Approach to Develop the Case Prediction API

Step 1: Set Up the Environment

1. **Ensure Python 3.8+ is installed**: The code requires Python version 3.8 or higher.
2. **Create a `.env` file**: Store environment variables securely in a `.env` file. This file will be used to load configuration settings like model paths or API keys.

Step 2: Initialize FastAPI Application

1. **Import Libraries and Modules**: Start by importing the necessary libraries for building the FastAPI application.
2. **Load Environment Variables**: Use `load_dotenv()` to load environment variables from the `.env` file.
3. **Create FastAPI Instance**: Initialize the FastAPI application using `app = FastAPI()`.
4. **Configure CORS Settings**: Set up CORS to allow requests from any origin. This is done to enable cross-origin requests to the API.

Step 3: Model and Tokenizer Configuration

1. **Define Model Path and Initialize HuggingFaceHub Client**: Specify the path where your model is stored and initialize the HuggingFaceHub client with appropriate parameters.
2. **Load Tokenizer and Model**: Load the tokenizer and model using the specified paths. This step prepares the model and tokenizer for making predictions.

Step 4: Define Request and Response Models

1. **Create Pydantic Models**: Define Pydantic models for the request body and response. These models ensure the data structure is validated and correctly formatted for both incoming requests and outgoing responses.

Step 5: Create the Prediction Endpoint

1. **Define the POST Endpoint**: Create an endpoint to handle POST requests for case predictions. This endpoint will receive the case details, make predictions, and return the result.
2. **Tokenize Input Query**: Convert the input query into a format suitable for the model using the tokenizer.
3. **Make Predictions Using the Model**: Use the pre-trained model to get predictions for the input query.
4. **Determine the Label Based on Prediction**: Map the prediction result to a human-readable label, such as "accepted" or "rejected".

5. **Generate Reasoning for the Prediction**: Use the language model to generate reasoning based on the prediction. This provides an explanation for the predicted outcome.
6. **Prepare the Response Content**: Structure the prediction and reasoning into the response format.
7. **Return the Response as JSON**: Encode the response content and return it as a JSON response to the client.

How and Why We Are Creating the Case Prediction Pipeline

Why We Are Creating the Case Prediction API

1. **Objective**: The primary goal is to build a system that can predict the outcome of legal cases (whether they will be accepted or rejected) based on the details provided. Additionally, it provides a reasoning for the prediction.
2. **Use Case**: This can be useful for legal professionals, researchers, or anyone interested in getting an initial assessment of a legal case. It can save time and provide valuable insights before proceeding with detailed legal analysis.

How We Are Creating the Case Prediction API

The creation of the Case Prediction API involves several steps, which together form a pipeline. Here's a step-by-step explanation of how we achieve this:

Step 1: Setting Up the Environment

1. **Install Necessary Libraries**:
    - We need several Python libraries to build our API. These libraries include:
        - `fastapi`: To create the web API.
        - `uvicorn`: To run the FastAPI application.
        - `transformers` and `torch`: To handle the machine learning models.
        - `langchain`: To interact with language models.
        - `python-dotenv`: To manage environment variables.
        - `pydantic`: For data validation.
2. **Create a `.env` File**:
    - This file securely stores configuration details such as paths to model files and API keys.

Step 2: Initializing the FastAPI Application

1. **Import Libraries**:

    - We import all the necessary libraries at the beginning of our code to use their functionalities.
2. **Load Environment Variables**:
    - Use `load_dotenv()` to load settings from the `.env` file. This helps in keeping sensitive information secure and manageable.
3. **Create the FastAPI Application**:
    - Initialize a FastAPI app using `app = FastAPI()`.
4. **Set Up CORS**:
    - Configure CORS settings to allow requests from any origin. This is necessary to ensure our API can be accessed from different domains.

Step 3: Configuring the Model and Tokenizer

1. **Define Model Path and Initialize HuggingFaceHub Client**:
    - Specify the path where our machine learning model is stored.
    - Initialize a client for the language model using `HuggingFaceHub`.
2. **Load Tokenizer and Model**:
    - Load the tokenizer and model. The tokenizer converts text into a format the model can understand, and the model is used to make predictions.

Step 4: Defining the Data Models

1. **Request and Response Models**:

Define data models using Pydantic. This ensures the incoming request data and the outgoing response data are in the correct format. For instance:
python
Copy code

```
class CaseQuery(BaseModel):
    query: str

class PredictionResult(BaseModel):
    prediction: int
    reasoning: str
```

    -

Step 5: Creating the Prediction Endpoint

1. **Create the POST Endpoint**:
    - Define an endpoint to handle POST requests. This is where users will send their case details for prediction.
2. **Tokenize the Input**:
    - Convert the input text (case details) into a format the model can understand.

3. **Make Predictions**:
    - Use the pre-trained model to predict whether the case will be accepted or rejected.
4. **Determine the Label**:
    - Map the prediction result to a human-readable label, such as "accepted" or "rejected".
5. **Generate Reasoning**:
    - Use a language model to generate a reasoning based on the prediction. This provides context and justification for the prediction.
6. **Prepare and Return the Response**:
    - Structure the prediction and reasoning into the response format and send it back to the user.

Step 6: Handling Errors

1. **Error Handling**:
    - Ensure any errors during request processing are caught and handled gracefully. If an error occurs, send back a detailed error message.

Backend + Frontend

Documentation for Legal Case Prediction Service

Overview

This documentation describes the approach and components used to build a Legal Case Prediction Service. This service predicts the outcome of a legal case based on the provided input and generates reasoning for the prediction using a language model.

Backend Explanation

FastAPI Setup

1. **FastAPI Framework**: The backend is built using FastAPI, a modern, fast (high-performance) web framework for building APIs with Python 3.7+ based on standard Python type hints.
2. **CORS Middleware**: Configured to allow cross-origin requests from any origin. This is done to ensure the frontend, which may be hosted on a different domain, can interact with the backend without any issues.

Loading Models

1. **Language Model Client**: The service uses HuggingFaceHub to load a language model (`Nous-Hermes-2-Mixtral-8x7B-DPO`) for generating reasoning texts. Parameters such as temperature and max_length are set to control the output.
2. **Tokenizer and Sequence Classification Model**: The backend loads a pre-trained tokenizer and sequence classification model from a specified directory. These are used to process the input queries and predict the outcomes.

API Endpoint

1. **Case Query Input**: The input to the service is a legal query provided by the user. This is structured using a Pydantic BaseModel.
2. **Prediction and Reasoning**: The query is tokenized and passed through the sequence classification model to predict the outcome (accepted or rejected). Based on the prediction, a prompt is generated and sent to the language model to create a reasoning text.
3. **Response**: The prediction and reasoning are packaged into a response model and returned to the frontend.

Frontend Explanation

API Interaction

1. **API Call**: The frontend interacts with the backend using a function (`casePrediction`) that sends a POST request to the backend endpoint. This function handles authorization using a token, error handling, and parsing the response.

User Interface

1. **Components**: The frontend is built with Svelte and includes components such as a Navbar, MessageInput for user queries, and Message for displaying interactions.
2. **File Upload and Text Extraction**: Users can enter a query directly or upload a PDF file. If a PDF is uploaded, the text is extracted from the PDF using the `PDFDocument` library, which processes each page and extracts text content.
3. **Progress Indication**: The frontend provides visual feedback during various stages of the process. A loading indicator displays the progress percentage, and an option to cancel the prediction request is available.
4. **Displaying Results**: The query, prediction, and reasoning are displayed in a chat-like interface. Each interaction is formatted and appended to the message list.

User Experience Enhancements

1. **Loading Progress**: The progress bar updates incrementally to give users real-time feedback on the status of their request.

2. **Cancellation**: Users can cancel a prediction request, and appropriate messages are shown to indicate the cancellation.

Connection Between Backend and Frontend

1. **HTTP Requests**: The frontend sends HTTP POST requests to the backend with the user query or extracted text from a PDF.
2. **Token Authorization**: The frontend includes an authorization token in the request headers to ensure secure communication with the backend.
3. **Response Handling**: The backend processes the request, performs the prediction and reasoning, and sends back a structured response, which the frontend then processes and displays to the user.

Usage Guide

1. **Setting Up the Backend**:
    - Install FastAPI and necessary dependencies.
    - Load the pre-trained models and configure the FastAPI app.
    - Implement the prediction and reasoning logic in the `/caseprediction` endpoint.
    - Handle errors and ensure proper response formatting.
2. **Setting Up the Frontend**:
    - Build the user interface with Svelte, including components for input, message display, and progress indication.
    - Implement the `casePrediction` function to interact with the backend.
    - Handle user inputs, file uploads, and display the results in an intuitive manner.
3. **Running the Service**:
    - Start the FastAPI backend server.
    - Deploy the Svelte frontend application.
    - Use the frontend interface to input queries or upload PDF files for prediction.
    - View the prediction results and reasoning generated by the backend service.