LLM Apps

Full Stack Level 1 App: To Do App

Goal

- Understand the complete cycle of full-stack web application development to apply it in LLM web applications like those analyzed in LlamaIndex.

Recommendations

- The most important thing is to understand the basic concepts. Code versions can change, but concepts remain.

- You don't need to learn everything about Python, FastAPI, Postgress, Next.js, Vercel, and Render.com. Learn the basic techniques and then keep learning as needed for each project you do. Keep an eye on the “big picture,” don't lose focus on small details now.

- Remember: no software engineer knows everything needed to develop a project beforehand. They have basic knowledge and progress from there according to the needs of each project.

- Remember where to find answers in case of doubt or error: ChatGPT, Google, StackOverflow, etc.

Essential backend logic

- We want to create a CRUD application that responds to 5 instructions:
  - Create an item in a database.
  - Show all items in the database.
  - Show an item from the database located by its ID.
  - Update an item in the database located by its ID.
  - Delete an item from the database located by its ID.

- Each time one of these instructions is ordered:
  - A Python function (CRUD helper) that carries out the selected CRUD action is executed.

- The result is displayed on a web URL (CRUD route).

Part 1: Backend with FastAPI and Postgres

1. Initial Setup.
2. Database.
3. CRUD Helpers.
4. Routes and Endpoints.
5. Test the backend on a local server.

1.1. Initial Setup

1. Create a virtual environment.
2. Install the necessary packages.

1.2. Database

1. Create the database.
2. Enter credentials in the .env file.
3. Validate data types with Pydantic.
4. Create a table in the database with SQLAlchemy and Alembic.

1.3. CRUD Helpers

1. Create the schema to determine the validation criteria for the data entered by the user.
2. Create the data model to determine the format of the data to be entered into the database table.
3. Using the schema and data model, create the CRUD helpers that manage the data input and output operations in the database:
   a. Create: create a new item.
   b. Read: show all items or a specific one.
   c. Update: update an item.
   d. Delete: delete an item.

1.4. Routes (URLs) and Endpoints (Executed Function)

1. FastAPI CRUD route: URL associated with each CRUD action.
2. Endpoint: Python function executed when each route is called.
    a. Each CRUD endpoint makes use of a CRUD helper.

1.5. Test the backend on a local server

1. Create a local server with unicorn.
2. Open a second terminal window to test the backend.

Part 2: Frontend with Next.js

1. Create a Next.js starter template.
2. Enter the backend API URL in the .env file.
3. Create the main page.
4. Create the components included in the main page.
5. Details of the key component.
6. Create the CSS styles of the application.

2.1. Create a Next.js Starter Template

- Basic version

2.2. Backend API URL in .env

- Key to connect front and backend.

2.3. Create the main page of the application

- Composition: insert one component into another.
- JSX: React looks like HTML.

2.4. Create React/Next.js Components

- Layout component to apply to all pages with composition.
- ToDoList component to show to-do functionality.
- ToDo component to display each to-do task.

2.5. Details of the Key Component

1. Initialization of variables.
2. Load initial data from the database.
3. Functions

Function addTodo(): activates the C in CRUD

- Activates the "/todos" endpoint of the API.
- Sends the new to-do task to the backend server.

Function fetchTodos(): activates the R in CRUD

- Activates the "/todos" endpoint of the API.
- Loads all to-do tasks from the backend.
- Converts that data into JSON format.

Function updateTodo(): activates the U in CRUD

- Activates the "/todos/{id}" endpoint of the API.
- Sends changes in the to-do task to the backend server.
- Function handleToDoChange: updates the edited to-do task in the list of tasks stored by the frontend.

Function handleDeleteTodo(): activates the D in CRUD

- Activates the "/todos/{id}" endpoint of the API.
- Sends the deleted to-do task to the backend server.
- Removes the to-do task from the list of tasks stored by the frontend.

2.6. Create CSS Styles

- CSS modules are used.

Part 3: Full Stack

1. Run the frontend and backend simultaneously locally.
2. Deploy the backend on Render.com.
3. Deploy the frontend on Vercel.

3.1. Run frontend and backend simultaneously

1. Open a terminal window in the backend directory:
    - uvicorn main:app --reload
2. Open another terminal window in the frontend directory:
    - npm run dev
3. Open your browser at http://localhost:3000
4. Test the app locally:
    - Enter new tasks
    - List pending and completed tasks
    - Edit a task from pending to completed
    - Delete a task

3.2. Deploy the backend on Render.com

1. Upload the backend to GitHub.
2. Create a free account on Render.com.
3. Create a Postgres database on Render.com and save the credentials.
4. Create a web service on Render.com by importing the backend from GitHub.
5. Enter the environment variables (database credentials) on Render.com.
6. Create the todos table in the remote database from your terminal.
7. Check the Render.com log to confirm that the backend loads without problems or error messages.
8. Remember where to find answers in case of doubts or problems.

3.3. Deploy the frontend on Vercel

1. Upload the frontend to GitHub.
2. Create a free account on Vercel.
3. Create a new project on Vercel by importing the frontend from GitHub.
4. Enter the environment variables on Vercel (backend URL on Render.com).
5. Check that the CORS configuration is correct so that it does not block communication between frontend and backend.
6. Remember where to find answers in case of doubts or problems.