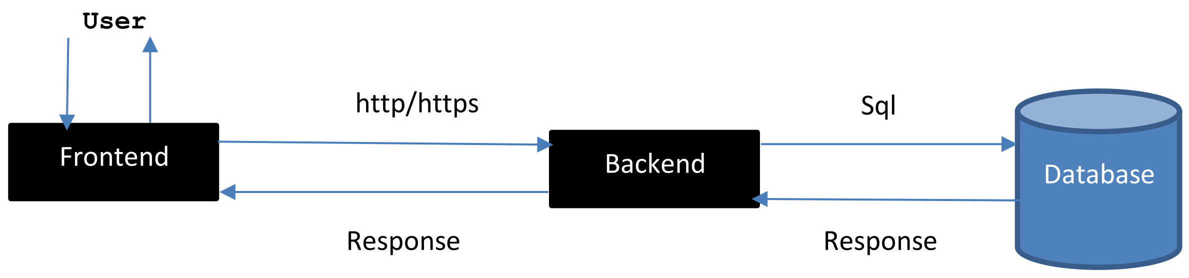
1. Architecture Diagram

A simple diagram for the recipe collection app



- **User**: Visits the app in a browser
- **Frontend**: HTML/CSS/JS running in browser
- **Backend**: Server (e.g., Node.js) that handles logic
- **Database**: Stores recipes permanently
- **HTTP**: Protocol used to send requests (GET, POST, PUT and DELETE)

---

2. Identify the Components

Where is the recipe data stored?

The recipes are stored in a **database** (e.g., SQL database like PostgreSQL / MySQL, or even a simple JSON file). The backend reads from and writes to this database.

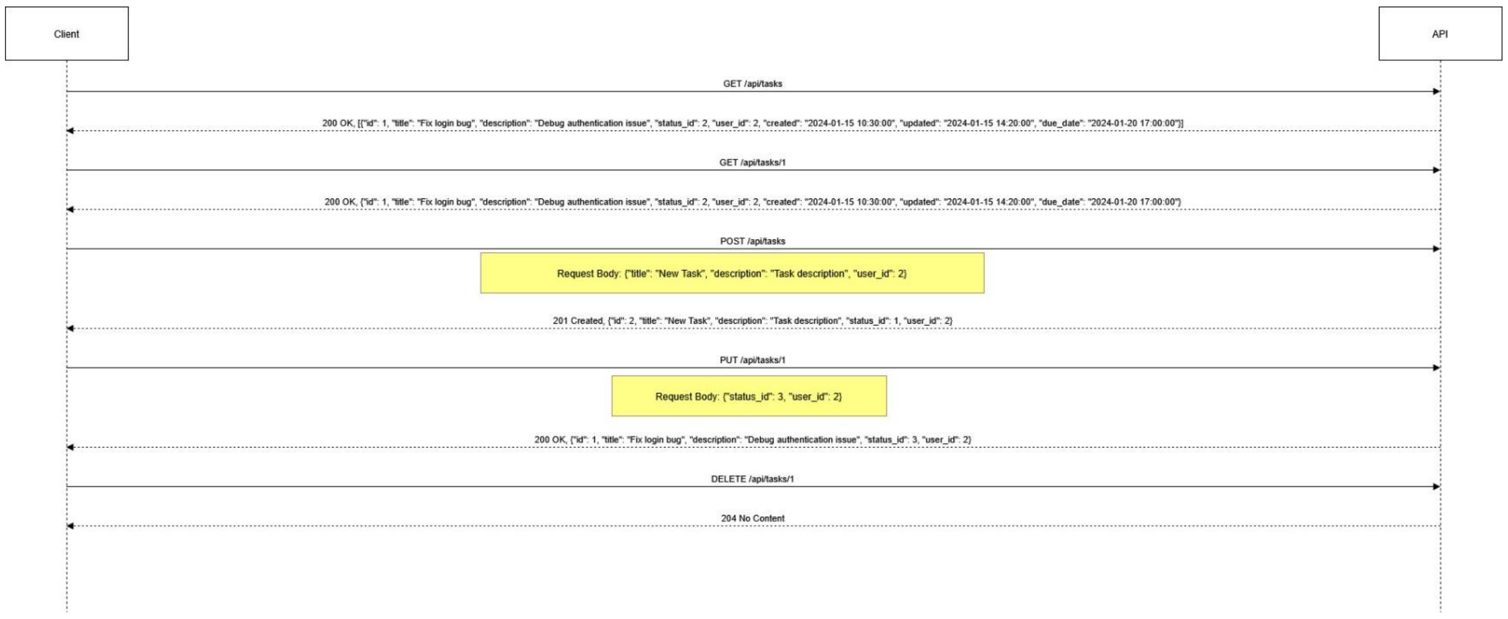
How does the frontend communicate with the backend?

The frontend communicates with the backend using **HTTP requests** (Such as GET, POST, PUT, DELETE) to a set of API endpoints such as:

- `GET /recipes` → fetch all recipes
- `GET /recipes/:id` → fetch one recipe
- `POST /recipes` → add new
- `PUT /recipes/:id` → edit
- `DELETE /recipes/:id` → remove
  These requests are sent by the frontend (in JavaScript fetch calls) and received by the backend API.

What happens when a user adds a new recipe?

1. User fills form in the frontend (title/ingredients/instructions)
2. Frontend sends `POST /recipes` to backend with recipe data
3. Backend receives it, validates, and saves to database
4. Backend returns success to frontend
5. Frontend shows updated recipe list or confirmation message


Client
API
GET /api/tasks
200 OK, [{"id": 1, "title": "Fix login bug", "description": "Debug authentication issue", "status_id": 2, "user_id": 2, "created": "2024-01-15 10:30:00", "updated": "2024-01-15 14:20:00", "due_date": "2024-01-20 17:00:00"}]
GET /api/tasks/1
200 OK, {"id": 1, "title": "Fix login bug", "description": "Debug authentication issue", "status_id": 2, "user_id": 2, "created": "2024-01-15 10:30:00", "updated": "2024-01-15 14:20:00", "due_date": "2024-01-20 17:00:00"}
POST /api/tasks
Request Body: {"title": "New Task", "description": "Task description", "user_id": 2}
201 Created, {"id": 2, "title": "New Task", "description": "Task description", "status_id": 1, "user_id": 2}
PUT /api/tasks/1
Request Body: {"status_id": 3, "user_id": 2}
200 OK, {"id": 1, "title": "Fix login bug", "description": "Debug authentication issue", "status_id": 3, "user_id": 2}
DELETE /api/tasks/1
204 No Content

3. Sequence Diagrams

1. Adding a New Recipe

```
User → Frontend → Backend → Database
User clicks "Add Recipe"
Frontend sends POST /recipes with recipe data
Backend saves data to database
Backend sends success response
Frontend updates page
```

Key HTTP method: `POST`
Flow focuses on saving new data

---

2. Viewing a Specific Recipe

```
User → Frontend → Backend → Database
User opens specific recipe
Frontend sends GET /recipes/:id
Backend fetches recipe from database
Backend returns recipe data
Frontend displays recipe
```

Key HTTP method: `GET`
Flow focuses on retrieving existing data

---

3. Editing an Existing Recipe

```
User → Frontend → Backend → Database
User clicks "Edit"
Frontend sends PUT /recipes/:id with updated data
Backend updates record in database
Backend sends success response
Frontend refreshes the display
```

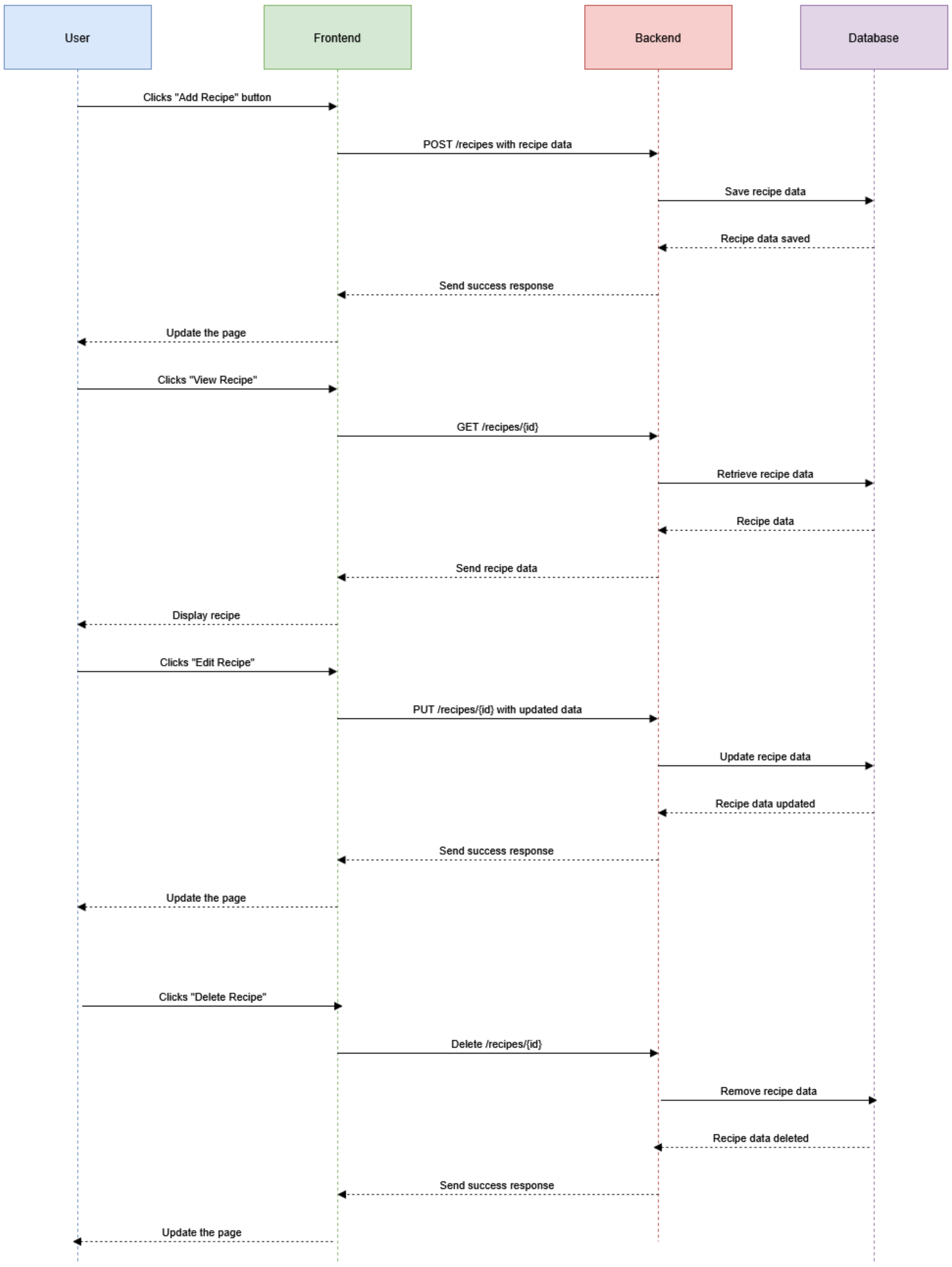
Key HTTP method: `PUT`
Flow focuses on updating stored data

---

4. Delete an Existing Recipe

```
User → Frontend → Backend → Database
User clicks "Delete"
Frontend sends DELETE /recipes/:id
Backend Delete record in database
Backend sends success response
Frontend refreshes the display
```

Key HTTP method: DELETE
Flow focuses on deleting stored data


User
Frontend
Backend
Database
Clicks "Add Recipe" button
POST /recipes with recipe data
Save recipe data
Recipe data saved
Send success response
Update the page
Clicks "View Recipe"
GET /recipes/{id}
Retrieve recipe data
Recipe data
Send recipe data
Display recipe
Clicks "Edit Recipe"
PUT /recipes/{id} with updated data
Update recipe data
Recipe data updated
Send success response
Update the page
Clicks "Delete Recipe"
Delete /recipes/{id}
Remove recipe data
Recipe data deleted
Send success response
Update the page