Project Documentation: Revolutionizing Liver Care

1. Introduction

Project Title: Revolutionizing Liver Care : Predicting Liver Cirrhosis using Advanced Machine Learning Techniques using AI/ML

Team Members:

- **Jhansi [member 1] – Frontend Developer**
  (Development of HTML pages, user interface for prediction and result display)

- **Siddhardha Kalisetti [member 2] – Flask Backend Devoloper**
  (API integration, Flask app development, model deployment on local server)

- **Donipudi Nandini [member 3] - Documentation and Testing**
  (Requirements gathering, documentation and user guide)

- **Nandini Devika [member 4] – Machine Learning Engineer**
  (Collected and Cleaned dataset, Built and trained the ML model using XGBoost)

The "Revolutionizing Liver Care" project uses AI and ML techniques to predict the possibility of liver disease based on medical data. The goal is to support early diagnosis using data-driven models. The system is designed to accept clinical input data and return real-time prediction results using a trained ML model.

By combining healthcare insights and machine learning algorithms like XGBoost, the system enhances the accuracy and speed of liver disease detection. The system supports doctors in identifying risks early and provides patients with useful health insights.

- Highlights:
- A machine learning model trained on real patient data.
- Prediction of liver disease likelihood from input fields.
- Web app interface for user interaction and data input.
- Deployment-ready design for real-world usage.

2. Project Overview

The project's main aim is to create an intelligent system that helps predict liver diseases from structured patient health records. This is especially useful in remote or under-equipped healthcare settings where AI can serve as a digital assistant to doctors.

This system supports uploading or entering medical parameters like bilirubin levels, enzyme counts, and protein measurements to determine liver condition. The integration of ML with web-based UI allows quick and easy access to diagnosis tools for both medical practitioners and patients.

- Key Features:
- Real-time liver disease prediction using AI/ML models.
- User-friendly web form for data entry and result display.
- Backend logic connected to Python ML model via API.
- Results stored for future analysis and patient records.

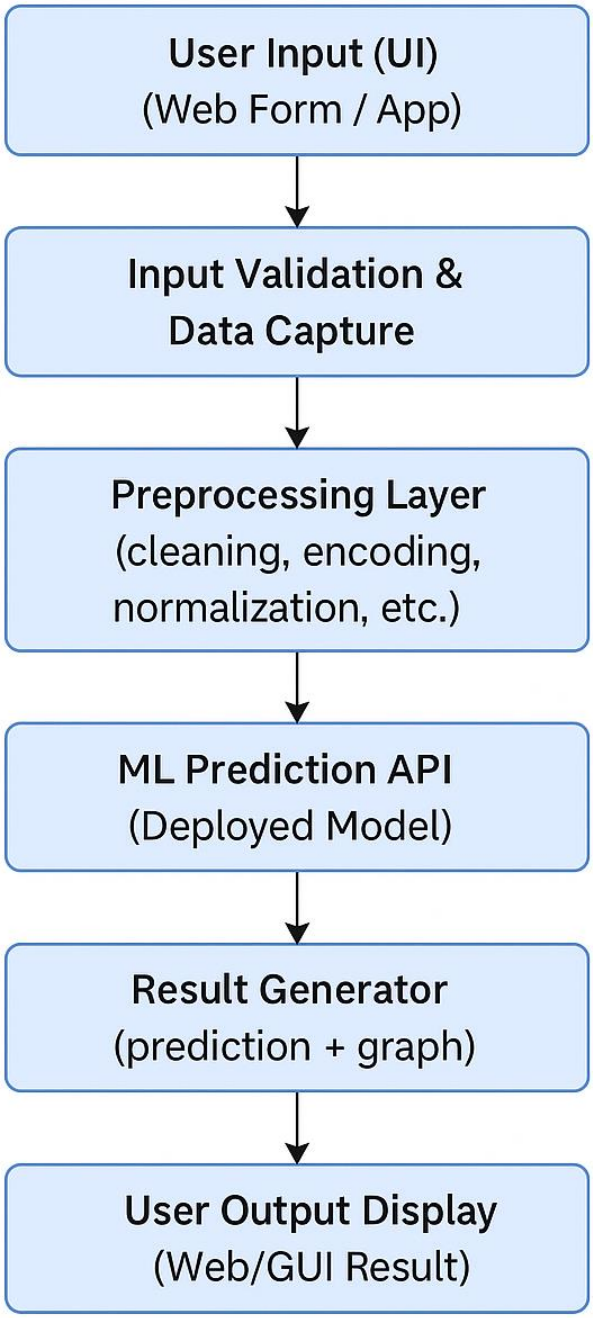
3. Architecture

The project architecture consists of a React-based frontend, Node.js backend, and Python-based ML model for predictions. MongoDB is used to store user and prediction data securely.

The backend acts as a bridge between the web interface and the trained model. It receives input, processes it through the Python script, and sends back results. The model used is trained using the XGBoost algorithm which is highly effective for medical data.

- Components:
- Frontend: React-based input form for patient data and display of prediction results.
- Backend: Node.js and Express manage API routes, session tokens, and Python script interaction.
- ML Model: Python-trained XGBoost model saved with joblib.
- Database: MongoDB stores user data and prediction history.

Technical Architecture: Revolutionizing Liver Care Using ML



4. Setup Instructions

To run the project, both the frontend and backend environments must be set up. Node.js and Python dependencies need to be installed, and MongoDB must be running either locally or using cloud services like MongoDB Atlas.

A .env file should be configured with MongoDB connection strings and any backend secrets. Python dependencies like xgboost, scikit-learn, and joblib must also be installed to run the model.

- Steps:
- Install Node.js, MongoDB, and Python 3 on your system.
- Clone the repository and install dependencies using npm install and pip install.
- Start MongoDB and set the environment variables in the .env file.
- Run backend, frontend, and ensure the Python model is reachable.

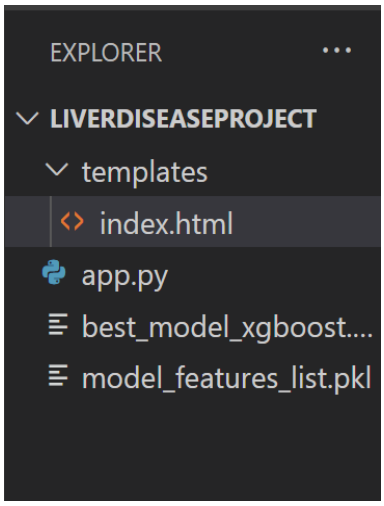
5. Folder Structure

The project folder is divided into a client folder (React frontend), a server folder (Node.js backend), and a ml_model or predict.py file (Python ML model).

Organizing the project like this allows for easy debugging and separation of concerns between logic layers.

- Structure Overview:
- Client: src/components, src/pages, App.js, and Form.js
- Server: routes, controllers, models, app.js
- Model: predict.py, best_model_xgboost.pkl, model_features_list.pkl
- Others: .env, README.md, package.json

Structure:



6. Running the Application

To run the application, start the backend server to handle requests and frontend server to display the web interface. Make sure Python model services are available and linked correctly.

It is important to test if all servers are running in sync and the model responds to requests before performing actual predictions.

- Steps to Run:
- Frontend:

  cd client
  npm start
- Backend:

  cd server
  npm start
- ML Model: Ensure Python file (predict.py) runs and responds via route integration.

7. API Documentation

The backend exposes endpoints that connect to the model. These endpoints validate inputs, call the model, and return responses.

Postman or similar tools can be used to test these endpoints during development and deployment.

- Endpoints:
- POST /api/predict – Predict liver condition.
- POST /api/register – Register user account.
- POST /api/login – User login and token generation.
- GET /api/history – Retrieve past predictions (if implemented).

8. Authentication

User login and access control are handled through JWT tokens. Sensitive data like passwords are encrypted using bcrypt and stored in MongoDB.

Each session includes a token that must be passed with secure endpoints to ensure only logged-in users can access the model.

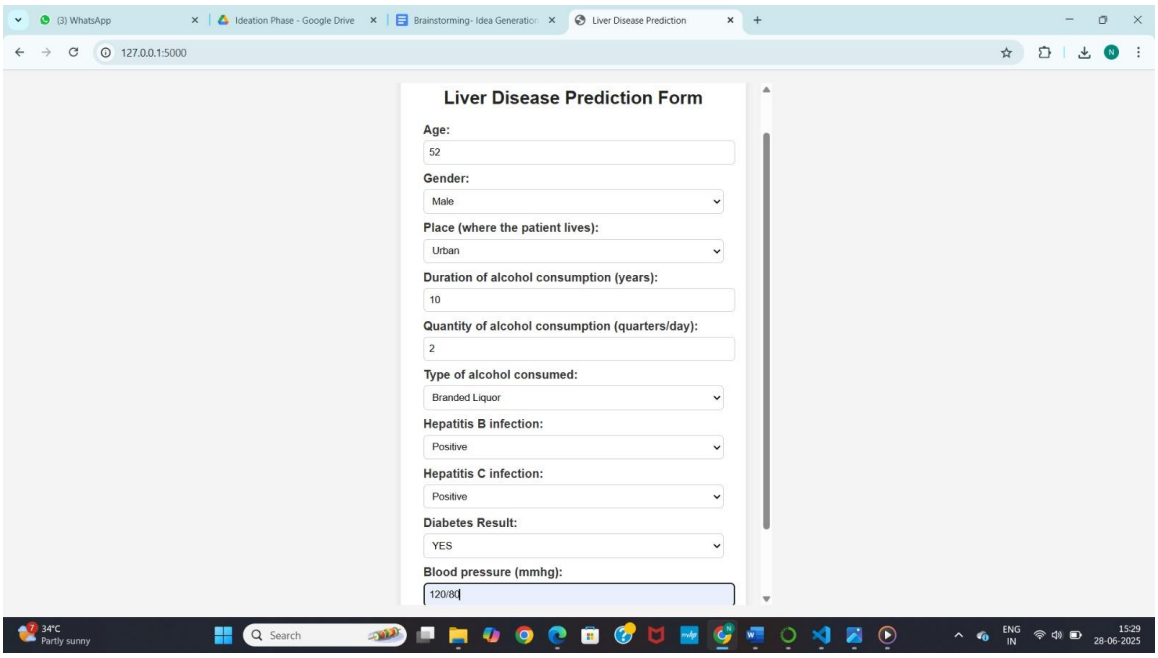
- Security Measures:
- JWT token-based authentication for login sessions.
- bcrypt hashing for password security.
- Protected routes for prediction access.
- Secure cookies or local storage token handling.

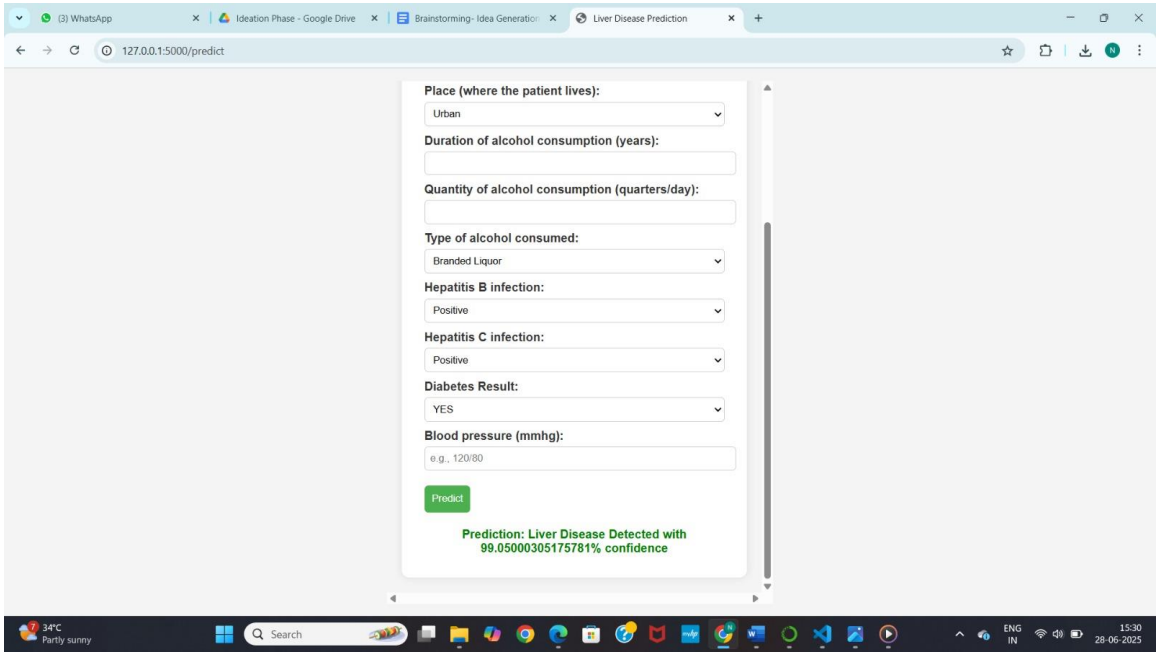
9. User Interface

The UI is minimal and intuitive. Users can register/login, enter medical values, and receive prediction results within seconds. The UI also displays alerts and results visually.

The design supports responsive behavior, making it usable on both desktops and mobiles.

- UI Screens:
- User Registration/Login screen
- Medical Form page for inputs
- Results page for predictions
- Error and alert messages





10. Testing

Manual testing was performed for each module to ensure data flow was correct. Backend endpoints were tested using Postman. Model predictions were verified using test data.

Test cases focused on validating form fields, backend responses, and overall integration.

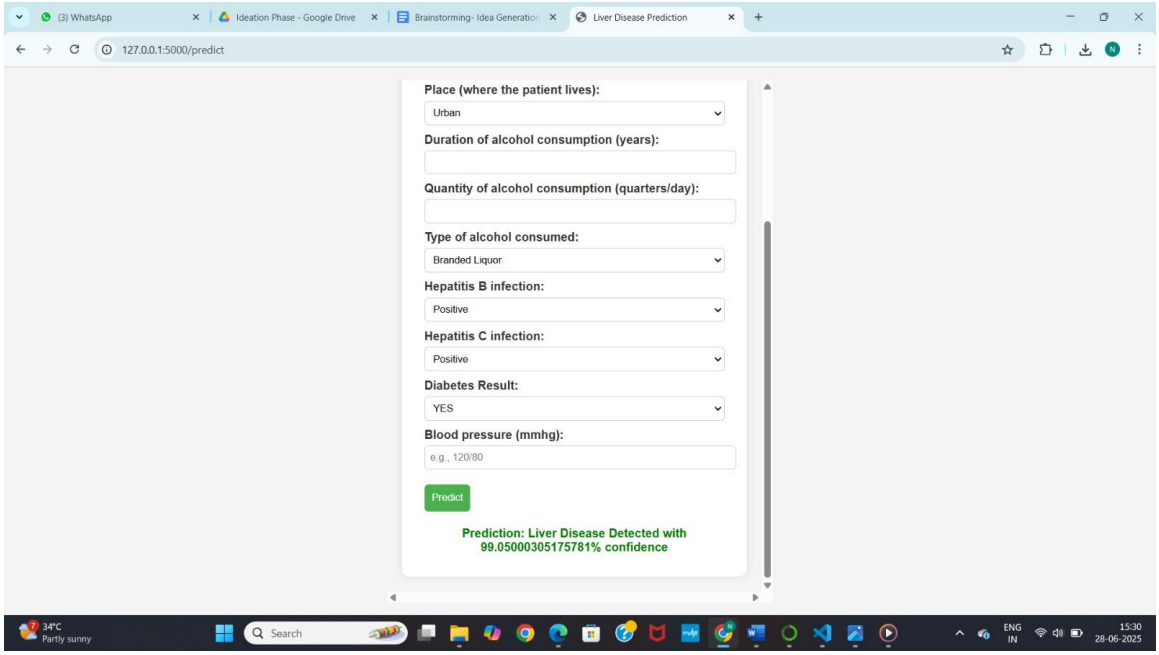
- Testing Activities:
- Postman testing for APIs.
- Field validation on form input.
- Cross-browser compatibility testing.
- ML prediction accuracy check using validation data.

11. Screenshots or Demo

Screenshots visually show the project flow and UI. They help stakeholders understand the user experience and how data is passed to the model.

You can also link a video demo or a hosted version for live testing.

- Suggestions:
- Add screenshots of home, form, prediction result page.
- Include a short video/GIF of the full flow.
- Embed demo link (if deployed online).
- Use captions to describe screenshots.



12. Known Issues

Some issues still exist with deployment and performance. Python model communication may lag if the server is slow. In some browsers, form field validation errors may not show properly.

These issues do not affect core functionality but can be improved in future updates.

- Reported Bugs:
- Delay in model response time.
- Browser-based form input issues.
- Missing validation for edge cases.
- Model accuracy drops slightly on unseen datasets.

13. Future Enhancements

To make the project production-ready, several improvements are planned. Features like user history tracking, chart-based insights, and real-time health dashboards can enhance usability.

Model accuracy will also be improved by training on more diverse and high-quality datasets.

- Upcoming Improvements:
- Visualization dashboard with charts for doctors.
- History of past predictions per user.
- Cloud deployment (e.g., Render, Heroku, AWS).
- Better model integration and error handling.

-----THANK YOU-----