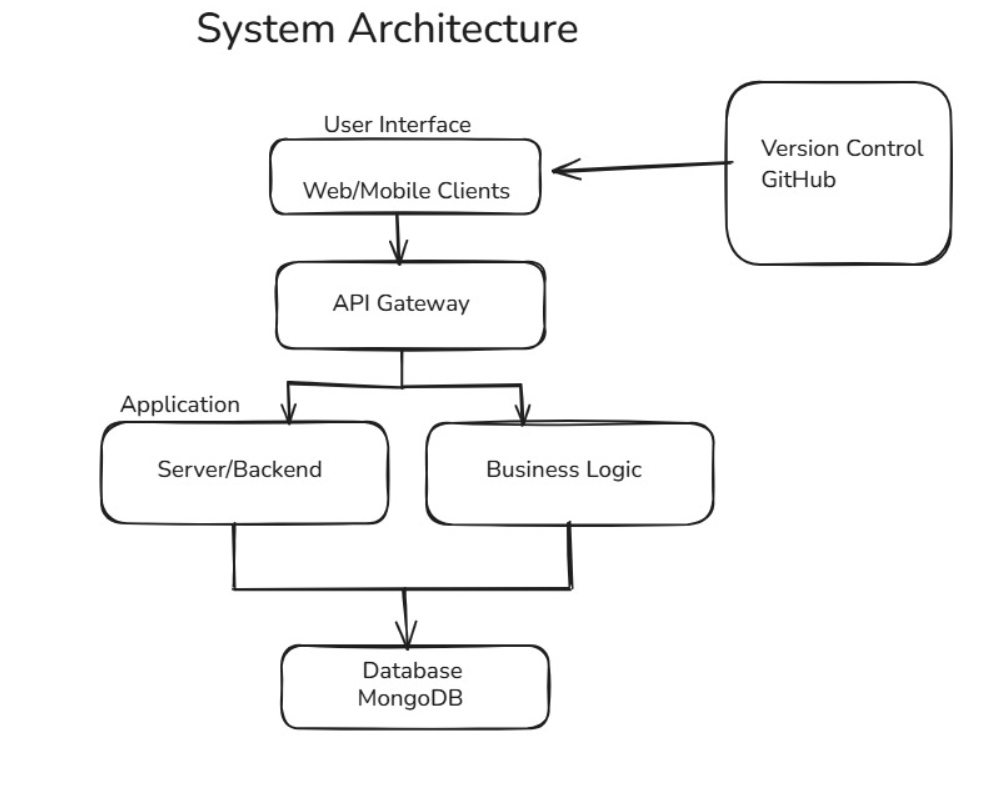
***Architecture***

**System Architecture:**



**ER-Diagram:**

A diagram of a company

AI-generated content may be incorrect.

**Database Tables:**

**Users**

* id (PK): Unique identifier for the user
* name: User’s name
* email: Email address
* password: Hashed password
* created\_at: Account creation timestamp
* updated\_at: Last profile update timestamp

**Recipes**

* id (PK): Unique identifier for the recipe
* user\_id (FK): References Users.id
* title: Title of the recipe
* description: Brief description
* image\_url: Link to recipe image
* created\_at: Creation timestamp
* updated\_at: Last update timestamp

**Recipe\_Comments**

* id (PK): Unique identifier for the comment
* recipe\_id (FK): References Recipes.id
* user\_id (FK): References Users.id
* comment: Comment text
* rating: Rating value
* created\_at: Timestamp when comment was added

**Ingredients**

* id (PK): Unique identifier for each ingredient
* name: Name of the ingredient

**Recipe\_Ingredients**

* id (PK): Unique identifier for the relation
* recipe\_id (FK): References Recipes.id
* ingredient\_id (FK): References Ingredients.id
* quantity: Amount of the ingredient
* unit: Measurement unit

**Recipe\_Steps**

* id (PK): Unique identifier for the step
* recipe\_id (FK): References Recipes.id
* step\_number: Order of the step
* description: Step instructions

**Categories**

* id (PK): Unique identifier for the category
* name: Name of the category

**MongoDB Collections:**

**Users**

{

"\_id": "ObjectId",

"name": "John Doe",

"email": "john@example.com",

"password": "hashed\_pw",

"created\_at": "2024-01-01T00:00:00Z",

"updated\_at": "2024-01-05T12:00:00Z"

}

**Recipes**

{

"\_id": "ObjectId",

"user\_id": "ObjectId",

"title": "Spaghetti",

"description": "Classic Italian pasta",

"image\_url": "url",

"created\_at": "2024-06-01T12:00:00Z",

"updated\_at": "2024-06-01T12:30:00Z"

}

**Comments:**

{

"\_id": "ObjectId",

"recipe\_id": "ObjectId",

"user\_id": "ObjectId",

"comment": "Loved it!",

"rating": 5,

"created\_at": "2024-06-02T10:00:00Z"

}

* ***Developer Guide***

**Folder Structure:**

* **/backend**
  + src/models: Contains Mongoose schemas for Users, Recipes, Comments, etc.
  + src/routes: Express route handlers organized by feature (e.g., recipeRoutes.js, userRoutes.js)
  + src/services: Custom services (e.g., Google Cloud Storage integration, Gemini AI interaction)
  + server.js: Entry point; sets up Express app, middleware, routes, DB connection
* **/frontend**
  + src/components: Reusable UI components (Navbar, RecipeCard, etc.)
  + src/pages: Page-level components (HomePage, UploadPage, RecipeDetailPage)
  + src/routes: Routing definitions and protected route wrappers
  + src/services: Axios-based service layer for API communication
  + layouts: Layout components used for consistent page structure
  + main.jsx: Entry point for the React app; mounts App component

**Codebase Walkthrough:**

* **Backend:**
  + server.js initializes Express, connects to MongoDB, sets up CORS, JSON parsing, and routes.
  + Routes are modularized (e.g., /add-recipe, /like-recipe, /comments) and handle logic for each feature.
  + Multer handles file uploads; images are stored in Google Cloud Storage using a utility in services/gcsUploader.js.
  + Gemini AI integration in a dedicated route under /api/gemini/recipe which sends a prompt and returns a recipe suggestion.
* **Frontend:**
  + App.jsx sets up routing and base layout.
  + Navbar.jsx manages navigation with links to core pages (e.g., Upload, Browse).
  + RecipeForm.jsx in pages handles form submission with file/image uploads.
  + Auth is managed with Clerk’s components (e.g., <SignIn />, <SignedIn />, etc.).
  + RecipeCard.jsx renders individual recipes using props passed from API responses.
  + services/api.js handles all API calls to the backend with axios.