Project setup :

\* git clone <github rul>

\* First open the terminal and go inside the modelAPI directory and run the python api.py

\* Then open another directory and in HumanDiseasePredictor directory type the command npm run dev

Command : npm run dev

Your site is ready .

Introduction :

The **Human Disease Predictor** is an AI-powered healthcare solution designed to assist medical professionals and patients in predicting diseases based on symptoms. By integrating **Machine Learning (ML), FastAPI, MongoDB, and IPFS (InterPlanetary File System)**, this system ensures **fast, secure, and decentralized** data management.

### ****Key Features****

✅ **AI-Based Diagnosis** – Uses ML models to predict diseases based on input symptoms.  
✅ **Secure & Decentralized Storage** – Stores patient records securely using **IPFS via Pinata API**.  
✅ **Fast and Scalable** – Built on **FastAPI**, ensuring **high-speed processing** and **low-latency responses**.  
✅ **Privacy-Preserving** – Only **Content Identifiers (CID)** are stored in **MongoDB**, ensuring patient data remains decentralized and secure.

Project details :

The **Human Disease Predictor**  is a **FastAPI-based** web service that predicts diseases based on symptoms using **machine learning models**. It also securely **stores and retrieves patient records** using **MongoDB** and **IPFS (InterPlanetary File System) via Pinata API**.

## ****(Step-by-Step Workflow)****

This will help to understand how a patient or doctor interacts with the system.

### ****Scenario 1: Disease Prediction****

The user (patient or doctor) enters symptoms in the **web application**.

The application sends the symptoms to the **FastAPI backend**.

The **ML model** predicts the most probable disease.

The result is displayed to the user.

### ****Scenario 2: Storing Patient Data****

The system generates a **patient record** containing the predicted disease and other details.

The data is **uploaded to IPFS via Pinata API**.

IPFS returns a **CID (Content Identifier)** for the stored data.

The system stores **(Patient ID, CID)** mapping in **MongoDB**.

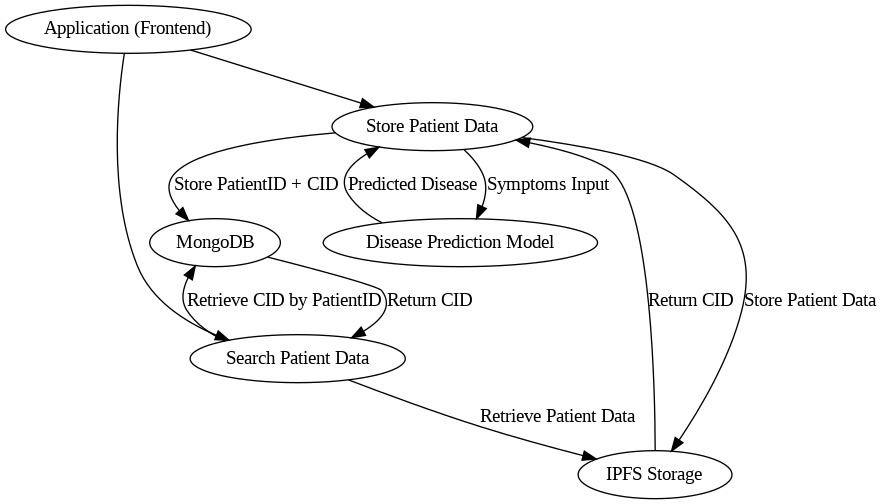
### ****Scenario 3: Retrieving Patient Data****

The user searches for a patient record by **Patient ID**.

The system retrieves the corresponding **CID from MongoDB**.

Using the CID, the data is **fetched from IPFS** and displayed to the user.

Architecture :



Your architecture involves a **Human Disease Predictor** system that integrates multiple technologies to store and retrieve patient data securely while leveraging machine learning for disease prediction. Here are the key concepts you can mention in your document:

### ****Workflow Explanation****

1️⃣ **User Input**: The patient or doctor enters symptoms through the **frontend application**.  
2️⃣ **ML Model Prediction**: The **FastAPI backend** processes the symptoms and predicts the possible disease using an ML model.  
3️⃣ **Data Storage**:

The patient’s details and disease predictions are uploaded to **IPFS via Pinata API**.

**IPFS returns a unique Content Identifier (CID)**.

**MongoDB stores (Patient ID, CID) mapping**, ensuring efficient retrieval. 4️⃣ **Data Retrieval**:

When a **doctor searches for a patient**, MongoDB retrieves the **CID based on Patient ID**.

The **backend fetches data from IPFS** and displays the record in the frontend.

### ****Architecture Overview****

The system consists of a frontend application, a disease prediction model, and a hybrid data storage mechanism using **MongoDB** and **IPFS (InterPlanetary File System)**. The workflow involves patient data input, disease prediction, and secure data storage with decentralized access.

### ****Key Concepts & Components****

#### ****1. Application (Frontend)****

Acts as the user interface where patients or medical professionals enter symptoms and retrieve disease predictions.

Sends data for disease prediction and storage.

#### ****2. Disease Prediction Model (Machine Learning)****

Receives symptoms input from the frontend.

Processes the data to predict the disease using an AI/ML model.

Returns the predicted disease to the application.

#### ****3. Storage Mechanism****

The system utilizes a **hybrid storage approach** for security and efficiency:

**a) MongoDB (Relational Storage)**

Stores metadata such as **Patient ID** and **CID (Content Identifier)**.

Enables searching patient data using the Patient ID.

Facilitates CID retrieval for accessing patient records.

**b) IPFS Storage (Decentralized Storage)**

Stores patient medical records securely in a decentralized manner.

Returns a **CID**, which acts as a unique reference to the stored data.

Ensures tamper-proof and immutable records for enhanced security.

#### ****4. Workflow****

**Store Patient Data:**

The frontend sends patient data and symptoms to the backend.

The Disease Prediction Model processes the symptoms and predicts the disease.

The processed data is stored in **IPFS**, returning a unique CID.

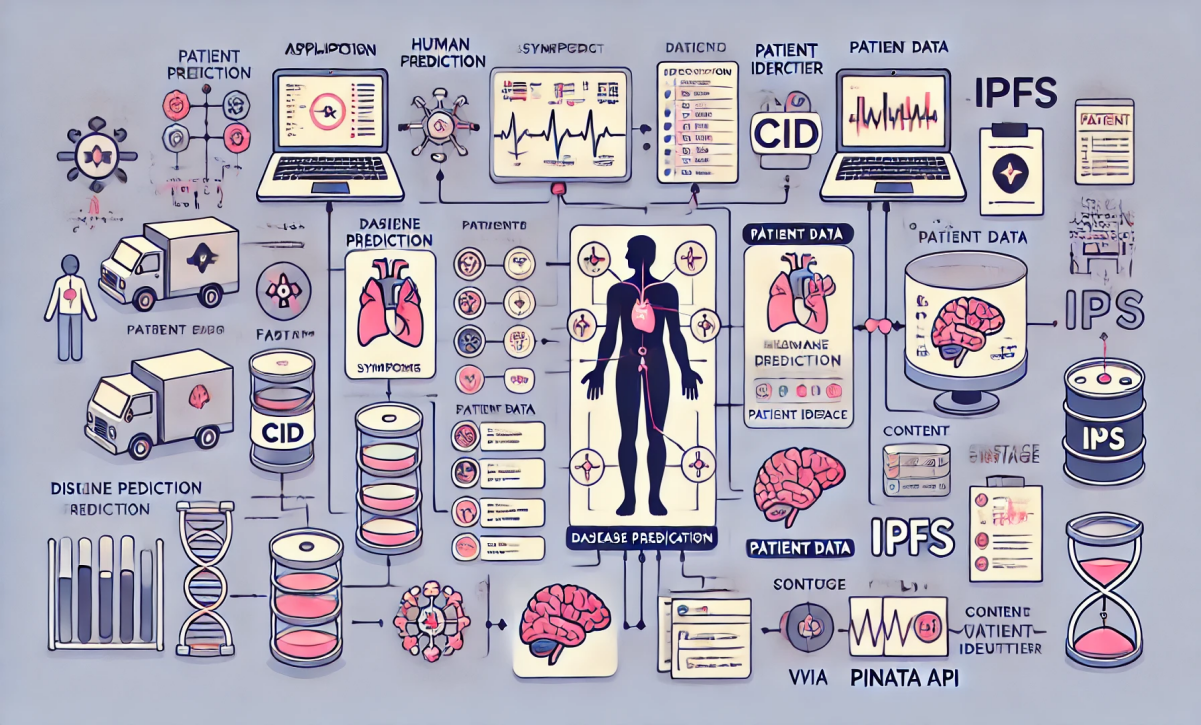
The CID is stored in **MongoDB** alongside the Patient ID for quick retrieval.

**Search & Retrieve Patient Data:**

The frontend allows searching by **Patient ID**.

The backend fetches the corresponding **CID** from **MongoDB**.

Using the CID, the system retrieves the actual patient data from **IPFS**.



## ****Role of Pinata API****

### ****Why Use Pinata API for IPFS?****

**Pinata API** is a service that simplifies the use of **IPFS** by providing an easy way to upload, store, and retrieve files without requiring users to run their own IPFS nodes.

### ****How Pinata Works in This Project:****

When a **patient record** is created, the data is uploaded to IPFS via **Pinata API**.

Pinata returns a **CID (Content Identifier)**, which acts as a permanent reference to the stored data.

The system then stores this **CID in MongoDB**, allowing for quick patient data retrieval.

When needed, the **CID is used to fetch data** from IPFS, ensuring decentralized, tamper-proof storage.

### ****Benefits of Using Pinata API****

✅ No need to maintain an IPFS node  
✅ Secure, immutable, and decentralized storage  
✅ Easy integration with MongoDB and backend APIs

# ****Key Concepts & Technologies Used****

## ****1.) FastAPI - Web Framework****

* FastAPI is a **Python-based web framework** that provides:
  + High performance
  + Automatic **data validation**
  + Interactive API documentation (**Swagger UI**)
* It is used to create **RESTful APIs** for handling **patient data, symptom input, and disease prediction**.

### ****Why FastAPI?****

✅ Faster than Flask  
✅ Supports asynchronous programming for better performance  
✅ Built-in validation using **Pydantic**

## ****2.) Machine Learning Models for Disease Prediction****

This system utilizes **three pre-trained machine learning models** to predict diseases based on symptoms:

### ****A. Decision Tree Model****

* A tree-like structure that makes decisions based on conditions.
* Works well for structured data with **if-else logic**.
* Used in **medical diagnostics** where decisions follow a hierarchical approach.

### ****B. Naïve Bayes Model****

* Based on **Bayes' Theorem**, it assumes that symptoms are **independent**.
* **Fast and effective** for disease classification tasks.

### ****C. Random Forest Model****

* An **ensemble learning technique** that builds multiple decision trees and **takes a majority vote**.
* More accurate than a single decision tree.
* Less likely to **overfit** the data.

### ****How Predictions Work?****

1. The API **receives a list of symptoms** from the user.
2. The symptoms are converted into a numerical format that the models understand.
3. Each of the three models makes a **disease prediction**.
4. The results are stored and returned to the user.

## ****3.) MongoDB - Patient Database****

* MongoDB is a **NoSQL database** used to store **patient data and IPFS CIDs** (Content Identifiers).
* The system maintains a **mapping between patient phone numbers and their IPFS-stored medical records**.

### ****Data Model (MongoDB)****

class PatientToCidMap(Document):

phoneNo = StringField(required=True) # Patient's phone number (unique identifier)

cid = StringField(required=True) # IPFS Content Identifier (CID) for patient record

✅ **Why MongoDB?**

* Schema flexibility (NoSQL structure).
* Handles large amounts of unstructured patient data.
* Fast **read/write operations**.

## ****4️⃣ IPFS (InterPlanetary File System) via Pinata API****

* Instead of storing patient records **directly in MongoDB**, we use **IPFS** for **decentralized and secure storage**.
* **Pinata API** is used to interact with IPFS, allowing us to **upload and retrieve patient records**.

### ****How IPFS Works in This Project?****

**When adding a patient:**

* + The patient's data (symptoms, predicted disease, etc.) is **stored in IPFS**.
  + An **IPFS CID (Content Identifier)** is returned and stored in MongoDB.

**When retrieving a patient:**

* + The system fetches the CID from MongoDB.
  + It then retrieves the **actual patient data from IPFS** using the CID.

### ****Why Use IPFS?****

✅ **Decentralized storage** (No single point of failure).  
✅ **Tamper-proof** and immutable records.  
✅ **Efficient data retrieval** with **content-based addressing**.

## ****5️⃣ CORS (Cross-Origin Resource Sharing)****

* This API includes **CORS Middleware**, allowing it to be accessed by **frontend applications from different domains**.
* **Example:** A React or Angular frontend can call this API without CORS restrictions.

app.add\_middleware(

CORSMiddleware,

allow\_origins=["\*"], # Allows all origins

allow\_credentials=True,

allow\_methods=["\*"], # Allows all HTTP methods

allow\_headers=["\*"], # Allows all headers

)

✅ **Why is CORS important?**

* Without CORS, **web applications hosted on different domains** cannot call the API.
* This setting ensures that the API can be used **by any frontend application**.

| **Component** | **Purpose** |
| --- | --- |

|  |  |
| --- | --- |
| **FastAPI** | Backend framework for handling API requests. |

|  |  |
| --- | --- |
| **Machine Learning Models** | Predict diseases based on symptoms. |

|  |  |
| --- | --- |
| **MongoDB** | Stores mappings of patient phone numbers and IPFS CIDs. |

|  |  |
| --- | --- |
| **IPFS (Pinata)** | Stores patient records securely in a decentralized manner. |

|  |  |
| --- | --- |
| **CORS Middleware** | Enables cross-origin API access for frontend applications. |

## ****Security & Privacy Considerations****

Since patient data is highly sensitive, the system employs several security measures:

### ****1️⃣ Data Storage & Privacy****

✅ **MongoDB** stores only metadata (Patient ID + CID), avoiding direct exposure of sensitive data.  
✅ **IPFS** ensures **data immutability**—once stored, it cannot be altered or tampered with.  
✅ **Pinata API** provides secure IPFS interactions, preventing unauthorized access.

### ****2️⃣ API Security****

✅ **CORS Middleware** ensures controlled access from frontend applications.  
✅ **FastAPI validation** ensures only valid and expected data is processed.  
✅ **JWT Authentication (optional)** can be implemented for role-based access.

### ****3️⃣ Decentralization Benefits****

✅ **No single point of failure**—even if MongoDB is compromised, patient records remain secure on IPFS.  
✅ **Immutable storage** ensures patient records cannot be altered or deleted maliciously.

Summary :

The **Human Disease Predictor** is an AI-powered healthcare system that predicts diseases based on symptoms entered by users. It securely stores patient records using **MongoDB and IPFS (InterPlanetary File System)** to ensure privacy, security, and fast retrieval. The system integrates **machine learning models** for accurate disease prediction and leverages decentralized storage to protect patient data from unauthorized access or data loss.

With this system:

Patients can get disease predictions based on symptoms.

Doctors can store and retrieve patient data efficiently.

The use of **IPFS + MongoDB** ensures secure, decentralized, and immutable data storage.