Full Stack Development with MERN - Project Documentation

1. Introduction

• Project Title: Liver Disease Prediction System
• Team Members:
- Moraboina Anusha: Full Stack Developer and ML Engineer

2. Project Overview

• Purpose:
The Liver Disease Prediction System is a MERN-based application aimed at predicting liver disease using machine learning models and providing real-time predictions to users via a clean and interactive interface. The goal is to help users understand their health status and potentially reduce diagnostic delays.

• Features:
- User registration and login
- Form to input health metrics
- Liver disease prediction result
- Dashboard to view previous results
- Admin interface (optional)
- REST API integration
- MongoDB-based user and result storage

3. Architecture

• Frontend:
Built using React.js. The UI includes routes for login, signup, prediction form, and result dashboard. Components are organized for reusability with state managed via React hooks or Redux.

• Backend:
Implemented using Node.js and Express.js. Handles user management, authentication, and prediction result storage. Integrates with the machine learning model through a Python service or child process.

• Database:
MongoDB is used for storing user data and prediction records. Mongoose schemas define User and Prediction models.

4. Setup Instructions

• Prerequisites:
- Node.js (v14 or above)
- MongoDB
- Python (for ML model service)

• Installation:
1. Clone the repository
2. Navigate to the root folder
3. Run `npm install` in both /client and /server folders
4. Set environment variables (MongoDB URI, JWT secret)
5. Run Python Flask server for ML model
6. Start client and server

5. Folder Structure

• Client:
- /components: Reusable components
- /pages: Route-based pages
- /services: API handlers
- /assets: Styles/images

• Server:
- /routes: Express routes
- /controllers: Business logic
- /models: Mongoose schemas
- /middleware: Auth functions

6. Running the Application

Frontend:
```
npm start (inside /client)
```

Backend:
```
npm start (inside /server)
```

ML Flask API (Optional):
```
python app.py (inside /ml_model)
```

7. API Documentation

- POST /api/auth/register: Register user
- POST /api/auth/login: Login user
- GET /api/predictions: Get user history
- POST /api/predict: Submit input data and return prediction result

8. Authentication

Authentication is implemented using JWT tokens. Upon login, the backend issues a token stored in localStorage or HTTP-only cookies. Routes are protected using middleware to verify JWTs.

9. User Interface

Includes login/register forms, prediction input form with validations, and a dashboard for previous results.
Screenshots provided in accompanying files.

10. Testing

Frontend tested using Jest and React Testing Library.
Backend API tested using Postman and automated tests with Mocha/Chai.

11. Screenshots or Demo

Screenshots are included in the /screenshots folder.
Demo Link: https://your-demo-link.com (if deployed)

12. Known Issues

- Model sometimes produces false positives
- Mobile responsiveness needs improvement
- No role-based access implemented yet

13. Future Enhancements

- Add email notifications
- Expand health metrics for better prediction
- Add role-based admin panel
- Cloud deployment for Flask ML API