish).
- **Functions:** Select/Deselect allergens, Persist selection to User database record.
- **Logic:**
    - **Persistence:** Choices are saved to the database (e.g., user_allergies table).
    - **Filtering:** Frontend hides ingredients matching allergens. Backend includes allergy parameters when calling the External Recipe API (to prevent fetching unsafe recipes).
- **API Endpoints:** GET /api/users/allergies, PUT /api/users/allergies.

Feature: Pantry/Stock Management (Core Business Module)

- **Screens:** My Pantry, Search Ingredients.
- **Display:** List of currently owned ingredients.
- **Functions:** Add to stock, remove from stock, search master list.
- **API Endpoints:** `GET /api/stock`, `POST /api/stock`, `DELETE /api/stock/{id}`.

Feature: Recipe Suggestion & Favorites

- **Screens:** Suggested Recipes, My Favorites.
- **Display:** List of currently owned ingredients (Filtered by active allergy preferences).
- **Functions:** Fetch recipes from External API based on stock, save to DB, view saved.
- **API Endpoints:** `GET /api/recipes/suggest`, `POST /api/favorites`, `GET /api/favorites`, `DELETE /api/favorites/{id}`.

Feature: Recipe Suggestion & Favorites

- **Screens:** Suggested Recipes, My Favorites.
- **Functions:** Fetch recipes from External API based on stock AND user allergy constraints, save to DB, view saved.
- **API Endpoints:** GET /api/recipes/suggest (Must accept or retrieve user allergy settings), POST /api/favorites, GET /api/favorites, DELETE /api/favorites/{id}.

2.5 Acceptance Criteria

AC-1: Successful Recipe Suggestion

➢ Given I am a logged-in user with ingredients in my stock list,

- ➢ When I click the "Suggest Recipes" button,
- ➢ Then the system should successfully consume the external public API,
- ➢ And display a list of recipes that utilize my selected ingredients and filter out recipes that contain allergens selected in the allergen filter.

AC-2: Adding to Favorites

- ➢ Given I am viewing a suggested recipe,
- ➢ When I click "Add to Favorites",
- ➢ Then the recipe details should be saved to the database,
- ➢ And a success notification should appear,
- ➢ And the recipe must appear in my "My Favorites" screen.

AC-3: Role-Based Access Control (Admin)

- ➢ Given I am logged in as a Regular User,
- ➢ When I attempt to access the Admin ingredient management endpoint,
- ➢ Then the system should reject the request with a 403 Forbidden status code,
- ➢ And display an appropriate UI access denied message.

AC-4: Persistent Allergy Filtering

- ➢ Given I have selected "Peanuts" in the allergy filter and logged out,
- ➢ When I log back in and search for recipes or ingredients,
- ➢ Then the "Peanuts" filter should remain active (Persistent),
- ➢ And no recipes or ingredients containing peanuts should appear in the results,
- ➢ And the External API call must include the exclusion parameter for peanuts

3.0 NON-FUNCTIONAL REQUIREMENTS

3.1 Performance Requirements

- **API response time:** ≤ 10 seconds for standard queries (External API fetching may vary slightly based on third-party constraints).
- **Database queries:** Complete within 500ms using proper indexing.
- **Mobile app cold start:** ≤ 30 seconds.

3.2 Security Requirements

- **Authentication:** Stateless JWT token authentication.
- **Password Storage:** Hashing using BCrypt (No plain-text passwords in the DB).
- **Data Exposure:** Use of Data Transfer Objects (DTOs) to ensure sensitive data (like passwords) is never exposed in API responses.
- **Role Verification:** Admin endpoints strictly require role verification at the controller layer.

3.3 Compatibility Requirements

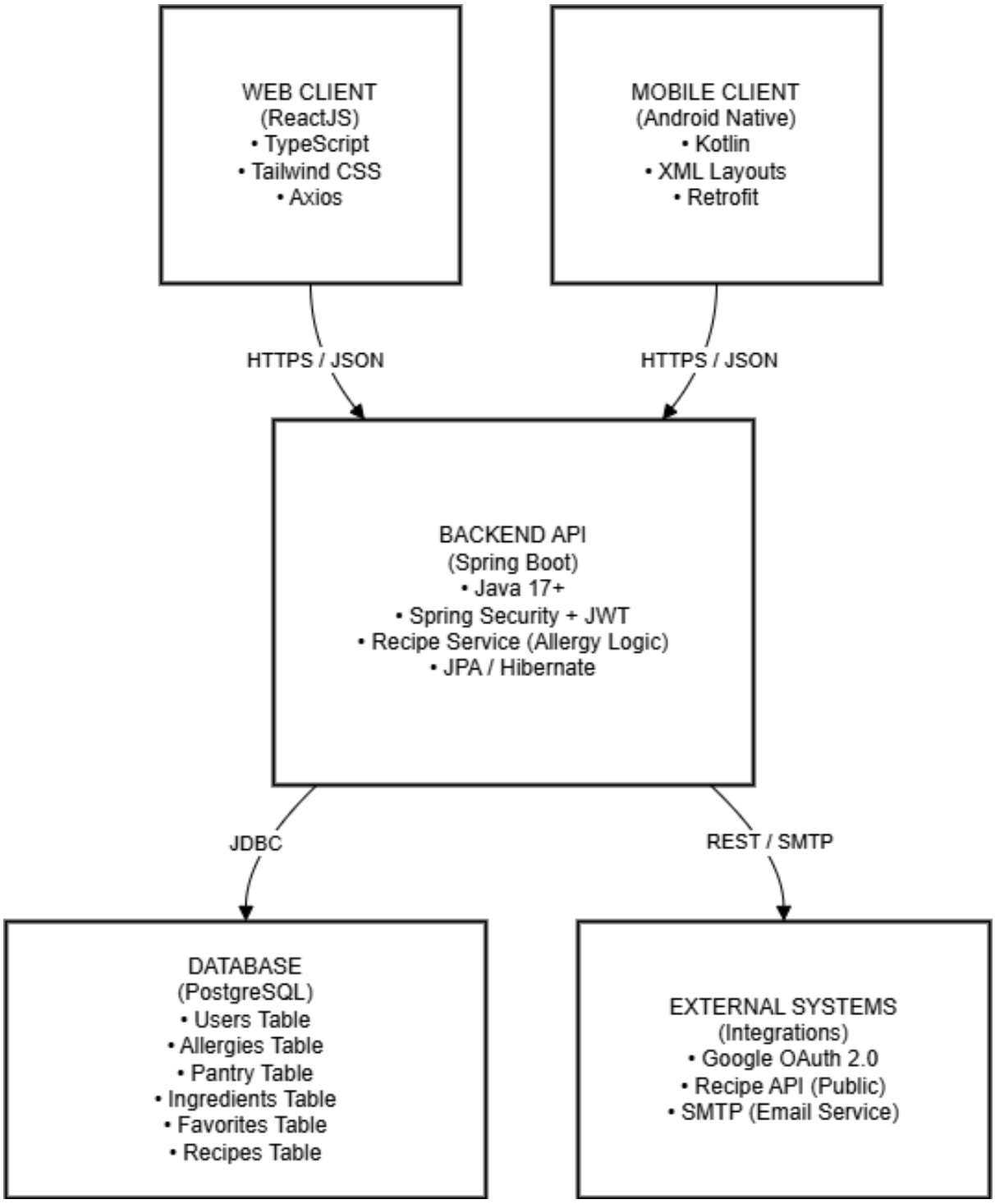
- **Web Browsers:** Chrome, Firefox, Safari, Edge (latest versions).
- **Android:** Native application supporting modern Android API levels.
- **Screen Sizes:** Responsive design across Mobile, Tablet, and Desktop platforms.

3.4 Usability Requirements

- Clean, intuitive user interface focused on reducing friction when adding ingredients.
- Clear error messages (e.g., "Ingredient already in pantry") with fast recovery options.
- Console prints are strictly forbidden for user feedback or backend logging (proper logging frameworks must be used).

4.0 SYSTEM ARCHITECTURE

4.1 Component Diagram



Technology Stack:

- **Backend:** Java 17, Spring Boot 3.x, Spring Security (JWT), Spring Data JPA
- **Database:** PostgreSQL 14+
- **Web Frontend:** React 18, TypeScript, Tailwind CSS, Axios
- **Mobile:** Kotlin, XML Layouts (View System), Retrofit, Room Database
- **Build Tools:** Maven (Backend), npm (Web), Gradle (Android)
- **External Services:** Google OAuth 2.0, SMTP (JavaMailSender), Spoonacular API
- **Deployment:** Railway/Render (Backend & DB), Vercel/Netlify (Web), APK Side-loading (Mobile)

5.0 API CONTRACT & COMMUNICATION

5.1 API Standards

Base URL	https://[server_hostname]:[port]/api/v1
Format	JSON for all data exchange (except file uploads)
Authentication	Bearer token (JWT) passed in the Authorization header

Response Structure	{
	"success": boolean,
	"message": string,
	"data": object | array | null,
	"error": {
	"code": string,
	"details": object | null
	},
	"timestamp": string
	}

5.2 Endpoint Specifications

Authentication & User Management Endpoints

User Registration

Description	User Registration

API URL	/auth/register
HTTP Request Method	POST
Format	JSON for all requests/responses
Authentication	None

Request Payload	{
	"email": "user@example.com",
	"password": "SecurePass 123!",
	"fullName": "John Doe"
	}

Response Structure	{
	"success": true,
	"data": {
	"token": "jwt_string...",
	"user": {
	"id": 1,
	"email": "...",
	"role": "USER"
	}
	}
	}

Standard User Login

Description	User Login
API URL	/auth/login
HTTP Request Method	POST
Format	JSON for all requests/responses
Authentication	None

Request Payload	{
	"email": "user@example.com",
	"password": "SecurePass123!"
	}

Response Structure	{
	"success": true,
	"data": {
	"token": "jwt_string...",
	"user": {...}
	}
	}

Google OAuth UserLogin

Description	Google OAuth Login
API URL	/auth/google
HTTP Request Method	POST
Format	JSON for all requests/responses
Authentication	None (Requires valid Google ID Token in payload)

Request Payload	{
	"idToken": "google_id_token_from_client"
	}

Response Structure	{
	"success": true,
	"data": {
	"token": "system_jwt_string...",
	"user": {...}
	}
	}

Get Current User

Description	Get Current Authenticated User
API URL	/auth/me
HTTP Request Method	GET
Format	JSON
Authentication	Bearer token (JWT) in Authorization header
Request Payload	None

Response Structure	{
	"success": true,
	"data": {
	"id": 1,
	"email": "user@example.com",
	"fullName": "John Doe",
	"role": "USER",
	"avatarUrl": "https://..."

	} }

Upload User Avatar

Description	Upload User Avatar
API URL	/users/avatar
HTTP Request Method	POST
Format	Multipart Form Data
Authentication	Bearer token (JWT) in Authorization header
Request Payload	Form data with key file (image file)

Response Structure	{
	"success": true,
	"message": "Avatar uploaded successfully",
	"data": {
	"avatarUrl": "https://server.com/uploads/user_1.jpg"
	}
	}

Pantry & Ingredients Management Endpoints

Get User's Pantry Ingredients

Description	Get User Stock
API URL	/stock
HTTP Request Method	GET
Format	JSON
Authentication	Bearer token (JWT)
Request Payload	None
Response Structure	{ "success": true, "data": [{ "id": 101,

	"name": "Tomato", "category": "Vegetable", "addedAt": "2026-03-01T10:00:00Z" }] }

Add Ingredient to Pantry

Description	Add Ingredient to Stock
API URL	/stock
HTTP Request Method	POST
Format	JSON
Authentication	Bearer token (JWT)

Request Payload	{
	"ingredientId": 101
	}

Response Structure	{
	"success": true,
	"message": "Ingredient added to pantry",
	"data": {
	"ingredientId": 101,
	"name": "Tomato"
	}
	}

Delete Ingredient from Pantry

Description	Remove Ingredient from Stock
API URL	/stock/{ingredientId}
HTTP Request Method	DELETE
Format	JSON
Authentication	Bearer token (JWT)
Request Payload	None

Response Structure	{
	"success": true,
	"message": "Ingredient removed from stock"
	}

Recipe Suggestion & Favorites Endpoints

Get Recipe Suggestions

Description	Get Recipe Suggestions
API URL	/recipes/suggest
HTTP Request Method	GET
Format	JSON
Authentication	Bearer token (JWT)
Request Payload	None

Response Structure	{
	"success": true,
	"data": [
	{
	"externalId": 7493,
	"title": "Tomato Chicken Curry",
	"imageUrl": "https://api.spoonacular.com/...",
	"usedIngredientCount": 2,
	"missingIngredientCount": 1
	}
	]
	}

Add Recipe to Favorites

Description	Add Recipe to Favorites
API URL	/favorites
HTTP Request Method	POST
Format	JSON

Authentication	Bearer token (JWT)

Request Payload	{
	"externalId": "7493",
	"title": "Tomato Chicken Curry",
	"imageUrl": "https://...",
	"summary": "A delicious curry..."
	}

Response Structure	{
	"success": true,
	"message": "Recipe added to favorites"
	}

Get Favorite Recipe

Description	Get Favorite Recipes
API URL	/favorites
HTTP Request Method	GET
Format	JSON
Authentication	Bearer token (JWT)
Request Payload	None

Response Structure	{
	"success": true,
	"data": [
	{
	"id": 1,
	"externalId": "7493",
	"title": "Tomato Chicken Curry",
	"savedAt": "2026-03-01T12:00:00Z"
	}
	]
	}

Remove Recipe from Favorites

Description	Remove Recipe from Favorites

API URL	/favorites/{id}
HTTP Request Method	DELETE
Format	JSON
Authentication	Bearer token (JWT)
Request Payload	None

Response Structure	{
	"success": true,
	"message": "Recipe removed from favorites"
	}

Administration Endpoints

Add Master Ingredient

Description	Add Master Ingredient
API URL	/admin/ingredients
HTTP Request Method	POST
Format	JSON
Authentication	Bearer token (JWT); Admin role strictly validated

Request Payload	{
	"name": "Quinoa",
	"category": "Grains"
	}

Response Structure	{
	"success": true,
	"data": {
	"id": 105,
	"name": "Quinoa",
	"category": "Grains"
	}
	}

Delete Master Ingredient

Description	Delete Master Ingredient
API URL	/admin/ingredients/{id}
HTTP Request Method	DELETE
Format	JSON
Authentication	Bearer token (JWT); Admin role strictly validated
Request Payload	None

Response Structure	{
	"success": true,
	"message": "Master ingredient deleted"
	}

Session Management Endpoints

User Logout

Description	User Logout
API URL	/auth/logout
HTTP Request Method	POST
Format	JSON
Authentication	Bearer token (JWT) in Authorization header
Request Payload	None

Response Structure	{
	"success": true,
	"message": "Successfully logged out. Please remove local tokens."
	}

5.3 Error Handling

HTTP Status Codes

- 200 OK: Request processed successfully.
- 201 Created: Resource (User, Stock Item, Favorite) successfully created.
- 400 Bad Request: Validation failure (e.g., invalid email, missing ingredient name).
- 401 Unauthorized: Missing or invalid JWT token.
- 403 Forbidden: User attempted to access Admin endpoints.
- 404 Not Found: Resource (Recipe ID, Stock ID) does not exist.
- 500 Internal Server Error: Unexpected backend failure or External API downtime.

Error Code Examples

Example 1	{
	"success": false,
	"message": "This ingredient is already in your pantry.",
	"data": null,
	"error": {
	"code": "STOCK-001",
	"details": {
	"ingredientId": 101,
	"conflictReason": "Duplicate entry detected"
	}
	},
	"timestamp": "2026-03-01T10:30:15Z"
	}

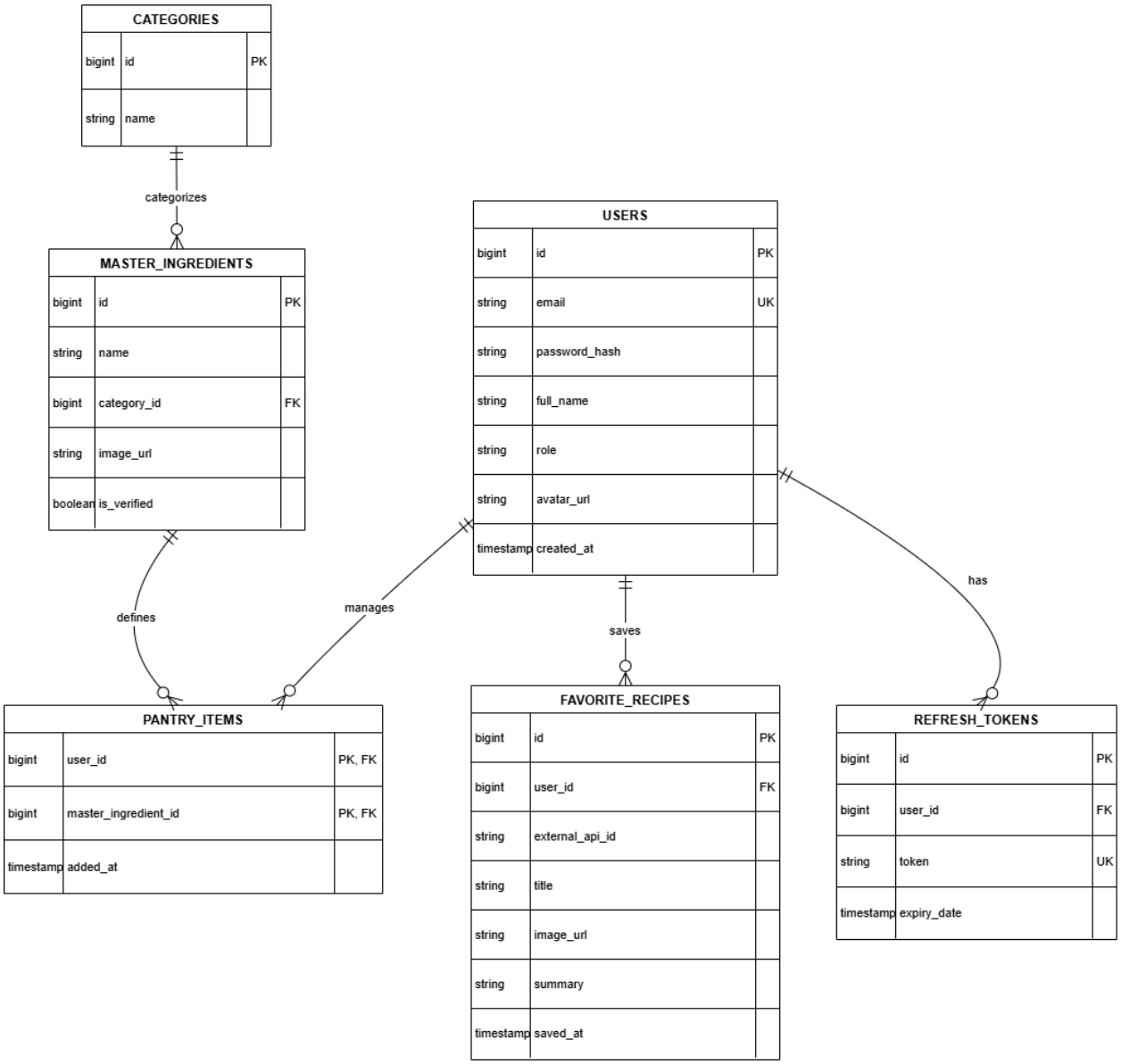
Example 2	{
	"success": false,
	"message": "We couldn't fetch new recipes right now. Please check your saved favorites or try again later.",
	"data": null,
	"error": {
	"code": "RECIPE-001",
	"details": {
	"provider": "SpoonacularAPI",
	"failureType": "Connection Timeout",
	"retryAfterSeconds": 60
	}
	},
	"timestamp": "2026-03-01T10:35:22Z"
	}

Common Error Codes

- AUTH-001: Invalid credentials
- AUTH-002: Token expired
- STOCK-001: Duplicate Ingredient
- RECIPE-001: External API Failure
- FILE-001: Invalid File Type
- RBAC-001: Access Denied

6.0 DATABASE DESIGN

6.1 Entity Relationship Diagram



Detailed Relationships:

- **One-to-Many:** Category → Master_Ingredients (A category like "Dairy" contains multiple ingredients).
- **One-to-Many:** User → Pantry_Items (A user can have multiple ingredients in their pantry)
- **One-to-Many:** Master_Ingredient → Pantry_Items (A master ingredient can exist in multiple users' pantries).
- **Many-to-Many:** User ↔ Master_Ingredient (Resolved cleanly via the Pantry_Items composite key join table).
- **One-to-Many:** User → Favorite_Recipes (A user can save multiple recipes fetched from the external API).
- **One-to-Many:** User → Refresh_Tokens (A user can have multiple active sessions across web and mobile).

Key Tables:

1. **users** - User accounts, core authentication data, and role assignments.
2. **categories** - Groupings to help organize and filter the master ingredient list.
3. **master_ingredients** - The central catalog of valid ingredients, managed exclusively by Administrators to ensure data integrity.
4. **pantry_items** - The associative join table tracking the boolean existence of ingredients in a user's local inventory (resolving the Many-to-Many relationship without tracking unnecessary quantities).
5. **favorite_recipes** - Cached metadata for recipes a user has saved from the external recipe API.
6. **refresh_tokens** - Secure storage for JWT refresh tokens to handle session persistence and secure logouts.

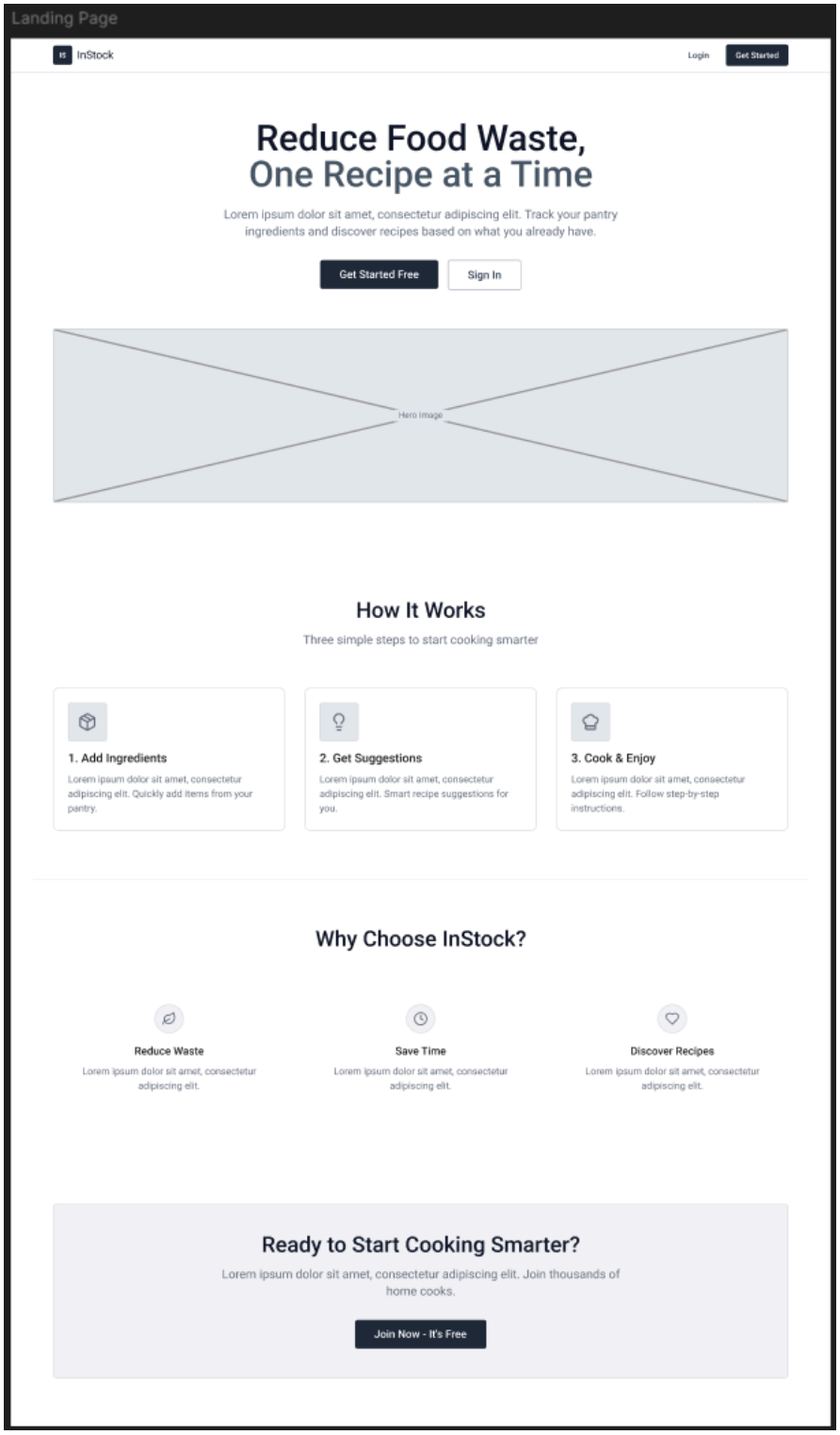
Table Structure Summary:

- **users:** id (PK), email (Unique), password_hash, full_name, role, avatar_url, created_at
- **categories:** id (PK), name
- **master_ingredients:** id (PK), name, category_id (FK), image_url, is_verified
- **pantry_items:** user_id (PK, FK), master_ingredient_id (PK, FK), added_at (Note: Primary Key is a composite of user_id and master_ingredient_id)
- **favorite_recipes:** id (PK), user_id (FK), external_api_id, title, image_url, summary, saved_at
- **refresh_tokens:** id (PK), user_id (FK), token (Unique), expiry_date

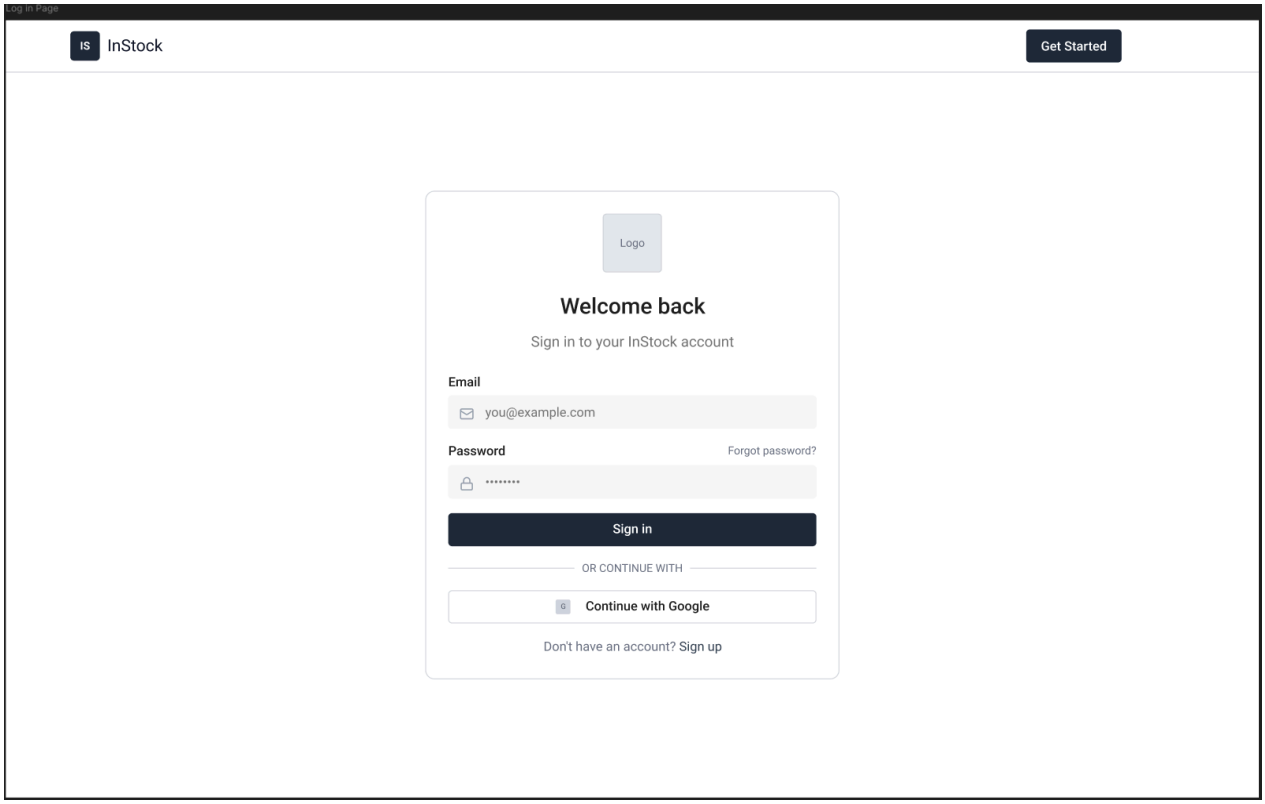
7.0 UI/UX DESIGN

7.1 Web Application Wireframes

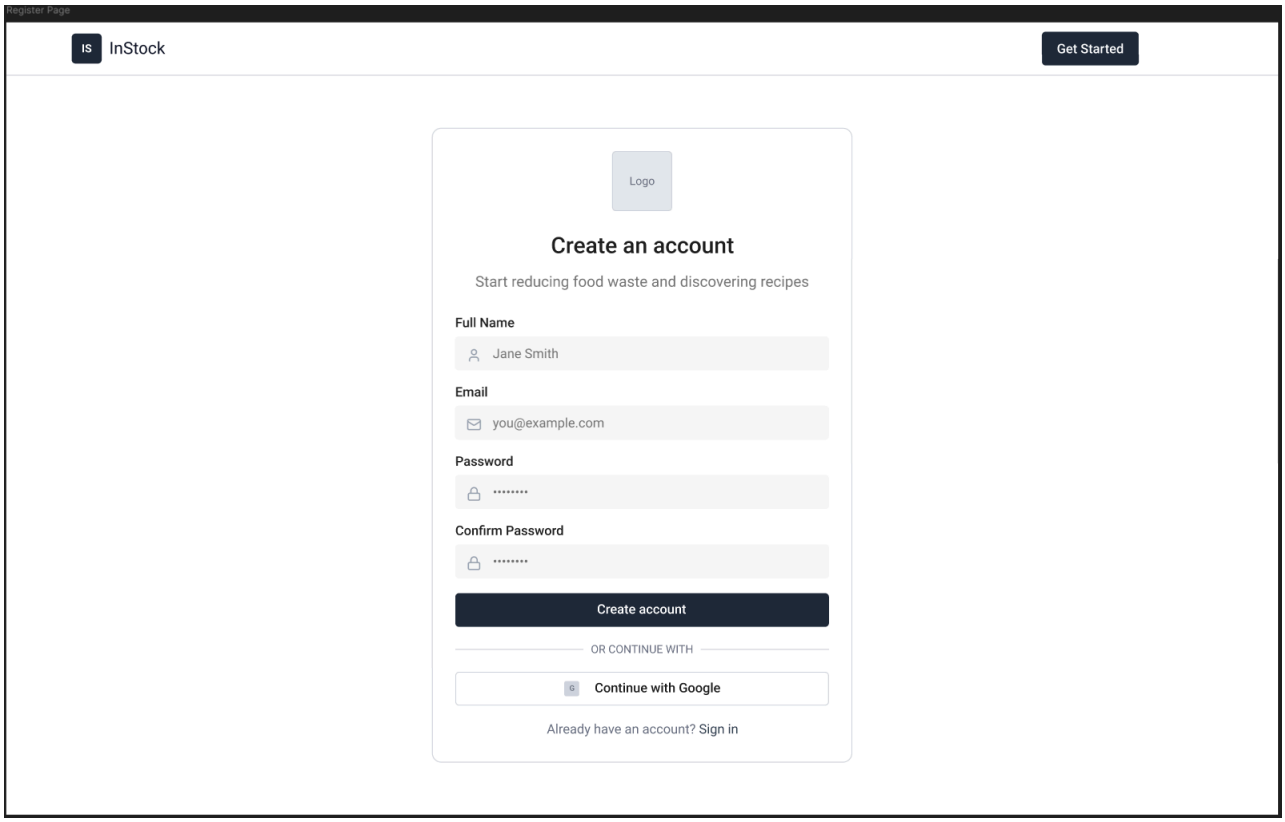
Landing Screen



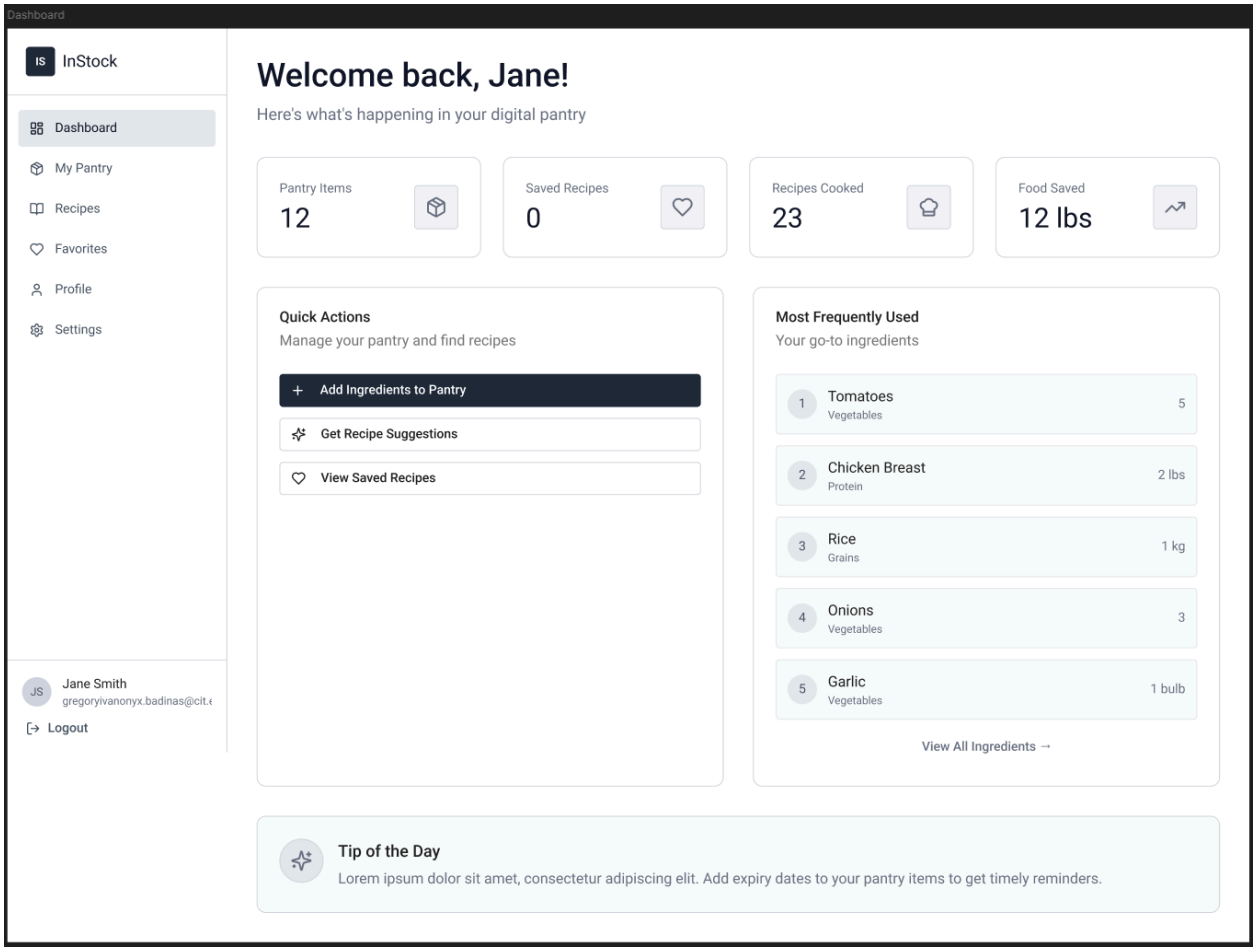
Log in Screen



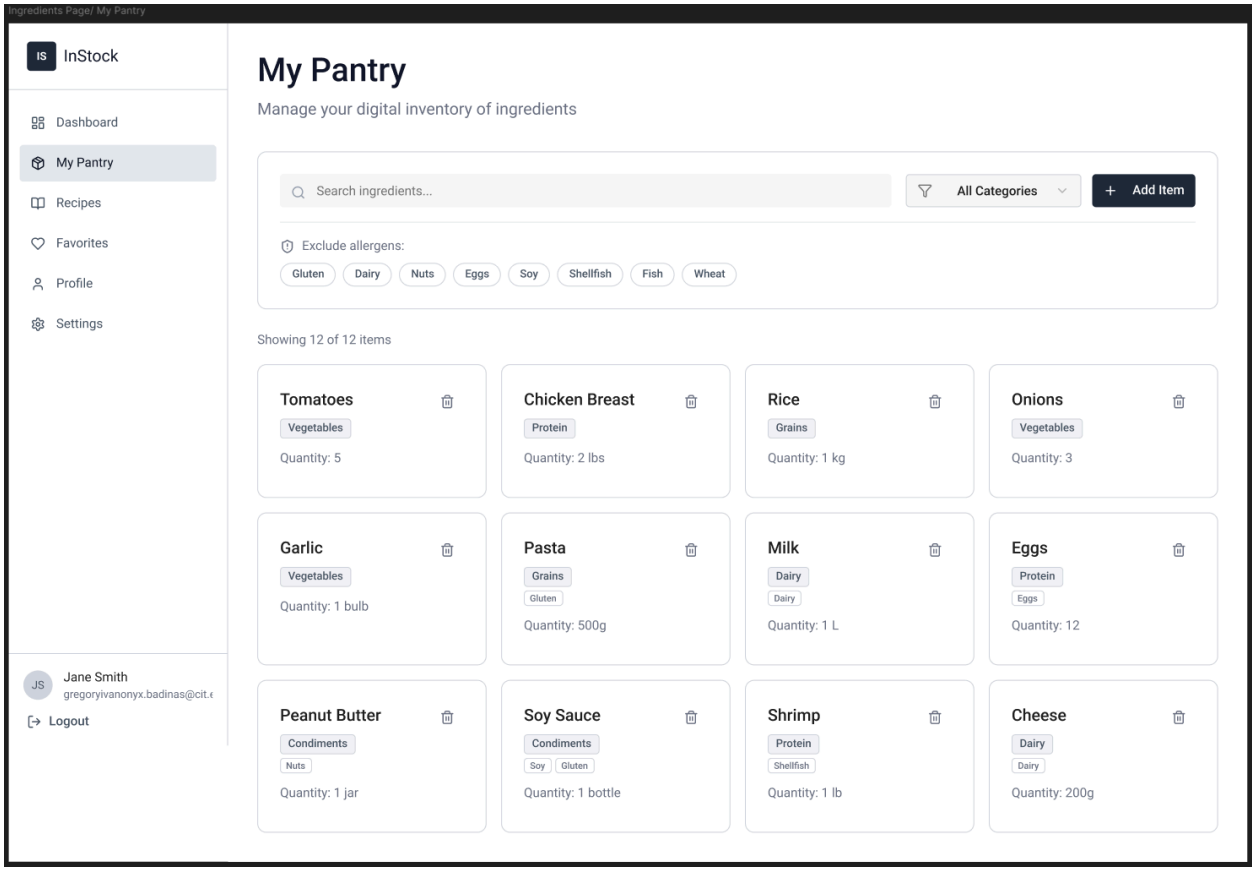
Register Screen


Register Page
IS
InStock
Get Started
Logo
Create an account
Start reducing food waste and discovering recipes
Full Name
Jane Smith
Email
you@example.com
Password
Confirm Password
Create account
OR CONTINUE WITH
Continue with Google
Already have an account? Sign in

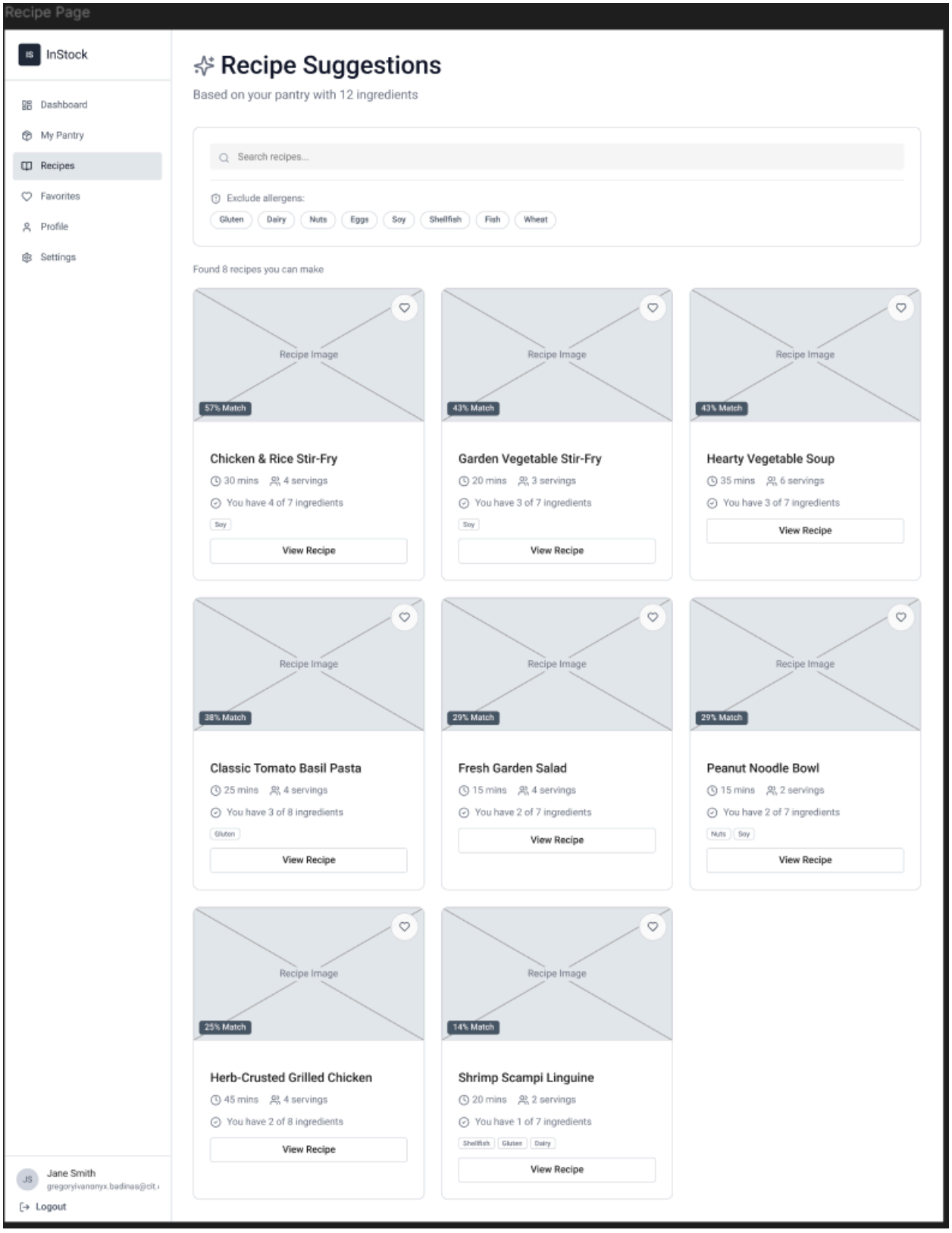
Dashboard Screen



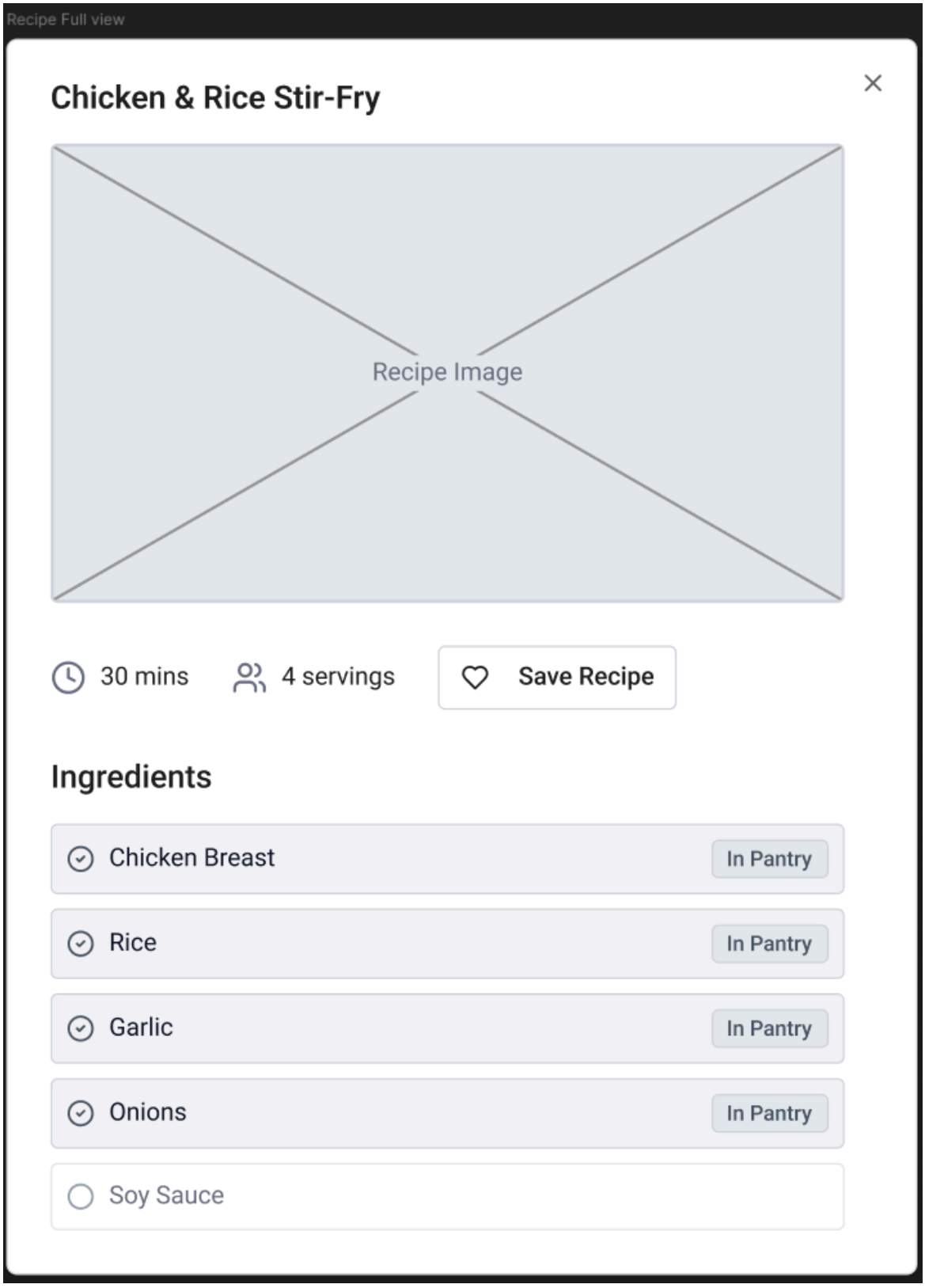
Ingredients Screen/My Pantry Screen



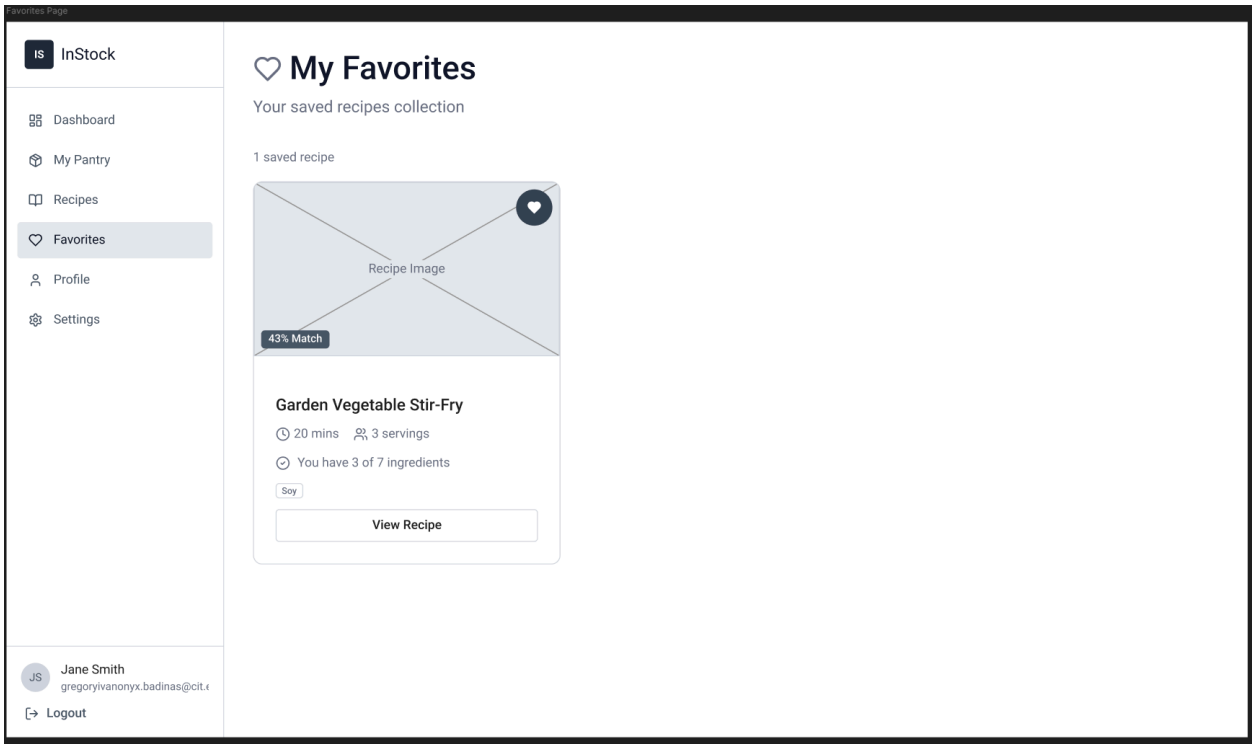
Recipe Screen



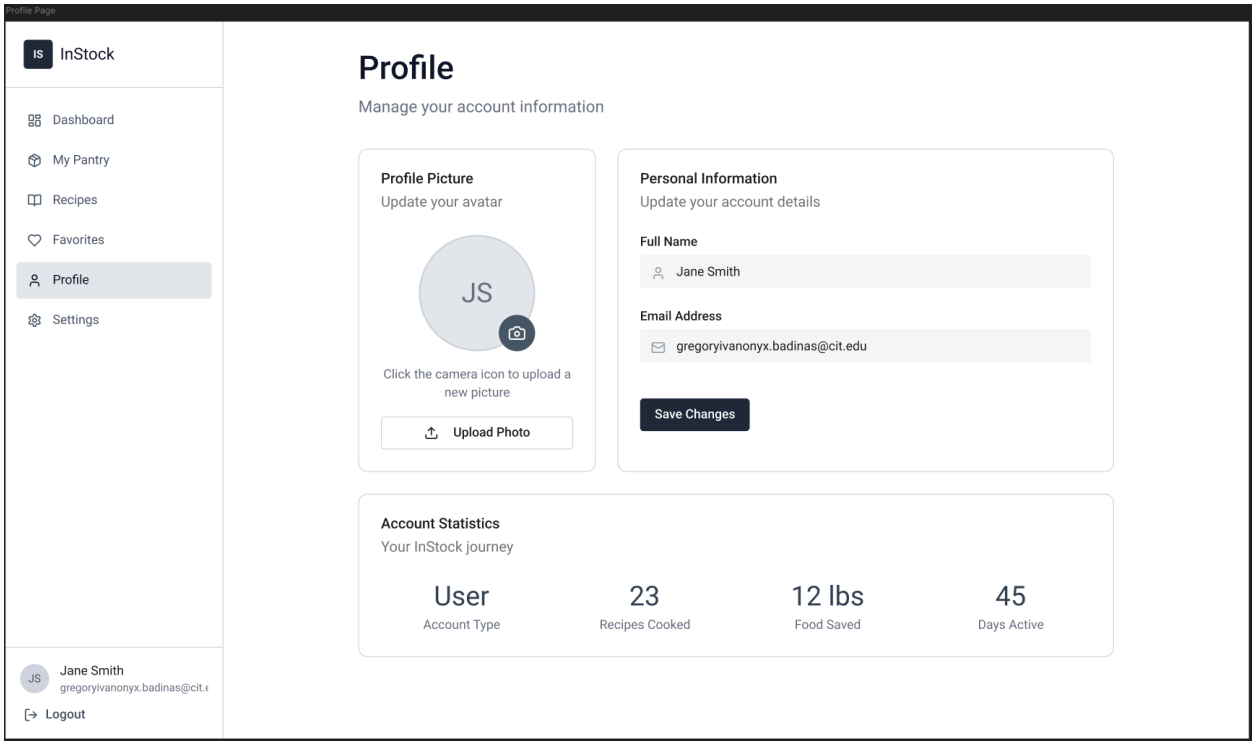
Recipe Full View Screen



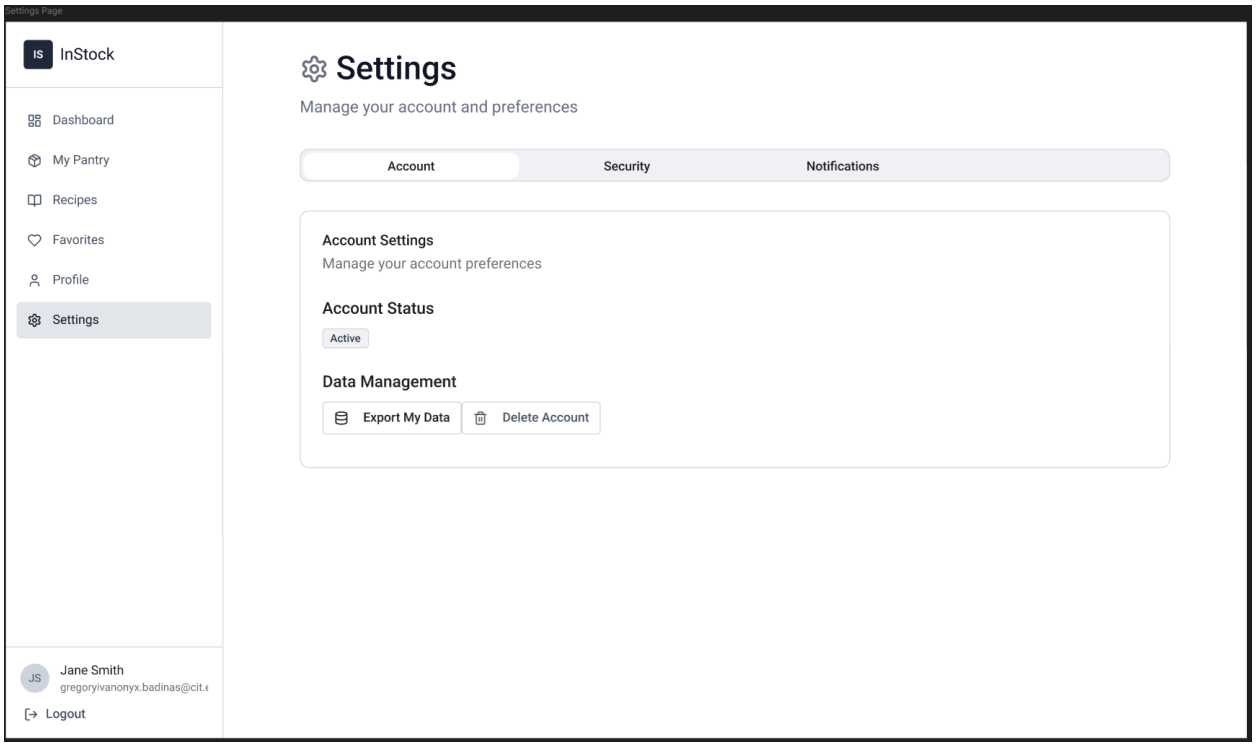
Favorites Screen



Profile Screen

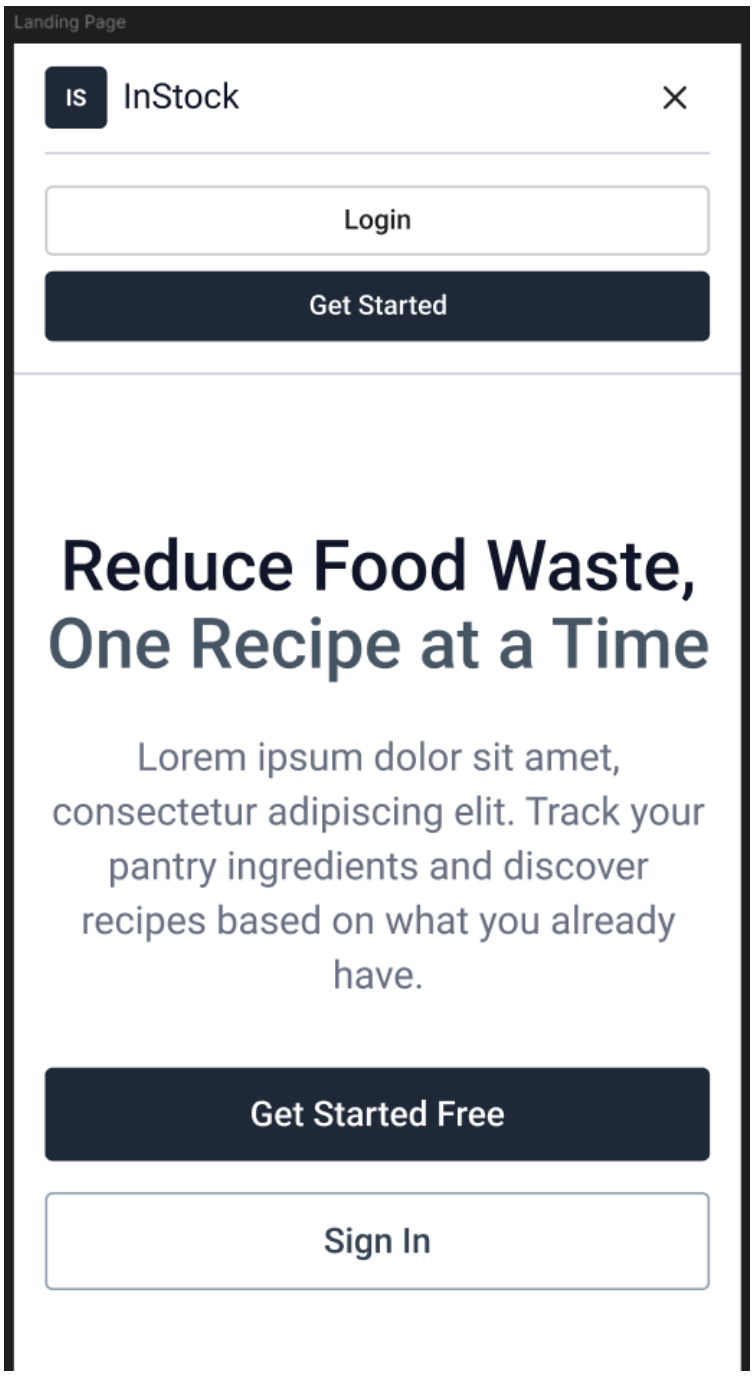


Settings Screen

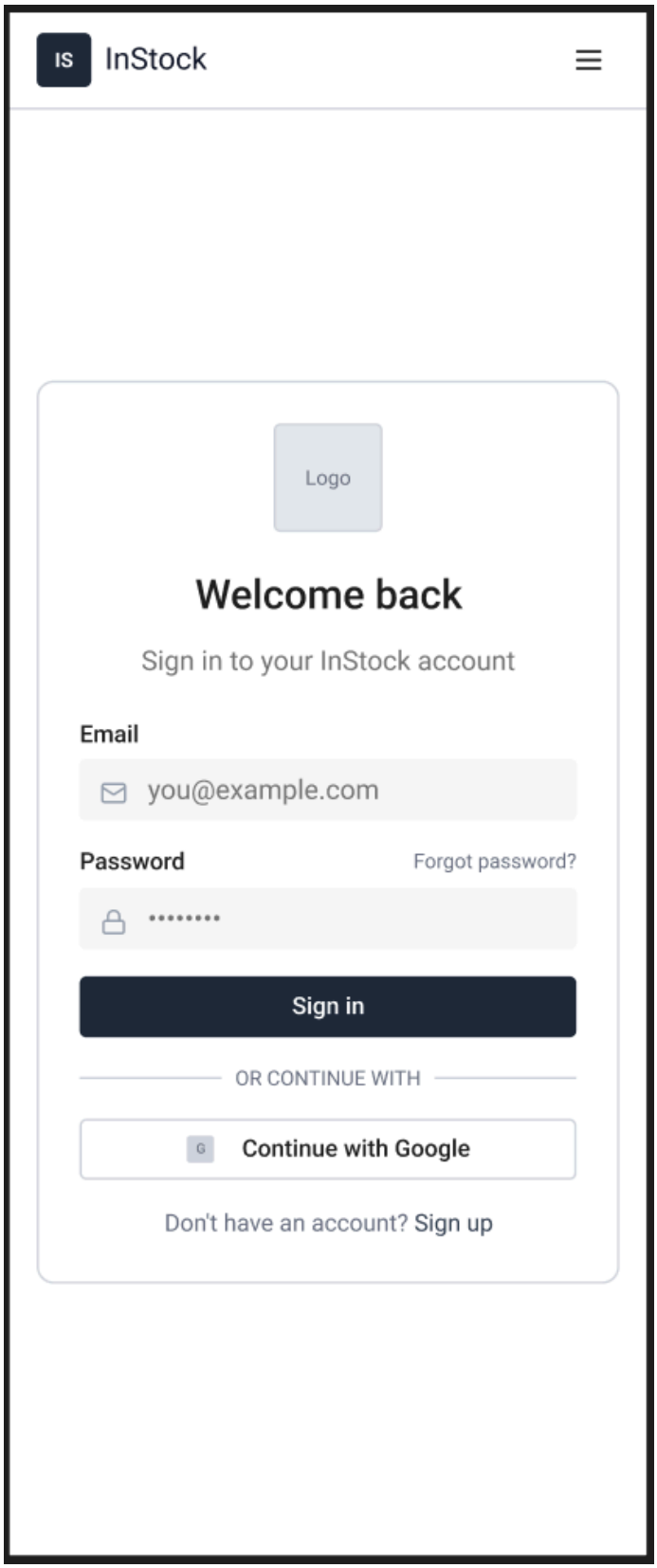


7.2 Mobile Application Wireframes

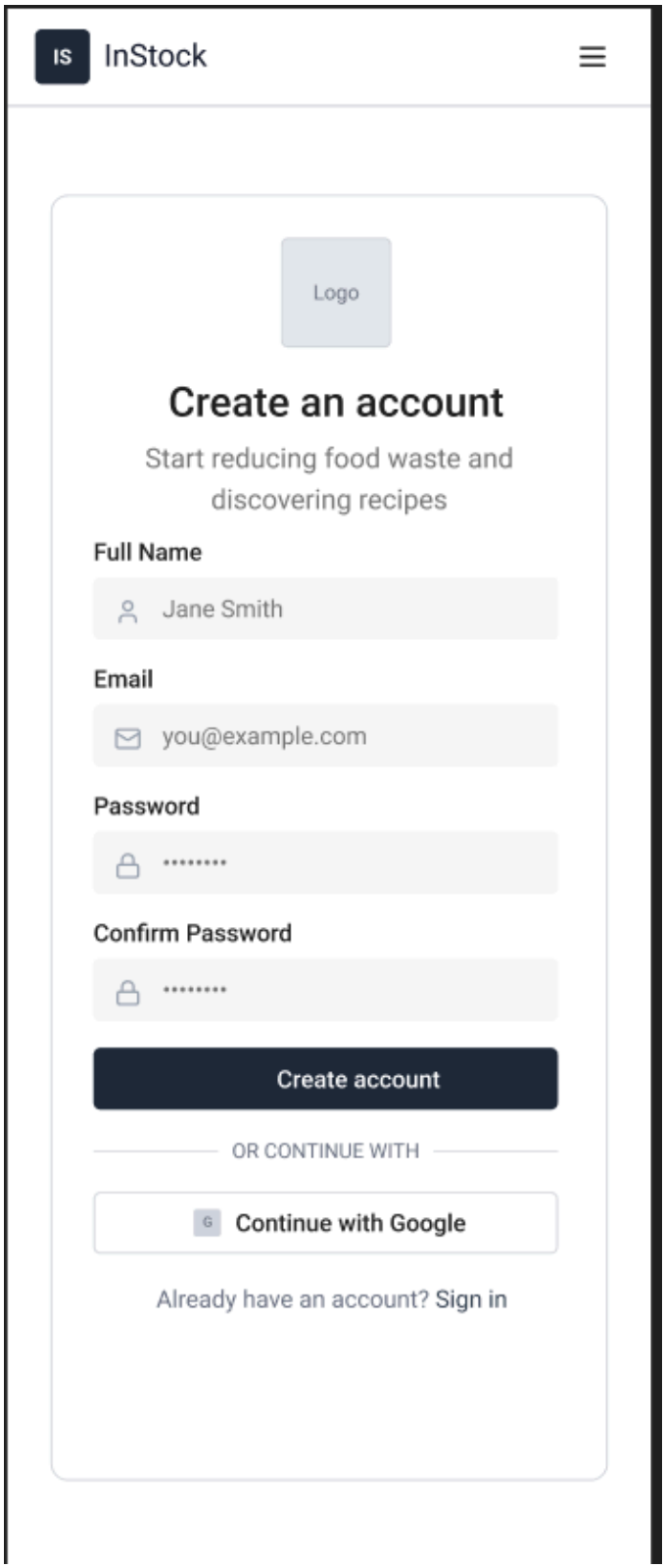
Landing Screen



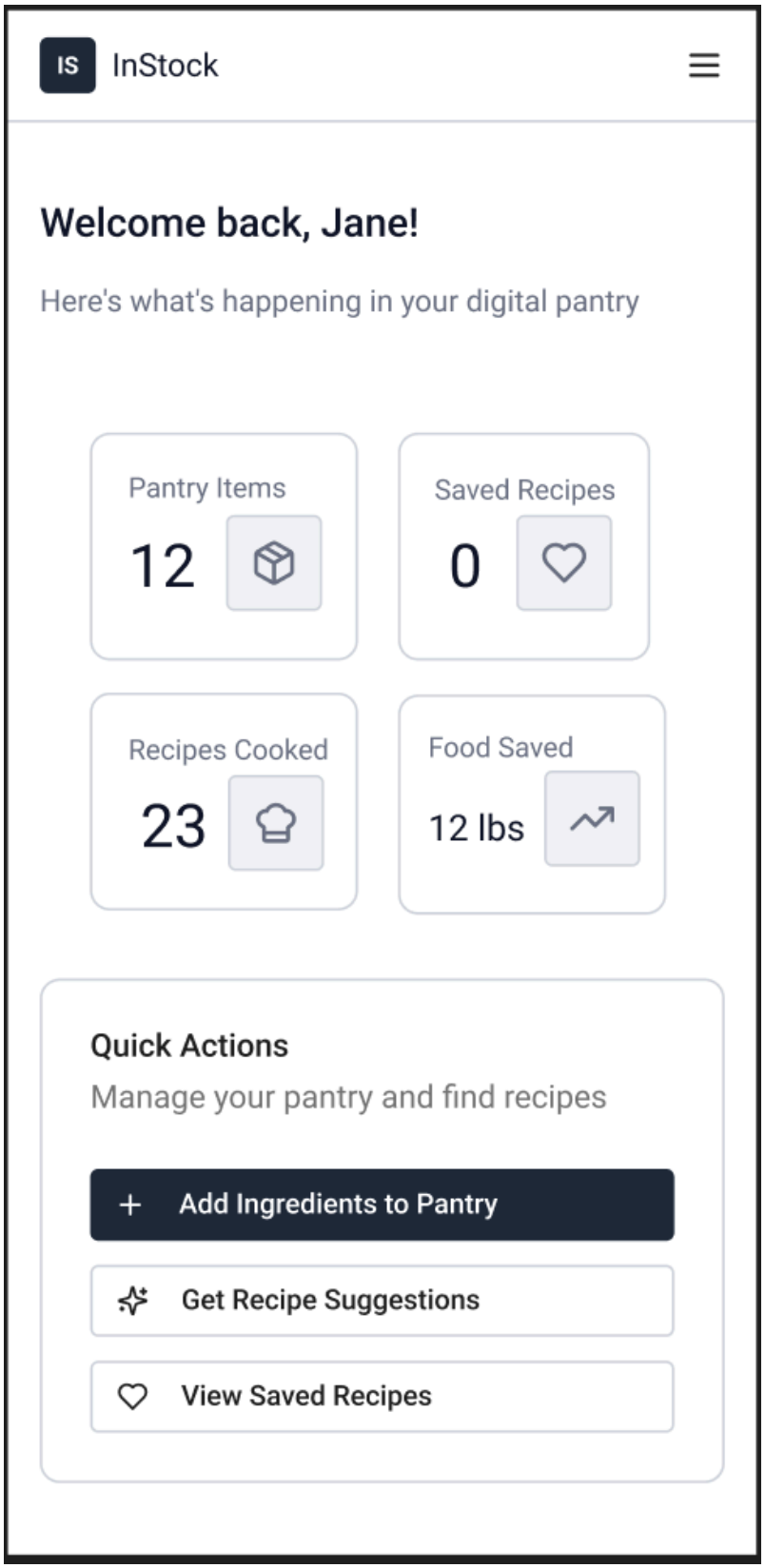
Log in Screen



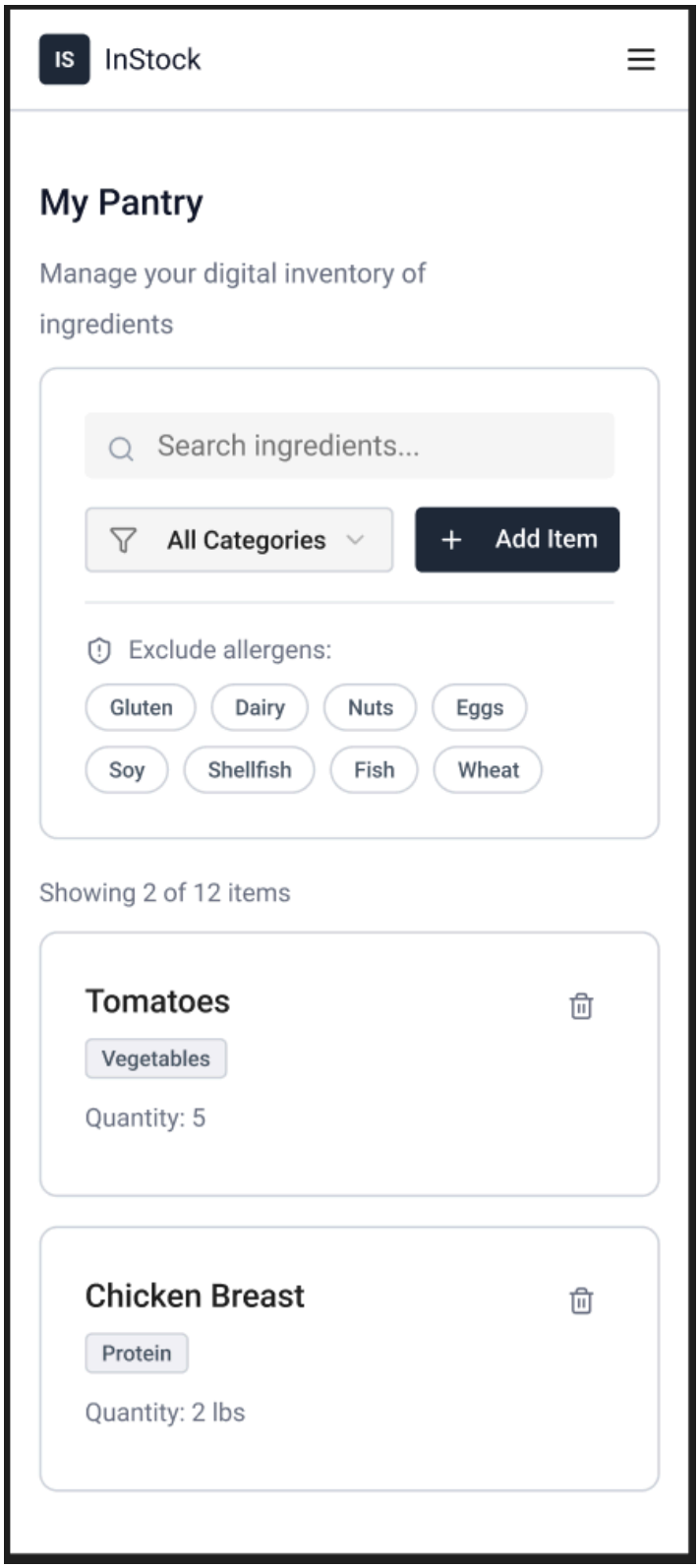
Register Screen



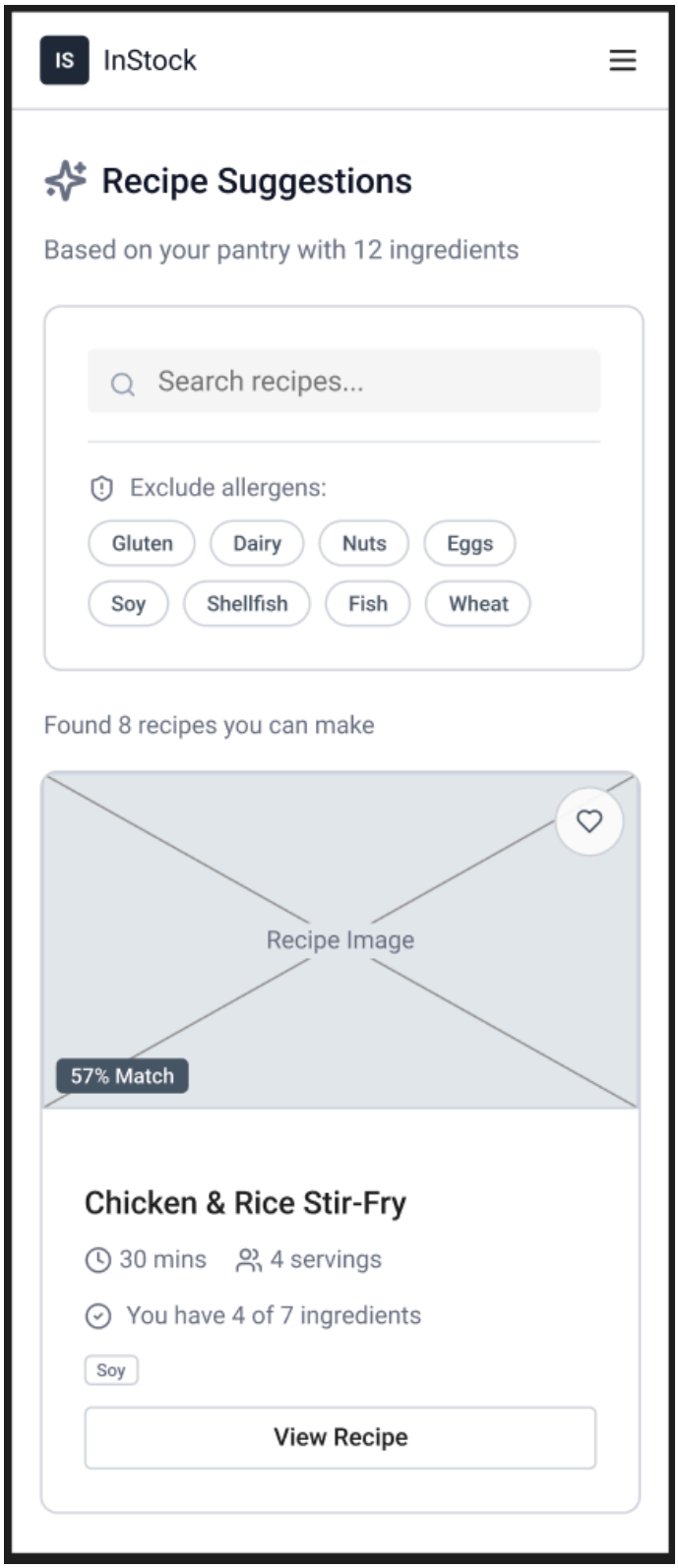
Dashboard Screen



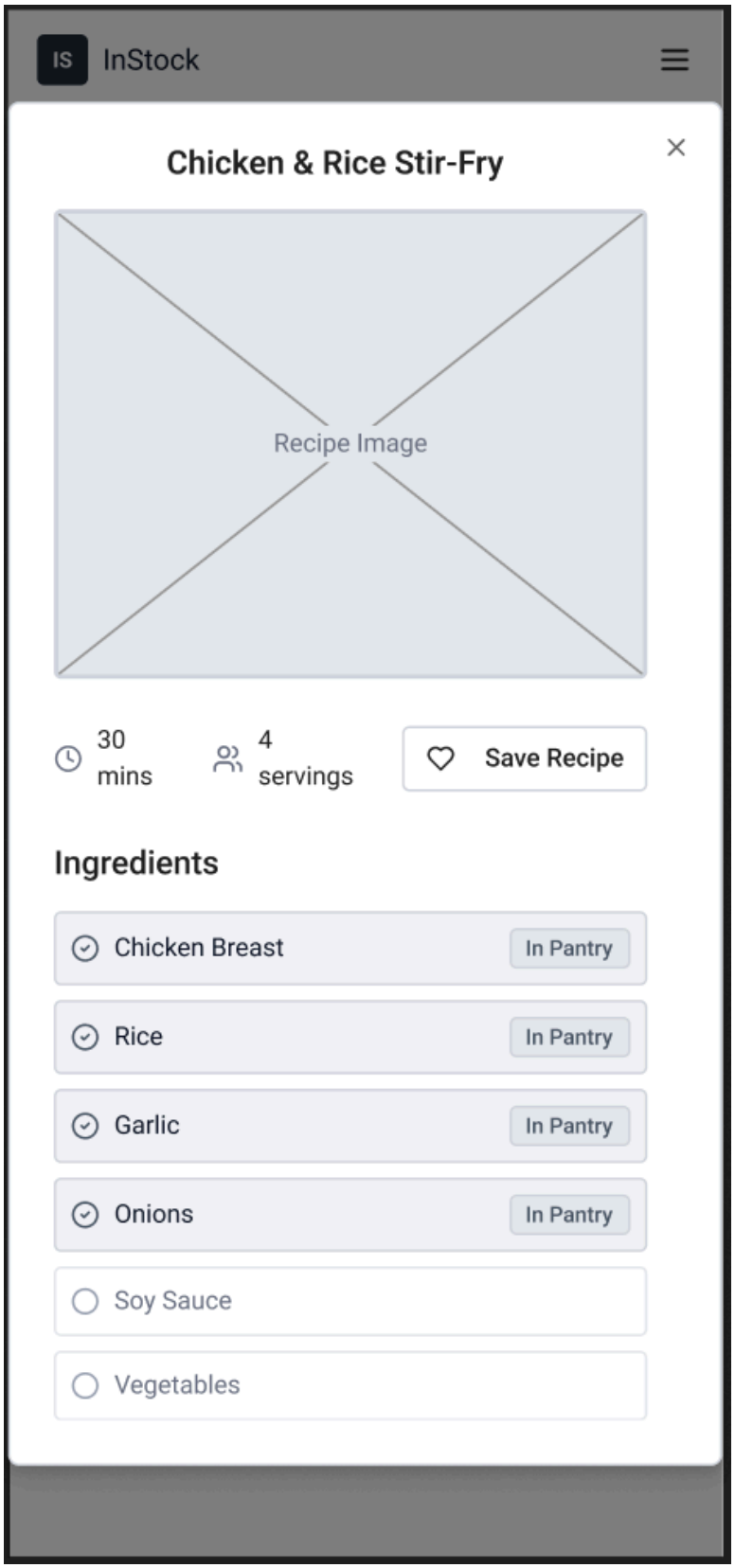
Ingredients Screen/My Pantry Screen



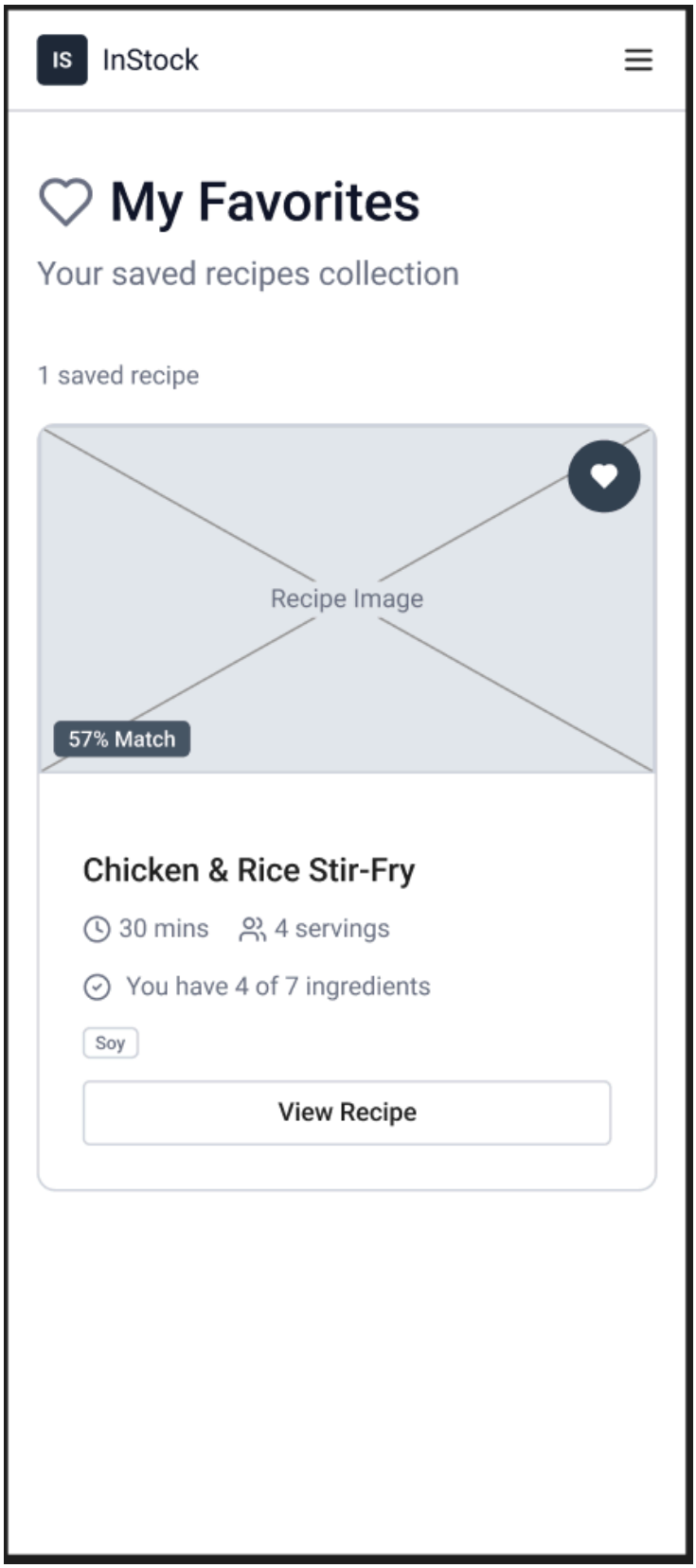
Recipe Screen



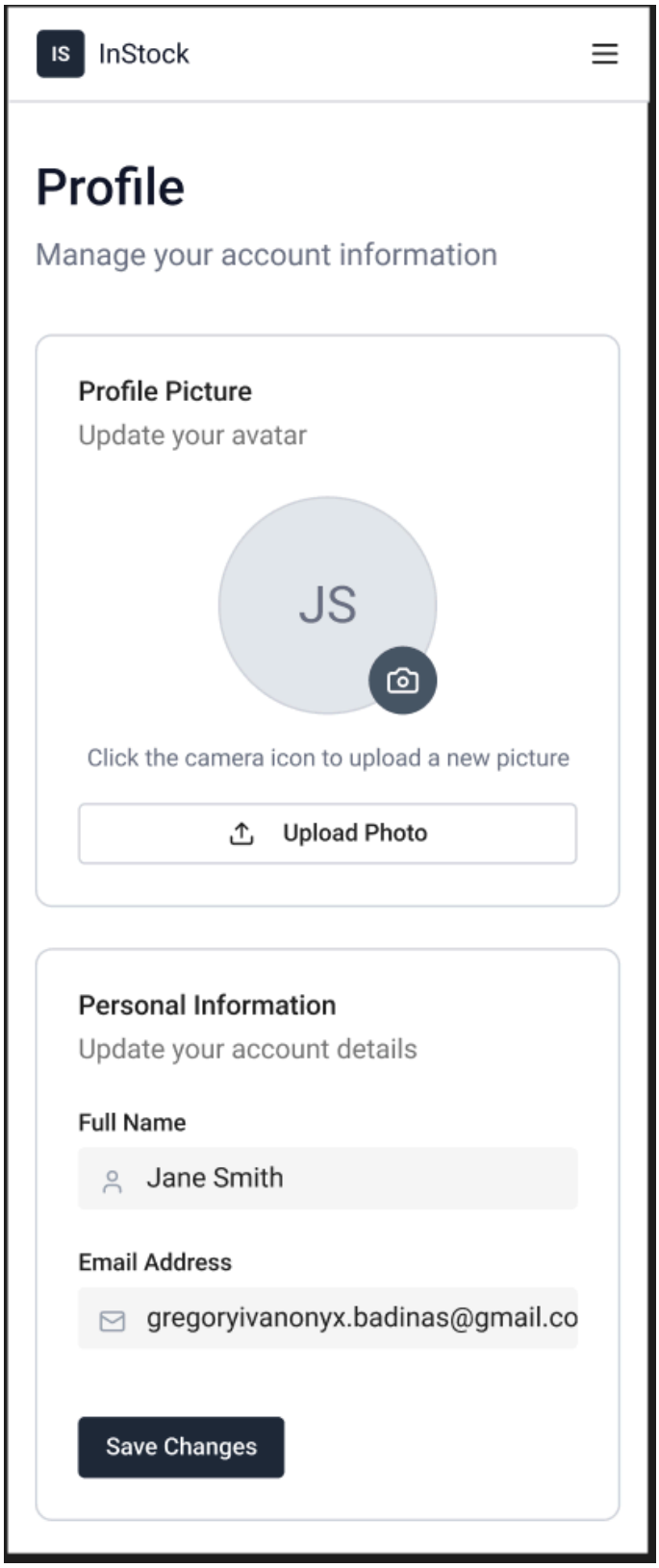
Recipe Full View Screen



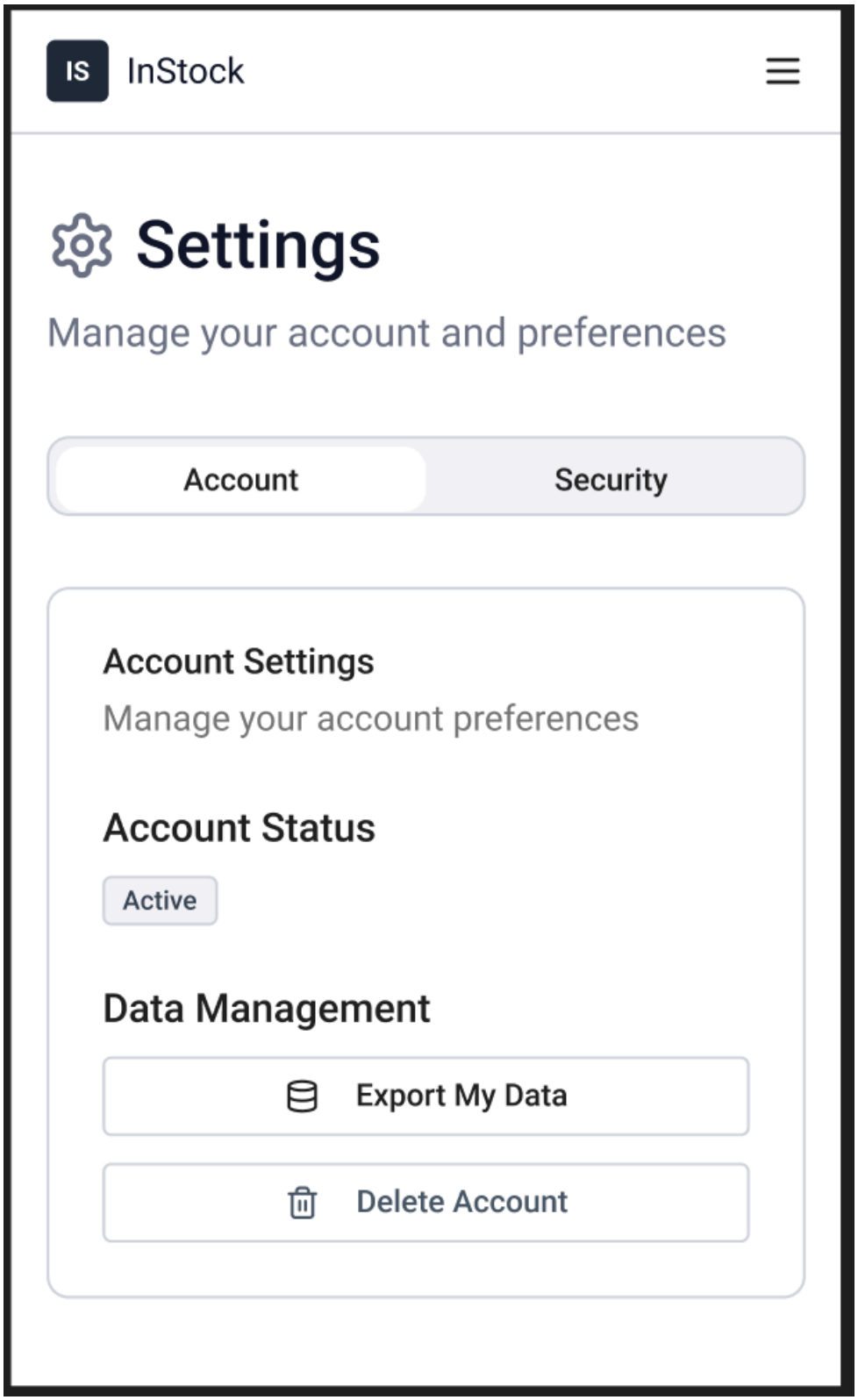
Favorites Screen



Profile Screen



Settings Screen



Mobile-Specific Features

- Touch-optimized buttons (min 44x44px)
- Offline caching
- XML-Based UI Layouts
- Role-Aware UI

Design System

- **Colors:** Primary (#2E7D32 - Fresh Basil), Secondary (#E9C46A - Harvest Gold), Success (#10B981 - Emerald, Error (#EF4444 - Tomato Red)
- **Typography:** Inter or Roboto font family with responsive sizing to maintain readability across both the React web application and the Android mobile app.
- **Spacing:** 8px grid system to maintain a clean and uncluttered visual hierarchy.
- **Components:** Standardized buttons, accessible text inputs for ingredient search, and consistent image cards for displaying the recipes fetched from the external API.
- **Responsive:** Mobile-first approach guaranteeing usability across Mobile, Tablet, and Desktop platforms.

8.0 PLAN

8.1 Project Timeline

Phase 1: Planning & Design (Week 1-2)

Week 1: Requirements & Architecture

Day 1-2: Project setup, Git repository structuring, and documentation.

Day 3-4: Complete Functional and Non-Functional Requirements, focusing on the cross-platform digital pantry concept.

Day 5-7: System architecture design (N-tier layered architecture) and UML diagramming.

Week 2: Detailed Design

Day 1-2: OpenAPI specification for backend endpoints.

Day 3-4: Relational database design (Minimum 5 tables: e.g., Users, Roles, MasterIngredients, UserPantry, FavoriteRecipes).

Day 5-6: UI/UX wireframes applying the Fresh Basil & Harvest Gold design system.

Day 7: Implementation plan finalization and "Analysis of Alternatives" review.

Phase 2: Backend Development (Week 3-4)

Week 3: Foundation

Day 1: Spring Boot setup with Java 17+ and JPA/Hibernate dependencies.

Day 2: Database configuration and core entity creation.

Day 3: Spring Security + JWT authentication implementation (including BCrypt hashing).

Day 4: User management endpoints and Role-Based Access Control (Admin vs. Regular User).

Day 5: Master Ingredient catalog CRUD operations.

Week 4: Core Business Modules & Integration

Day 1: User "My Pantry" stock management logic.

Day 2: External Public API integration (e.g., Spoonacular) for the recipe suggestion engine.

Day 3: Favorite recipes management functionality.

Day 4: File upload (User Avatar) and SMTP Email notifications (Welcome & System alerts).

Day 5: Global exception handling (P0/P2 categorization) and DTO validation.

Phase 3: Web Application (Week 5-6)

Week 5: Frontend Foundation

Day 1: React setup and routing configuration.

Day 2: Authentication pages (Login, Register) and Google OAuth 2.0 integration.

Day 3: "My Pantry" ingredient search and list management UI.

Day 4: Recipe suggestion dashboard (consuming External API data).

Day 5: "Favorites" catalog implementation.

Week 6: Complete Web Features

Day 1: User profile page with Avatar file upload testing.

Day 2: Admin Dashboard (Master ingredient list management & stats).

Day 3: UI/UX responsive design polish (Mobile, Tablet, Desktop).

Day 4: UI-level role restriction testing (hiding Admin features from Users).

Day 5: End-to-end web API integration and testing.

Phase 4: Mobile Application (Week 7-8)

Week 7: Android Foundation

Day 1: Android Studio setup using API Level 34 (Android 14), Kotlin, and strictly XML-based layouts.

Day 2: Retrofit API service layer and secure JWT storage (Android Keystore).

Day 3: Mobile Authentication screens.

Day 4: "My Pantry" interactive list with gesture support (swipe to remove).

Day 5: Ingredient search interface implementation.

Week 8: Complete Mobile App

Day 1: Recipe Suggestion UI mapping external API responses.

Day 2: Favorites management and Offline local database setup (Room/SQLite) for offline-first resilience.

Day 3: Role-Aware UI adjustments and visual polish.

Day 4: Error handling implementation (e.g., handling CORS or network fluctuations gracefully).

Day 5: Testing on emulator/physical device and APK generation.

Phase 5: Integration & Deployment (Week 9-10)

Week 9: Integration Testing

Day 1: End-to-end testing across all platforms (Web and Mobile sharing the exact same backend).

Day 2: Bug fixes, checking for "over-fetching" or "under-fetching" data.

Day 3: Security review (validating protected routes and DTOs).

Day 4: Performance testing.

Day 5: Software Design Document (SDD) final updates.

Week 10: Deployment

Day 1: Backend deployment (e.g., Render, Railway).

Day 2: React web app deployment (e.g., Vercel, Netlify).

Day 3: Mobile APK final distribution preparation.

Day 4: Final testing.

Day 5: Project submission.

Milestones:

- **M1 (End Week 2):** All design documents and API contracts complete.
- **M2 (End Week 4):** Spring Boot backend and 3rd-party integrations functional.
- **M3 (End Week 6):** React application complete and responsive.
- **M4 (End Week 8):** Android application (XML/Kotlin) complete.
- **M5 (End Week 10):** Full system deployed and successfully defended.

Critical Path:

1. JWT & Google OAuth authentication (Week 3).
2. Pantry/Ingredient relational mapping (Week 4).
3. External Recipe API integration (Week 4).
4. Mobile Retrofit integration with backend (Week 7-8).

Risk Mitigation:

- Start with the simplest working MVP of the pantry list before adding the recipe engine.
- Be mindful of External API rate limits during development; cache responses if necessary.
- Ensure CORS (Cross-Origin Resource Sharing) headers are configured early to prevent React web app connection failures.