Sample documentation

The documentation should contain the following sections:

- Author information (name, Neptun code, date)
- Project overview
  - topic
  - short description
  - domain-specific vocabulary
- Functional requirements
  - Roles
  - Features by roles
  - Additionally, features by roles and pages
  - Additionally, permissions for authorization
- Technology stack
- Database schema
  - entities and relationships
  - diagram
  - raw DBML description
  - database description by tables and fields
- Planned sitemap (endpoints for features)
- UI mockup
- AI appendix

RecipeBox

Győző Horváth, K81DNC
gyozo.horvath@inf.elte.hu
23.02.2026

Project Overview

RecipeBox is a comprehensive recipe management platform that allows users to discover, create, organize, and share recipes from around the world. The platform provides a user-friendly interface for browsing recipes, creating collections, and managing personal recipe libraries.

Domain Specific Vocabulary

- **Recipe**: a recipe is a set of instructions for preparing a particular dish, including ingredients, steps, and images.
- **Collection**: a collection is a group of recipes that a user can create to organize their recipes based on themes, occasions, or preferences.

Functional Requirements

1. Guest (Unauthenticated User)

Features by pages:

- Home (`/`)
  - Browse all recipes
  - Search recipes
- Recipe detail (`/recipes/{id}`)
  - View recipe details
- Collections (`/collections`)
  - Browse recipe collections
- Collection Detail (`/collections/{id}`)
  - View recipes in the collection
- Sign Up (`/signup`)
  - Register for an account
- Login (`/login`)
  - Log in to an existing account

Restrictions

- Cannot create/edit/delete recipes.

- Cannot create/edit/delete collections.

2. User (Authenticated User)

Features:

- All Guest features
- View personal dashboard
  - View all personal recipes
  - Quick stats (total recipes, collections, views)
  - Edit/Delete actions for each recipe
- Recipes (CRUD)
  - View own recipes (see personal dashboard)
  - Create new recipes
  - Edit own recipes
  - Delete own recipes
- Collections (CRUD)
  - View own collections
  - Create a collection
  - Edit own collections
  - Delete own collections
- Add recipes to collections
- Remove recipes from collections

Restrictions:

- Cannot manage other users' content
- Cannot access admin panel

Pages:

- All Guest pages
- Dashboard (`/dashboard`)
- My Recipes (`/recipes/my`)

- Create Recipe (`/recipes/create`)
- Edit Recipe (`/recipes/{id}/edit`)
- Delete Recipe (`/recipes/{id}/delete`)
- My Collections (`/collections/my`)
- Create Collection (`/collections/create`)
- Edit Collection (`/collections/{id}/edit`)
- Delete Collection (`/collections/{id}/delete`)

3. Admin (Superuser)

Features:

- All User features
- Access admin dashboard
- Manage recipes (all users)
  - view, edit, delete any recipe
- Manage collections (all users)
  - view, edit, delete any collection
- Manage users (view, edit, ban)

Pages:

- All User pages
- Admin Dashboard (`/admin`)
- User Management (`/admin/users`)
- Recipe Management (`/admin/recipes`)
- Collection Management (`/admin/collections`)

Technology Stack

- Database: SQLite
- Backend:
  - Framework: Laravel (PHP)
  - Authentication: Laravel Breeze

- Frontend:
  - Blade templates
  - Tailwind CSS for styling
  - DaisyUI for UI components

Database Schema

Entities

- users
- recipes
- categories

Relationships

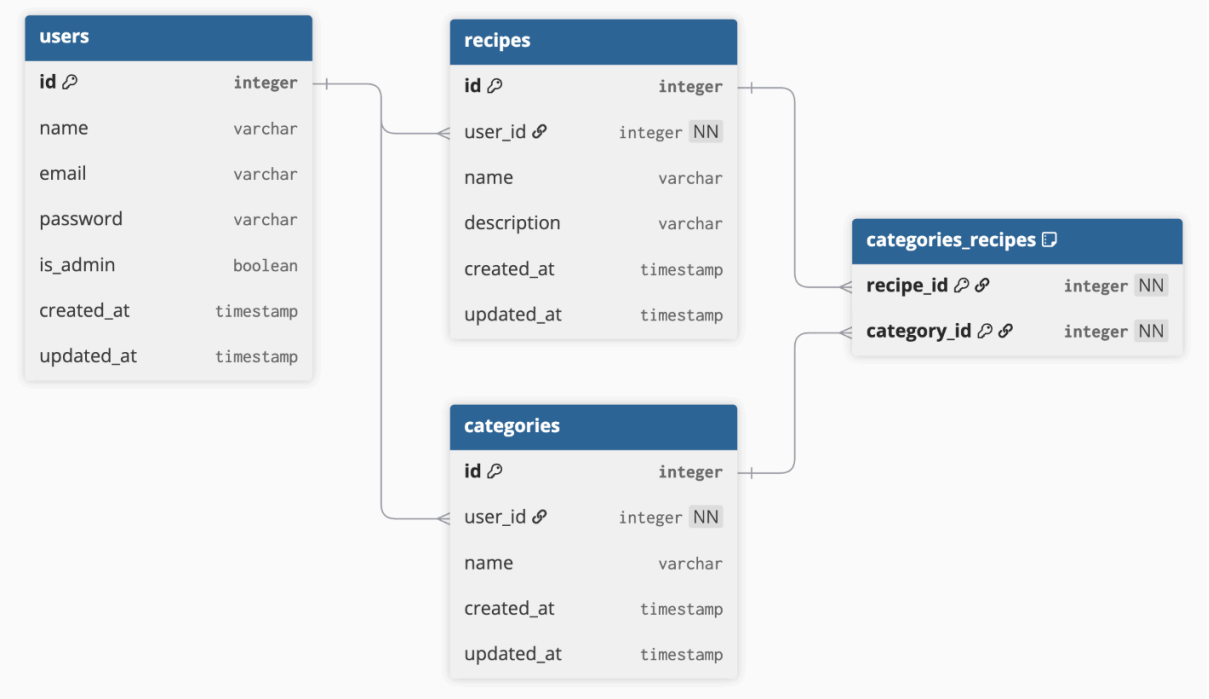
- **users** → **recipes** (One-to-Many): A user can create multiple recipes, but a recipe belongs to one user
- **users** → **collections** (One-to-Many): A user can create multiple collections, but a collection belongs to one user
- **recipes** ↔ **collections** (Many-to-Many via `collection_items`): Recipes can belong to multiple collections, collections can contain multiple recipes

Diagram

You can view the database schema diagram at dbdiagram.io by importing the following DBML description:

```
Table users {
  id integer [primary key]
  name varchar
  email varchar
  password varchar
  is_admin boolean

  created_at timestamp
  updated_at timestamp
}
```

```
Table recipes {
  id integer [pk]
  user_id integer [not null]
  title varchar
  description text
  image_url varchar
  difficulty enum('easy', 'medium', 'hard')
  cook_time integer
  instructions text
  ingredients text

  created_at timestamp
  updated_at timestamp
}

Table categories {
  id integer [pk]
  user_id integer [not null]
  name varchar

  created_at timestamp
  updated_at timestamp
}

Table categories_recipes {
  recipe_id integer [not null]
  category_id integer [not null]

  indexes {
    (recipe_id, category_id) [pk]
  }
}

Ref: recipes.user_id > users.id
Ref: categories.user_id > users.id
// Ref: categories.id <> recipes.id
Ref: categories_recipes.recipe_id > recipes.id
Ref: categories_recipes.category_id > categories.id
```



RecipeBox diagram

Database description by tables and fields (version 1)

`users`

Stores user account information.

Column	Type	Description
`id`	integer	Primary key
`name`	varchar	User's full name
`email`	varchar	Unique email address
`password`	varchar	Hashed password
`is_admin`	boolean	Admin flag (default: false)
`created_at`	timestamp	Account creation timestamp
`updated_at`	timestamp	Last update timestamp

`recipes`

Stores recipe information created by users.

Column	Type	Description
`id`	integer	Primary key

Column	Type	Description
`user_id`	integer	Foreign key referencing `users.id`
`title`	varchar	Recipe title
`description`	text	Brief description of the recipe
`image_url`	varchar	URL to an image of the dish
`difficulty`	enum('easy', 'medium', 'hard')	Difficulty level of the recipe
`cook_time`	integer	Estimated cooking time in minutes
`instructions`	text	Step-by-step instructions for preparing the dish
`ingredients`	text	List of ingredients required for the recipe
`created_at`	timestamp	Recipe creation timestamp
`updated_at`	timestamp	Last update timestamp

`categories`

Stores user-defined categories for organizing recipes.

Column	Type	Description
`id`	integer	Primary key
`user_id`	integer	Foreign key referencing `users.id`
`name`	varchar	Name of the category (e.g., "Desserts", "Vegetarian")
`created_at`	timestamp	Category creation timestamp
`updated_at`	timestamp	Last update timestamp

`categories_recipes`

Join table for the many-to-many relationship between recipes and categories.

Column	Type	Description
`recipe_id`	integer	Foreign key referencing `recipes.id`
`category_id`	integer	Foreign key referencing `categories.id`
Primary Key	(recipe_id, category_id)	Composite primary key ensuring unique associations

Database description by tables and fields (version 2)

- **users**

- `id`: integer, primary key
- `name`: varchar, user's full name
- `email`: varchar, user's email address (unique)
- `password`: varchar, hashed password for authentication
- `is_admin`: boolean, indicates if the user has admin privileges
- `created_at`: timestamp, when the user was created
- `updated_at`: timestamp, when the user was last updated

- **recipes**
  - `id`: integer, primary key
  - `user_id`: integer, foreign key referencing `users.id`, indicates the creator of the recipe
  - `title`: varchar, the name of the recipe
  - `description`: text, a brief description of the recipe
  - `image_url`: varchar, URL to an image of the dish
  - `difficulty`: enum('easy', 'medium', 'hard'), the difficulty level of the recipe
  - `cook_time`: integer, estimated cooking time in minutes
  - `instructions`: text, step-by-step instructions for preparing the dish
  - `ingredients`: text, list of ingredients required for the recipe
  - `created_at`: timestamp, when the recipe was created
  - `updated_at`: timestamp, when the recipe was last updated
- **categories**
  - `id`: integer, primary key
  - `user_id`: integer, foreign key referencing `users.id`, indicates the creator of the category
  - `name`: varchar, the name of the category (e.g., "Desserts", "Vegetarian")
  - `created_at`: timestamp, when the category was created
  - `updated_at`: timestamp, when the category was last updated

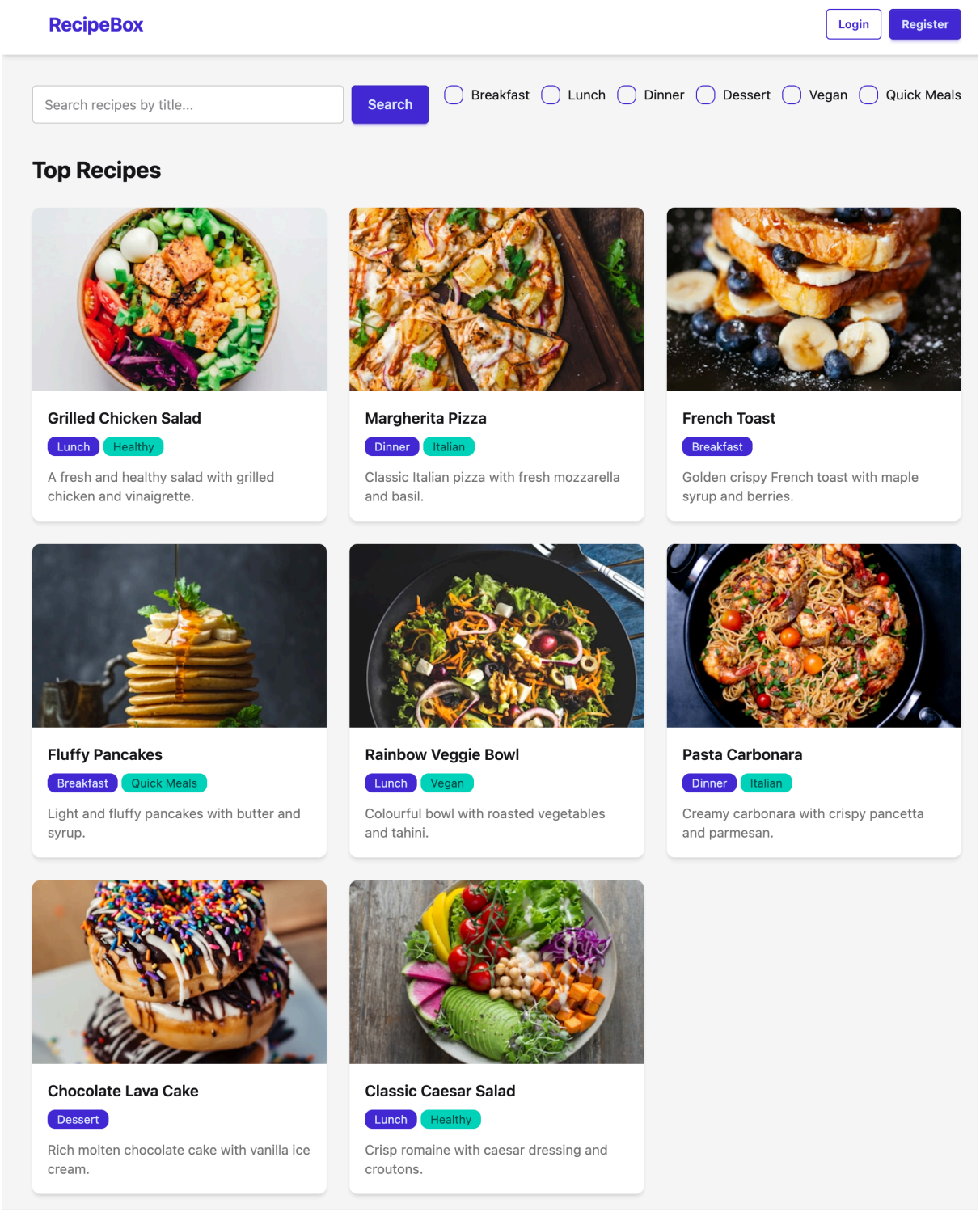
- **categories_recipes**
  - `recipe_id`: integer, foreign key referencing `recipes.id`, indicates the associated recipe
  - `category_id`: integer, foreign key referencing `categories.id`, indicates the associated category
  - Primary key is a composite of `recipe_id` and `category_id`, ensuring that each recipe-category association is unique.

Planned Sitemap (Endpoints for features)

```
RecipeBox
│
├── Public Pages (Guest)
│   ├── / (Home)
│   ├── /recipes/{id} (Recipe Detail)
│   ├── /collections (Browse Collections)
│   ├── /collections/{id} (Collection Detail)
│   ├── /login (Login)
│   └── /signup (Sign Up)
│
├── Authenticated User Pages
│   ├── /dashboard (User Dashboard)
│   ├── /recipes/my (My Recipes)
│   ├── /recipes/create (Create Recipe)
│   ├── /recipes/{id}/edit (Edit Recipe)
│   └── /collections/my (My Collections)
│
└── Admin Pages
    ├── /admin (Admin Dashboard)
    ├── /admin/users (User Management)
    ├── /admin/recipes (Recipe Management)
    └── /admin/collections (Collection Management)
```

UI Mockup

1. Landing Page (Home)



Home page

2. Recipe Detail Page

RecipeBox

Login Register

Recipes › Grilled Chicken Salad

Grilled Chicken Salad

Lunch Healthy 30 min Easy

Add to Favourites

Login to save recipes

A fresh and healthy salad with grilled chicken breast, mixed baby greens, cherry tomatoes, cucumbers, and a light honey-lemon vinaigrette.

Ingredients

- 2 chicken breasts
- 4 cups mixed greens
- 1 cup cherry tomatoes
- 1 cucumber, sliced
- 1/4 red onion, thinly sliced
- 1/4 cup feta cheese
- 2 tbsp olive oil
- 1 tbsp lemon juice
- 1 tbsp honey
- Salt & pepper to taste

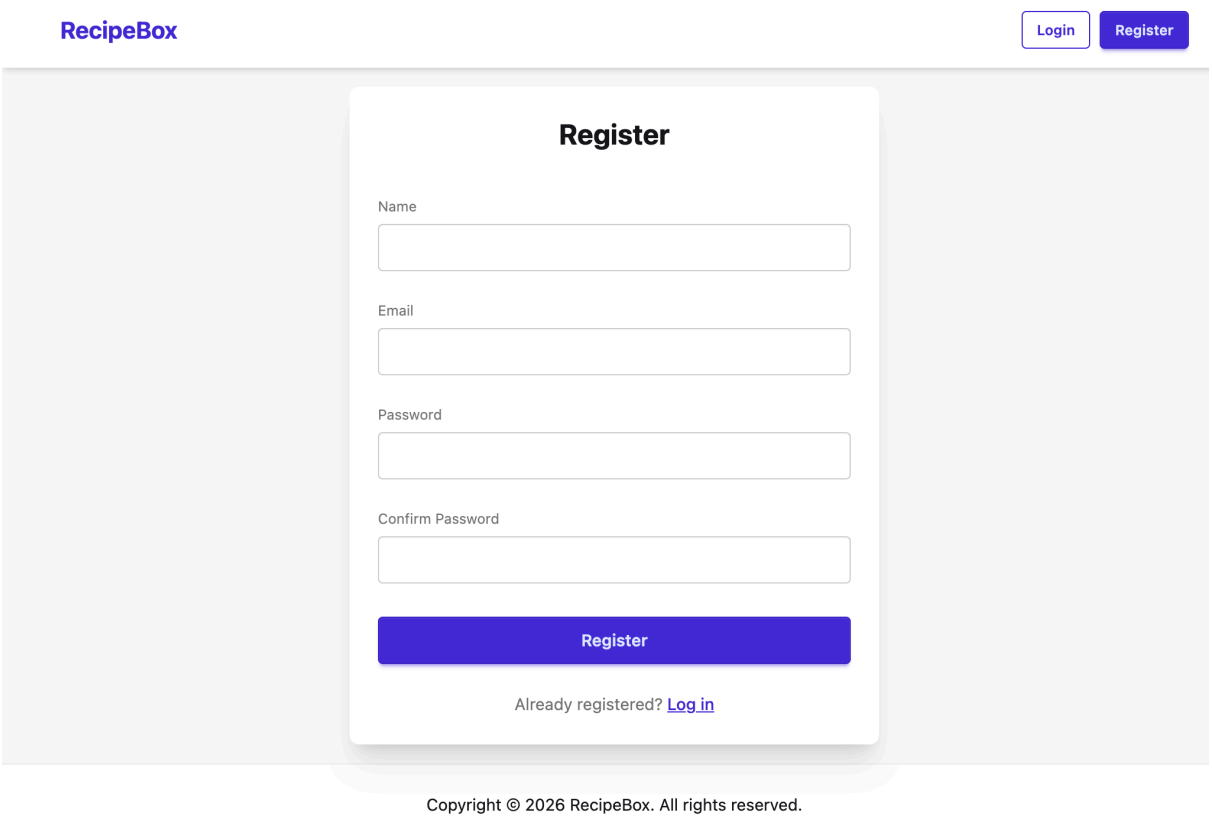
Instructions

1. **Prepare the chicken** — Season chicken breasts with salt, pepper, and olive oil. Let rest for 10 minutes.
2. **Grill the chicken** — Preheat grill to medium-high. Grill 6-7 minutes per side until done.
3. **Make the vinaigrette** — Whisk olive oil, lemon juice, honey, salt, and pepper.
4. **Assemble the salad** — Arrange greens, tomatoes, cucumber, onion, and feta on plates.
5. **Serve** — Slice chicken on top. Drizzle with vinaigrette.

Copyright © 2026 RecipeBox. All rights reserved.

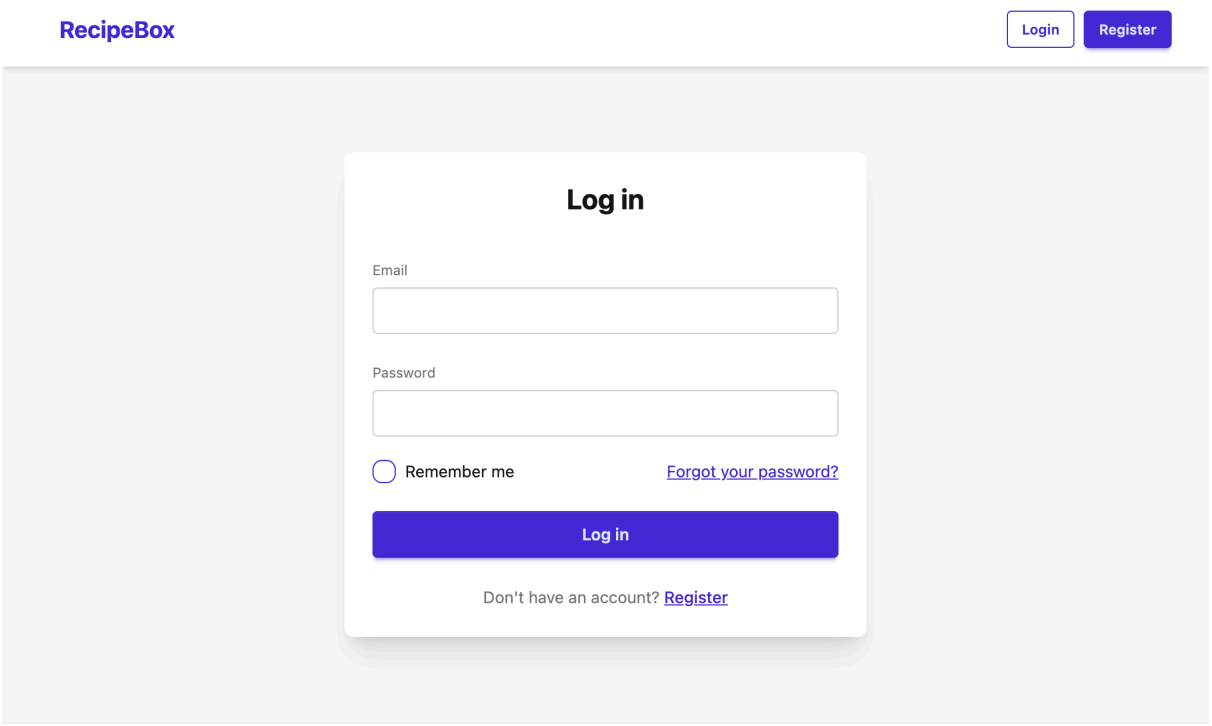
Recipe detail page

3. Sign Up Page



Sign up page

4. Login Page



Copyright © 2026 RecipeBox. All rights reserved.

Login page

5. User Dashboard

RecipeBox

Home | My Recipes | Favourites | My Categories | + Add Recipe

JD

Welcome back, John!

Discover new recipes or share your own creations.

+ Add New Recipe

My Recipes	Favourites	My Categories
12	24	6

Search recipes by title...

Search

Breakfast | Lunch | Dinner | Dessert | Vegan

Top Recipes

Grilled Chicken Salad

Lunch · Healthy

A fresh and healthy salad with grilled chicken and vinaigrette.

Margherita Pizza

Dinner · Italian

Classic Italian pizza with fresh mozzarella and basil.

French Toast

Breakfast

Golden crispy French toast with maple syrup and berries.

Fluffy Pancakes

Breakfast · Quick Meals

Light and fluffy pancakes with butter and syrup.

Rainbow Veggie Bowl

Lunch · Vegan

Colourful bowl with roasted vegetables and tahini.

Pasta Carbonara

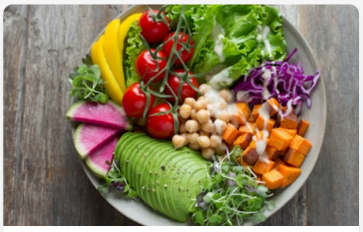
Dinner · Italian

Creamy carbonara with crispy pancetta and parmesan.

Chocolate Lava Cake

Dessert

Rich molten chocolate cake with vanilla ice cream.



Classic Caesar Salad

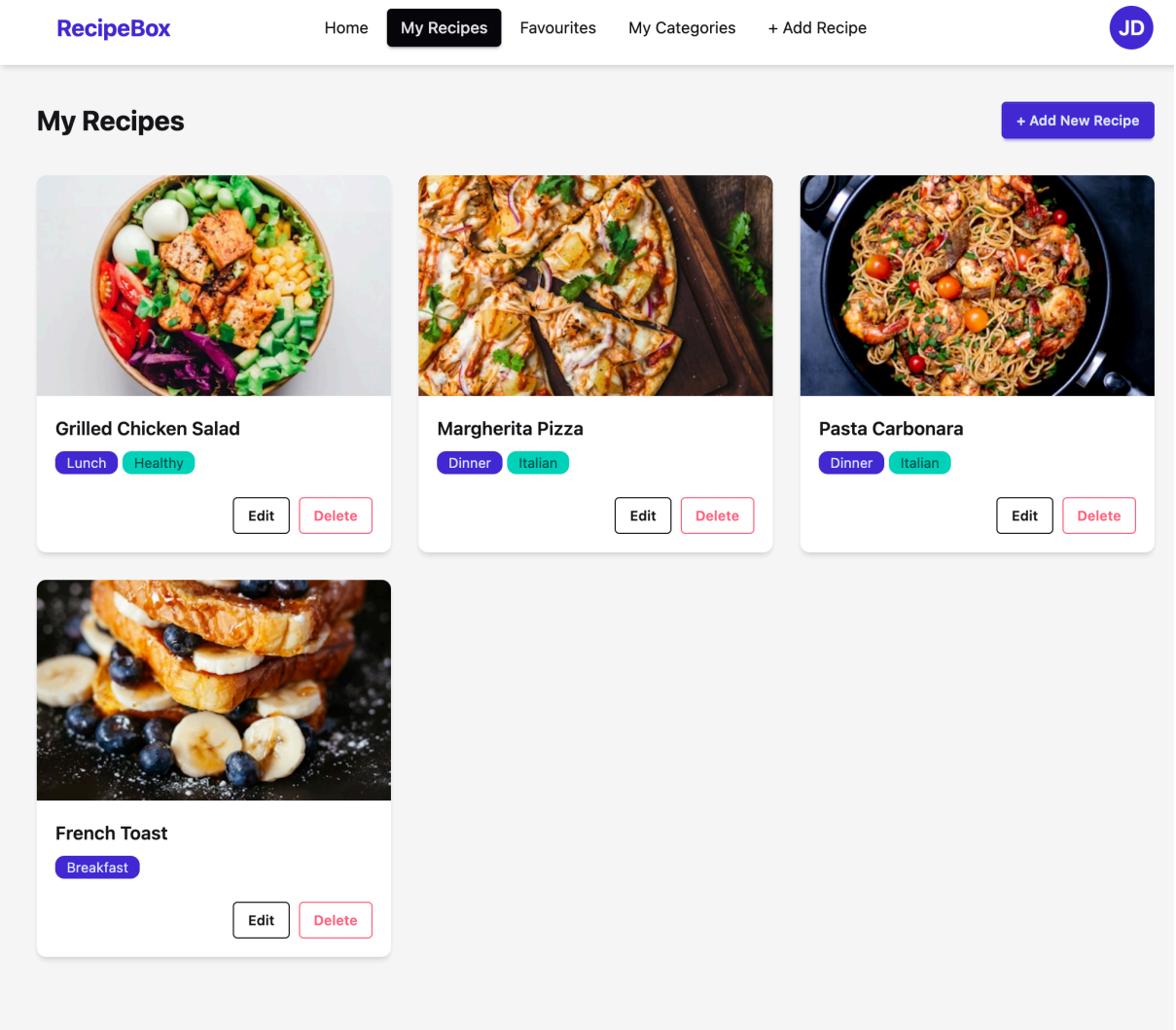
Lunch · Healthy

Crisp romaine with caesar dressing and croutons.

Copyright © 2026 RecipeBox. All rights reserved.

User dashboard

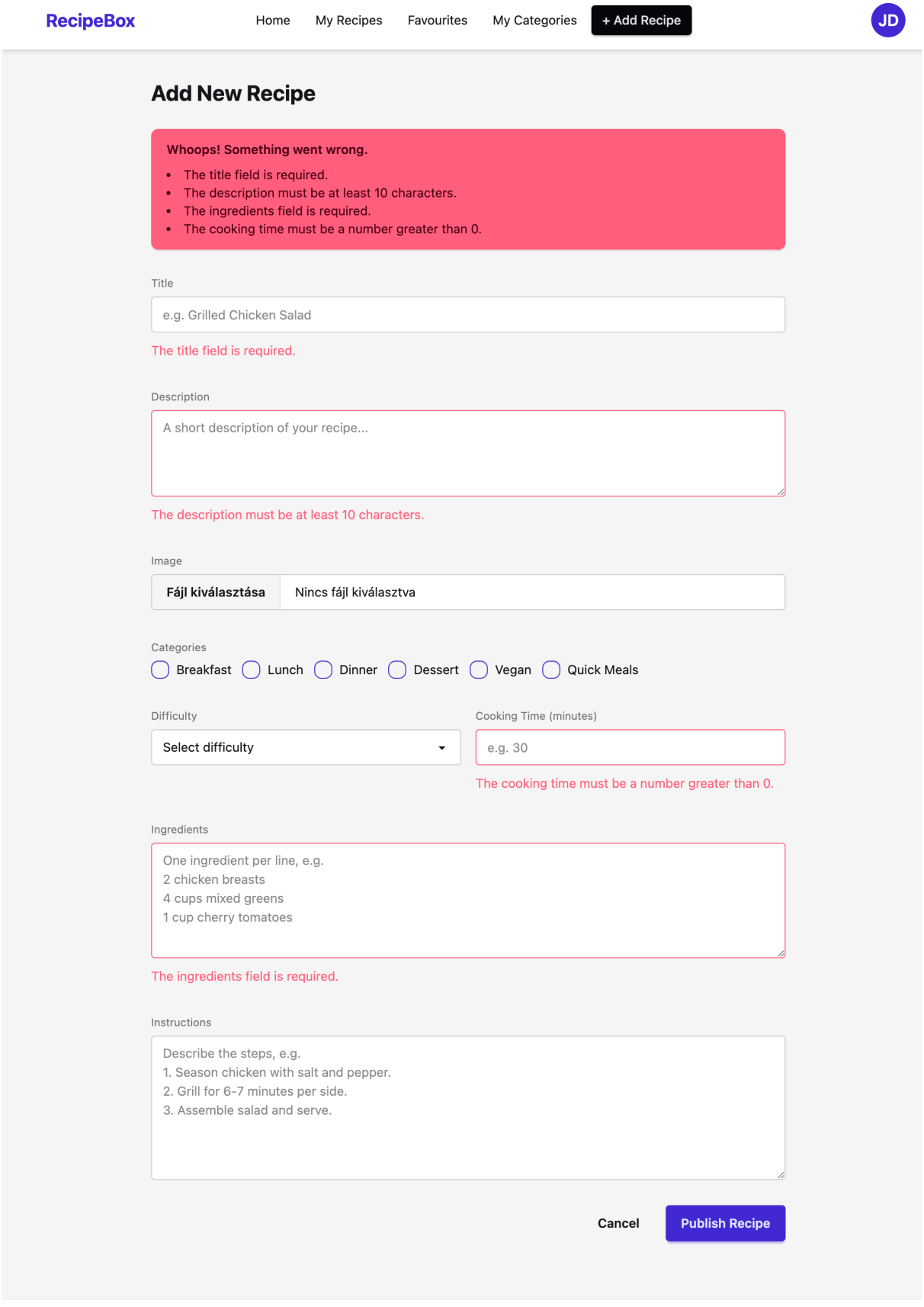
6. My Recipes Page



Copyright © 2026 RecipeBox. All rights reserved.

My recipes page

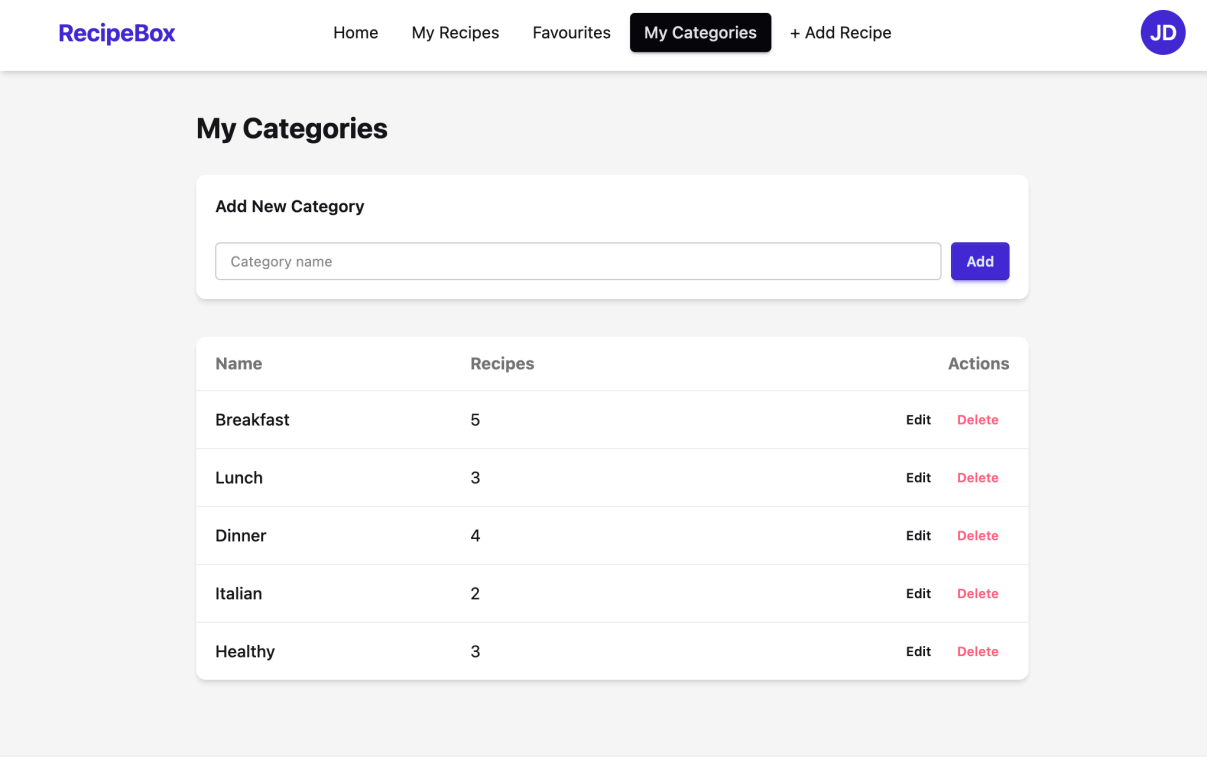
7. Create Recipe Page



Copyright © 2026 RecipeBox. All rights reserved.

Create recipe page

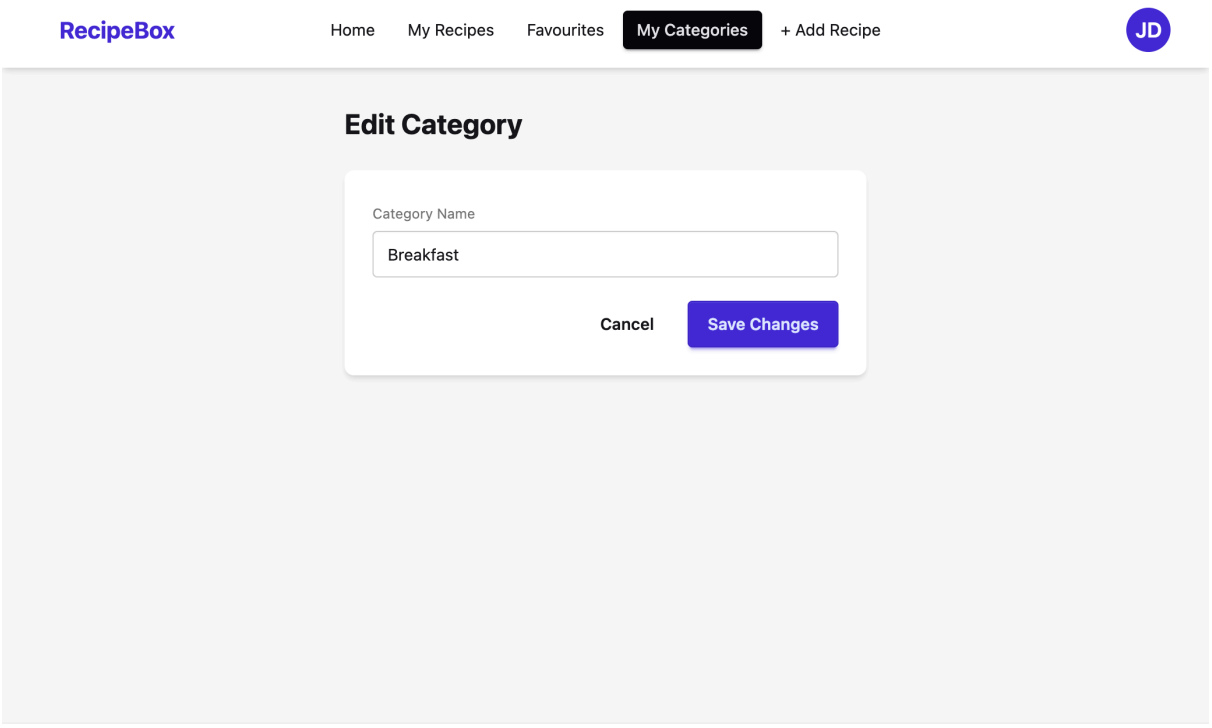
8. My Collections Page



Copyright © 2026 RecipeBox. All rights reserved.

My collections page

9. Edit Collection Page



Edit collection page

AI Appendix

UI Mockup Generation

With O365 Copilot I generated the prompt for the UI mockups as follows:

> We would like to create a recipe application based on the functional requirements below. Do not implement, but create a prompt, which I will give to an AI to generate the user interface. I would like to use daisyui, and it will serve as a mockup, so just html and css from cdn should be used.## Functional requirements
>
> - Guest
>   - Main page
>   - recipe list
>   - top or picked recipes
>   - search/filter recipes
>   - filter by category

    - Recipe detail page
    - Authentication
        - Register
        - Login
- Authenticated user
    - Can do what a guest user can do
    - Add a new recipe
        - Assign recipe to categories
    - Edit/delete own recipes
    - List own recipes
    - Save a recipe as favourite
    - Categories CRUD
- Admin
    - user management
    - edit/delete any recipes/categories

I used the generated prompt in v0 as a starting point and asked the AI to do further improvements to the UI mockups.

> Your task:
> Create a complete UI mockup (non-functional, static HTML) for a Recipe Application using only HTML and CSS, with TailwindCSS CDN + DaisyUI CDN.
> No JavaScript logic is required — only layout, mock data, and example components.
>
> 🎨 Tech Requirements
>
> Use TailwindCSS CDN
> Use DaisyUI CDN
> Use only HTML (static mockup)
> Use DaisyUI components for styling (cards, menus, navbar, inputs, modals, badges, drawers, etc.)
> Include multiple pages as separate HTML sections (they do not need to link)
> Add mock content (fake photos, fake recipe names, example text)

> 📌 Pages & Requirements to Mock
> 1. Guest Features
> Main Page
>
> Navbar with:
>
> Login button
> Register button
>
> Recipe list:
>
> Show “Top recipes” or “Picked recipes”
> Grid of recipe cards (image, title, short description)
>
> Search / filter section:
>
> Search input
> Category filter dropdown or chips
>
> Footer
>
> Recipe Detail Page
>
> Large recipe image
> Title + category tags
> Ingredients list
> Steps / instructions
> “Add to favourites” button (disabled for guests)
> “Edit / Delete” buttons hidden for guest
>
> Authentication Pages
>
> Login page
> Register page
>
> 1. Authenticated User Features
>   Include UI elements only visible to logged-in users:
>
> Navbar changes:
>
> Profile menu
> “My Recipes”
> “Add Recipe”
> “Logout”
>
> Add Recipe Page:
>
> Title, image upload placeholder, description
> Ingredients input repeater (static mock elements ok)

> Steps input
> Category selection (checkboxes or multi-select)
>
> Edit Recipe Page (same as Add, prefilled)
> Own Recipes Page
>
> List of user's recipes with Edit / Delete buttons
>
> Favourite Recipes Page
>
> Grid list of saved recipes
>
> 1. Admin Features
>   Indicate admin-only controls clearly:
>   User Management Page
>
> Table of users
> Edit / Delete buttons
>
> Admin Recipe/Category Management
>
> Page for editing/deleting any recipe
> Page for managing categories (CRUD)
>
> List of categories
> Add category modal
> Edit / Delete category buttons
>
> 📁 Layout Expectations
>
> Use consistent DaisyUI theme
> Clean navigation layout
> Example data for all recipes/users/categories
> Buttons, forms, and cards styled with DaisyUI
>
> ❗ Deliverables
> Put pages into separate html files and connect them together with links.
> Do not use JavaScript functionality — only static layout.

> We need some adjustments.
> On the landing page (index.html) we dont need the hero part, and the picked part. Top recipes are good. Beside the search bar make it possible to select multiple categories. So we can search by titles and categories at the same time.
> We dont need the public menu (Home, recipes), because recipes points to one recipe page. It does not make any sense.

> Later, not now in this mockup, we will use Laravel, so please align the data on the register and login page to Laravel Breeze’s expectations. We dont need google sign in.
> On the Add and Edit recipe page ingredients and descriptions should be in a textarea. Difficulty and cooking time is missing.
> Create a Categories page as well, where we can add/edit/delete categories.
> Use daisyUI, make it responsive, and the HTML shoild be as clean and small and less noisy, as possible, because it will be used for demonstration in a class.

> The search button is missing. This will be a classical server side app, without JS. So we need a search button. Align the pages according to this expectations. There is no edit category page (or popup). An dcategories should be my categories on the dashboard, because the public page will show all the distinct categories, but every user will have its own categories.

> Use the actual daisy ui and tailwind, and on the add recipe page prepare the form to display error messages like it was an error generated by a server application like laravel.

After this I downloaded and edited the generated HTML and CSS to fit my needs. The generated mockups were a good starting point, but I had to make some adjustments to better align with the functional requirements and the overall design vision for the application.