***API Building Plan***

**Overview**

Our API development team focuses on building a robust system that handles file uploads, processes model inference using AI, and communicates seamlessly between the back-end and front-end. The system employs Python’s FastAPI for model inference using Hugging Face models, C# ASP.NET Core for building the Web API, and React as the front-end framework. The solution is designed to integrate well across these components and allows users to interact with AI models through a simple and effective web interface.

**Key Components of the API Backend**

FastAPI (Python):

* Purpose: FastAPI serves as the model inference engine, loading Hugging Face models and exposing an API for generating responses based on user input. It is the central service responsible for handling the AI processing tasks.
* Functionality: The Python API listens for HTTP requests, processes the user input, runs it through the loaded Hugging Face models, and returns the result to the C# API. The models can be swapped based on different requests or use cases.

C# ASP.NET Core Web API:

* Why ASP.NET Core? ASP.NET Core is used to build the Web API due to its flexibility, performance, and strong integration capabilities. It acts as a mediator between the front-end (React) and the FastAPI service (Python), forwarding the user’s requests and retrieving AI-generated responses.
* Key Operations:
  + HTTP Client Integration: The Web API makes HTTP requests to the FastAPI service for model inference using the HttpClient package.
  + Endpoints: The API exposes endpoints that accept user input, forwards it to the Python service, and returns the processed results back to the front-end.

React Front-End:

* Why React? React was selected for its simplicity in creating dynamic user interfaces. It allows users to easily interact with the API by providing inputs through a web form and displaying the results.
* User Interaction: A simple form allows users to input prompts, which are then forwarded to the C# API for processing. The front-end listens for responses and dynamically updates the UI with the generated content.

**Implementation Steps**

Set Up Python API (FastAPI) for Model Inference

* Install Dependencies: Install libraries like transformers, torch, fastapi, and uvicorn to load Hugging Face models and run the API.
* Run the FastAPI service on localhost, exposing endpoints for model inference that will be used by the C# backend.

Set Up ASP.NET Core Web API

* Create ASP.NET Core Project: Use dotnet new webapi to generate the Web API project that will interact with the Python service.
* Add Dependencies: Ensure HttpClient is added to the project to allow communication with the Python API.
* Configure API: In Startup.cs (or Program.cs), set up the necessary configurations to handle API requests and responses.

Connect the C# API to FastAPI

* Controller Setup: Create a controller in C# that forwards requests to the Python API and returns the response to the front-end.
* Data Flow: User input is forwarded from the front-end to the C# API, which calls the FastAPI service for model inference and returns the generated response.

Set Up Front-End (React)

* Create React Project: Initialize a new React project and create a user-friendly interface to send requests to the C# API.
* Handle API Requests: Use fetch or axios to send user input to the C# API and display the responses on the web page.

Testing and Debugging

* Test the Full Stack: Ensure seamless communication between the React front-end, C# API, and Python FastAPI. Test with different user inputs and model outputs to verify functionality.
* Debugging: Check the logs of both services to troubleshoot issues.