Custom Recipe Manager Project Documentation

Course: CSE 183 Web Applications

Date: June 13, 2025

Team Members:
Dhriti Avala
Varun Golusupudi (Team Lead)
Hana Jahangiri
Trisha Sriram
Jose Valencia
Praneet Varade

GitHub Repository: github.com/ucsc2025-cse183/project-1

Table of Contents

1. Project Overview

Custom Recipe Manager is a full-stack web application that allows users to create, manage, and discover recipes. Built with a modern React frontend and py4web backend, the application provides comprehensive recipe management capabilities with advanced search functionality, nutritional information calculation, and integration with external recipe databases.

Key Capabilities

- **User Authentication**: Secure registration and login system
- **Recipe Management**: Full CRUD operations for personal recipes
- **Advanced Search**: Search by recipes or ingredients
- **Nutrition Tracking**: Automatic calculation of nutritional information
- **Public Recipe Discover**: Browse community shared recipes
- **Ingredient Database**: Comprehensive ingredient management system
- **Multi-Image Support**: Upload multiple images per recipe
- **External API Integration**: Import recipes from TheMealDB
- **Responsive Design**: Mobile friendly interface

2. Setup Instructions

Prerequisites

- Python 3.8+
- Node.js 16+
- npm
- py4web framework

Setup Steps

1. Clone the Repository

```
Shell
git clone [https://github.com/ucsc2025-cse183/project-1.git]
```

2. Install Dependencies

```
Shell
cd ..
cd class-code              # Folder with py4web dependencies  installed
nix-shell                  # Run script to enter shell
```

3. App Setup

```
Shell
cd ~/project-1             # Navigate to project directory
./start-server.sh          # Run script to start py4web server

# Access frontend here: http://127.0.0.1:8000/CustomRecipeManager/static/
```

3. Core Features

3.1 Database Schema

This backend for this application uses py4web's Database Abstraction Layer (DAL) to define and manage its relational schema. The database includes all required core tables, extended and extra credit features, ensuring support for nutrition tracking, recipe management, user interaction, and enhanced usability. They are as follows:

ingredient: Stores global ingredients shared by all users, including detailed nutrition data.

Field	*Type*	*Description*
name	string	Unique name of the ingredient
unit	string	Measurement unit (g, tsp, cup, etc.)
description	text	Short ingredient description
calories_per_unit	double	Calories per unit (kcal)
protein_per_unit	double	Protein per unit (g)
fat_per_unit	double	Fat per unit (g)
carbs_per_unit	double	Carbohydrates per unit (g)
sugar_per_unit	double	Sugar per unit (g)
fiber_per_unit	double	Fiber per unit (g)
sodium_per_unit	double	Sodium per unit (mg)
auth.signature	metadata	Tracks created_on/by, updated_on/by

recipe: Represents a public recipe created by a user, includes instructional content and metadata.

Field	*Type*	*Description*
name	string	Name of the recipe
type	string	Recipe category (Lunch, Dinner, etc.)
description	text	Detailed recipe description

image	string	Path or URL to the main image
author	reference	Links to auth_user table
instruction_steps	text	Full step-by-step instructions
servings	integer	Number of servings
auth.signature	metadata	Tracks created_on/by, updated_on/by

recipe_ingredient: A many to many linking table connecting recipes and ingredients with serving-specific quantities. This table is essential for computing total calories and nutrition per recipe.

Field	*Type*	*Description*
recipe_id	reference	Links to recipe table
ingredient_id	reference	Links to ingredient table
quantity_per_serving	double	Quantity used per serving
auth.signature	metadata	Tracks created_on/by, updated_on/by

recipe_multiple_images: Supports uploading multiple images per recipe to enhance display and user experience.

Field	*Type*	*Description*
recipe_id	reference	Links to recipe table
multi_images	upload	Additional image file
quantity_per_serving	double	Quantity used per serving
auth.signature	metadata	Tracks created_on/by, updated_on/by

contact: Allows users to send feedback or support messages via a built-in contact form. The table supports user engagement and internal message tracking.

Field	*Type*	*Description*
name	string	Sender’s name
email	string	Sender’s email address for replies
subject	string	Message subject line

message	text	Full message body
status	string	Message status (new, in, resolved)
auth.signature	metadata	Tracks created_on/by, updated_on/by

Validation and Integrity

- All fields use strong validators, including IS_NOT_EMPTY, IS_FLOAT_IN_RANGE, and IS_LENGTH, to ensure consistent and valid data.
- Tables use auth.signature to automatically record creation and update metadata.
- Referential integrity is enforced via the reference field linking recipes and ingredients.
- Shared ingredients are designed to be immutable after creation.

TheMealDB Import Support

The existing database scheme supports automatic population of recipes and ingredients from TheMealDB API through a custom function auto_import_themealdb() defined in models.py. Import logic is explained in section 4.1 in detail.

3.2 Search Functionality

The application includes a responsive ingredient search feature that allows users to find ingredients by name and view detailed nutritional information. This functionality is implemented using a React frontend and a RESTful py4web backend.

Frontend Features

- A search bar allows users to filter ingredients by name
- Search results update in real time with 300ms debounce delay
- Results are displayed 10 per page
- Pagination controls ("Previous", "Next") allow users to navigate between pages
- A summary shows how many results are displayed and the total number of matches
- If no results are found, the UI displays a helpful message and example search terms
- The layout is built with Tailwind CSS and is responsive across devices

Additional Nutritional Information Display

Each ingredient result includes a full nutritional breakdown per unit:

- Calories (kcal)
- Protein, Fat, Carbohydrates, Sugar, Fiber (g)
- Sodium (mg)

Additional nutritional information is discussed further in section 5.3.

Backend API Endpoint

GET/api/ingredients/search?query=<name>&page=<number>&limit=<number>

The backend supports case-insensitive name filtering and full pagination using the page and limit parameters. It returns a JSON response with:

- Ingredients: List of matching results
- Total: Total number of matches
- Page, limit: Pagination metadata

CORS and preflight handling are implemented via set_cors_headers() and on the OPTIONS method.

3.3 Recipe Management

The application provides comprehensive recipe management capabilities that support CRUD (Create, Read, Update, Delete) operations for user-created recipes. This functionality is implemented using a React frontend and a RESTful py4web backend.

Frontend Features

- Create recipe multi-step form that collects recipe name, type, servings, description, images, ingredients, and instructions
- Recipes can be edited or deleted by their author directly from the dashboard or recipe detail view
- The interface supports uploading a main image and up to four additional images per recipe, with real-time previews and validation
- Step-by-step instructions are managed as an ordered list allowing users to add, remove, or record steps
- All recipe fields are validated on the client-side for completeness and correctness before submission
- Recipes are displayed in a card-based layout with a summary of details, and can be expanded to display either a pop up view of the recipe or the full page of the recipe on the public recipes section

Additional Features

- Recipes can only be edited or deleted by the original author
- Nutrition information is automatically calculated and stored for each recipe based on its ingredients and quantities
- All changes to recipes are tracked with metadata via auth.signature

Backend API Endpoints

POST /api/recipes: Creates a new recipe
GET /api/recipes: Retrieves a user's own recipe
GET /api/recipes/<id>: Retrieves the specified recipe
PUT/PATCH /api/recipes/<id>: Updates the recipe fields
DELETE /api/recipes/<id>: Deleted the user's specified recipe

3.4 Ingredient Management

The ingredient management system provides an extensive foundation for tracking, searching, and utilizing ingredients across all recipes. It is designed to ensure data consistency, accuracy regarding nutritional content, and ease of use for users.

Frontend Features

- Users can access ingredient details - including units, description, and nutritional breakdown - when creating or editing their recipes
- The interface supports pagination and displays the total number of matching ingredients
- Users can select ingredients from the global database and specify quantity per serving when creating or editing their recipes
- User can add their own custom ingredients with full nutritional information if the ingredient is not present

Additional Features

- All ingredient entries include detailed nutritional data (calories, protein, fat, carbs, sugar, fiber, sodium) and are globally shared across all users
- Ingredient data is validated for completeness and accuracy during creation so once created, ingredients are immutable
- The system supports automatic population of ingredients from TheMealDB API, with estimated nutritional values assigned during import
- Ingredient units are standardized and appended dynamically in the UI
- Any missing nutritional values default to zero

Backend API Endpoints

POST /api/ingredients: Creates a shared ingredient
GET /api/ingredients/<id>: Retrieves the specified ingredient

4. Full Grade Requirements

4.1 TheMealDB API Integration

The application supports the automatic population of recipes and ingredients from TheMealDB API, to provide users with a sufficient amount of initial data and to enhance the overall experience of using the application. This integration is implemented as an admin-only backend feature which makes sure that only authorized users can trigger large imports.

Backend Implementation

A dedicated endpoint is provided for importing data:

```python
@action('api/admin/import-themealdb', method=['POST'])
@action.uses(db, session, auth.user)
def import_themealdb():
       # Bulk import from TheMealDB API with error handling
```

The endpoint is protected by authentication and restricted to admin users, verified by checking the logged-in user's email address. The import logic is defined in the auto_import_themealdb() function in models.py.

Import Process

The import logic is executed on the first application startup and uses the existing recipe, ingredient, and recipe_ingredient tables without requiring any additional schema changes. It is as follows:

- Ingredients are pulled from the API's ingredient list endpoint and inserted with default units (g) and heuristically estimated nutritional values (calories, protein, fat, carbs, sugar, fiber, and sodium).
- Recipes are fetched by category (e.g. Chicken, Seafood, Dessert) and mapped into the schema with fields for name, type, instructions, image, and a default of four servings. The author field is assigned to a system-generated admin user (admin@themealdb.com)
- Ingredient quantities are parsed from strings (e.g. "1 cup", "3 tbsp") using a custom parse_quantity() function that estimates gram-equivalents.
- Recipe-ingredients linking is handled via the recipe_ingredient table using the parsed quantities

Value

The import process runs automatically on first startup. It populates the application with real recipe and ingredient data from TheMealDB using only the existing schema. The imported

content is fully usable for search, recipe display, nutrition calculation, and editing (where allowed), providing immediate value to the user experience without requiring manual data entry.

Security and Access Control

Only authenticated admin users can access the import endpoint and all imported data is validated before insertion. The system is guarded to prevent duplicate imports and to reinforce referential integrity between recipes and ingredients.

4.2 Professional UI

The application features a modern and professional user interface which was designed to deliver both an intuitive and visually appealing experience on all devices. The frontend is built with React and styled using a combination of Material-UI and Tailwind CSS, in order to provide a consistent and adaptable design.

Design Principles and Responsiveness

The application uses a clean, card-based layout for recipes, ingredient lists, and dashboards. This makes information easy to scan over and interact with.

The responsive design principles are applied throughout the application so layout components automatically adjust to the different screen sizes. This creates an adaptable design that allows for comfortable use on all devices.

The color palette chosen is minimal, primarily consisting of green and white, which allows for contrast and a modern look. In contrast, the use of emoticons throughout the application creates a more engaging environment but still while preserving the simple aesthetic.

User Flow

The navigation bar provides quick and easy access to the core features of the application, from information on Mealzi, such as the about section or contact form to user-specific buttons, such as the personal dashboard or login/registration.

The landing page features a prominent hero section, feature highlights, and various calls to action for registration and login.

Once a user is signed in, the recipe creation form guides the user through creating a recipe with real-time validation and progress indicators. When the recipe is successfully created, the user is redirected back to their personal dashboard, where they can interact with their created recipes or navigate to the public recipes page.

Interactive Features

The application provides users with real-time feedback through loading symbols, inline error/success messages, and toast notifications. Users are provided with this feedback throughout their interaction with the interface, such as during the sign-in process, completing the creation of a recipe, or confirming the deletion of a recipe. These features allow for the user to feel guided through their experience and more confident.

The interface for uploading images supports the drag-and-drop method, previews, and validation for the files type and size.

When interacting with recipe cards, there are interactive elements such as expandable details, the image gallery with lightbox functionality, and the adjustment for serving size buttons.

Accessibility

All of the interactive elements are keyboard accessible and have corresponding ARIA labels, and all form fields have clear labeling.

4.3 Public Search API

The application provides a set of public RESTful API endpoints that allow users and external applications to search and browse recipes and ingredients without authentication. These endpoints are designed for flexibility and performance, supporting advanced filtering and pagination.

Key Features

- Recipe Search: Users can search for public recipes by name, type, or ingredient list, with support for both “match all” and “match any” for ingredients. These results are paginated for efficient browsing.
- Ingredient Search: The public ingredient API allows searching by name and returns detailed nutritional information for each result.
- CORS Support: All public endpoints are configured with global CORS headers, enabling safe access from the frontend and third-party applications.
- Structured Responses: APIs return JSON responses with pagination metadata and detailed data for each recipe or ingredient.

Backend API Endpoints

GET /api/recipes/public: Browse all public recipes
GET /api/recipes/search: Search recipes by name and type
GET /api/recipes/search_by_ingredients: Search recipes by ingredient list
GET /api/ingredients/search: Search ingredients by name

These endpoints power the application’s advanced search interfaces and support integration with external tools. Creating, editing, and deleting operations remain protected and require authentication, making sure that only non-sensitive and public data is exposed to all users.

4.4 Secure Editing Logic

The application enforces strict security and access control for all editing operations to protect user data and maintain integrity. Both the backend and frontend layers work together to make sure that only authorized users can edit or delete content.

Backend Implementation

All sensitive backend endpoints, such as recipe creation, editing, deletion, and image uploads, are protected using @action.uses(auth.user), ensuring that only authenticated users can access these routes. This author-based access control is enforced; only the original creator of a recipe or ingredient can update or delete it. It is implemented by comparing the author field of the record with the current user's id at runtime. Additionally, administrative actions, such as importing data from TheMealDB, are restricted to the designated admin accounts, verified by the user's email address.

Frontend Features

The frontend hides or disables editing and deleting controls for users who are not the original author of a recipe or ingredient via protected routes and UI elements that ensure that only authenticated users can access personal dashboards and recipe management features.

All user input is validated client-side before submission to prevent any malicious data from reaching the backend.

Input Validation and Error Handling

The backend validates all incoming data for required fields, correct data types, and user permissions. Invalid requests result in structured error messages and appropriate HTTP status codes (e.g., 401 Unauthorized, 403 Forbidden, 400 Bad Request). An instance of this is as follows:

```python
if recipe.author != auth.current_user['id']:
    response.status = 403
    return {"error": "You are not the author of this recipe"}
```

Session and CSRF Protection

Sessions are managed securely via cookies, and sensitive operations are only permitted within an authenticated user context. CSRF protection is implemented to prevent unauthorized actions from outside sources on important operations like editing or deleting data.

4.5 Automatic Nutrition Calculation

The application automatically calculates and displays nutritional information for each recipe based on its ingredients and specified quantities. This feature is fully integrated into both the backend and frontend, providing users with immediate feedback on the profile of their recipes.

Backend Implementation

When a recipe is created or updated, the backend aggregates nutritional values - including calories, protein, fat, carbohydrates, sugar, fiber, and sodium - from each ingredient used in the recipe. The system multiplies each ingredient's nutritional values by the quantity specified per serving, then sums these values across all ingredients to compute the total nutrition for the recipe. Nutrition is calculated both per serving and for the entire recipe, and these values are stored and returned in API responses for use in the frontend. All calculations are performed automatically and updated in real time whenever a recipe's ingredients or serving size are changed.

Frontend Features

The frontend displays a nutrition summary for each recipe, showing calories, protein, fat, carbohydrates, sugar, fiber, and sodium per serving and for the total recipe. Nutrition information is visible on recipe cards, full recipe views, and during the recipe creating/editing process, allowing users to be aware of these values as they build or modify recipes. The UI uses clear units and formatting to keep nutritional data easy to understand.

Value

This feature allows for users to instantly see the nutritional impact of their ingredient choices and serving sizes, supporting healthy cooking and dietary tracking. The feature is available for both user-created and imported recipes, so that users can easily understand and interact with one another's recipes.

5. Extra Credit Features

5.1 Multiple Images Per Recipe

The application allows users to upload and display multiple images for each recipe, enhancing the visual presentation and making recipes more engaging.

Backend Implementation

The database includes a recipe_multiple_images table, which links each recipe to up to four additional images, in addition to the main image. Images are uploaded through secure API endpoints and stored in the backend's uploads directory. The backend enforces a maximum of five images per recipe (one main image and up to four additional images) and validates file type and size during upload. When a recipe is updated, any images removed by the user are deleted from both the database and the filesystem to maintain data integrity.

Frontend Features

The frontend provides an intuitive interface for uploading, previewing, and managing multiple images. The recipe creation form allows users to upload a main image and up to four gallery images when originally creating a recipe or editing one, with real-time previews and drag-and-drop support. The UI provides clear feedback on upload status, file limits, and validation errors.

Value

On recipe detail pages, all images are displayed in a responsive gallery with lightbox functionality, allowing users to view larger versions and easily browse through all images. Multiple images help users better present their recipes, show preparation steps, and make their creations more appealing to others. This feature helps users better communicate their cooking process, increases the appeal of shared recipes, and supports both user-created and imported recipes for a consistent experience.

5.2 Search by Ingredients

The ingredient-based search feature allows users to discover recipes that include specific ingredients they have on hand or wish to use. This makes it easier for users to find relevant recipes, reduce food waste, and plan meals based on available ingredients.

Frontend Features
The frontend offers an intuitive interface for selecting multiple ingredients from the global ingredient database, allowing users to toggle between “match all” and “match any” modes to refine their search results.

Ingredient-based search can be combined with other filters, such as recipe name or type, for more targeted discovery. Search results update in real time as filters are applied, and are displayed in a paginated, card-based layout with key recipe details. This feature helps users quickly find recipes that fit their available ingredients, dietary needs, or preferences, making meal planning more flexible and reducing food waste.

Backend API Endpoint
GET /api/recipes/search_by_ingredients

The endpoint accepts a list of ingredient id’s and supports both “match all” and “match any” logic. Meaning recipes that contain all selected ingredients or at least one selected ingredient.

This endpoint can be combined with additional filters, such as recipe name or type, and supports pagination for efficient browsing of large result sets. The search logic is optimized to efficiently search the database and return only recipes that meet the specified criteria, so it provides both fast and accurate results even as the dataset grows.

Value
Users can quickly find recipes that fit their dietary needs, preferences, or available pantry items. The search results display recipe cards with key details, and users can further explore recipes by viewing full instructions, images, and nutrition information. This feature enhances the flexibility and usefulness of the application, like for everyday cooking with what little or random ingredients one may have at their disposal.

5.3 Additional Nutritional Information

To provide users with a more complete understanding of ingredients, the application supports additional nutritional data beyond just calories. This extra data is stored in the database, displayed in the UI, and included in all relevant API responses.

Database Support

The ingredient table includes the following nutritional fields, all stored per unit:

- calories_per_unit
- protein_per_unit
- fat_per_unit
- carbs_per_unit
- sugar_per_unit
- sugar_per_unit
- fiber_per_unit
- sodium_per_unit

These values are entered when a new ingredient is created and remain immutable after insertion.

Frontend Features

The frontend displays the nutritional breakdown for each ingredient in the ingredient search table. Units are automatically appended using a helper function, and missing values default to zero for a consistent and clean layout.

Backend Implementation

Both the ingredient search (GET /api/ingredients/search) and recipe-related endpoints include all nutritional fields in their JSON responses. These values are used in:

- Ingredient detail listings
- Recipe nutrition calculations (per serving and total)

5.4 Automatic Ingredient Searching

Automatic ingredient searching streamlines the process of finding and selecting ingredients for recipe creation and editing. This makes the application more user-friendly and also more efficient.

Frontend Features

The ingredient search bar allows users to filter ingredients by name as they type, with results updating in real time using a 300ms debounce delay to minimize unnecessary requests. Search results are displayed 10 per page, with pagination controls (“Previous”, “Next”) for easy navigation. Each result includes the ingredient’s name, unit, and a full nutritional breakdown per unit (calories, protein, fat, carbohydrates, sugar, fiber, sodium).

The interface is responsive and built with Tailwind CSS, ensuring usability across devices. However, if no results are found, the UI displays a helpful message and helps guide the user.

Backend API Endpoint

GET /api/ingredients/search?query=<name>&page=<number>&limit=<number>

This endpoint supports case-insensitive name filtering and full pagination using the page and limit parameters. The response includes a list of matching ingredients, total number of matches, and pagination metadata. All ingredient data returned includes detailed nutritional information.

CORS headers and preflight handling are implemented via set_cors_headers() and the OPTIONS method, allowing safe access from the frontend and external applications.

Value

Automatic ingredient searching reduces manual effort, speeds up recipe creation, and helps users make informed choices by providing instant access to nutritional data. This feature is tightly integrated with the recipe management workflow, supporting both the selection of existing ingredients and the addition of new custom ingredients if needed.

5.5 Public Ingredient API

The application implements a public API for ingredient data, making the ingredient database accessible not only to users within the app but also to external developers.

Open Access and Integration
The public API allows anyone to search and retrieve ingredient information without authentication. Developers can use this API to build new features or external apps that leverage the comprehensive ingredient database and nutritional data.

Frontend Features
Ingredient data is available throughout the application, including in search, recipe creation, and ingredient exploration interfaces. The UI presents ingredient details in a clear, paginated format, with nutritional information and units displayed for each entry.

Backend API Endpoint
GET /api/ingredients/search?query=<name>&page=<number>&limit=<number>

For technical details on the endpoint, see section 5.4.

Value
By making ingredient data publicly accessible, the application promotes transparency and encourages users to be creative. It extends its utility beyond recipe management to a wide range of health and nutrition related applications.

6. Technical Architecture

6.1 Backend Structure

This application's backend is built using py4web, a lightweight Python web framework that provides routing, database access, authentication, session handling, and RESTful API design. It supports all core functionality, including user authentication, recipe and ingredient management, nutrition computation, image uploads, contact messaging, search, and integration with TheMealDB API. It also serves the React frontend with client-side routing support.

Architecture Overview

```
Unset
backend/
├── apps/
│   └── CustomRecipeManager/
│       ├── controllers.py       # API endpoints and business logic
│       ├── models.py            # Database schema definitions
│       ├── settings.py          # Configuration
│       ├── static/              # Frontend build files
│       └── uploads/             # User-uploaded images
```

Key Modules and Responsibilities
The backend is organized into several key modules:

models.py defines five main tables:

- ingredient: Shared ingredient entries with unit, description, and detailed nutrition (calories, protein, fat, carbs, sugar, fiber, sodium)
- recipe: User-authored recipes with name, type, description, instructions, servings, and image fields
- recipe_ingredient: Links recipes to ingredients, including quantity per serving
- recipe_multiple_images: Allows up to four additional images per recipe
- contact: Stores user-submitted messages and status

It also includes auto_import_themealdb(), which imports recipes and ingredients from TheMealDB API, assigns estimated nutrition, and avoids duplicates.

controllers.py defines backend logic via @action routes. It includes:

- Full CRUD for recipes and ingredient creation
- Nutrition calculations
- Authentication and session handling

- Recipe search (by name, type, or ingredients)
- Contact form submission
- Admin-only TheMealDB import
- Static frontend serving and CORS support

common.py, settings.py, and __init__.py handle configuration, authentication setup, app registration, and global services.

uploads/ stores uploaded images securely. Access is controlled via a dedicated download route.

API Design and Routing

All backend functionally is exposed via RESTful JSON APIs, organized by domain and implemented using @action decorators. The API supports both application/JSON and multipart/form data, and CORS headers and OPTIONS handlers are applied globally.

APIs are provided for:

- User authentication (register, login, logout, session check)
- Ingredient management (add, search with pagination)
- Recipe management (create, view, update, delete, image upload)
- Recipe search by name, type, or ingredients
- Contact form submissions
- Admin-only data imports from TheMealDB
- Securing serving of upload files and frontend routes

A complete list of routes is documented in Section 9: API Documentation.

Authentication and Access Control

Authentication is managed using py4web's built-in auth system. Sessions are stored securely via cookies. Protected endpoints use @action.uses(auth.user) to enforce login requirements.

Users may only update or delete their own recipes, verified through a comparison of recipe.author and auth.current_user['id']. Admin-only functionality, such as importing from TheMealDB, is restricted by checking the authenticated user's email address.

Business Logic and Features

The backend implements key business logic to ensure consistent data handling:

- Nutrition Calculation: Recipes automatically compute total and per-serving values for calories, protein, fat, and carbs.
- Ingredient Immutability: Once created, ingredients cannot be edited or deleted.

- Image Management: Recipes support one main image and up to four additional images. Uploads are validated, and unused images are deleted during updates.
- Advanced Search: Recipes can be searched by name, type, or ingredients, with both “match any” and “match all” logic and pagination support.
- Contact Messaging: Messages submitted through the contact form are stored in the database and may optionally trigger email notifications to admins and users.
- TheMealDB Integration: Recipes and ingredients can be imported using an admin-only endpoint. The system assigns default nutrition values, downloads images, and prevents duplicate entries.

Supporting Utilities and Static Serving

A helper script (import_mealdb.py) supports importing data from TheMealDB and complements the admin API. Uploaded images are served securely through /uploads/<filename>. The React frontend is delivered through static route handlers like/, /static/, and /static/<route>.

6.2 Frontend Structure

The frontend of Custom Recipe Manager is a modern, single-page application built with React and powered by Vite for fast development and optimized builds. It provides a responsive, user-friendly interface for all recipe management features, integrating with the py4web backend via RESTful APIs.

Architecture Overview

```
Unset
frontend/
├── src/
│   ├── components/              # Reusable UI components
│   ├── pages/                         # Route-based page components
│   ├── services/                      # API service layer
│   ├── hooks/                   # React hooks for state management
│   ├── context/                       # Context providers
│   ├── config.js                      # Application configuration
│   └── App.jsx                        # Main application component
├── public/                            # Static assets
└── dist/                              # Production build output
```

Key Modules and Responsibilities
The frontend is organized into several key modules:

components/ contains all 22 reusable UI elements:

- RecipeDetailsSection.jsx: Displays detailed recipe information
- RecipeCard.jsx: Card layout for recipe previews
- RecipeDetailModal.jsx: Modal for viewing recipe details
- Navbar.jsx: Top navigation bar
- FeaturesSection.jsx: Highlights app features
- Footer.jsx: Application footer
- HeroSection.jsx: Prominent landing page section
- AboutSection.jsx: Mealzi information section
- CTASection.jsx: Call-to-action prompts
- RecipeDashboardHeader.jsx: Dashboard header for recipes
- IngredientSearchInput.jsx: Input for searching ingredients
- RecipeList.jsx: Lists recipes
- SelectedIngredientsList.jsx: Displays selected ingredients
- PromoBannerCard.jsx: Displays banners with image, title, description
- ProtectedRoute.jsx: Route protection for authenticated pages

- RecipeForm.jsx: Multi-step form for creating/editing recipes
- HowItWorksSection.jsx: Explains app workflow
- IngredientSearch.jsx: Ingredient search interface
- IngredientSearchBar.jsx: Search bar for ingredients
- InstructionsSection.jsx: Displays recipe instructions
- NutritionSummary.jsx: Shows nutrition information
- AuthModals.jsx: Authentication (login/register) modals

pages/ contains seven top-level views mapped to application routes:
- CreateRecipePage.jsx: Page for creating new recipes
- ShareableRecipe.jsx: Publicly shareable recipe view
- RecipeDashboard.jsx: User's recipe dashboard
- PublicRecipes.jsx: Browse/search public recipes
- LandingPage.jsx: Main landing page
- DashboardPage.jsx: General user dashboard

services/ implements all API calls using Axios, abstracting backend communication for authentication, recipe and ingredient management, search, and image uploads. Centralizes error handling and token management.

config.js stores environment specific settings, such as API base URLS, pagination defaults, and feature flags.

App.jsx serves as the root component, setting up React Router for client-side navigation and providing user session, theme, etc.

UI/UX and Styling

The interface is styled using Material-UI to provide a professional look. The layout is responsive and adapts to all device sizes, with touch-friendly controls. Real-time feedback is provided to the user through loading screens and error/success messages. The UI is discussed in more detail in section 4.2.

API Integration and Routing

Client-side routing is managed by React Router and the protected routes make sure that only authenticated users can access personal dashboards and recipe management features, such as editing or deleting.

Public routes allow for anyone to browse and search community recipes. All core features - including authentication, recipe and ingredient management, search, nutrition display, and image uploads - are powered by RESTful API calls to the py4web backend.

The frontend manages CORS, session state, and secure token storage, supporting advanced, real-time search and filtering pagination.

Build and Deployment

The frontend uses Vite for development and optimized production builds. Production assets are output to the dist/ directory and served by the backend, while static assets (images, icons, etc.) are managed in the public/ directory and referenced throughout the app.

6.3 Security Features

The backend structure includes multiple layers of security to ensure authenticated access, data protection, and safe handling of user-generated content. These safeguards are implemented throughout the system using py4web's built-in tools and additional logic in key routes.

Route-Level Protection
All sensitive endpoints - including recipe creation, editing, deletion, and image uploads - are protected using @action.uses(auth.user). This ensures that only authenticated users can access these routes. Unauthenticated requests return a 401 Unauthorized response.

Author-Based Access Control
Only the original author of a recipe is allowed to modify or delete it. This is enforced by comparing recipe.author with auth.current_user['id'] at runtime. Unauthorized access attempts return a 403 Forbidden response.

Admin-Only Actions
Critical administrative functions, such as importing recipes from TheMealDB, are restricted to a specific user account. This check is performed using the logged-in user's email address (e.g., admin@example.com).

Input Validation and Error Handling
All API endpoints validate required fields, expected data types, and user permissions. Invalid request results in structured error messages and appropriate HTTP status codes.

Secure File Upload and Access
Uploaded images are stored in a server-controlled uploads/directory and are not directly accessible via the web server. Instead, they are served through a secure route (/uploads/<filename>). Each recipe is limited to a total of five images. During recipe edits, old or unused images are deleted from both the database and filesystem to prevent clutter or leakage.

CORS and Session Security
Global CORS headers are configured via set_cors_headers() to allow cross-origin requests from the frontend. Sessions are managed securely via cookies, and sensitive operations are only permitted within an authenticated user context.

7. API Documentation

Authentication APIs

Method	*Endpoint*	*Description*
POST	/api/auth/register	Registers a new user and starts a session
POST	/api/auth/login	Authenticates the user and stores the session cookie
POST	/api/auth/logout	Logs out the user and clears the session
GET	/api/auth/user	Returns the currently authenticated user

Ingredient APIs

Method	*Endpoint*	*Description*
POST	/api/ingredients	Creates a new shared ingredient, author access required
GET	/api/ingredients/search	Searches ingredients by name with pagination

Recipe APIs

Method	*Endpoint*	*Description*
POST	/api/recipes	Creates a new recipe with ingredients and optional images
GET	/api/recipes	Retrieves recipes created by the logged-in user
GET	/api/recipes/public	Retrieves all public recipes
GET	/api/recipes/<id>	Retrieves a specific recipe with ingredients, images, and nutrition
PUT/PATCH	/api/recipes/<id>	Updates recipe fields and images, author access required
DELETE	/api/recipes/<id>	Deleted a recipe and associated data, author access required
POST	/api/recipes/<id>/upload_images	Uploads one or more additional images to a recipe

Search APIs

Method	*Endpoint*	*Description*
GET	/api/recipes/search	Searches public recipes by name or type, supports pagination

GET	/api/recipes/search_by_ingredients	Searches recipes by ingredient list, supports match-all/any and filters

Contact API

Method	*Endpoint*	*Description*
POST	/api/contact	Submits a contact form message, stores it and supports optional email notifications

Admin API

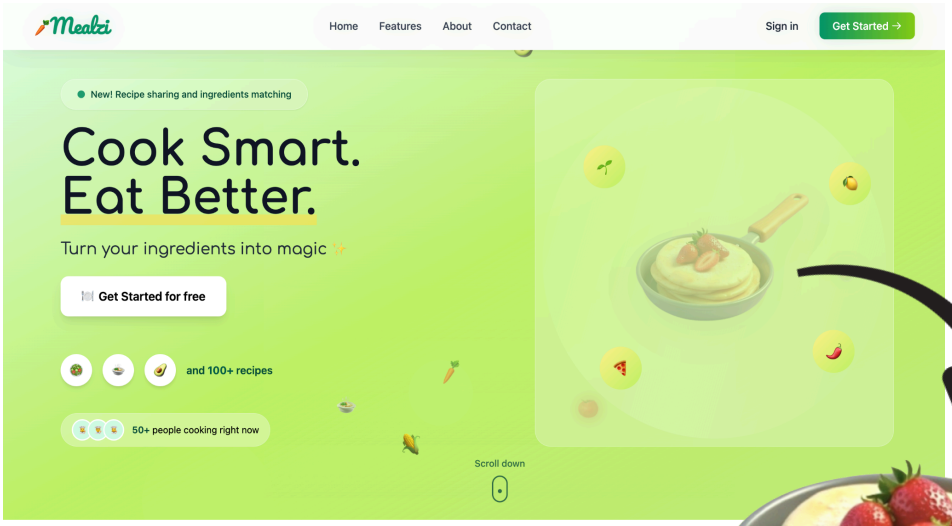
Method	*Endpoint*	*Description*
POST	/api/admin/import-themealdb	Imports recipes and ingredients from TheMealDB, restricted to admin user

Static and File Serving

Method	*Endpoint*	*Description*
GET	/uploads/<filename>	Serves uploaded image files securely from the backend
GET	/	Serves the main React app from the root route
GET	/static/<route>	Serves frontend routes for navigation

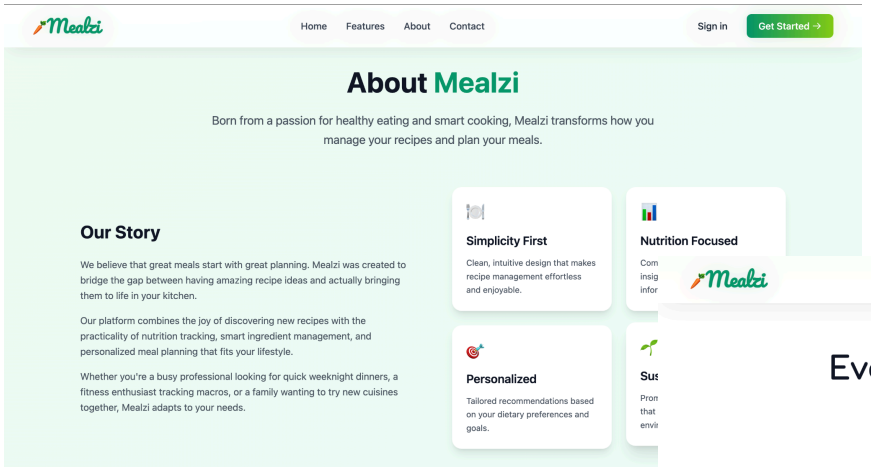
8. Screenshots

Landing Page

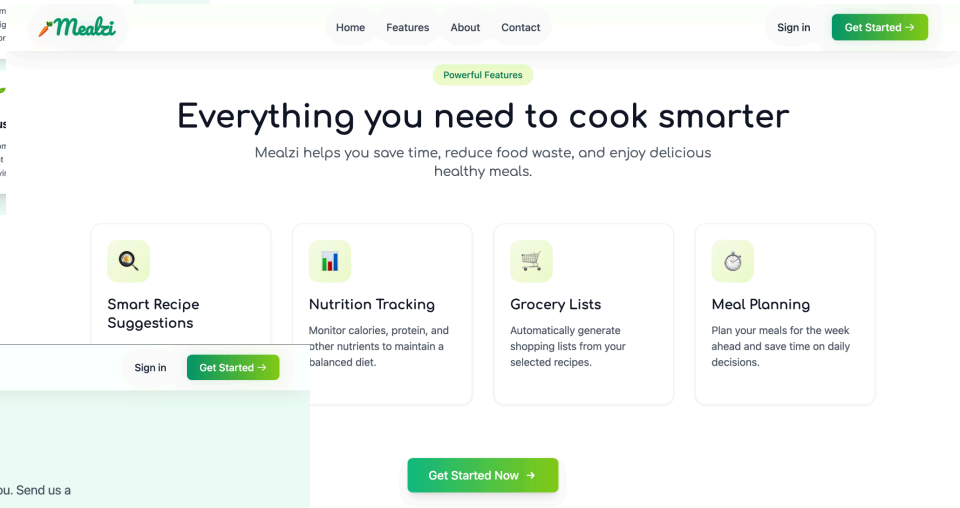


← Home

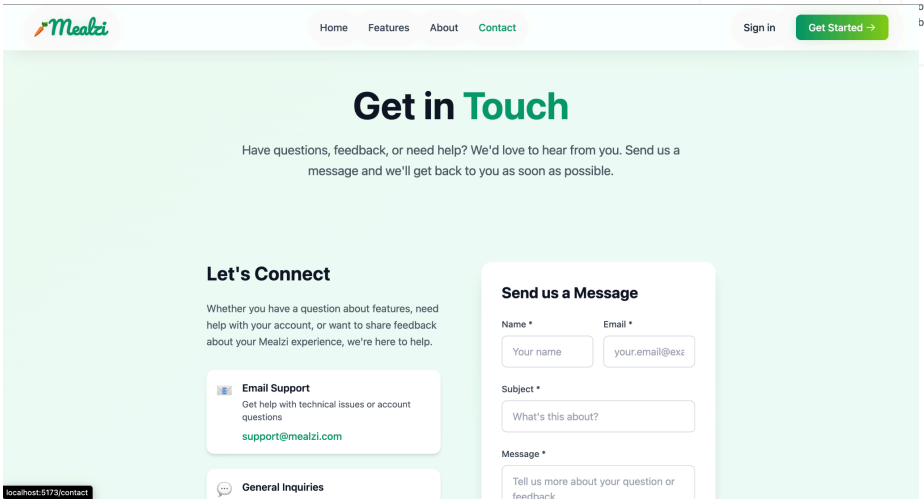
Interactive Icon →



← About Section



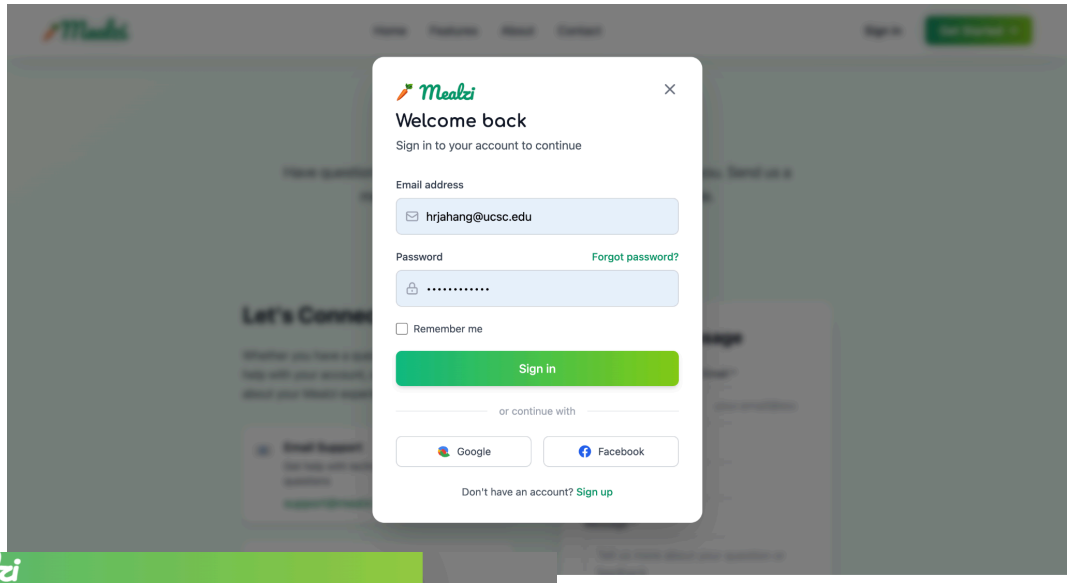
Features Section →

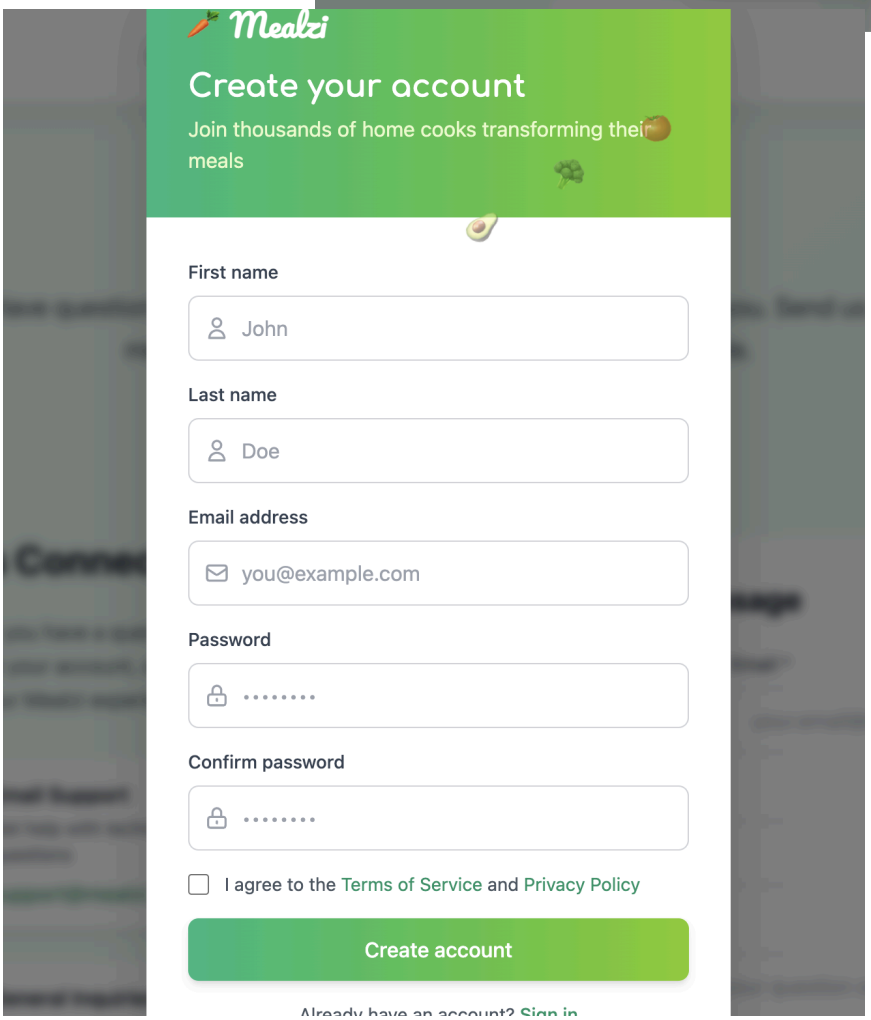


← Contact Page

Authentication

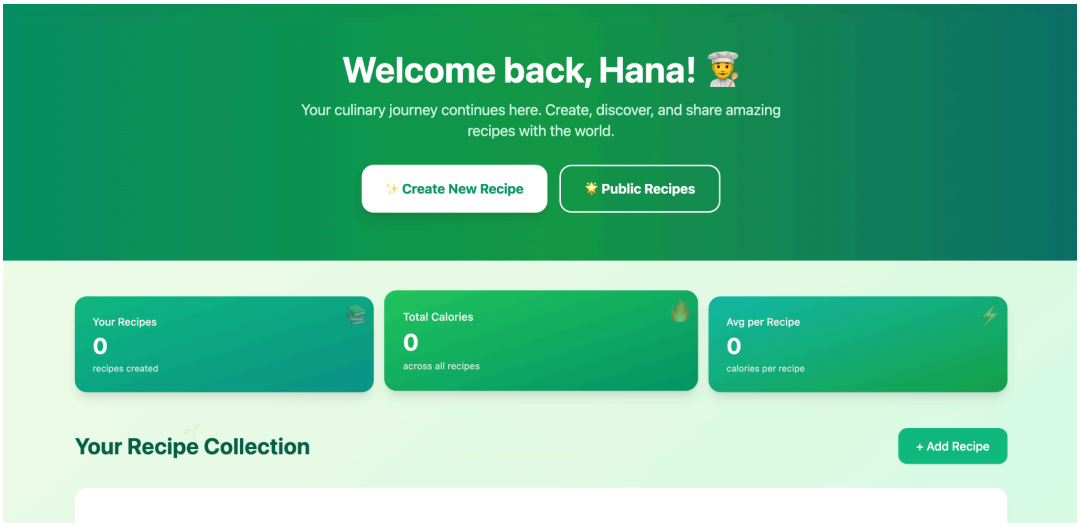
Login screen →





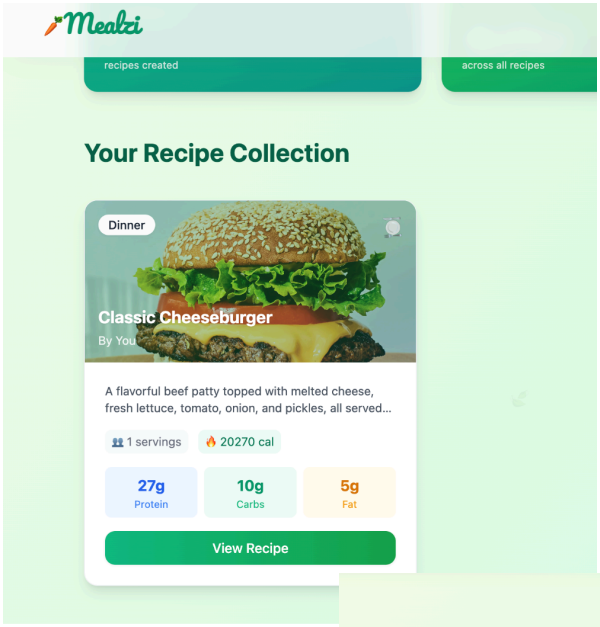
← Registration form

Dashboard



← Welcome message

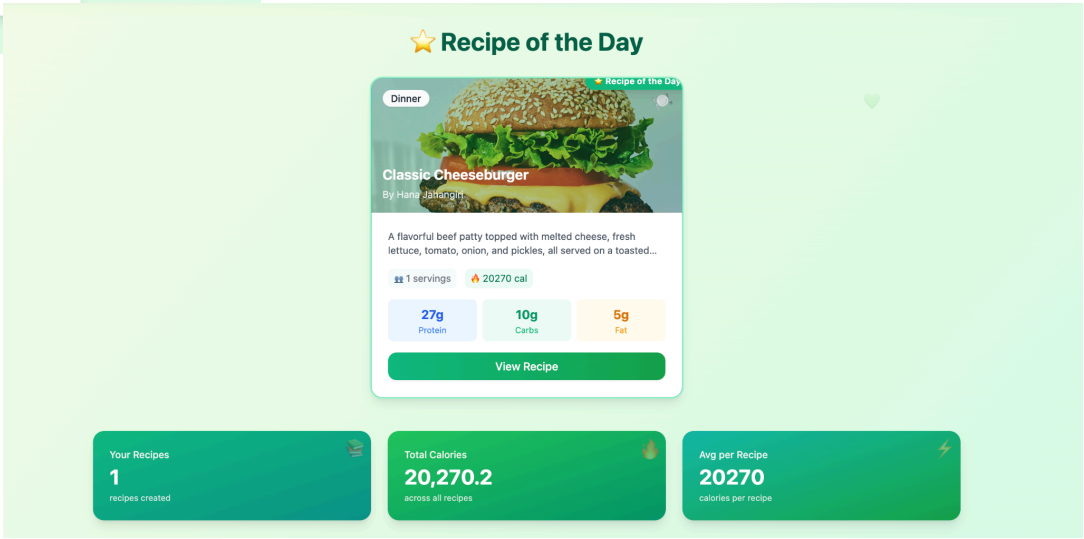
← Quick action buttons



← Recipe collection with nutrition summary

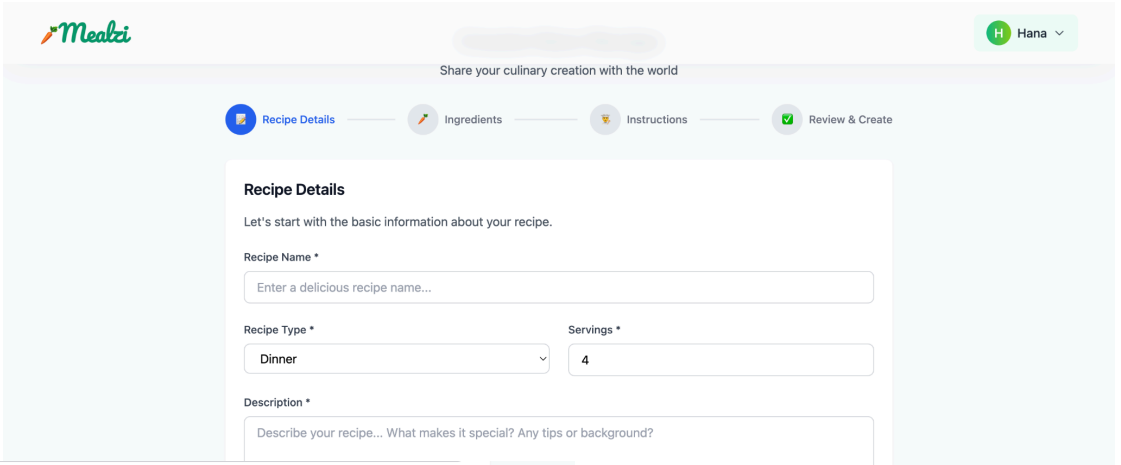
Recipe of the day →

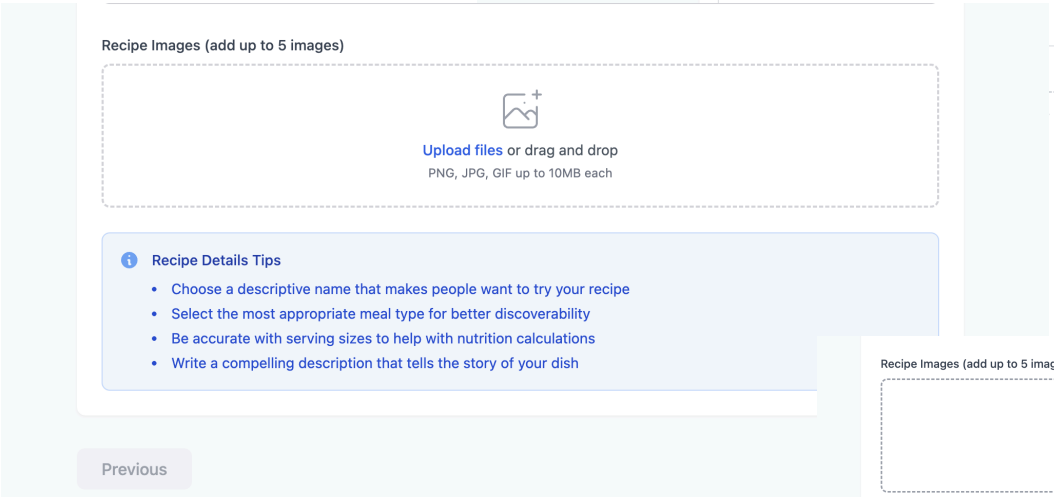
User recipe statistics →



Recipe Creation

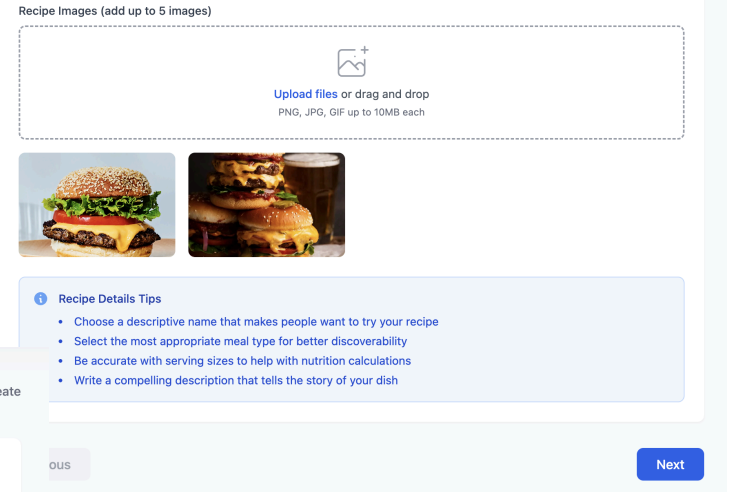
Step-by-step form →

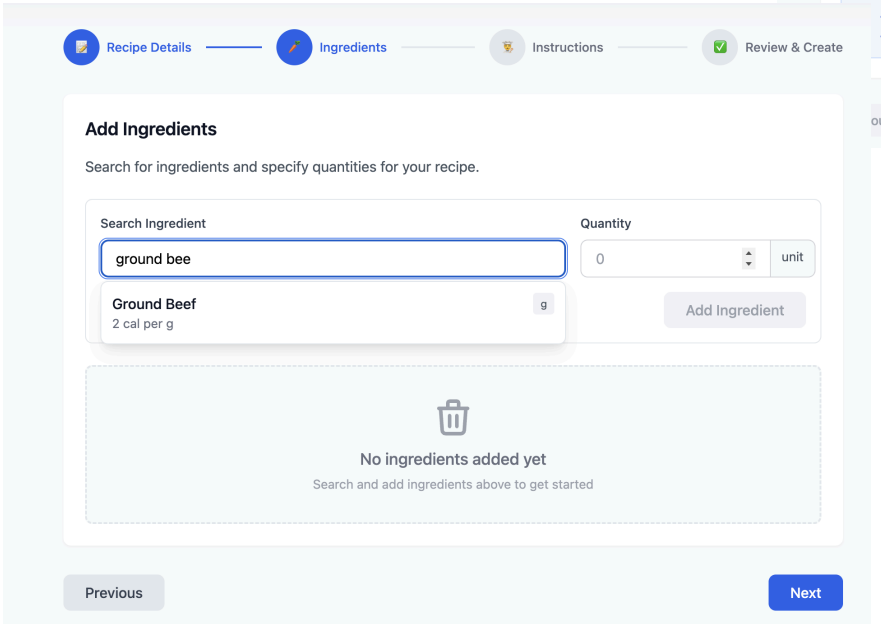




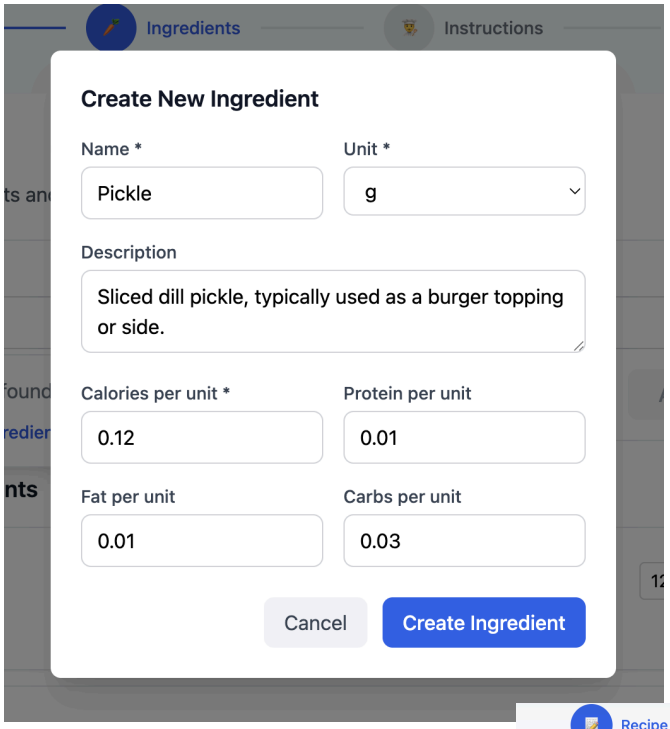
← Image upload interface

Support for multiple images →

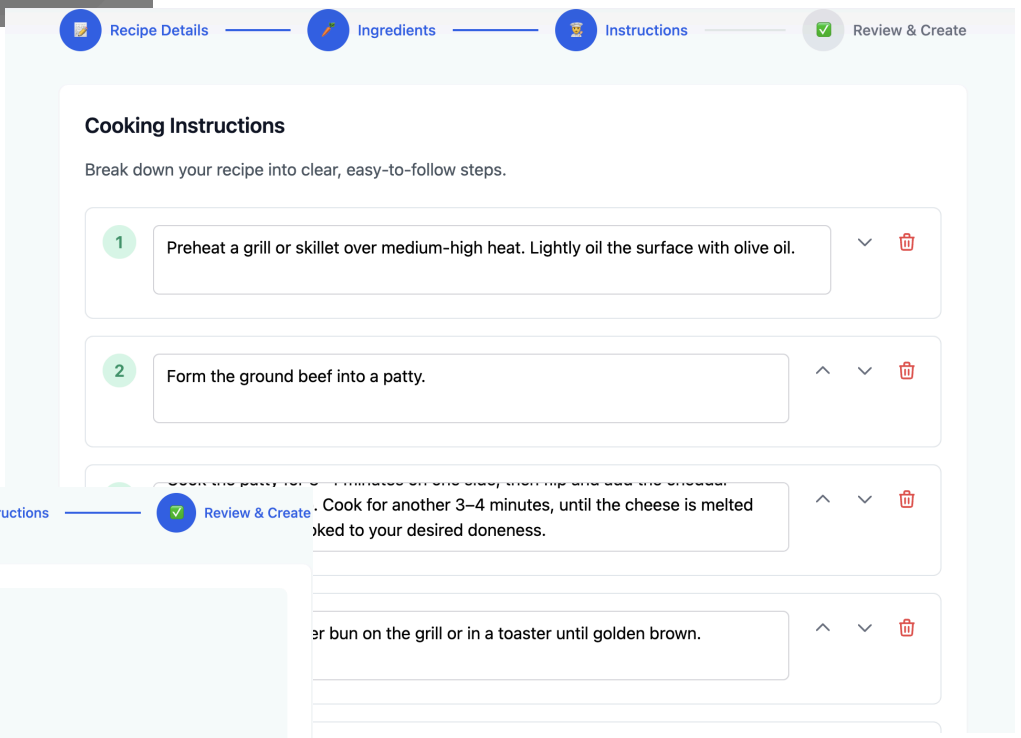




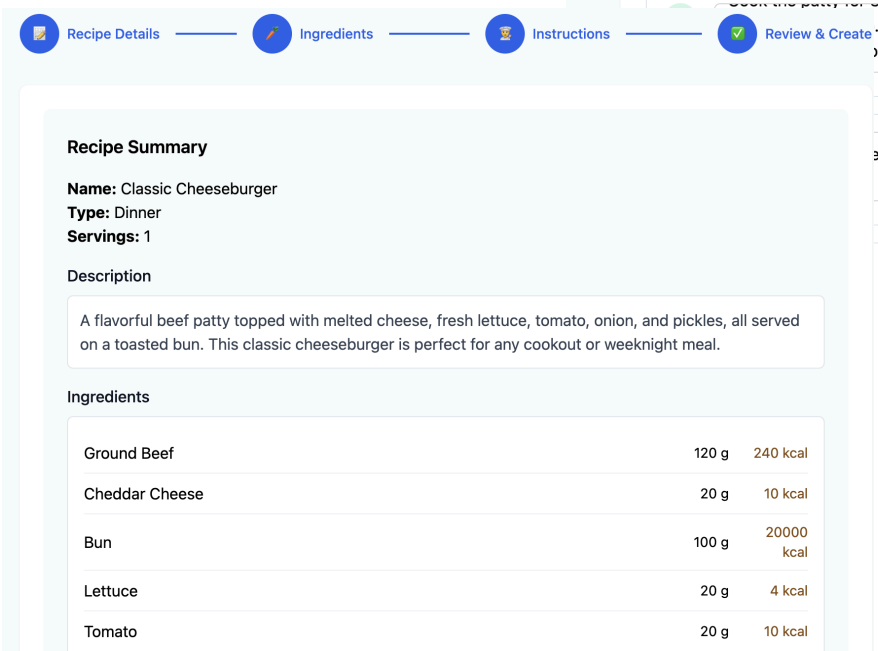
← Ingredient search



← Ingredient creation form



Rearrangeable instructions →



← Recipe summary

Red Onions	10 g	5 kcal
Pickle	10 g	1.2 kcal

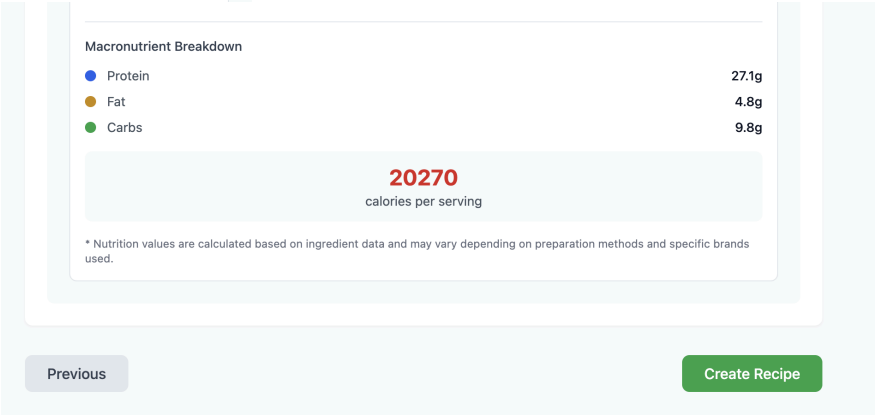
Instructions

1. Preheat a grill or skillet over medium-high heat. Lightly oil the surface with olive oil.
2. Form the ground beef into a patty.
3. Cook the patty for 3–4 minutes on one side, then flip and add the cheddar cheese slice on top. Cook for another 3–4 minutes, until the cheese is melted and the patty is cooked to your desired doneness.
4. Toast the hamburger bun on the grill or in a toaster until golden brown.
5. Assemble the burger.

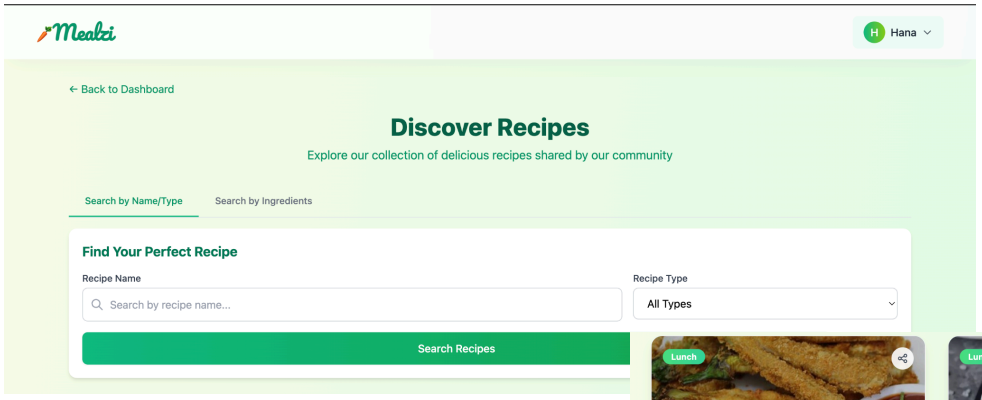
Nutrition Per Serving

Calories	20270 kcal
Protein	27.1 g
Fat	4.8 g
Carbs	9.8 g
Sugar	2.3 g
Fiber	1.9 g
Sodium	21 mg

← Recipe summary continued

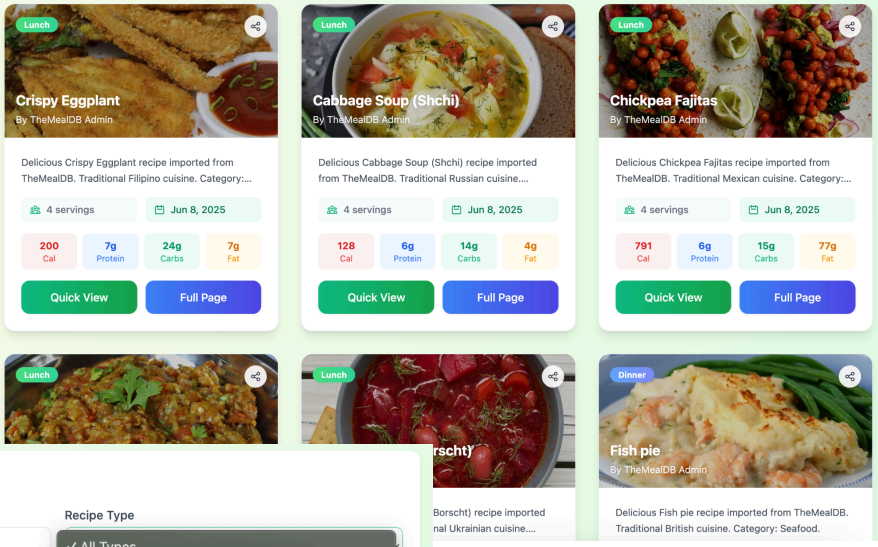


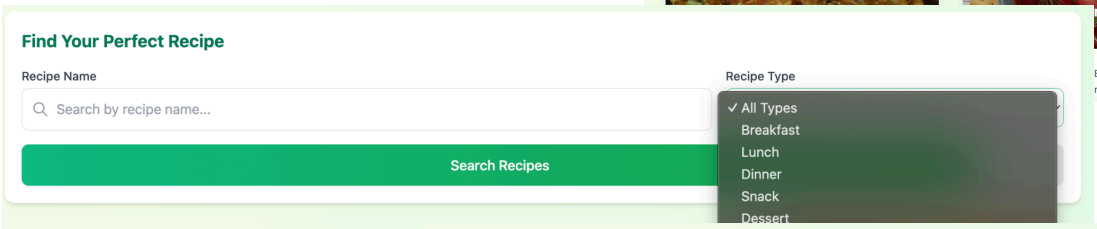
Recipe Discovery



← Public recipe dashboard

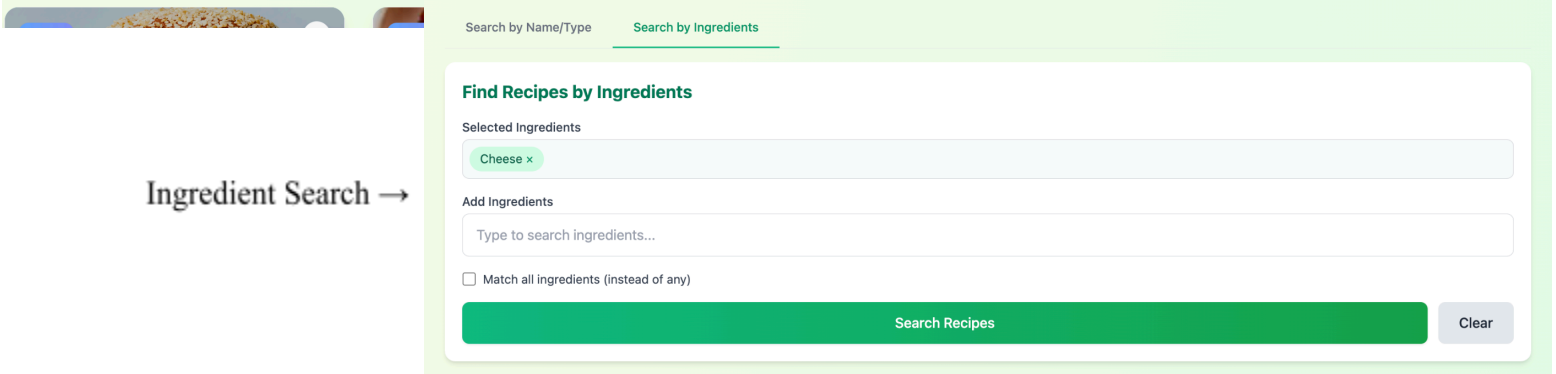
Meal DB recipes imported →



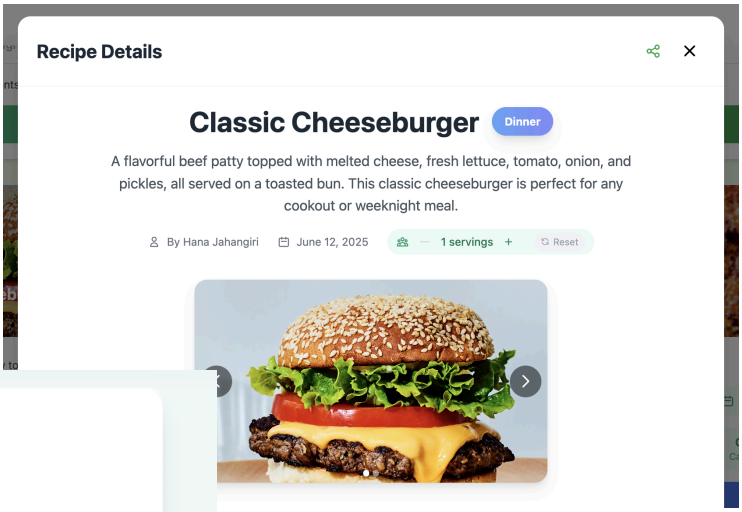


← Recipe Search

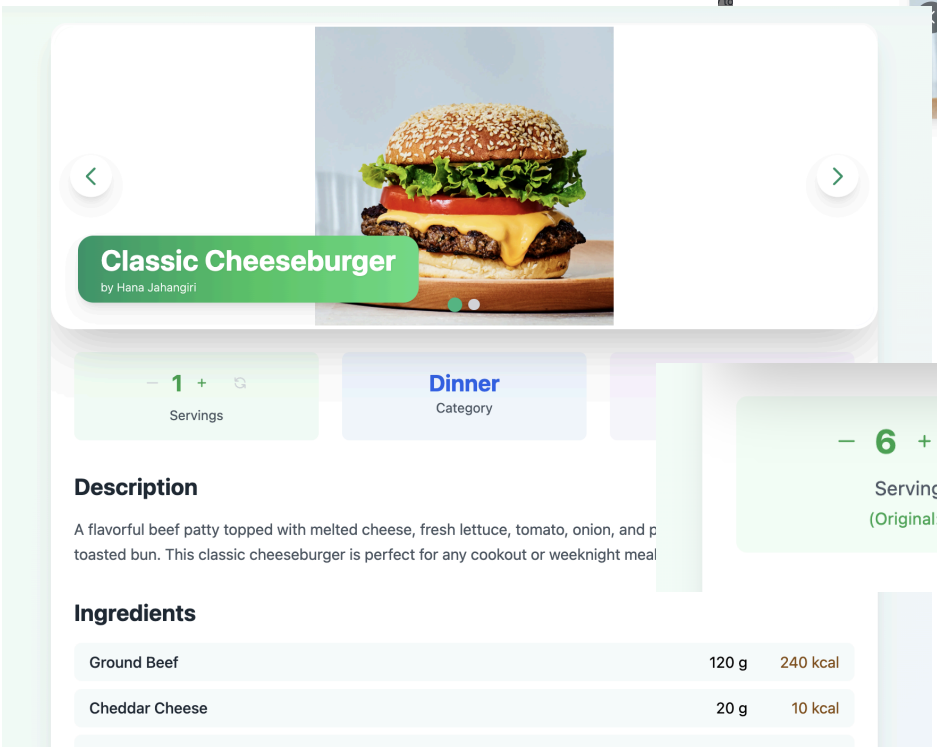
Ingredient Search →



Recipe details quick view →

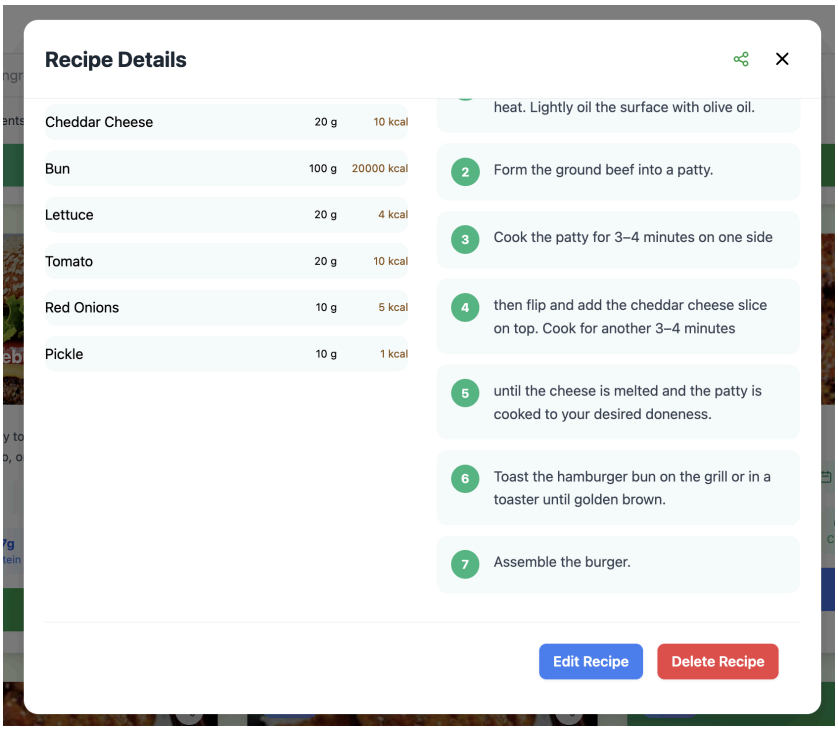


← Recipe details full page



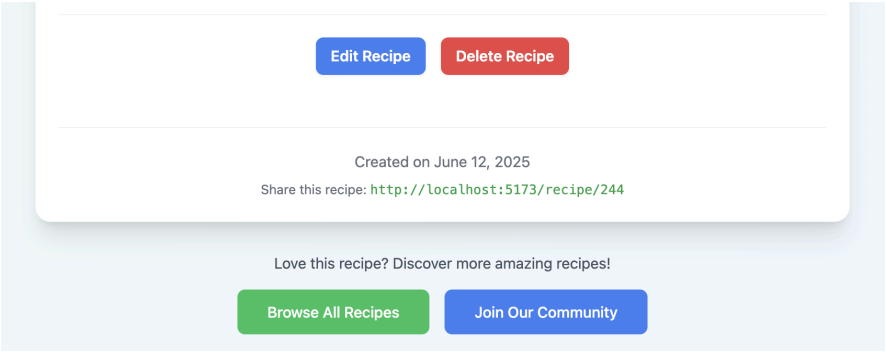
Adjust serving size

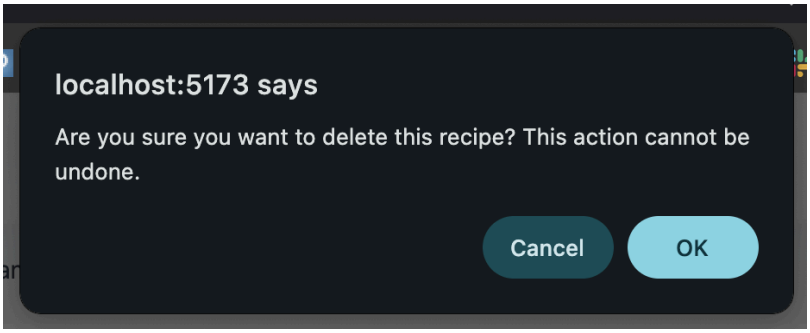
Recipe Management



← Edit or delete recipe on quick view

Edit or delete recipe on full page →





← Confirmation for deleting recipe

9. Individual Contributions

Dhriti Avala **1,035 ++ 380 --**

- Set up the database schema
- Ability to search ingredients by name
- A public search API for ingredients
- Support for multiple images per recipe, backend
- Store additional nutritional info beyond just calories
- Support for multiple images per recipe
- Documentation 3.1, 3.2, 5.3, 6.1, 6.3, 7
- Setup py4web backend

Varun Golusupudi **10,629 ++ 954 --**

- A professional, self-documenting, and intuitive user interface
- Import data from TheMealDB API to populate recipes
- Set up py4web backend
- Set up frontend with React and Tailwind
- Connect frontend to backend API
- User Authentication
- Py4web migration
- start-server.sh script

Hana Jahangiri **1,517 ++ 162 --**

- Ingredients are shared across users and cannot be edited once created
- Ability to edit recipes authored by the logged-in user only
- Secure recipe editing logic: only allow authors to modify their own recipes
- Automatically compute total calories per recipe based on ingredients and quantities
- Documentation 3.3, 3.4, 4.1, 4.2, 4.4. 4.5, 5.1, 5.2, 5.5, 6.2, 8, 9

Trisha Sriram **5,215 ++ 2,016 --**

- Ability to create new recipes
- Ability to view “My Recipes” by the logged-in user only
- Build recipe creation form in React
- Allow multiple images per recipe, frontend and backend integration
- Allow image uploads via form
- Py4web migration

Jose Valencia **1,418 ++ 216 --**

- Ability to search recipes by name and type
- Public API endpoint for recipe search

- Automatically scale ingredients when changing number of servings
- Ability to search recipes by ingredients (any subset)
- Documentation 4.3, 5.4

Praneet Varade **3,941 ++ 814 --**

- All recipes are public
- Import data from TheMealDB API to populate recipes
- Shareable recipe links

10. Conclusion

Our implementation of the Custom Recipe Manager successfully delivers a comprehensive recipe management platform that exceeds the assignment's requirements.

Assignment Requirements

Database Schema:

- A table for ingredients with fields: name, unit, calories_per_unit, description ✅
- A table for recipes with fields: name, type, description, image, author, instruction_steps, servings ✅
- A linking table to connect recipes and ingredients, with fields: recipe_id, ingredient_id, quantity_per_serving ✅

Functionality:

- Ability to search ingredients by name ✅
- Ability to add new ingredients ✅
- Ingredients are shared across users and cannot be edited once created ✅
- Ability to search recipes by name and type ✅
- All recipes are public ✅
- Ability to create new recipes ✅
- Ability to edit recipes authored by the logged-in user only ✅
- Users cannot edit others' recipes ✅

Documentation:

- A PDF with instructions and screenshots showing functionality ✅

Requirements for Full Grade:

- Import data from TheMealDB API to populate recipes (import only once; the import code must be included) ✅
- A professional, self-documenting, and intuitive user interface ✅
- A public search API for recipes (JSON format) ✅
- Secure recipe editing logic: only allow authors to modify their own recipes ✅
- Automatically compute total calories per recipe based on ingredients and quantities ✅

Optional Features (for extra credit):

- Support for multiple images per recipe ✅
- Ability to search recipes by ingredients (any subset) ✅
- A public search API for ingredients ✅
- Store additional nutritional info beyond just calories ✅
- Automatically scale ingredients when changing number of servings ✅