Full Stack Development with MERN

Project Documentation format

1. Introduction

- **Project Title: Revolutionizing Liver Care: Predicting Liver Cirrhosis.**
- **Team Members: 1. Team Leader-Sara Gogulamudi**
  2. Team Member-Ryali Sai Krishna Harshith
  3. Team Member- S Dhanalakshmi

2. Project Overview

Purpose: The purpose of this project is to develop an intelligent and accurate liver cirrhosis prediction system using machine learning techniques. It aims to assist healthcare professionals in identifying liver cirrhosis at an early stage using routine clinical data, enabling faster diagnosis, early treatment, and better patient outcomes especially in settings where specialist resources are limited.

- **Goals:**
- Build a predictive machine learning model to assess the risk of liver cirrhosis based on clinical and biochemical features.
- Improve early detection of liver cirrhosis to prevent disease progression and reduce mortality.
- Develop a user-friendly interface for doctors and healthcare workers to input patient data and receive instant predictions.
- Ensure accuracy, scalability, and interpretability of the prediction system for clinical use.

Features: The system provides real-time liver cirrhosis risk prediction using advanced machine learning models. By analyzing patient data such as bilirubin levels, liver enzymes, and albumin, the system can predict whether a patient is at high or low risk for liver cirrhosis, enabling timely medical intervention.

- A user-friendly web-based interface allows healthcare professionals to easily input patient data and receive instant prediction results. The interface is designed with simplicity and clarity in mind, ensuring it can be used efficiently by both technical and non-technical users.
- To ensure transparency and trust, the system includes model interpretability features using techniques like SHAP (SHapley Additive exPlanations) or LIME (Local Interpretable Model-Agnostic Explanations). These tools highlight which input features contributed most to each prediction, supporting informed clinical decisions.
- Data security and privacy are top priorities. The platform ensures that all patient data is encrypted, securely stored, and only accessible to authorized users. It follows standard healthcare data protection regulations such as HIPAA and GDPR to maintain confidentiality.
- The platform supports both real-time and batch processing. This means doctors can evaluate individual patients instantly or upload large datasets (e.g., in CSV format) for mass screening and hospital-wide risk analysis.
- It features automatic performance monitoring, allowing administrators to track key metrics such as model accuracy, precision, and recall over time. Alerts can be configured to notify if prediction accuracy drops or if data anomalies are detected.
- Designed for scalability and cloud deployment, the system can handle increasing numbers of users and data volume. It can be deployed across multiple hospitals or clinics with ease, maintaining performance and accessibility regardless of location.
- The system also includes automated report generation, producing downloadable

and printable patient reports that summarize prediction results, contributing factors, and suggested next steps for clinical review or patient communication.

- Finally, the model is built to support continuous improvement through retraining. As new anonymized patient data becomes available, the model can be updated to maintain and enhance its predictive performance over time.

3. Architecture

Frontend: The frontend architecture of the system is built using React, a modern JavaScript library for building dynamic user interfaces. It follows a component-based structure, where each part of the application—such as the input form, prediction result display, and data visualization—is developed as an independent, reusable component. React's virtual DOM ensures efficient rendering and fast updates, while state management (using tools like React Context or Redux) handles data flow between components. The interface is styled using CSS modules or Tailwind CSS, and it communicates with the backend via RESTful APIs to send patient data and receive prediction results. This architecture ensures a clean, modular, and scalable frontend that offers a smooth user experience for doctors and healthcare professionals.

- **Backend:** The backend of the system is developed using Node.js and Express.js, providing a lightweight and scalable server-side framework. At its core, the architecture follows a RESTful API structure, where different endpoints handle operations such as patient data submission, prediction requests, user authentication, and report generation. Express.js manages routing, middleware integration, and request handling efficiently. The backend interacts with the machine learning model, either by calling a locally deployed model (via Python shell or child process) or accessing it through a separate microservice. Data is stored and retrieved from a secure database (e.g., MongoDB or PostgreSQL), with all inputs validated using middleware to ensure data integrity. For security, the system implements JWT-based authentication, role-based access control, and HTTPS. The architecture is modular, with separate folders for routes, controllers, services, and utilities, making it maintainable and extendable for future features like logging, analytics, or multi-user support.
- **Database:** In this project, MongoDB is used as the primary database due to its flexibility in handling semi-structured medical data. It stores records in collections as JSON-like documents, making it ideal for storing patient information and prediction results without the rigid structure of traditional relational databases.
- The **patients** collection stores individual patient data, including details such as name, age, gender, and important clinical features like bilirubin, liver enzyme levels, and albumin values. This data is essential for the prediction process and serves as the input to the machine learning model.
- The predictions collection holds the output of the machine learning model. Each prediction entry is linked to a patient via their ID and includes a risk score, classification (e.g., High or Low risk), model version, and optionally a feature explanation (e.g., SHAP values) to show what influenced the prediction.
- The **users** collection manages user accounts for doctors, administrators, and healthcare staff. It stores login credentials, user roles, and access permissions, enabling secure access control within the system.
- To ensure security and efficiency, the database implements input validation, role-based access control, and indexed fields for fast queries. Passwords are hashed using bcrypt, and sensitive data is protected according to HIPAA and GDPR standards.

4. Setup Instructions

- **Prerequisites:** **Backend (Node.js & Express.js)**

- **Node.js** – To run backend JavaScript code
- **Express.js** – To create APIs and handle HTTP requests
- **Mongoose** – To connect and manage MongoDB database

Machine Learning (Python)

- **Python 3.x** – For developing the ML model
- **Scikit-learn** – To build and train the prediction model
- **Pandas** – To clean and process medical data
- **NumPy** – For handling numeric operations

Frontend (React.js)

- **React.js** – To build the user interface
- **Axios** – To send patient data to backend and get results
- **React Router** – For navigation between pages (e.g., home, results)
- **Tailwind CSS** – For styling the web pages

Database

- **MongoDB** – To store patient details, prediction results, and users
- **MongoDB Atlas** *(optional)* – For cloud-based MongoDB hosting

Tools for Development

- **Postman** – To test APIs during development
- **Installation:** Step-by-step guide to clone, install dependencies, and set up the environment variables.

5. Running the Application

- Provide commands to start the frontend and backend servers locally.
  - **Frontend:** `npm start` in the client directory.
  - **Backend:** `npm start` in the server directory.

6. API Documentation

- Document all endpoints exposed by the backend.
- Include request methods, parameters, and example responses.

7. Authentication

- Explain how authentication and authorization are handled in the project.
- Include details about tokens, sessions, or any other methods used.

8. Testing

- Ensure **accuracy of predictions**, **security of patient data**, **smooth UI/UX**, and **reliable integration** between components.

9. Known Issues

Small Dataset: The project currently uses a limited dataset, which may affect the model's accuracy and performance. A small amount of data can cause the model to make less reliable predictions**.**

Imbalanced Data: In many liver disease datasets, there are more healthy cases than cirrhosis cases. This imbalance can cause the model to predict more negative results, even when the patient is actually at risk.

Hard to Understand Predictions: Although tools like SHAP or LIME are used to explain the model's output, non-technical users like doctors might still find it hard to interpret why the model made a certain prediction

Not Yet Tested in Hospitals: The system has not been tested in real hospitals. Without real-world clinical validation, it may not be fully trusted or used in actual patient care yet.

Browser Display Issues: Some small UI issues may appear when the app is opened in older browsers or on mobile devices with smaller screens.

Security in Development Mode: If best practices are not followed during development, environment variables or login details may not be properly secured.

Free Hosting Limits: Using free hosting services (like MongoDB Atlas or basic cloud plans) might lead to slower performance or limits on the number of users the system can handle**.**

Internet Required: If the system is deployed online, it needs a stable internet. This could be a problem in rural or low-network areas.

10. Future Enhancements:

In the future, this system can be significantly improved by integrating a larger and more diverse dataset sourced from multiple hospitals and regions to enhance the model's accuracy and reliability across various patient groups. One major enhancement would be the integration with clinical systems such as hospital management software (HMS) and electronic health records (EHR), enabling seamless and real-time access to patient data for live prediction support.

A mobile application version of the platform can be developed to allow healthcare providers to make predictions on the go, which would be especially beneficial in rural and low-resource areas. The system could also be expanded to support multi-disease prediction, allowing it to diagnose related liver conditions like hepatitis or fatty liver using similar input features**.**

To improve user experience and reduce manual data entry, features like voice input and OCR (Optical Character Recognition) could be introduced, allowing doctors to speak or scan test reports to input data. Adding a feedback loop where healthcare professionals confirm or correct predictions would help the model learn and improve over time through active learning techniques.

The interface can be enhanced by providing role-based dashboards with specific access and functionalities for doctors, lab technicians, and administrators. Additionally, an advanced explainable AI module can be added to present predictions in a more visual and easy-to-understand way for non-technical users. Lastly, adding multilingual support will make the platform more inclusive and user-friendly for medical staff and patients in non-English-speaking regions.