Recipe Studio

Architecture

Frontend:

We will use react-native on frontend with mutations and queries to call api endpoint. We will have login screen, a feed, a saved screen to display saved recipes and a post button for user to post a new recipe.

Backend:

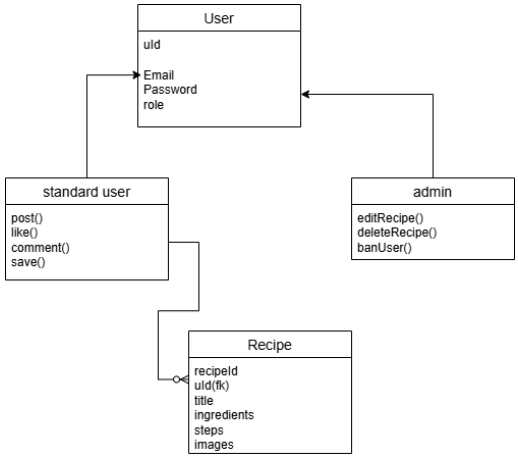
Backend will be in node js. We will have endpoints like
login
fetch-user-data
fetch-recipes
post-recipes
delete-recipes
ban-user (Admin)
like-recipe
comment
Save-recipe
fetch-saved-recipe

Database:

Data regarding the flows will be stored in pgAdmin

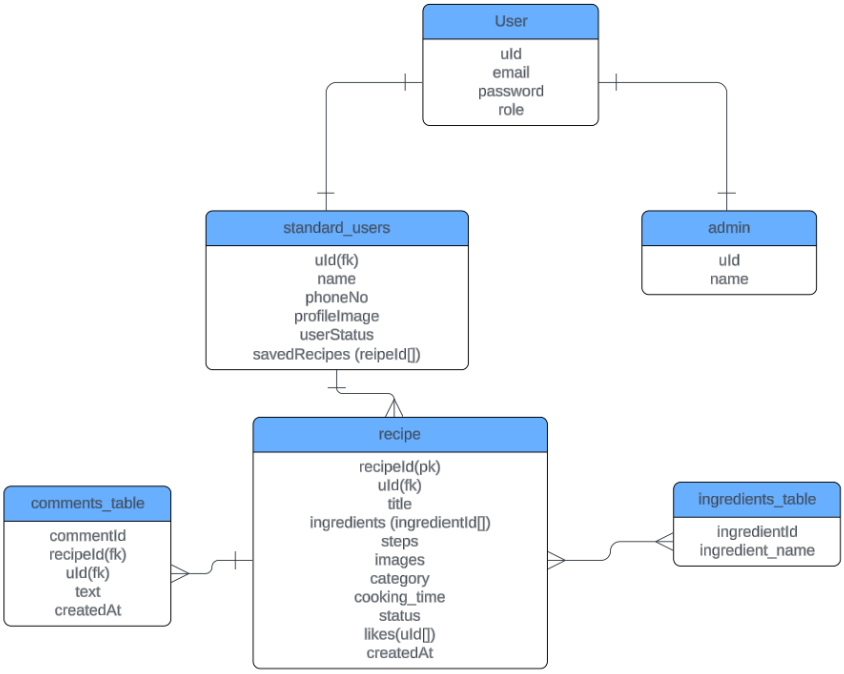
Overall Structure:

Here is an overall structure of relation of all the entities involved in this flow:



Firstly we have a generalized user, standard user and admins will inherit from it. Standard users can post, like, comment and save recipes while admins can edit and delete a recipe and ban a user. Recipes will have a one to many relation with standard user. One user can post multiple recipes. Recipe will have recipeId as primary key and uId as foreign key from the user. Broadly viewing the recipe entity, it will have title, ingredients, steps and images etc. more details on each of these entities will be mentioned in database schema.

Database Design:



In db, we will have a user table which will store userID, email, password and role. This will be used during user login and data of user will be fetched based on the role.
In standard user table we will have username, phoneNumber, user's profile image url, user status to track whether user is active or banned. And an array of recipeIds by name savedRecipes to keep track of recipes saved by user.
In admin table we will only have uId as foreign key and admin name.
In recipe table, we have recipeId as primary key to uniquely identify each recipe and we have uId to track which recipe was posted by which user. Then we have title, steps, images of recipe. For ingredients we will have an array of ingredientsIds. We have status field to track if recipe is a draft or if it has been posted. For likes we have an array of uIds. Based on these ids, data regarding likes will be fetched from user tables. Lastly we have createdAt, which will be a timestamp to track the date a recipe was posted.
Ingredients table will simply hold an id as primary key and name of ingredient, this will help in searching recipes on basis of ingredients. Will can run a query to get ingredient id and search recipe table to check if ingredient array contains that id.
Comments table will have a one to many relation with recipe. Each recipe can have multiple comments. Then we have uId in comments table to keep track of users who posted these comments.

In recipe table we have created at and likes array which will help us in tunning users feed based on recent posts and most liked posts.

Scalability:

The proposed database schema is designed with scalability in mind by ensuring a clear separation of concerns and leveraging relational and array-based data structures for efficient data retrieval. The use of a **role-based user system** allows flexibility in handling different types of users (standard users and admins), making it easier to expand functionality without restructuring the core user data. By storing references such as recipeIds and ingredientsIds as arrays within respective tables, the schema minimizes data redundancy and allows for efficient one-to-many relationships. The inclusion of timestamps (createdAt) and status fields enables efficient querying for feed personalization, making it easier to scale the application by fetching relevant content without overloading the database. Additionally, by using foreign key relationships between tables like uId in multiple tables, the schema ensures data integrity while allowing horizontal scaling strategies such as sharding, replication, and partitioning to distribute the data load effectively as the user base grows.

Performance Optimization Strategies:

To optimize performance, several strategies can be implemented based on the schema design. **Indexing** on frequently queried fields such as uId, recipeId, and ingredientId will speed up lookup operations, particularly when filtering recipes based on saved items or ingredient searches. **Caching mechanisms** such as Redis can be employed to store frequently accessed data like user profiles and popular recipes, reducing the load on the database. Query optimization techniques such as using **JOINs efficiently**, avoiding unnecessary complex queries, and applying pagination for large datasets can enhance response times. Additionally, implementing **denormalization** for read-heavy operations—such as storing aggregated like counts within the recipe table—can minimize the need for repeated joins.