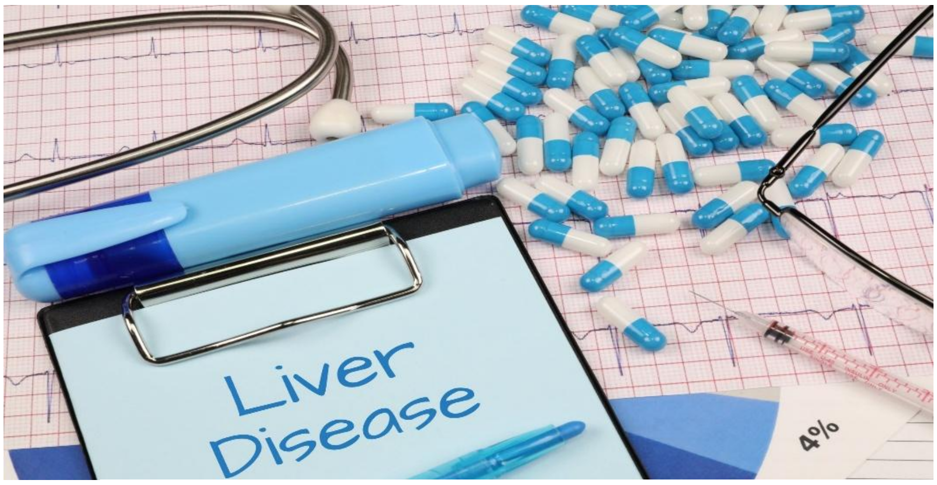
Project Title:

Revolutionizing Liver Care: Predicting Liver Cirrhosis Using Advanced Machine



TEAM MEMBERS:

Team Leader: Paritala Sri Venkata Gopinadh

Team member: Majeti Sudheer

Team member: Thoka Naveen Babu

Team member: Vempati Lakshmi Priya

Introduction:

Liver cirrhosis is a chronic and potentially life-threatening condition characterized by progressive scarring of the liver tissue. Early detection and intervention can significantly improve patient outcomes. However, traditional diagnostic methods often identify the disease at advanced stages. This project aims to harness the power of advanced machine learning (ML) techniques to accurately predict liver cirrhosis in its early stages, enabling timely and personalized treatment strategies.

Project Overview:

Purpose:

The primary purpose of this project is to develop an intelligent, non-invasive, and efficient diagnostic tool that leverages advanced machine learning algorithms to predict liver cirrhosis at an early stage. The goal is to assist medical professionals in identifying high-risk patients using clinical and biochemical data, thus enabling timely treatment, reducing healthcare costs, and improving patient outcomes.

Features:

Early Prediction System: Accurately predicts the likelihood of liver cirrhosis using patient health records and lab results.

User-Friendly Interface: Provides an intuitive dashboard for doctors and healthcare providers to input data and receive predictions.

Multi-Model Analysis: Utilizes various ML algorithms (Random Forest, SVM, XGBoost, etc.) to ensure robust prediction performance.

Visualization Tools: Offers graphical representation of patient risk profiles, model performance metrics, and feature importance.

Data Preprocessing Module: Automatically handles missing data, normalization, and feature selection for cleaner model input.

Model Evaluation Panel: Displays accuracy, precision, recall, F1-score, and ROC-AUC to help assess model effectiveness.

Real-time Prediction: Enables fast and efficient diagnosis support during clinical decision-making.

Architecture:

Frontend: React.js

- The frontend is developed using React.js, offering a responsive and modular user interface.
- **Component-based Structure**:
    - **InputForm**: Allows healthcare professionals to enter patient details and test results.
    - **PredictionResult**: Displays cirrhosis risk prediction and confidence score.
    - **VisualizationPanel**: Shows data charts (e.g., feature importance, trend graphs).

    - **Navigation & Routing**: Implemented using React Router for smooth page transitions.
- **State Management**: Utilizes **React Context API** or **Redux** (optional) for managing shared data (e.g., patient input).
- **API Integration**: Fetches predictions and results from the backend via **Axios** or **Fetch API**.

Backend: Node.js with Express.js:

The backend is powered by Node.js with Express.js to handle API requests and responses.

RESTful API Design:

- POST /predict: Accepts patient data and returns cirrhosis prediction from the ML model.
- GET /patients: Retrieves stored prediction records.
- POST /patients: Stores new prediction results in the database.

ML Model Integration:

- The machine learning model is served either using a Python Flask microservice (via HTTP) or pre-exported with ONNX/PMML and invoked within Node using a wrapper.

Security & Validation:

- Input validation using Joi or Express-validator.
- CORS policy and authentication can be added using JWT for secure access.

Setup Instructions:

Prerequisites:

Ensure the following software dependencies are installed before setting up the project:

Tool	**Version**	**Description**
Node.js	>= 16.x	JavaScript runtime environment for the backend
npm	>= 8.x	Node package manager (comes with Node.js)
MongoDB	>= 6.x	NoSQL database to store patient records
Git	Latest	For cloning the repository
Python	>= 3.8 (optional)	If using Python-based ML model integration

Installation Guide:

1. Clone the Repository
2. Setup Backend (Node.js + Express)
3. Set Up Environment Variables
4. Start MongoDB Server
5. Run the Backend Server
6. Setup Frontend (React.js)
7. Run Python ML Microservice

Folder Structure:

Client (React Frontend):

The frontend is organized using a modular, component-based structure, following standard React practices to ensure reusability and maintainability:

- **/public**: Contains the base HTML file (index.html) used as the template for rendering the React app in the browser.
- **/src:** Main source folder where all application logic and UI code reside**.**
    - **components/:** Houses reusable UI elements such as form fields, prediction result displays, and chart components.
    - **pages/:** Contains full-page views like Home, Dashboard, and error pages, built using the components.
    - **services/:** Includes API interaction logic (e.g., Axios calls to the backend).
    - **App.js:** Root component that defines the overall layout and routes.
    - **index.js**: Entry point that renders the React app into the DOM.
    - **App.css or styles/:** For global or modular styling of the interface.

Server (Node.js + Express Backend):

The backend is structured using a layered MVC (Model-View-Controller) pattern to enhance maintainability and logical separation:

- **config/**: Contains configuration files such as MongoDB connection settings using Mongoose.
- **models/**: Defines the data schema for storing patient details and prediction results in MongoDB.
- **controllers/**: Implements the logic for processing API requests and communicating with the ML model or database.
- **routes/**: Defines the application’s RESTful API endpoints (e.g., /predict, /patients).
- **services/**: Handles interaction with the external machine learning model (e.g., sending data to a Python microservice and receiving predictions).
- **middleware/**: Custom Express middleware for error handling, logging, or authentication (if implemented).
- **server.js / app.js**: Main server entry files where the Express app is initialized, middleware is loaded, and routes are registered.

Running the Application:

To run the full-stack application locally, make sure you have completed the setup and installation steps. Follow the commands below to start both the frontend and backend servers.

Start the Backend Server:

Navigate to the backend directory and run the server using:

```
cd backend

npm start
```

- This will start the Node.js + Express.js server on http://localhost:5000
- Ensure MongoDB is running and properly connected.

Start the Frontend Server:

Open a new terminal window, navigate to the frontend directory, and run:

```
cd frontend

npm start
```

- This will launch the React development server at http://localhost:3000.
- The frontend communicates with the backend via REST APIs for prediction and data storage.

API Documentation:

Base URL: http://localhost:5000/

1. Predict Liver Cirrhosis

Endpoint: /api/predict
Method: POST
Description: Submits patient clinical data to the machine learning model and returns a prediction.

2. Save Prediction Record

Endpoint: /api/patients
Method: POST
Description: Stores the patient's data and prediction result into the MongoDB database.

3. Get All Patient Records

Endpoint: /api/patients
Method: GET
Description: Retrieves a list of all previously stored patient prediction records.

Authentication:

To secure access to sensitive endpoints (such as viewing or storing patient prediction data), the project implements token-based authentication using JSON Web Tokens (JWT). This ensures that only authorized users (e.g., doctors, medical staff) can access protected routes.

How It Works:

1. User Login:

A user (e.g., doctor or admin) logs in using valid credentials via a dedicated authentication endpoint (e.g., /api/auth/login).

2. JWT Generation:

- On successful login, the server generates a JWT token using a secret key (stored in.env as JWT_SECRET).
- This token includes encoded user information and an expiration time.

3. Token Storage:

- The token is sent to the frontend and typically stored in:
    - LocalStorage or SessionStorage
    - Or managed securely in HTTP-only cookies (optional for enhanced security)

4. Authenticated Requests:

For protected routes, the token must be included in the Authorization header:

```
Authorization: Bearer <your_token_here>
```

5. Token Verification Middleware (Backend):

A middleware function intercepts incoming requests and verifies the token:

```
const jwt = require('jsonwebtoken');

function verifyToken(req, res, next) {

  const authHeader = req.headers['authorization'];

  const token = authHeader && authHeader.split(' ')[1];
  if (!token) return res.sendStatus(401); // Unauthorized
  jwt.verify(token, process.env.JWT_SECRET, (err, user) => {
    if (err) return res.sendStatus(403); // Forbidden
    req.user = user;
    next();
  });
}
```

6. Role-Based Authorization:

You can restrict routes further based on user roles (e.g., only admins can delete records).

User Interface:

The user interface of the **"Revolutionizing Liver Care"** project is designed to be intuitive, responsive, and medical-professional friendly. Below are suggested screenshots or GIFs to include in your documentation or project presentation:

1. Home / Landing Page:

- **Features**:
    - Project title and tagline
    - Brief description or mission statement
    - Navigation bar with links to Dashboard, Predict, and About

2. Patient Data Input Form:

- **Features**:

    - Fields for patient name, age, gender, and clinical parameters (e.g., bilirubin, albumin, etc.)
    - Submit button to predict cirrhosis risk

3. Prediction Result Panel:
    - **Features**:
        - Displays risk level (e.g., Low, Medium, High)
        - Shows prediction score (e.g., 0.78 confidence)
        - May include a health advisory or recommendation
4. Patient Dashboard / Record Table:
    - **Features**:
        - Table showing past predictions with patient info
        - Filter/search capabilities by name or risk level

5. Data Visualization Panel:
    - **Features**:
        - Charts showing trends (e.g., how many patients fall into each risk category)
        - Feature importance visualization.
6. Login/Authentication Page:
    - **Features**:
        - Input fields for email and password
        - Forgot password and sign-up options

Testing:

A comprehensive testing strategy is implemented to ensure the reliability, correctness, and robustness of both frontend and backend components of the application. Testing is done at multiple levels: unit, integration, and end-to-end (E2E).

Testing Strategy:

- **Unit Testing:** Checks individual components and functions.
- **Integration Testing:** Verifies interaction between frontend, backend, and database.
- **End-to-End Testing:** Simulates user behaviour across the full application.

Tools Used:

- **Frontend:** Jest, React Testing Library
- **Backend:** Mocha, Chai, Supertest
- **E2E Testing:** Cypress or Playwright
- **ML Model (Python):** Pytest
- **Mocking:** MSW, Sinon

Example:

- Test form input submission in React.
- Test /api/predict route for valid prediction response.

Goal:

- Ensure app reliability with 80%+ test coverage and automated testing in CI/CD pipeline.

Screenshots or Demo:

1. **Home Page**
   Description: Clean landing page with navigation bar and project overview.
   Screenshot: home_page.png
2. **Patient Data Input Form**
   Description: Form to enter patient details and clinical values.
   Screenshot: input_form.png
3. **Prediction Result Page**
   Description: Displays risk level (e.g., High/Low) with a confidence score.
   Screenshot: prediction_result.png
4. **Dashboard**
   Description: List of past predictions with filters and search functionality.
   Screenshot: dashboard.png
5. **Login Page**
   Description: Secure login interface for doctors/admins.
   Screenshot: login_page.png

Known Issues:

1. **Model Prediction Delay**

- Issue: Predictions may take a few seconds if the machine learning model is hosted on a slow or remote server.
- Workaround: Optimize model size or deploy on a faster cloud instance.

2. **Limited Input Validation**
    - Issue*:* Current input form lacks advanced validation (e.g., for negative values or missing fields).
    - Workaround: Users must ensure all fields are filled with realistic values.

3. **Authentication Not Fully Implemented**
    - Issue: Role-based access control (e.g., admin vs. doctor) is planned but not fully active.
    - Workaround*:* All logged-in users currently have the same access level.
4. **No File Upload Support**
    - Issue: Bulk upload via CSV or Excel is not available.
    - Planned Fix*:* Future update will allow file-based data input.
5. **Mobile Responsiveness**
    - Issue: Some dashboard elements may not render well on small screens.
    - Workaround*:* Use the app on desktop or tablet for best experience.

Future Enhancements:

1.Clinical Data & Model Quality

- Integrate imaging-derived features (e.g., ultrasound elastography scores).
- Support longitudinal data to model disease progression over time.
- Add automatic feature engineering and hyperparameter tuning pipelines.

2. Explainability & Trust

- SHAP- or LIME-based feature contribution views per patient.
- Global vs. patient-level risk factor comparison dashboards.
- Confidence intervals and model uncertainty visualization.

3. Workflow & EHR Integration

- FHIR/HL7 connectors to pull lab values directly from hospital systems.
- Auto-import recent lab panels to pre-fill prediction forms.
- Send flagged high-risk patients to clinician task queues.

4. Advanced Analytics

- Cohort analytics (risk distribution by age, gender, comorbidities).
- Outcome tracking: link predictions to actual biopsy/imaging follow-up.
- Model drift detection and automated re-training alerts.

5. User Experience

- Mobile-optimized provider view.
- Bulk CSV/Excel upload for screening camps.
- Multi-language UI (start with English + regional languages).

6. Security & Compliance

- Full role-based access (doctor, lab tech, admin).
- Audit logging for all record views/edits.
- Optional end-to-end encryption of PHI and regional compliance modes (HIPAA, NDHM/ABDM-ready).

7. Deployment & Scaling

- Containerized microservices (Docker + Kubernetes).
- Serverless prediction endpoint for bursty workloads.
- Edge deployment option for offline clinics with periodic sync.

8. Alerts & Notifications

- Threshold-based SMS/email alerts for high-risk patients.
- Scheduled patient follow-up reminders.

- Integration with WhatsApp Business API (where allowed).

9. Data Quality Tools

- Built-in lab unit normalization (mg/dL vs µmol/L).
- Outlier detection and clinician review queue.
- Missing-data imputation guidance UI.

10. Research Extensions

- Comparative benchmarking against existing cirrhosis scoring systems (e.g., MELD, Child-Pugh).
- Multi-class modeling (fibrosis staging vs. binary cirrhosis).
- Federated learning across hospitals without centralizing raw data.

Conclusion:

This project demonstrates the potential of machine learning in transforming liver disease diagnosis, particularly in predicting liver cirrhosis at an early stage. By leveraging patient clinical data and deploying intelligent predictive models, the system offers a non-invasive, efficient, and accessible tool for healthcare professionals. The integration of a user-friendly interface, secure backend, and scalable architecture ensures a reliable experience for both developers and end users. With further enhancements and clinical validation, this application can significantly contribute to proactive liver care and early medical intervention—ultimately improving patient outcomes and reducing healthcare burdens.