**Backend Tech Stack Breakdown**

**API Implementation (Backend)**

The API is implemented using FastAPI and interacts with an SQLite database named TutorConnect.db. Below is an explanation of how the API accesses and manages the database:

**1. Database Connection**

The API connects to the SQLite database using the sqlite3 module. The database file TutorConnect.db is used to store user information.

conn = sqlite3.connect(db\_path)

cursor = conn.cursor()

Every time an endpoint requires database access, a connection is established and closed after executing queries.

**2. Creating a User**

The endpoint /users/create allows the frontend to send user details (name, email, and password) via an HTML form. This data is inserted into the database. A computer screen shot of a program code

AI-generated content may be incorrect.

**3. Retrieving Users HTMX**

The /users endpoint retrieves all users from the database and returns the data as an HTML table for display in the frontend. This or the next method can be used.

A computer screen shot of a program

AI-generated content may be incorrect.**4. Retrieving Users as JSON**

For API clients that need a JSON response, /users/json returns the list of users as JSON objects. We can use either type of endpoint.A computer screen shot of a program code

AI-generated content may be incorrect.

**Frontend Implementation (HTML)**

The frontend interacts with the API using **HTMX**, which allows dynamic updates without full page reloads and cuts down a lot of the JS required. The HTML page (usersTest.html) includes buttons and forms to trigger API calls. This is just a test page to show how to implement it to the actual application pages.

**1. Loading Users**

A button is provided to fetch user data from /users and dynamically update the #user-table div.

A black screen with orange text

AI-generated content may be incorrect.When clicked, HTMX sends a GET request to /users, and the response (an HTML table) replaces the contents of the #user-table div.

**2. Adding a New User**

A form is provided to create a new user. It submits data via an hx-post request to /users/create, and the response (updated user table) is displayed in #user-table.

A computer screen shot of text

AI-generated content may be incorrect.When the form is submitted, an HTTP POST request is sent to /users/create. The response replaces the existing #user-table with the updated user list.

**Summary**

1. The **API** (FastAPI) connects to an SQLite database to create and fetch user data.
2. **HTMX** in the frontend makes requests to API endpoints (Read,Create,Update,Delete etc).
3. The /users endpoint returns an **HTML table** of users, which is directly inserted into the page.
4. The /users/create endpoint accepts form submissions, adds users to the database, and updates the displayed user list automatically.
5. The /users/json endpoint provides a **JSON API** alternative for structured data retrieval.

We have not decided on which API technique we should use.

**Pros and Cons of HTMX:**

**HTMX Pros:**

Less JavaScript – Avoids writing complex JavaScript logic for making AJAX requests and updating the DOM.

Simpler Codebase – Keeps most of the logic in HTML attributes (hx-get, hx-post, etc.), reducing the need for separate JavaScript files.

Easier to Maintain – Cleaner, more readable code as interaction logic stays within the HTML markup.

Built-in AJAX Support – Handles HTTP requests natively with minimal setup.

Declarative Approach – Uses attributes like hx-trigger, hx-target, and hx-swap for event-driven UI updates, making it intuitive.

**HTMX Cons:**

Limited Client-Side Logic – Not suitable for complex client-side applications requiring significant state management or dynamic updates.

SEO Concerns – If content is loaded dynamically, search engines may not index it properly (depends on backend rendering).

Debugging Can Be Tricky – Debugging issues within hx- attributes can sometimes be harder than plain JavaScript.

Less Popular – Fewer resources, tutorials, and third-party libraries compared to React, Vue, or even jQuery.

Not a Full Framework – Lacks built-in state management, routing, or component-based UI structure.