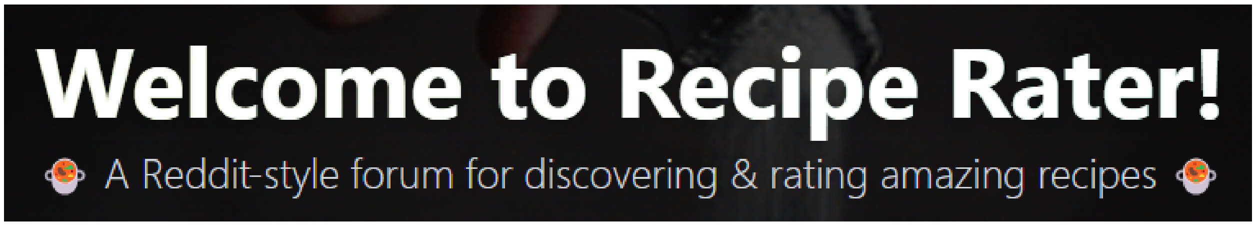
Aoi Ueki - UCID 30179305
Ali Al Yasseen - UCID 30151000
Vincent Iyegbuye UCID:30211518



Welcome to Recipe Rater!

A Reddit-style forum for discovering & rating amazing recipes

[Recipe Rater Deployed Live Demo](#)
[Recipe Rater Github](#)

Table of contents

Abstract

Recipe Rater is a community-first recipe application where users can upload their own recipes, rate and comment on other users' creations, and follow their favorite home cooks. This application aims to improve and simplify existing systems by focusing on user-generated content and real-time interaction, creating a social-media-style platform where users feel fully immersed in a shared culinary community. Under the hood, Recipe Rater leverages React for a dynamic, component-based frontend styled with TailwindCSS, a RESTful Node.js/Express backend, and MySQL for structured data storage. Customers and Admins authenticate securely via session-based login (Express-Session, bcrypt), while Admins maintain quality and safety through a dedicated moderation dashboard. Users can enrich recipes with drag-and-drop image uploads (Multer), and explore dishes using powerful search and filtering by ingredients, cuisine, prep time, or rating. A mobile-first layout powered by React Router ensures seamless browsing and recipe creation on any device.

Introduction

Current systems fail in having community driven content as some applications may have community ratings that are hard to find, some may also have comments that are hard to find, or that community driven content simply is not the primary focus of the application. Our system aims to improve on this by simplifying the user interface to allow users to easily and quickly find ratings and comments made by other users, ultimately creating an environment that is more interactive and immersive with the community. Recipe Rater employs a Reddit-style upvote/downvote mechanism on recipes, allowing the community to surface top-rated content organically. Items with the highest net upvotes rise to the top of feeds and search results, while lesser-voted content remains accessible but unobtrusive. This democratic approach ensures that users can quickly find tried-and-true recipes and insightful commentary, driving quality contributions and keeping the community engaged.

Our system

Recipes consist of the title of their recipe, an optional short description, and necessary ingredients and steps to re-create their recipe. Recipes also contain pre-defined common categories and optional user uploaded images of their recipes. Recipes are uploaded in ease with the dedicated uploading page as long as the user is registered with an account. Creating accounts only requires the publicly visible name, username to login with, and the password to login with. User uploaded recipes will be uploaded to a database where non-registered users may simply view, or where registered users will be able to view, as well as rate and comment on other users recipes. Specific recipes can be found by sorting by newest first, by popularity, by category, or by searching via the search icon

System users

Recipe Rater features three core roles, each with distinct objectives and capabilities:

---

1. Guest (Unauthenticated Visitor)

Objectives:

- Explore featured, trending, and newly added recipes
- View recipe details, threaded comments, and top-voted content
- Search and filter by keyword, date, rating, or category

Capabilities:

- Read-only access to all public recipes and comments
- Cannot submit recipes, rate, comment, or vote

---

2. Customer (Registered Cook)

Objectives:

- Share and manage personal recipes with image uploads
- Provide feedback via upvotes/downvotes and comments

Capabilities:

- Create, read, update, and delete their own recipes and profile
- Cast upvotes/downvotes on any recipe or comment
- Edit or remove their own comments

---

3. Administrator

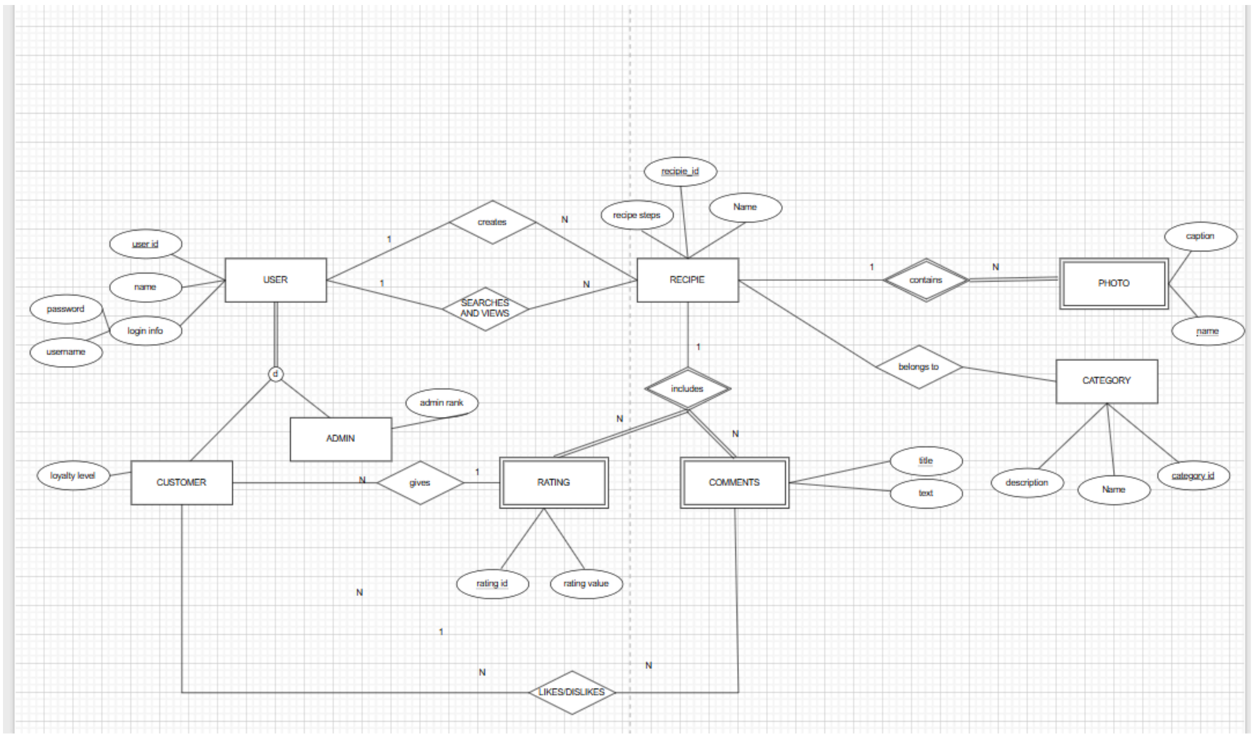
Objectives:

- Uphold content quality, security, and community guidelines
- Address policy violations, disputes, and spam

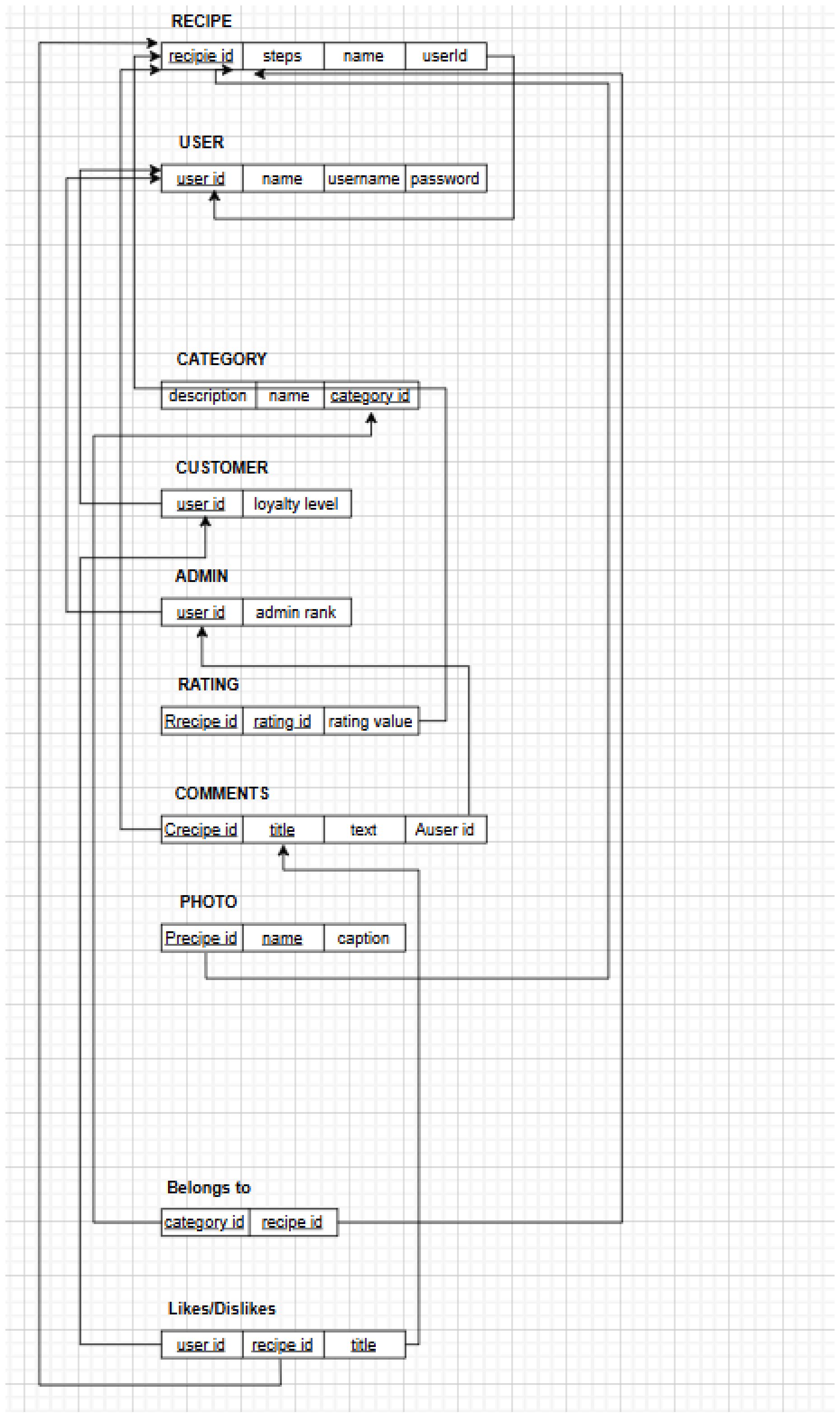
Capabilities:

- All Customer privileges, plus:
    - Edit or delete any recipe or comment site-wide

ER Diagram


recipie_id
recipe steps
Name
creates
1
N
user id
name
password
login info
username
USER
SEARCHES AND VIEWS
1
N
RECIPIE
1
contains
N
PHOTO
caption
name
belongs to
CATEGORY
1
includes
N
N
d
admin rank
ADMIN
loyalty level
CUSTOMER
N
gives
1
RATING
COMMENTS
title
text
description
Name
category id
rating id
rating value
N
1
N
N
LIKES/DISLIKES

RM Diagram


RECIPE
recipie id
steps
name
userId
USER
user id
name
username
password
CATEGORY
description
name
category id
CUSTOMER
user id
loyalty level
ADMIN
user id
admin rank
RATING
Rrecipe id
rating id
rating value
COMMENTS
Crecipe id
title
text
Auser id
PHOTO
Precipe id
name
caption
Belongs to
category id
recipe id
Likes/Dislikes
user id
recipe id
title

SQL:

```sql
CREATE DATABASE recipe;
use recipe;

CREATE TABLE user (
    user_id  INT AUTO_INCREMENT PRIMARY KEY,
    name     VARCHAR(255) NULL,
    username VARCHAR(100) NOT NULL,
    password VARCHAR(100) NOT NULL,
    CONSTRAINT username UNIQUE (username)
);

CREATE TABLE category (
    category_id INT AUTO_INCREMENT PRIMARY KEY,
    name        VARCHAR(100) NOT NULL,
    description TEXT NULL
);

-- "ADMIN" is a subtype of user
CREATE TABLE admin (
    user_id    INT NOT NULL PRIMARY KEY,
    admin_rank INT NULL,
    CONSTRAINT admin_ibfk_1 FOREIGN KEY (user_id) REFERENCES user (user_id)
);

-- "CUSTOMER" is a subtype of user
CREATE TABLE customer (
    user_id       INT NOT NULL PRIMARY KEY,
    loyalty_level INT NULL,
    CONSTRAINT customer_ibfk_1 FOREIGN KEY (user_id) REFERENCES user (user_id)
);

CREATE TABLE recipe (
    recipe_id INT AUTO_INCREMENT PRIMARY KEY,
    name      VARCHAR(255) NOT NULL,
    steps     TEXT NULL,
    user_id   INT NULL,
    CONSTRAINT fk_user_recipe
        FOREIGN KEY (user_id) REFERENCES user (user_id)
        ON UPDATE CASCADE ON DELETE CASCADE
);

CREATE TABLE ingredient (
    ingredient_id INT AUTO_INCREMENT PRIMARY KEY,
    name          VARCHAR(255) NOT NULL,
    amount        VARCHAR(50)  NULL,   -- e.g. "2 cups", "3 tbsp"
    type          VARCHAR(50)  NULL,   -- e.g. "spice"
    recipe_id     INT NOT NULL,
    CONSTRAINT ingredient_ibfk_1 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id)
);

-- ====================================== -- 3. RATING, COMMENTS, PHOTO --
======================================

CREATE TABLE rating (
    rating_id    INT AUTO_INCREMENT PRIMARY KEY,
    recipe_id    INT NOT NULL,
    user_id      INT NOT NULL,
    rating_value INT NOT NULL,
    CONSTRAINT recipe_id UNIQUE (recipe_id, user_id),
    CONSTRAINT rating_ibfk_1 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id),
    CONSTRAINT rating_ibfk_2 FOREIGN KEY (user_id) REFERENCES user (user_id)
);

CREATE TABLE comments (
    comment_id INT AUTO_INCREMENT PRIMARY KEY,
    recipe_id  INT NOT NULL,
    user_id    INT NOT NULL,
    title      VARCHAR(255) NULL,
    text       TEXT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NULL,
    CONSTRAINT comments_ibfk_1 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id) ON DELETE CASCADE,
    CONSTRAINT comments_ibfk_2 FOREIGN KEY (user_id) REFERENCES user (user_id)
);

CREATE TABLE photo (
    photo_id   INT AUTO_INCREMENT PRIMARY KEY,
    recipe_id  INT NOT NULL,
    name       VARCHAR(255) NOT NULL,  -- filename or URL
    caption    TEXT NULL,
    created_at DATETIME DEFAULT CURRENT_TIMESTAMP NULL,
    CONSTRAINT photo_ibfk_1 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id) ON DELETE CASCADE
);
```

```
-- ====================================== -- 4. Relationship Tables --
======================================

-- SUBMITS:  many-to-many between user and recipe
CREATE TABLE submits (
    user_id     INT NOT NULL,
    recipe_id   INT NOT NULL,
    submit_date DATETIME DEFAULT CURRENT_TIMESTAMP NULL,
    PRIMARY KEY (user_id, recipe_id),
    CONSTRAINT submits_ibfk_1 FOREIGN KEY (user_id) REFERENCES user (user_id),
    CONSTRAINT submits_ibfk_2 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id)
);

-- BelongsTo:  many-to-many between recipe and category
CREATE TABLE belongs_to (
    category_id INT NOT NULL,
    recipe_id   INT NOT NULL,
    PRIMARY KEY (category_id, recipe_id),
    CONSTRAINT belongs_to_ibfk_1 FOREIGN KEY (category_id) REFERENCES category
(category_id) ON DELETE CASCADE,
    CONSTRAINT belongs_to_ibfk_2 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id) ON DELETE CASCADE
);

-- Likes/Dislikes:  many-to-many between user and recipe, plus boolean or enum
CREATE TABLE likes_dislikes (
    user_id   INT NOT NULL,
    recipe_id INT NOT NULL,
    liked     TINYINT(1) NULL,  -- TRUE = like, FALSE = dislike
    PRIMARY KEY (user_id, recipe_id),
    CONSTRAINT likes_dislikes_ibfk_1 FOREIGN KEY (user_id) REFERENCES user
(user_id),
    CONSTRAINT likes_dislikes_ibfk_2 FOREIGN KEY (recipe_id) REFERENCES recipe
(recipe_id) ON DELETE CASCADE
);

CREATE TABLE admin_removes_rating (
    admin_id    INT NOT NULL,   -- same as user_id in ADMIN
    rating_id   INT NOT NULL,
    remove_time DATETIME DEFAULT CURRENT_TIMESTAMP NULL,
    PRIMARY KEY (admin_id, rating_id),
    CONSTRAINT admin_removes_rating_ibfk_1 FOREIGN KEY (admin_id) REFERENCES
admin (user_id),
    CONSTRAINT admin_removes_rating_ibfk_2 FOREIGN KEY (rating_id) REFERENCES
```

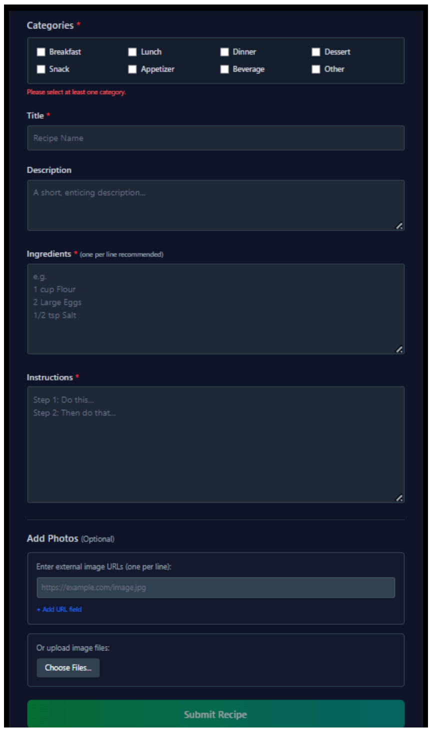
```sql
rating (rating_id)
);

-- ======================================
-- OPTIMIZATION: Add indexes for performance
-- ======================================

-- For sorting/filtering recipes
CREATE INDEX idx_recipes_id ON recipe (recipe_id);

-- For username lookups during authentication
CREATE INDEX idx_user_username ON user (username);

-- For comment-related queries
CREATE INDEX idx_comments_recipe ON comments (recipe_id);
CREATE INDEX idx_comments_user ON comments (user_id);

-- For rating aggregations
CREATE INDEX idx_ratings_recipe ON rating (recipe_id);
CREATE INDEX user_id ON rating (user_id);

-- For likes/dislikes performance
CREATE INDEX idx_likes_dislikes_recipe ON likes_dislikes (recipe_id);
CREATE INDEX idx_likes_dislikes_user ON likes_dislikes (user_id);

-- For ingredient queries
CREATE INDEX idx_ingredients_recipe ON ingredient (recipe_id);

-- For photo loading
CREATE INDEX idx_photos_recipe ON photo (recipe_id);

-- For category relationships
CREATE INDEX idx_belongs_to_recipe ON belongs_to (recipe_id);
CREATE INDEX idx_belongs_to_category ON belongs_to (category_id);

-- For admin removes rating
CREATE INDEX rating_id ON admin_removes_rating (rating_id);

-- For submits relationship
CREATE INDEX recipe_id ON submits (recipe_id);

-- Category data population
INSERT INTO category (name, description) VALUES
    ('Breakfast',  NULL),
    ('Lunch',      NULL),
    ('Dinner',     NULL),
    ('Dessert',    NULL),
    ('Snack',      NULL),
    ('Appetizer',  NULL),
    ('Beverage',   NULL),
    ('Other',      NULL);
```

Selected DBMS:

For our project, we selected MySQL as the database management system (DBMS) primarily due to its wide adoption and robust support for relational database models. As part of the class requirements, we were instructed to work with an SQL setup, and MySQL provides a stable and efficient solution for managing and querying relational data. Its integration with various programming languages and frameworks, along with features like referential integrity and ACID compliance, made it a suitable choice for implementing the Recipe Rater application. Additionally, the support for complex queries and its ability to handle a large number of concurrent connections ensure scalability as the platform grows.

Visual interface description

1. Add Recipe Page



Categories *

☐ Breakfast	☐ Lunch	☐ Dinner	☐ Dessert
☐ Snack	☐ Appetizer	☐ Beverage	☐ Other

Please select at least one category.

Title *

Recipe Name

Description

A short, enticing description...

Ingredients * (one per line recommended)

e.g.
1 cup Flour
2 Large Eggs
1/2 tsp Salt

Instructions *

Step 1: Do this...
Step 2: Then do that...

Add Photos (Optional)

Enter external image URLs (one per line):

https://example.com/image.jpg

Or upload image files:

Visual Hierarchy:

- Centered dark-themed form with gradient headers
- Category selection grid with glowing checkboxes
- Split image upload section (URLs + file uploads)
- Progressively enhanced form validation
- Gradient submit button with loading animation

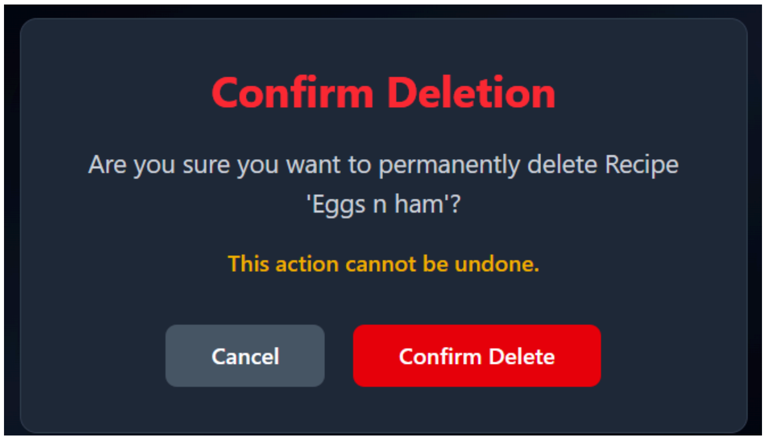
Key Interactions:

- Dynamic category validation
- Expandable URL input fields

- File preview with size indicators
- Real-time error feedback

---

2. Delete Confirmation Modal



Confirm Deletion

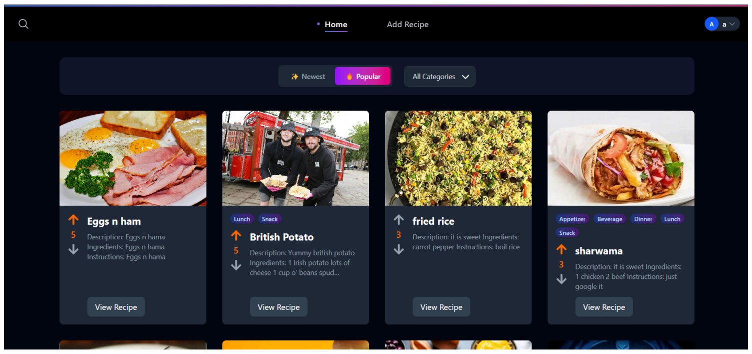
Are you sure you want to permanently delete Recipe 'Eggs n ham'?

This action cannot be undone.

Visual Elements:

- Radial gradient backdrop
- Glowing red warning icon
- Animated confirmation dialog
- Destructive action emphasis with red gradients

3. Home Feed



Eggs n ham
Description: Eggs n hama Ingredients: Eggs n hama Instructions: Eggs n hama

British Potato
Description: Yummy british potato Ingredients: 1 Irish potato lots of cheese 1 cup o' beans spud...

fried rice
Description: it is sweet Ingredients: carrot pepper Instructions: boil rice

sharwama
Description: it is sweet Ingredients: 1 chicken 2 beef Instructions: just google it

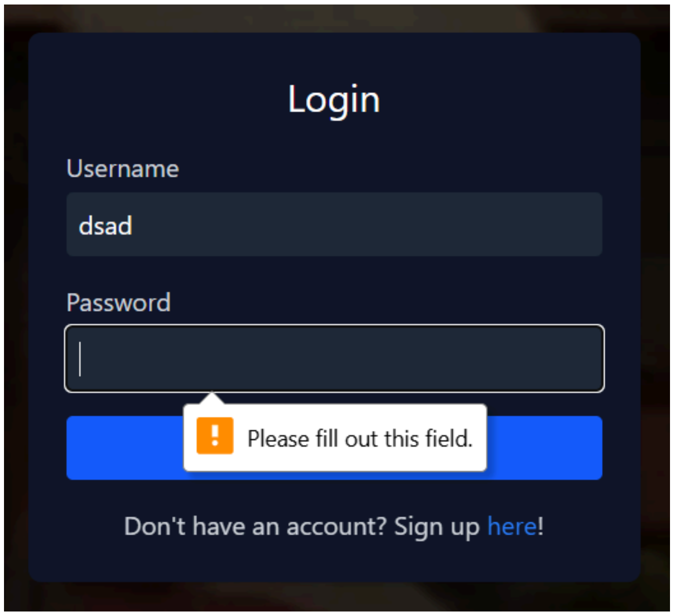
Layout Structure:

- Responsive grid of recipe cards
- Floating filter bar with sort controls
- Search results overlay with instant preview
- Loading skeletons with shimmer effects
- Empty state illustrations

Visual Features:

- Card hover effects with 3D transforms
- Category tag clouds with gradient backgrounds
- Interactive voting buttons with micro-animations
- Progressive image loading with blur-up effect

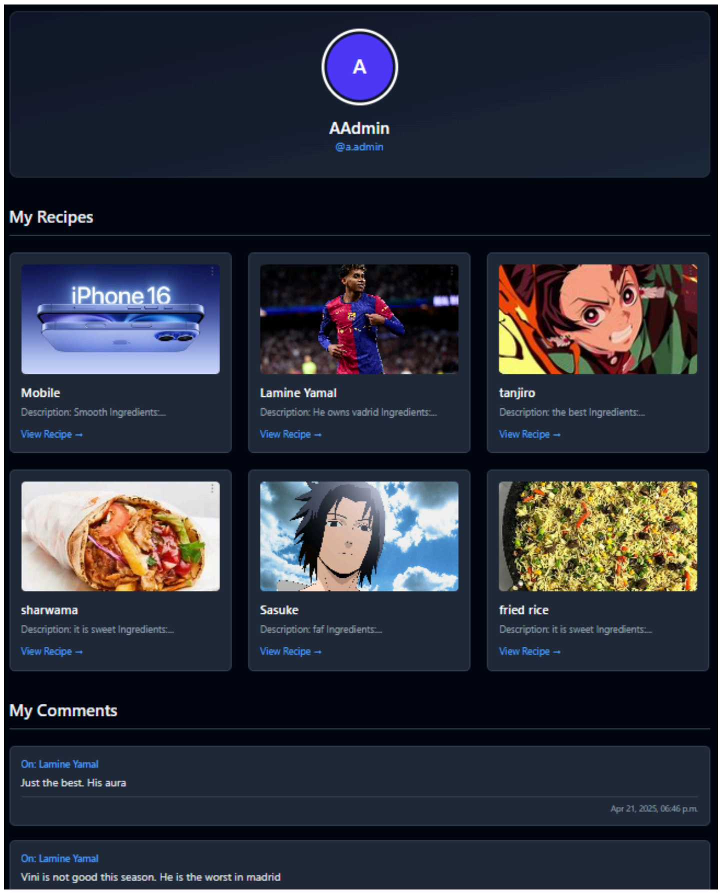
4. Authentication Flows


Login
Username
dsad
Password
Please fill out this field.
Don't have an account? Sign up here!

Shared Characteristics:

- Animated gradient backgrounds
- Floating form containers with glassmorphism
- Error message pulsing
- Social login integration points

5. User Profile



A

AAdmin

@a.admin

My Recipes

Mobile

Description: Smooth Ingredients:...

View Recipe →

Lamine Yamal

Description: He owns vadrid Ingredients:...

View Recipe →

tanjiro

Description: the best Ingredients:...

View Recipe →

sharwama

Description: it is sweet Ingredients:...

View Recipe →

Sasuke

Description: faf Ingredients:...

View Recipe →

fried rice

Description: it is sweet Ingredients:...

View Recipe →

My Comments

On: Lamine Yamal

Just the best. His aura

Apr 21, 2025, 06:46 p.m.

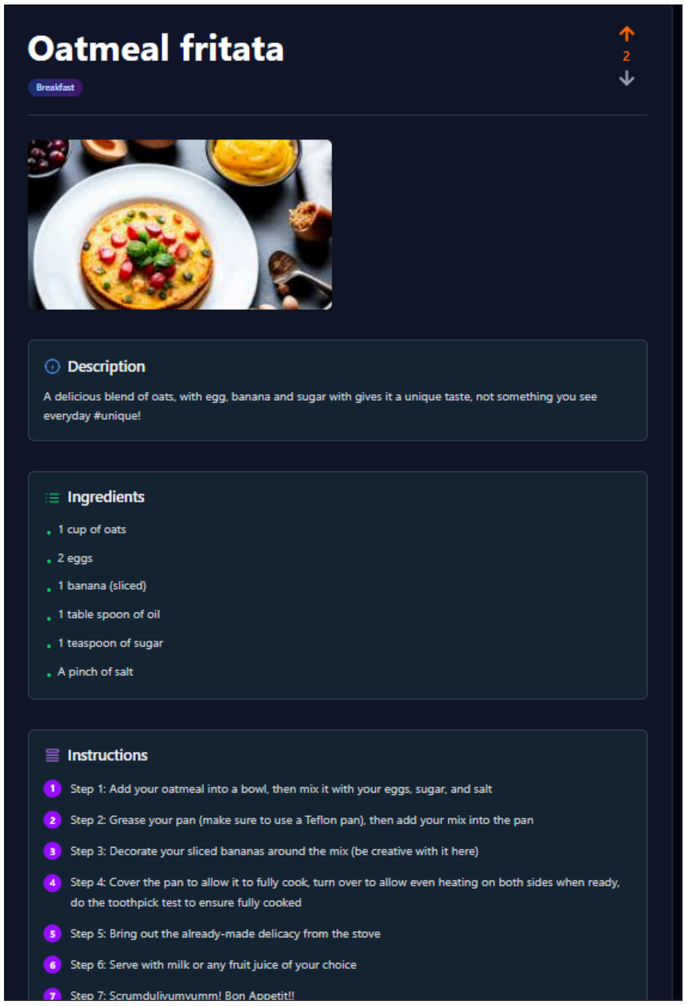
On: Lamine Yamal

Vini is not good this season. He is the worst in madrid

Interface Components:

- Avatar with dynamic initials/colors
- Expandable recipe cards with context menus
- Comment timeline with highlight edit/delete buttons

6. Recipe Detail View



Oatmeal fritata

Description

A delicious blend of oats, with egg, banana and sugar with gives it a unique taste, not something you see everyday #unique!

Ingredients

- 1 cup of oats
- 2 eggs
- 1 banana (sliced)
- 1 table spoon of oil
- 1 teaspoon of sugar
- A pinch of salt

Instructions

1. Step 1: Add your oatmeal into a bowl, then mix it with your eggs, sugar, and salt
2. Step 2: Grease your pan (make sure to use a Teflon pan), then add your mix into the pan
3. Step 3: Decorate your sliced bananas around the mix (be creative with it here)
4. Step 4: Cover the pan to allow it to fully cook, turn over to allow even heating on both sides when ready, do the toothpick test to ensure fully cooked
5. Step 5: Bring out the already-made delicacy from the stove
6. Step 6: Serve with milk or any fruit juice of your choice
7. Step 7: Scrumdulivumyumm! Bon Appetit!!

Content Presentation:

- Interactive ingredient checklist
- Step-by-step cooking mode

Visual Treatments:

- Directional lighting effect on images

---

7. Navigation System

UI Patterns:

- Sticky header with blur effect
- Context-aware search with typeahead
- Breadcrumb navigation with micro-interactions
- Adaptive menu system (desktop/mobile)
- Progress indicators for auth states

---

8. Card Components

Visual Design System:

- Consistent elevation shadows
- Gradient borders with animated states
- Responsive typography scaling
- Iconography with unified stroke weights
- placeholder images

Global Design Language

Core Principles:

- **Dark Theme**: #0F172A base with slate/indigo accents
- **Motion Design**: Spring physics for interactions
- **Accessibility**: WCAG 2.1 AA compliant contrasts
- **Responsive**: Fluid layouts from 320px to 4K
- **Microcopy**: Friendly tone with emoji integration

Signature Elements:

- Gradient overlays (blue → purple → pink)
- Geometric pattern backgrounds
- Animated transition between routes

Controllers/SQL query Setup:

---

1. `adminController.js`

Manages administrator records and permissions

- `getAllAdmins`
  - **Method**: GET
  - **Purpose**: Retrieves paginated list of admins with user details
  - **Parameters**: `limit` (default 10), `page` (default 1)
  - **Response**: JSON with admin list + pagination metadata
- `getAdminByUserId`
  - **Method**: GET
  - **Purpose**: Gets single admin by user ID
  - **Parameters**: `userId` (URL param)
  - **Response**: Admin details or 404
- `createAdmin`
  - **Method**: POST
  - **Purpose**: Promotes existing user to admin
  - **Parameters**: `user_id` (required), `admin_rank`
  - **Checks**: User existence, duplicate admin entries
- `updateAdmin`

    - **Method**: PATCH/PUT
    - **Purpose**: Updates admin rank
    - **Parameters**: `userId` (URL), `admin_rank` (body)
    - **Response**: Success message or 404
- `deleteAdmin`
    - **Method**: DELETE
    - **Purpose**: Removes admin privileges
    - **Parameters**: `userId` (URL)
    - **Note**: Doesn't delete user account

---

2. `adminRemovesRatingController.js`

Tracks rating removals by admins

- `getAllAdminRemoves`
    - **Method**: GET
    - **Purpose**: Lists all rating removal records
    - **Features**: Pagination, timestamp sorting
- `getAdminRemovesByAdmin`
    - **Method**: GET
    - **Purpose**: Shows all removals by specific admin
    - **Parameters**: `adminId` (URL)

- `getAdminRemovesByRating`
    - **Method**: GET
    - **Purpose**: Shows removal history for specific rating
    - **Parameters**: `ratingId` (URL)
- `createAdminRemoves`
    - **Method**: POST
    - **Purpose**: Logs a rating removal
    - **Parameters**: `admin_id`, `rating_id` (body)
    - **Constraint**: Unique (admin_id, rating_id) pair

---

3. `belongsToController.js`

Manages recipe-category relationships

- `getAllBelongsTo`
    - **Method**: GET
    - **Purpose**: Lists all category-recipe associations
    - **Features**: Pagination
- `createBelongsTo`
    - **Method**: POST
    - **Purpose**: Links recipe to category
    - **Parameters**: `category_id`, `recipe_id` (body)

    - **Constraint**: Prevents duplicate associations
- `deleteBelongsTo`
    - **Method**: DELETE
    - **Purpose**: Removes recipe from category
    - **Parameters**: Composite key (URL params)

---

4. `categoryController.js`

Manages recipe categories

- `searchCategories`
    - **Method**: GET
    - **Purpose**: Fuzzy search by category name
    - **Parameters**: `query` (URL query)
- `createCategory`
    - **Method**: POST
    - **Purpose**: Adds new category
    - **Parameters**: `name` (required), `description`
- `updateCategory`
    - **Method**: PATCH
    - **Purpose**: Modifies category details
    - **Parameters**: Partial updates supported

---

5. `commentController.js`

Handles recipe comments

- `getCommentsByRecipe`
  - **Method**: GET
  - **Purpose**: Gets all comments for a recipe
  - **Features**: Includes username
- `createComment`
  - **Method**: POST
  - **Purpose**: Adds new comment
  - **Validation**: Checks recipe/user existence
- `updateComment` **(INSECURE)**
  - **Method**: PATCH
  - **Security Note**: Relies on client-provided user ID
  - **Logic**: Allows owner or admin edits

---

6. `customerController.js`

Manages customer profiles

- `createCustomer`
  - **Method**: POST
  - **Purpose**: Converts user to customer

    - **Parameters**: `user_id` (must exist in users)

- `updateCustomer`
    - **Method**: PATCH
    - **Purpose**: Updates loyalty level
    - **Parameters**: `loyalty_level` (body)

---

7. `ingredientController.js`

Manages recipe ingredients

- `searchIngredients`
    - **Method**: GET
    - **Purpose**: Searches by name/type
    - **Parameters**: `query` (URL query)
- `createIngredient`
    - **Method**: POST
    - **Purpose**: Adds ingredient to recipe
    - **Parameters**: `name`, `recipe_id` (required)

---

8. `likesDislikesController.js`

Handles recipe reactions

- `createLikesDislikes`

    - **Method**: POST
    - **Purpose**: Records like/dislike
    - **Parameters**: `user_id`, `recipe_id`, `liked` (boolean)
- **`updateLikesDislikes`**
    - **Method**: PATCH
    - **Purpose**: Changes reaction type
    - **Parameters**: `liked` (new boolean value)

---

9. `photoController.js`

Manages recipe photos

- **`createPhoto`**
    - **Method**: POST
    - **Features**: File upload handling
    - **Security**: Deletes orphaned files on error
    - **Response**: Returns accessible URL
- **`addPhotoFromUrl`**
    - **Method**: POST
    - **Purpose**: Links external images
    - **Validation**: Basic URL format check

---

10. `ratingController.js`

Manages recipe ratings

- `getAverageRatingForRecipe`
  - **Method**: GET
  - **Purpose**: Calculates recipe's avg rating
  - **Parameters**: `recipeId` (URL)
- `createRating`
  - **Method**: POST
  - **Constraint**: Unique (user, recipe) pair
  - **Parameters**: 1-5 rating value

---

11. `recipeController.js`

Core recipe management

- `getAllRecipes`
  - **Method**: GET
  - **Features**: Sorting (newest/top-rated), pagination
  - **Includes**: Main photo URL, vote counts
- `combinedSearch`
  - **Method**: GET
  - **Purpose**: Unified search across recipes/ingredients/categories
  - **Parameters**: `query` (URL query)
- `getRecipeById`

- **Method**: GET
- **Features**: Includes aggregated category list

---

12. `userController.js`

Handles authentication

- `registerUser`
  - **Method**: POST
  - **Features**: Optional admin creation
  - **Security**: Session initialization
- `loginUser`
  - **Method**: POST
  - **Authentication**: Plain-text comparison (demo only)
  - **Session**: Stores user role (admin/user)
- `getProfile`
  - **Method**: GET
  - **Purpose**: Returns current session data

Installation Guide

Overview

This guide will walk you through the process of setting up the Recipe Rater application on your local machine. The application consists of a frontend built with React and a backend API connected to a MySQL database.

Prerequisites

- Git
- Node.js and npm
- MySQL server

Installation Steps

Step 1: Clone the Repository

First, clone the repository to your local machine:

```
git clone https://github.com/alia720/Recipe-Rater.git
cd Recipe-Rater
```

Step 2: Install Dependencies

Frontend Setup

Navigate to the frontend directory and install the dependencies:

```
cd frontend
npm install
```

Backend Setup

Next, go to the backend directory and install the necessary dependencies:

```
cd ../backend
npm install
```

Step 3: Configure the Environment

In the `backend/.env` file, configure your database connection settings:

```
DB_HOST=127.0.0.1
DB_PORT=3306
DB_USER=root
DB_PASSWORD=ingredient
DB_DATABASE=recipe_db
```

Configuration Details:

- **DB_HOST**: Your MySQL server address (127.0.0.1 for local setup)
- **DB_PORT**: The port MySQL runs on (default is 3306)
- **DB_USER**: Your MySQL username (typically `root` for local setups)
- **DB_PASSWORD**: Db password which is `ingredient`
- **DB_DATABASE**: The name of the database (`recipe_db`)

Step 4: Initialize the Database

Create the Database

Create a MySQL database named `recipe_db` by running the following command in your MySQL shell or client:

CREATE DATABASE recipe_db;

Import the Schema

Next, import the schema provided in the `backend/recipe.sql` file into your `recipe_db` database:

```
mysql -u root -p recipe_db < backend/recipe.sql
```

This will populate your database with the necessary tables for the Recipe Rater application.

Step 5: Running the Application

Start the Backend Server

Navigate to the backend directory and start the server:

```
cd backend
npm start
# Or just run
# node index.js
```

The backend will be running at: `http://localhost:5000`

Start the Frontend Development Server

Navigate to the frontend directory and start the development server:

```
cd frontend
npm run dev
```

The frontend will be running at: `http://localhost:5173`

Verification

Once both servers are running, open your browser and navigate to `http://localhost:5173`. You should see the Recipe Rater application interface.

Troubleshooting

If you encounter any issues during installation:

- **Database Connection Errors**: Verify your MySQL credentials and ensure the MySQL service is running
- **Port Conflicts**: If ports 5000 or 5173 are in use, you can modify the port settings in the respective configuration files
- **Dependency Issues**: Ensure you have the correct versions of Node.js and npm installed