**Recipe Sharing Platform**

**Overview:**

A web-based platform where users can share, browse, and review recipes. Users can register, log in, submit their recipes, comment on and rate others' recipes, and categorize their favorite recipes.

**Collections:**

**Recipes**

* \_id (ObjectId)
* title (String)
* ingredients (Array of Strings)
* instructions (String)
* cook\_time (Number)
* servings (Number)
* category (ObjectId, referencing Categories)
* author\_id (ObjectId, referencing Users)
* rating (Number)
* reviews (Array of ObjectIds, referencing Comments)

**Users**

* \_id (ObjectId)
* name (String)
* email (String)
* password (String, hashed)
* favorite\_recipes (Array of ObjectIds, referencing Recipes)
* submitted\_recipes (Array of ObjectIds, referencing Recipes)
* comments (Array of ObjectIds, referencing Comments)

**Comments**

* \_id (ObjectId)
* user\_id (ObjectId, referencing Users)
* recipe\_id (ObjectId, referencing Recipes)
* content (String)
* timestamp (Date)

**Categories**

* \_id (ObjectId)
* name (String)
* description (String)

**API Routes:**

1. Recipes

* GET /recipes
  + Description: Retrieve all recipes.
  + Response: Array of recipe objects.
* GET /recipes/:id
  + Description: Retrieve a single recipe by its ID.
  + Response: Recipe object.
* POST /recipes
  + Description: Create a new recipe.
  + Request Body: Recipe data (title, ingredients, instructions, cook\_time, servings, category).
  + Response: Created recipe object.
  + Validation: Ensure all required fields are present and valid.
* PUT /recipes/:id
  + Description: Update an existing recipe by its ID.
  + Request Body: Updated recipe data.
  + Response: Updated recipe object.
  + Validation: Ensure fields are valid.
* DELETE /recipes/:id
  + Description: Delete a recipe by its ID.
  + Response: Success message.

1. Users

* GET /users/:id
  + Description: Retrieve user profile.
  + Response: User object.
* POST /users
  + Description: Register a new user.
  + Request Body: User data (name, email, password).
  + Response: Created user object.
  + Validation: Ensure email is unique and password is hashed.
* PUT /users/:id
  + Description: Update user profile.
  + Request Body: Updated user data.
  + Response: Updated user object.
  + Validation: Ensure valid email format and password is hashed.
* DELETE /users/:id
  + Description: Delete a user account.
  + Response: Success message.

1. Comments

* GET /comments
  + Description: Retrieve all comments.
  + Response: Array of comment objects.
* GET /comments/:id
  + Description: Retrieve a single comment by its ID.
  + Response: Comment object.
* POST /comments
  + Description: Create a new comment.
  + Request Body: Comment data (user\_id, recipe\_id, content).
  + Response: Created comment object.
  + Validation: Ensure all required fields are present and valid.
* PUT /comments/:id
  + Description: Update an existing comment by its ID.
  + Request Body: Updated comment data.
  + Response: Updated comment object.
  + Validation: Ensure content is valid.
* DELETE /comments/:id
  + Description: Delete a comment by its ID.
  + Response: Success message.

1. Categories

* GET /categories
  + Description: Retrieve all categories.
  + Response: Array of category objects.
* GET /categories/:id
  + Description: Retrieve a single category by its ID.
  + Response: Category object.
  + POST /categories
  + Description: Create a new category.
  + Request Body: Category data (name, description).
  + Response: Created category object.
  + Validation: Ensure name is unique.
* PUT /categories/:id
  + Description: Update an existing category by its ID.
  + Request Body: Updated category data.
  + Response: Updated category object.
  + Validation: Ensure name is unique.
* DELETE /categories/:id
  + Description: Delete a category by its ID.
  + Response: Success message.

**Authentication & User Management:**

* Implement OAuth for user authentication and management.
* Users can register, log in, and manage their profiles.
* Protect routes that require authentication (e.g., posting a recipe, commenting) using middleware.

**API Documentation:**

* Use a tool like Swagger to generate comprehensive API documentation.
* Ensure the documentation is accessible at /api-docs on Render.

**Publishing:**

* Deploy the API to Render and ensure it is accessible with all routes functional and documented.