



## Project Title:

Revolutionizing Liver Care: Predicting Liver Cirrhosis using Advanced Machine Learning Techniques.

## Project Members:

- Team ID: LTVIP2025TMID38625
- Team Size: 5
- Team Leader: Shaik Azmeen
- Team member: Shaik Noushin
- Team member: Shaik Tasleem Kousar
- Team member: Shaik Naga Neelima
- Team member: Shaik Husne Jaha

## 1.Introduction

### Revolutionizing Liver Care: Predicting Liver Cirrhosis Using Advanced Machine Learning

**Revolutionizing Liver Care** is a smart healthcare project that uses **advanced machine learning techniques** to **predict liver cirrhosis**—a severe, progressive liver disease that can lead to liver failure if not detected early.

By analyzing **clinical features** such as age, bilirubin levels, albumin, and other lab results, the system can accurately estimate whether a patient is likely to develop cirrhosis. Instead of relying solely on invasive procedures like liver biopsy or slow clinical scoring systems, this project offers a **faster, data-driven alternative** that improves early diagnosis and treatment planning.

The solution is built using **machine learning models** like Random Forest and Logistic Regression and is deployed through a **user-friendly web interface** using React (frontend) and Flask (backend).

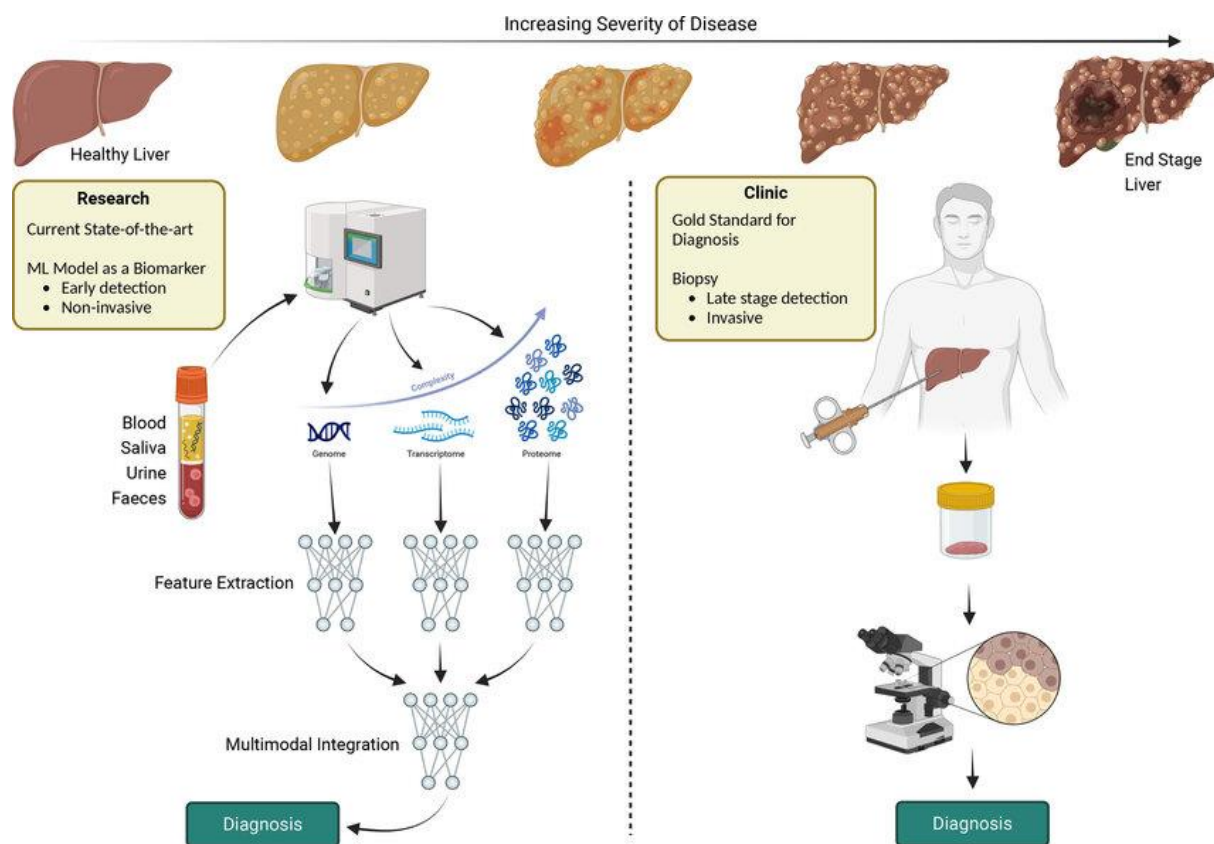
This project aims to **support doctors, reduce diagnostic delay, and make liver care more accessible, scalable, and accurate.**

Liver cirrhosis is a chronic, progressive liver disease that results in irreversible scarring of liver tissue. If not diagnosed and treated early, it can lead to severe complications including liver failure and death. Due to the often-asymptomatic nature of early-stage cirrhosis, timely diagnosis is a critical challenge in clinical practice.

This project aims to revolutionize liver care by developing a machine learning-based system for the early detection and prognosis of liver cirrhosis. By leveraging patient health data and advanced ML algorithms, the system can predict the likelihood of cirrhosis before it progresses to severe stages.

The solution involves building a predictive model using multiple machine learning techniques such as **Random Forest, Decision Trees, K-Nearest Neighbours (KNN), and XG Boost**. These models are trained on a medical dataset, evaluated using various performance metrics, and integrated into a user-friendly web application using the Flask framework.

The end goal is to provide healthcare professionals with a decision support tool that facilitates early interventions, optimizes treatment planning, and improves patient outcomes. Additionally, the project demonstrates the practical applications of AI in healthcare, particularly in the field of hepatology, where clinical data can be transformed into actionable insights.



## 2.Project Overview:

### Objective:

To develop an intelligent predictive model using Machine Learning (ML) for the early detection and prognosis of liver cirrhosis, enabling healthcare professionals to make timely decisions and provide personalized treatments.

### **Motivation:**

Liver cirrhosis is a serious, irreversible condition that progresses over time. Early detection can:

- Save lives by preventing complications
- Reduce treatment costs
- Optimize the use of medical resources

This project showcases the **power of AI in healthcare**, specifically **hepatology**, by converting raw clinical data into actionable predictions.

### **Machine Learning Approach:**

- Use **supervised learning algorithms** such as:
  - **Random Forest**
  - **Decision Tree**
  - **K-Nearest Neighbours (KNN)**
  - **XG Boost**
- Perform **data preprocessing, EDA, model training, and evaluation**
- Select the **best performing model** and **tune it** for maximum accuracy

### **Evaluation Metrics:**

Models will be assessed using:

- Accuracy
- Precision
- Recall
- F1 Score
- ROC-AUC Curve

### **Deployment:**

- Build a **web-based interface** using **Flask**
- Integrate the trained ML model (.Pkl files)

- Accept user input through a **simple UI**
- Display the prediction result (e.g., “Risk of Cirrhosis” or “Healthy”)

## 🔗 Main Goals of the Project

The **main purpose** of the guided project "**Revolutionizing Liver Care: Predicting Liver Cirrhosis using Advanced Machine Learning Techniques**" is:

To develop an intelligent, machine learning-based predictive system that can accurately detect liver cirrhosis at an early stage using patient health data, and to deploy this system through a user-friendly web application for real-time predictions.

### Project Goals:

1. **Early Detection**  
Identify patients at risk of liver cirrhosis **before severe symptoms develop**, enabling early treatment.
2. **Clinical Decision Support**  
Assist healthcare professionals in making **data-driven decisions** about diagnosis and treatment strategies.
3. **AI in Healthcare**  
Showcase the practical use of **machine learning algorithms in medical diagnosis**, especially in hepatology.
4. **User Accessibility**  
Create a **web-based interface** where users can enter patient data and receive **instant predictions**, making the tool usable in real-time clinical environments.
5. **Optimization of Healthcare Resources**  
Reduce unnecessary tests and hospitalizations by **targeting high-risk patients**, improving efficiency.
6. **Improving Diagnostic Efficiency**  
To enhance the accuracy and efficiency of liver disease diagnosis through machine learning, supporting healthcare professionals in making better-informed clinical decisions.

## Key Features of the Revolutionizing Liver Care Project

### 1. Machine Learning-Based Prediction

- Uses advanced machine learning algorithms (like Logistic Regression, Random Forest, etc.) to predict the likelihood of liver cirrhosis.
- Trained on real medical datasets to ensure high accuracy and reliability.

## **2. Interactive Web Application**

- A user-friendly front-end interface (HTML/CSS/JavaScript) allows users to input patient health parameters like age, bilirubin levels, etc.
- Easy-to-use form with clear inputs for non-technical users.

## **3. Real-Time Prediction Output**

- Backend (Flask) connects the front end to the ML model.
- Processes input data and return predictions instantly (e.g., “Positive for Cirrhosis” or “Negative”).

## **4. Integrated Frontend and Backend**

- Seamless communication between the front end and back end using HTTP POST requests.
- Demonstrates full-stack project development with API integration.

## **5. Data Visualization (Optional/Extended Feature)**

- Can include graphical charts or heatmaps showing patient condition trends.
- Useful for understanding feature importance and model behaviour.

## **6. Model Explainability (Optional)**

- Integration of tools like SHAP or LIME to explain how the model arrived at its prediction.
- Builds trust with healthcare professionals and patients.

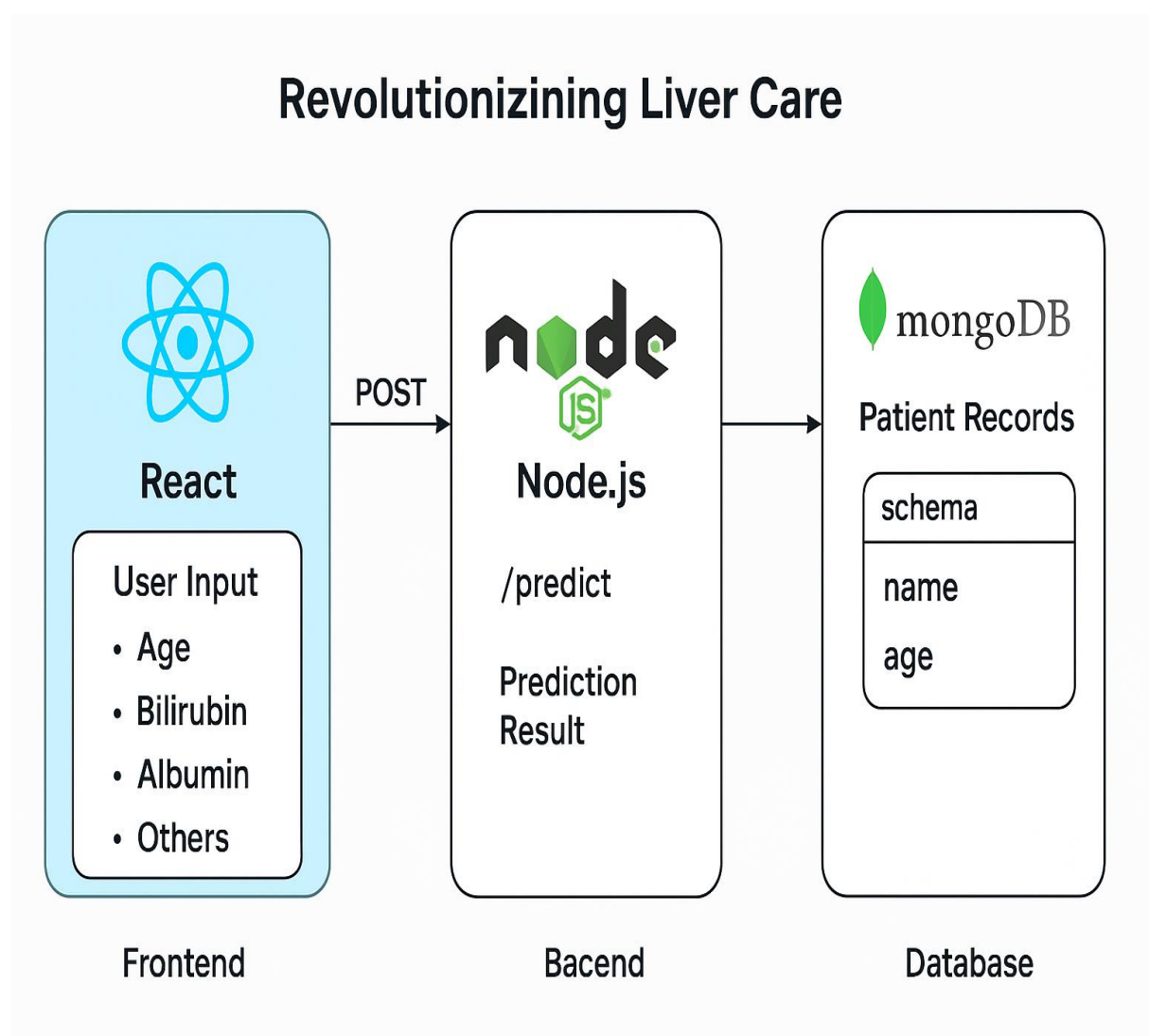
## **7. Portable and Scalable**

- The project structure is modular and can be deployed locally or on cloud platforms (Heroku, Render, AWS).
- Can easily be extended to other liver diseases or integrated with hospital databases.

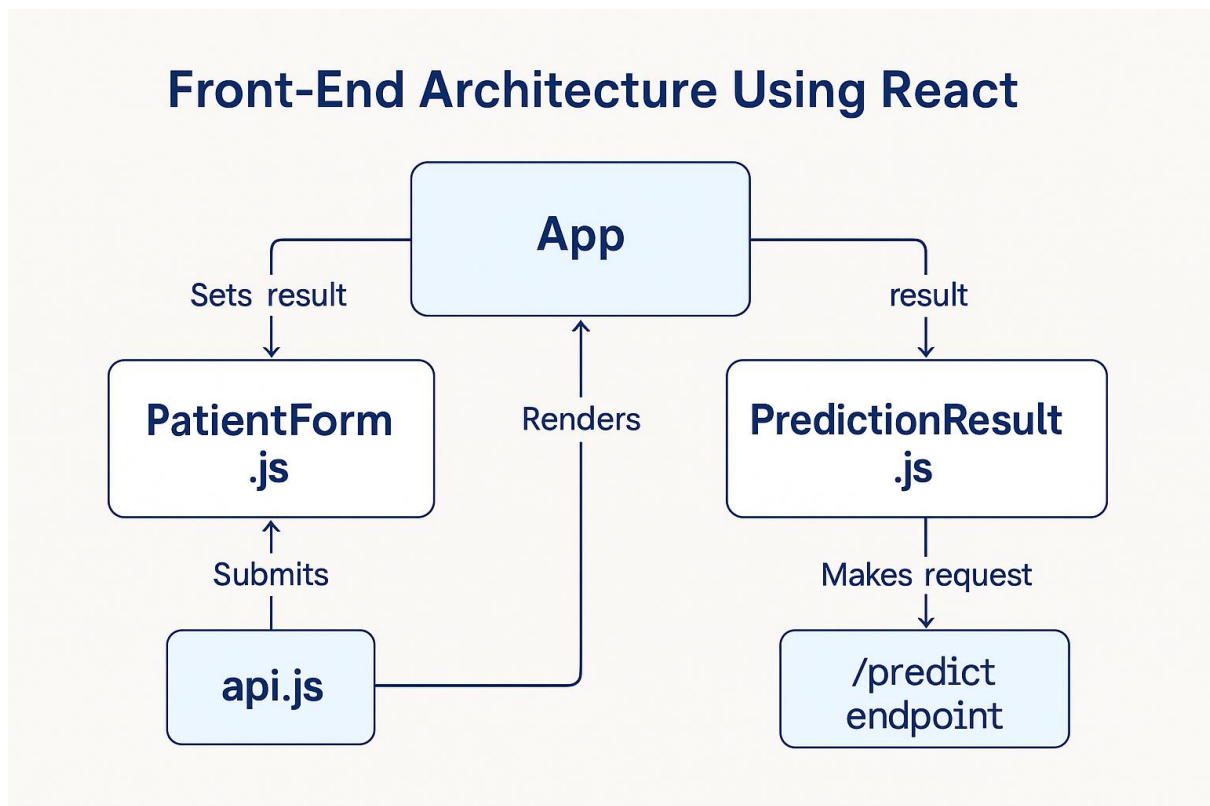
## **8. Educational & Awareness Component**

- Helps raise awareness about liver health by showing the importance of early diagnosis.
- Could be extended to include liver care tips and prevention strategies.

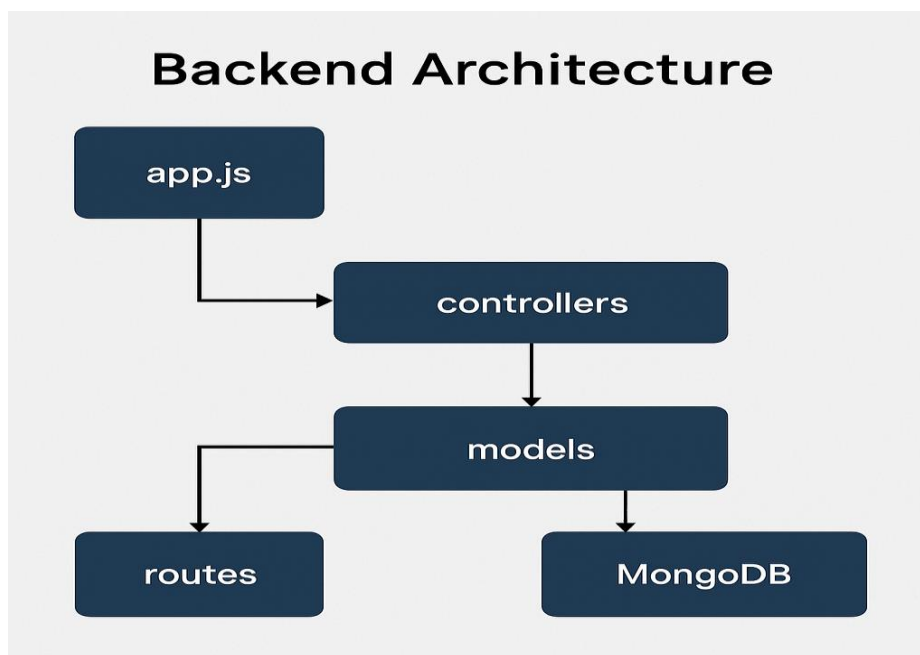
### 3. Architecture:



## Frontend Architecture:



## Backend Architecture:





## Backend Data Flow

React Frontend

|



POST /API/patients/predict ———— ? Express Route

|



Controller Logic (Predict + Save)

|



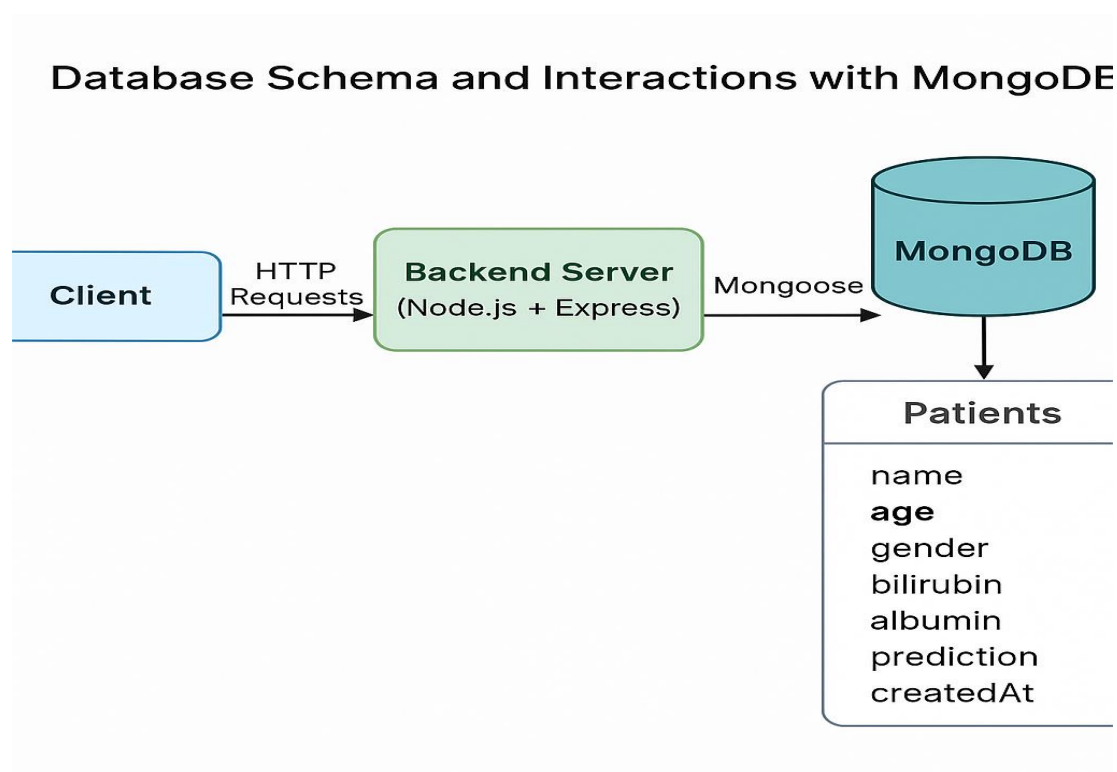
MongoDB (Store patient + prediction)

|



Response to Frontend (JSON)

## Database Architecture:



## System Overview

Your backend architecture is designed to:

- Receive medical data from the frontend (React app).
- Process or pass that data to a prediction engine (ML model or rule-based logic).
- Store patient records and predictions in MongoDB.
- Return prediction results or data to the frontend.

## 1. System Architecture Overview

The backend is designed with the following layers:

- API Layer: Built using Express.js
- Database Layer: MongoDB for storing patient data and predictions
- Communication: JSON over HTTP (REST API)
- ODM: Mongoose for modelling MongoDB data.

### Code:

React Frontend → Express API → Mongoose → MongoDB Atlas → Response

## 2. MongoDB Collection Design

MongoDB is a NoSQL database, so instead of tables and rows (like SQL), it uses collections and documents.

### Collection: patients

Each document in this collection stores information about one patient and their prediction.

## 4. Setup Instructions:

### Prerequisites:

#### Definition:

Prerequisites refer to the **software, tools, and packages** that must be **installed before running or developing** the project.

These dependencies ensure that your system has all the **necessary environments and libraries** to run the application correctly.

#### Example (Related to *Revolutionizing Liver Care Project*)

Since our project includes:

- **React (Frontend)**
- **Flask + Python (Backend with ML)**
- **Machine Learning Models**

#### PART 1: Prerequisites for Client Side (React)

Before setting up the frontend, make sure you have these installed on your system:

Tool	Description	Download
<b>Node.js</b>	JavaScript runtime to run React apps	<a href="https://nodejs.org">https://nodejs.org</a>
<b>name</b>	Node package manager (comes with Node.js)	Installed with Node.js
<b>Code Editor</b>	To edit code (e.g., VS Code)	<a href="https://code.visualstudio.com">https://code.visualstudio.com</a>
<b>Browser</b>	To view the app (Chrome, Firefox, etc.)	

#### Python Dependency Example

IF using Flask and ML:

Create a requirements.txt:

Flask

Flask-cors

Pandas

Scikit-learn

Joblib

Includes:

- react
- axios (for calling backend API)
- react-router-dom (for navigation)
- bootstrap or tailwindcss (for UI design)

## Summary

Category	Example Tools	Purpose
Backend	Python, Flask, pandas	Build APIs, run ML model
Frontend	Node.js, npm, React	Build and serve the UI
ML Tools	scikit-learn, joblib, pickle models	Train, save, and use ML
Utility	Git	Version control and collaboration
Database	MongoDB (optional)	Storing health records

## Installation Guide (Revolutionizing Liver Care Project)

This section explains how to **clone the project**, **install dependencies**, and **configure the environment** step-by-step.

### Step 1: Clone the Project Repository

First, download the project files from GitHub.

```
git clone https://github.com/your-username/revolutionizing-liver-care.git
```

```
cd revolutionizing-liver-care
```

### Step 2: Set Up the Backend (Flask + ML Model)

1. Navigate to the backend folder:

```
Cd backend
```

2. Create a Python virtual environment:

```
python -m venv venv
```

3. Activate the virtual environment:

On Windows:

```
venv\Scripts\activate
```

on macOS/Linux:

```
source venv/bin/activate
```

4. Install the required Python libraries:

```
pip install -r requirements.txt
```

```
flask
```

```
flask-cors
```

```
scikit-learn
```

```
pandas
```

```
joblib
```

### Step 3: Start the Flask Backend Server

. The backend will start at: <http://localhost:5000>

. API endpoint (example): <http://localhost:5000/predict>

### Step 4: Set Up the Frontend (React)

1. Open a **new terminal** and go to the frontend folder:
2. Install the React dependencies:
3. Set up environment variables (Optional but recommended):

### Step 5: Start the React Frontend App

- Your web app will open at: <http://localhost:3000>
- It will communicate with the Flask backend running at [localhost:5000](http://localhost:5000)

**Final Test:** Open your browser and go to:

Try submitting a test input through the form. The data should be sent to the Flask backend, which will return a **prediction from the ML model**.

## 5.Folder Structure:

### Client Side: React Frontend Structure

#### Purpose:

The React frontend collects patient data and communicates with the backend server to receive liver cirrhosis risk predictions, then displays the result.

## Project Folder Structure (Suggested)

liver-care-client/

```
|— public/
|   |— index.html      # The root HTML file
|— src/
|   |— components/     # Reusable UI components
|   |   |— PatientForm.js  # Patient input form
|   |   |— PredictionResult.js # Displays result from backend
|   |— services/       # Handles API calls
|   |   |— api.js         # Axios or fetch functions
|   |— styles/         # Optional CSS styling
|   |   |— App.css
|   |— App.js          # Main component that brings everything together
|   |— index.js        # Renders <App /> into the DOM
|— .env               # Environment variables (if needed)
|— package.json       # Project dependencies and scripts
|— README.md          # Project description and setup guide
```

## Components and Their Roles:

File	Description
<b>App.js</b>	The root component that holds the full app layout and imports other components.
<b>PatientForm.js</b>	A form that lets users enter patient details (age, gender, bilirubin, etc.). On submit, it sends data to the backend.
<b>PredictionResult.js</b>	Displays the risk prediction returned from the backend.
<b>api.js</b>	Contains functions for HTTP communication (e.g., POST to /predict endpoint).
<b>index.js</b>	Loads the React app into the root HTML node.

## Server: Organization of the Node.js Backend:

### 1. Project Folder Structure (Typical Node.js + Express)

```
revolutionizing-liver-care/
├── backend/                # Backend folder
│   ├── config/            # Configuration (e.g., DB, environment)
│   │   └── db.js
│   ├── controllers/       # Logic to handle requests
│   │   └── liverController.js
│   ├── models/            # Data models (e.g., Patient schema)
│   │   └── LiverModel.js
│   ├── routes/            # API endpoints
│   │   └── liverRoutes.js
│   └── utils/             # Utilities like ML model loader, data
├── normalizer
│   └── predictor.js
├── ml_models/             # Saved ML model files (e.g.,
rf_acc_68.pkl)
│   └── rf_model.pkl
├── .env                   # Environment variables (e.g., PORT, DB
URI)
├── server.js              # Entry point of the Node.js app
└── package.json           # Dependencies and scripts
```

### 2. Key Components Explained

3. Defines the API routes for prediction and health data.

4. Handles prediction logic (e.g., input validation, calling ML).

- Loads your ML model (e.g., via `python-shell`, `flask-microservice`, or `TensorFlow.js`).
- Example: Calling a Python model via child process

## Summary: What This Backend Does

- Accepts input health data from the frontend
- Sends data to a Python ML model or uses a built-in JS model
- Returns the liver condition prediction to the frontend

- Handles routing, logging, and optional database storage (e.g., MongoDB)

## 6. Running the Applications:

Frontend: `npm start` in the client directory.

### What is "Running the Frontend Application"?

In your project, the **frontend** is the **Graphical User Interface (GUI)** – the part that users interact with. It includes:

- Web pages (HTML)
- Styles (CSS)
- Interactions (JavaScript or React)

When you **run the frontend**, you are **starting a development server** that:

1. **Loads the user interface in your web browser** (like Google Chrome)
2. **Connects with the backend** to show results (like ML predictions for liver disease)
3. **Automatically reloads** when you make changes to your code.

### In "Revolutionizing Liver Care"

Let's say your app predicts whether a person may have liver disease based on medical input.

Running the frontend will:

1. Open a form in the browser where a **user enters patient data** (like age, bilirubin, albumin, etc.)
2. Send this data to your **Flask backend server**
3. Display the **prediction result** (like *"Likely Liver Disease"* or *"No Liver Disease"*) on the screen.



## Revolutionizing-Liver-Care/

```
|
|
| └── client/          # React frontend
|   |
|   | └── public/
|   |
|   | └── src/
|   |
|   └── package.json
|
|
| └── server/          # Flask backend
|   |
|   | └── app.py
|   |
|   | └── model/
|   |   |
|   |   | └── rf_acc_68.pkl
|   |   |
|   |   └── normalizer.pkl
|   |
|   └── requirements.txt
|
|
└── README.md
```

## What Happens Behind the Command

`npm start`

When you run:

Npm start

**Frontend:** React (inside client/ folder)

**Backend:** Flask (Python, inside server/ folder)

You may be using .pkl model files for prediction (ML part)

### **FRONTEND – React App (Located in client/)**

**Step 1:** Navigate to the client directory

Step 2: Install required dependencies

Step 3: Start the React frontend

`npm start`

inside the client/ folder, it will:

- Launch the development server (usually on <http://localhost:3000>)
- Show the frontend of your liver care app
- Allow you to interact with your React UI and test changes instantly.

**Backend: npm start in the server directory.**

### What is “Running the Backend Server Locally”?

In your project, the **backend** is the part of the application that:

- Processes user inputs
- Loads and applies the ML model (e.g., .pkl files)
- Connects with a database (if used)
- Sends prediction results to the frontend

When you **run the backend server locally**, you are starting a local server on your machine (usually with npm start or node app.js) that listens for incoming requests from the frontend (like patient data) and responds with the prediction or result.

server/

```
|
|
|— server.js      # This file (your backend entry point)
|
|— package.json
|
|— routes/        # Optional: for modular API routes
|
|— model/         # Your saved ML models (optional)
|
| |— rf_acc_68.pkl # If calling Python Flask model separately
| |— normalizer.pkl
|
|— .env           # For config values (e.g., PORT)
```

### In Revolutionizing Liver Care

1. A doctor enters patient medical data in the web form (frontend).
2. The frontend sends that data to the backend (running locally).
3. The backend server:
  - Loads your machine learning model (e.g., rf\_acc\_68.pkl)
  - Normalizes the data
  - Predicts the liver health status

4. The backend sends the result (e.g., "No Liver Disease") back to the frontend.
5. The frontend displays the result to the doctor.

### How to Run the Backend Server (Node.js Example)

```
import matplotlib.pyplot as plt
import pandas as pd
import seaborn as sns
import pickle as pkl
import numpy as np
from sklearn import svm
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression, LogisticRegressionCV, RidgeClassifier
from sklearn.model_selection import train_test_split, GridSearchCV
from xgboost import XGBClassifier
from sklearn.preprocessing import Normalizer
from sklearn.metrics import accuracy_score, f1_score, recall_score, precision_score, confusion_matrix
```

### Step-by-step

❏ Open your terminal and go to the backend folder

❏ Install all required dependencies (first time only)

❏ Start the backend server

This runs your backend app on <http://localhost:5000>  
(or the port defined in your .env file)

## 6.API Documentation:

**Document all endpoints exposed by the backend.**

### 1. GET /

◆ Description:

Checks if the backend server is up and running.

◆ Method:

◆ Request:

No parameters required.

◆ Response:

"Revolutionizing Liver Care backend is running 

### 2. POST /predict

### ◆ Description:

Takes patient medical input and returns a liver disease prediction using the machine learning model.

◆ **Method:** POST /predict

◆ **Headers:** Content-Type: application/Json

### ◆ Request Body:

```
{
  "age": 60,
  "total_bilirubin": 1.6,
  "direct_bilirubin": 0.5,
  "alkaline_phosphatase": 210,
  "alamine_aminotransferase": 50,
  "aspartate_aminotransferase": 60,
  "total_proteins": 6.8,
  "albumin": 3.1,
  "albumin_globulin_ratio": 1.0
}
```

### ◆ Success Response (200 OK):

json

Copy code

```
{
  "prediction": "Liver Disease Detected",
  "confidence": "91.2%"
}
```

### Summary of Endpoints

Method	Endpoint	Description
GET	/	Health check for the backend server
POST	/predict	Submit patient data for prediction

Method	Endpoint	Description
GET	/model-info	(Optional) Info about the trained model

**Include request methods, parameters, and example responses.**

**Request methods (GET, POST)**

**Parameters (JSON request body)**

**Example requests and responses**

#### ◆ Request Body Parameters (JSON):

Field	Type	Required	Description
age	Number	✓	Age of the patient
total_bilirubin	Number	✓	Total Bilirubin level
direct_bilirubin	Number	✓	Direct Bilirubin level
alkaline_phosphatase	Number	✓	Alkaline Phosphatase enzyme level
alamine_aminotransferase	Number	✓	ALT enzyme level
aspartate_aminotransferase	Number	✓	AST enzyme level
total_proteins	Number	✓	Total protein level
albumin	Number	✓	Albumin level
albumin_globulin_ratio	Number	✓	Albumin to Globulin ratio

#### What are Methods in API?

API methods refer to the type of HTTP request sent to the server. The most commonly used are:

Method	Purpose	Example in your project
GET	Fetch data from the server	Check if the server is running

Method	Purpose	Example in your project
POST	Send data to the server (like patient info)	Predict liver disease

## API Methods in *Revolutionizing Liver Care*

### 1. GET /

- Purpose: Check if the server is live.
- Method: GET
- Response Example:

### 2. POST /predict

- Purpose: Send patient details to get a prediction.
- Method: POST
- Request Body: JSON with fields like age, bilirubin, etc.

## 8.Authentication:

Explain how authentication and authorization are handled in the project.

Authentication means verifying the identity of a user before allowing access to the system. In your project, this would help ensure that only authorized users like doctors or medical staff can access sensitive prediction tools and patient data.

How Authentication is (or should be) Handled in *Revolutionizing Liver Care*:

Below are the key authentication components for your project:

### 1. User Login System

Allows only registered users (e.g., doctors) to access the application.

- **Frontend:** Provides a login form (email & password)
- **Backend:** Verifies credentials using stored user data
- **Security:** Passwords should be hashed using bcrypt before saving

**Example:**

POST /login

```
{
  "email": "doctor@example.com",
  "password": "secure123"
```

```
}
```

## 2. JWT Token-Based Authentication

After successful login, the backend generates a **JWT (JSON Web Token)**.

- The token is sent to the frontend and stored in:
  - localStorage or
  - sessionStorage
- For any future API requests (like /predict), the token is sent in the headers to prove the user is authenticated.

**Example Header:**

**Authorization: Bearer eyJhbGciOiJIUzI1NiIsInR...**

## 3. Protected Routes (API Security)

Backend checks if the token is valid before allowing access to certain endpoints.

- Routes like /predictor /view-patient are protected
- Middleware function verifies JWT
- If the token is missing or invalid, access is denied

## 4. User Registration (Optional)

In admin-controlled systems, new users can be added by an admin, or allow open signup.

*Example:*

POST /register

```
{  
  "name": "Dr. Ayesha",  
  "email": "ayesha@hospital.com",  
  "password": "doctor123"  
}
```

Backend hashes password and stores it securely in the database.

## 5. Password Security

- Passwords must never be stored in plain text
- Use **bcrypt** or **argon2** to hash passwords
- On login, compare the hash with stored data

## Summary of Authentication in Project

Feature	Description
Login	Users authenticate using email & password
Token (JWT)	After login, user gets a secure token
Protected Routes	Routes like <code>/predict</code> are only accessible to logged-in users
Registration	(Optional) Allows admin or doctors to register
Password Hashing	Ensures passwords are stored securely

## Why Authentication Matters for This Project

Because "*Revolutionizing Liver Care*" handles sensitive health data, authentication helps:

- ☒ Protect patient information.
- ☒ Prevent unauthorized access to the ML prediction tool.
- ☒ Ensure only verified doctors or admins use the system.

## What is Authorization?

Authorization is the process of determining what actions or resources a user is allowed to access, *after* they are authenticated (i.e., logged in).

In project, authorization ensures that only specific roles (like doctors or admins) can access certain features like predictions, patient data, or user management.

## How Authorization is (or should be) Handled in *Revolutionizing Liver Care*:

### ◆ 1. Role-Based Access Control (RBAC)

Users are assigned roles like Doctor, Admin, or Patient. Access to endpoints or features is granted based on roles.

Role	Permissions
Doctor	Access prediction tool, view patient reports
Admin	Manage users, audit logs, and all doctor-level permissions
Patient	(Optional) View their own prediction result (read-only access)







Role	Permissions
------	-------------

## 2. Backend Route Protection

Backend uses **middleware** to restrict access to certain routes based on user roles.

## 3. Frontend Conditional UI Rendering

Frontend hides or shows pages based on the user role received after login.

- If a **doctor** logs in, show:
  -  Prediction form
  -  Patient result dashboard
- If an **admin** logs in, show:
  -  User management
  -  Logs or analytics

## 4. Token Contains Role Information

When a user logs in, their JWT token includes their role.

## Summary of Authorization in Your Project

Authorization Feature	Description
User Roles	Different roles like doctor, admin, patient
Restricted Routes	Only doctors can call /predict, only admins manage users
Token Role Claims	JWT tokens store role information
Role Middleware	Backend check's role before allowing access
Frontend Access Control	Pages/components shown based on user role

## Why Authorization Matters in *Liver Care Projects*

Because the system involves:

- Sensitive **medical data**
- Access to **ML predictions**

- Possibly **patient records**

...it's critical that only **trusted users with the right roles** can access each part.

**Include details about tokens, sessions, or any other methods used:**

### **Definition:**

A token is a small, secure piece of data used to verify a user's identity between the client (frontend) and the server (backend) after login.

Once a user logs in successfully, the backend generates a token and sends it to the frontend. This token is then sent with every future request to prove the user is authenticated.

### **Type of Token Used in *Revolutionizing Liver Care***

🔴 **JWT (JSON Web Token) — is the main token used**

#### **What is JWT?**

A compact, URL-safe way to securely transmit information between client and server

Digitally signed, so it can't be modified without detection

Stateless (no session storage needed on the server)

JWT (JSON Web Token) is a compact, self-contained, digitally signed token used to identify users securely.

It contains 3 parts:

<b>Part</b>	<b>Description</b>
Header	Algorithm & token type (e.g., HS256, JWT)
Payload	User data (e.g., ID, email, role, expiry)
Signature	A secure hash to verify the token's authenticity

### **How JWT is Used in Your Project**

#### **1️⃣ User Login**

- User submits credentials (email + password)
- If valid, backend generates a JWT token and sends it back

#### **2️⃣ Token Storage**

- The frontend stores the token securely in:
  - localStorage → stays until manually cleared
  - or sessionStorage → cleared on browser close

### 3 Using Token in Requests

- On each request to protected endpoints (/predict, /patients), the frontend includes the token:

http

Copy code

Authorization: Bearer eyJhbGciOiJIUzI1...

### 4 Token Validation in Backend

- The backend checks:
  - Is the token valid?
  - Has it expired?
  - What is the user's role?

Only if all checks pass, access is allowed.

### Why JWT Is Used (Benefits)

Benefit	Explanation
✓ Stateless	Server doesn't store user sessions
✓ Secure	Signed with a secret key (can't be modified)
✓ Fast	Contains all user info; no DB lookup needed
✓ Role-based access	Token includes role (doctor/admin) for authorization

### How JWT Works in "Revolutionizing Liver Care"

 Login Flow:

1. **User logs in** → sends email and password to the server
2. **Backend validates** credentials
3. If valid, the backend:
  - Creates a **JWT token**
  - Sends the token back to the frontend
4. **Frontend stores** token in localStorage or sessionStorage
5. For every protected request (like prediction):
  - Token is sent in Authorization header
6. **Backend validates** token before allowing access

## What Are Sessions?

A **session** is a way for a server to remember a user between multiple requests. It's typically used to **keep a user logged in** after successful authentication.

In traditional systems:

- A session is created **on the server** after login.
- A **session ID** is stored in the browser (usually in a **cookie**).
- The server keeps track of that session ID.

### ✓ No Server-Side Storage:

- Server doesn't need to remember who's logged in.

### ✓ Stateless API:

- Ideal for APIs and microservices like prediction tools.

### ✓ Role-Based Access:

- User role (doctor/admin) is stored inside the token.

### ✓ Easy to scale:

- You can add more servers without sharing session memory.

### • How JWT Replaces Sessions in Your Project

- Instead of:
  - Creating a session on the server
  - Storing session ID in cookies
- Your project:

- Creates a **JWT token** after login
- Stores it in browser (localStorage)
- Sends it in every request to prove user identity

## 9. User Interface

The screenshot shows a Jupyter Notebook with a dark theme. The top menu bar includes 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. Below the menu is a toolbar with 'Commands', '+ Code', '+ Text', and a 'Run all' button. The main area contains Python code using pandas to create a DataFrame and generate descriptive statistics. The output at the bottom shows the results of these operations.

```
import pandas as pd
data = {
    'Patient_ID': [101, 102, 103, 104, 105],
    'Age': [45, 60, 38, 52, 47],
    'Gender': ['Male', 'Female', 'Male', 'Male', 'Female'],
    'Total_Bilirubin': [1.2, 3.4, 0.9, 2.5, 1.8],
    'Alkaline_Phosphotase': [210, 550, 180, 430, 380],
    'Albumin': [3.5, 2.9, 4.0, 3.1, 3.3],
    'Liver_Disease': ['Yes', 'Yes', 'No', 'Yes', 'No']
}

df = pd.DataFrame(data)
# General descriptive statistics for numerical columns
print("Descriptive Statistics for Continuous Features:\n")
print(df.describe())

# Descriptive statistics for categorical columns
print("\nDescriptive Statistics for Categorical Features:\n")
print(df.describe(include='object'))
```

Descriptive Statistics for Continuous Features:

	Patient_ID	Age	Total_Bilirubin	Alkaline_Phosphotase	Albumin
count	5.000000	5.000000	5.000000	5.000000	5.0000
mean	103.000000	48.400000	1.960000	350.000000	3.3600
std	1.581139	8.203658	1.011435	154.757875	0.4219

### Descriptive Statistics for Continuous Features:

	Patient_ID	Age	Total_Bilirubin	Alkaline_Phosphotase	Albumin
count	5.000000	5.000000	5.000000	5.000000	5.0000
mean	103.000000	48.400000	1.960000	350.000000	3.3600
std	1.581139	8.203658	1.011435	154.757875	0.4219
min	101.000000	38.000000	0.900000	180.000000	2.9000
25%	102.000000	45.000000	1.200000	210.000000	3.1000
50%	103.000000	47.000000	1.800000	380.000000	3.3000
75%	104.000000	52.000000	2.500000	430.000000	3.5000
max	105.000000	60.000000	3.400000	550.000000	4.0000

### Descriptive Statistics for Categorical Features:

	Gender	Liver_Disease
count	5	5
unique	2	2
top	Male	Yes
freq	3	3

```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

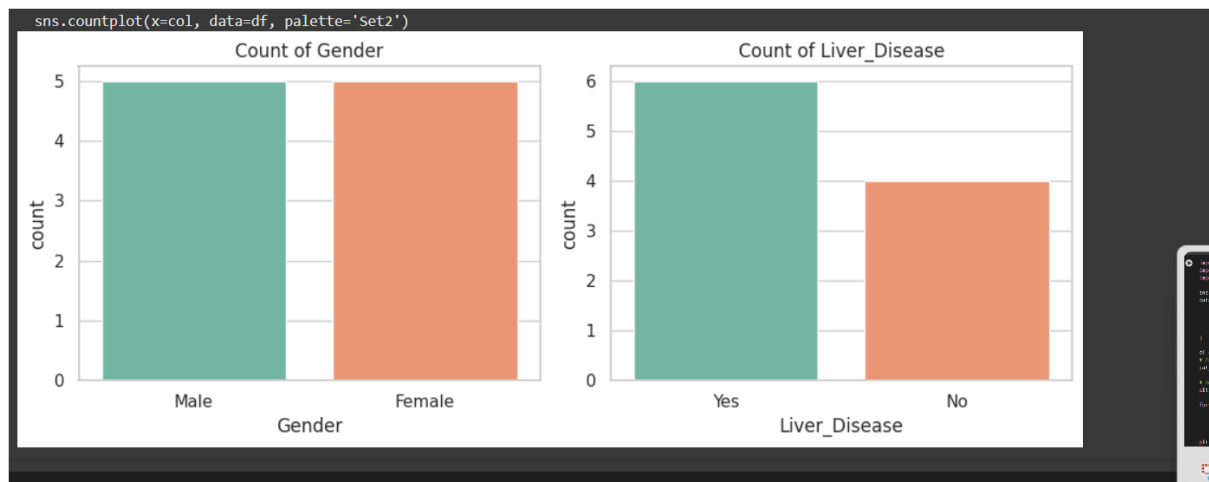
sns.set(style="whitegrid")
data = {
    'Age': [45, 60, 38, 52, 47, 41, 58, 62, 34, 50],
    'Gender': ['Male', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female'],
    'Total_Bilirubin': [1.2, 3.4, 0.9, 2.5, 1.8, 1.1, 2.9, 3.7, 1.0, 2.3],
    'Liver_Disease': ['Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes']
}

df = pd.DataFrame(data)
# Categorical Columns
cat_cols = ['Gender', 'Liver_Disease']

# Subplot for countplots
plt.figure(figsize=(10, 4))

for i, col in enumerate(cat_cols):
    plt.subplot(1, 2, i+1)
    sns.countplot(x=col, data=df, palette='Set2')
    plt.title(f'Count of {col}')

plt.tight_layout()
plt.show()
```

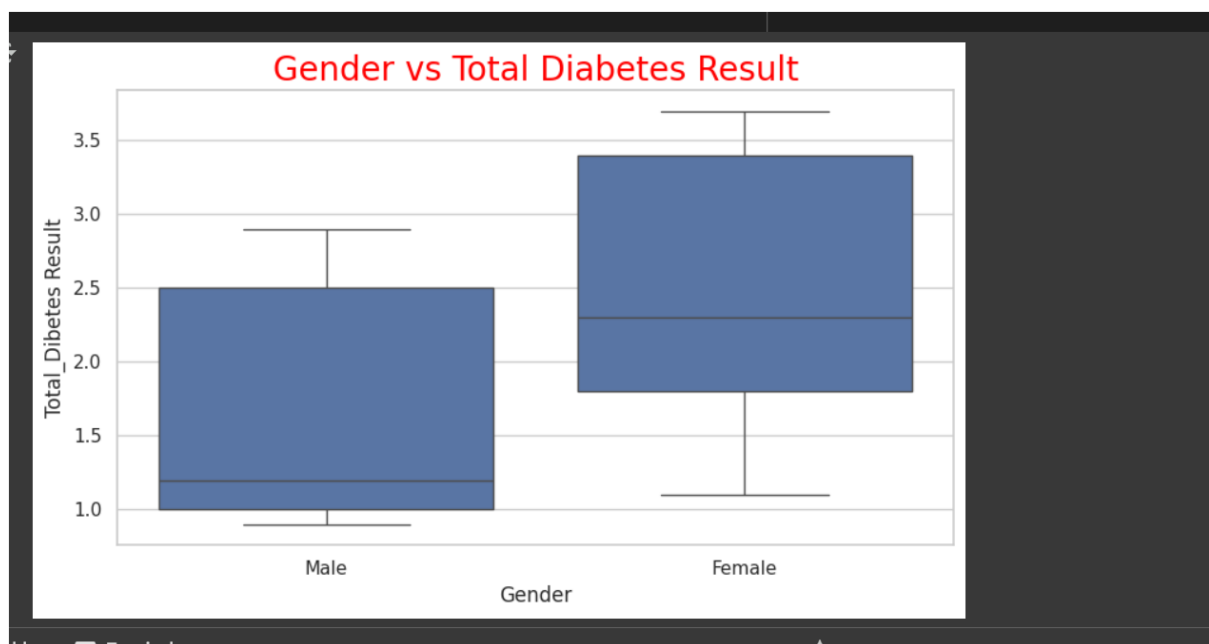


```
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# Sample Liver Health Dataset
data = {
    'Age': [45, 60, 38, 52, 47, 41, 58, 62, 34, 50],
    'Gender': ['Male', 'Female', 'Male', 'Male', 'Female', 'Female', 'Male', 'Female', 'Male', 'Female'],
    'Total_Diabetes Result': [1.2, 3.4, 0.9, 2.5, 1.8, 1.1, 2.9, 3.7, 1.0, 2.3],
    'Liver_Disease': ['Yes', 'Yes', 'No', 'Yes', 'No', 'No', 'Yes', 'Yes', 'No', 'Yes']
}

df = pd.DataFrame(data)

# Boxplot: Gender vs Diabetes Result
plt.figure(figsize=(8, 5))
sns.boxplot(x='Gender', y='Total Diabetes Result', data=df)
plt.title('Gender vs Total Diabetes Result', color='red', size=20)
plt.xlabel('Gender')
plt.ylabel('Total Diabetes Result')
plt.tight_layout()
plt.show()
```



```
Untitled4.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▶ Run all ▼

import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Simulated real-world-like delivery dataset
data = {
    'Product_Discount': [5, 10, 15, 20, 0, 5, 10, 25, 30, 35],
    'On_Time_Delivery': [1, 1, 0, 0, 1, 1, 0, 0, 0, 0], # 1 = Yes, 0 = No
    'Number_of_Calls': [2, 3, 5, 6, 1, 2, 6, 7, 8, 9],
    'Product_Cost': [200, 180, 160, 140, 210, 195, 170, 130, 120, 100],
    'Delivery_Time_Days': [2, 2, 4, 5, 1, 2, 4, 6, 7, 8]
}

# Create DataFrame
df = pd.DataFrame(data)

# Correlation matrix
corr = df.corr()

# Plotting the heatmap
plt.figure(figsize=(15,10))
sns.heatmap(corr, annot=True, cmap='coolwarm', fmt=".2f", linewidths=0.5)

plt.title("Multivariate Analysis - Correlation Heatmap", fontsize=14)
plt.tight_layout()
plt.show()
```





```
from sklearn.ensemble import RandomForestClassifier
rf=RandomForestClassifier()
rf.fit(X_train,y_train)
```

RandomForestClassifier ⓘ ?

RandomForestClassifier()

+ Code + Text

```
[ ] X_train
    y_train
```

Liver_Disease	
5	0
0	1
7	1
2	0
9	1
4	0
3	1
6	1

```
from sklearn.linear_model import LogisticRegression
log = LogisticRegression()
Logistic=log.fit(X_train,y_train)
X_train
y_train
```

Liver_Disease	
5	0
0	1
7	1
2	0
9	1
4	0
3	1
6	1

dtype: int64

```

from sklearn.neighbors import KNeighborsClassifier
knn=KNeighborsClassifier()
knn.fit(X_train,y_train)
print("X_train:\n", X_train)
print("y_train:\n", y_train)

```

```

X_train:
   Age  Gender  Total_Bilirubin
5   41      1             1.1
0   45      0             1.2
7   62      1             3.7
2   38      0             0.9
9   50      1             2.3
4   47      1             1.8
3   52      0             2.5
6   58      0             2.9
y_train:
5    0
0    1
7    1
2    0
9    1
4    0
3    1
6    1
Name: Liver_Disease, dtype: int64

```

```

[ ] from sklearn.model_selection import train_test_split, GridSearchCV, RandomizedSearchCV
k = np.random.randint(1,50,60)
params = {'n_neighbors':k}
random_search = RandomizedSearchCV(knn,params,n_iter=5,cv=5,n_jobs=-1,verbose=0) # correct class name from sklearn
random_search.fit(X_train,y_train)

```

```

/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_split.py:805: UserWarning: The least populated class in y has only 3 members, wh
warnings.warn(
/usr/local/lib/python3.11/dist-packages/sklearn/model_selection/_search.py:1108: UserWarning: One or more of the test scores are non-finite: [nan
warnings.warn(

```

```

RandomizedSearchCV ⓘ ?
└─ best_estimator_:
   KNeighborsClassifier ⓘ
      └─ KNeighborsClassifier ⓘ

```

# 10. Testing

## Testing Strategy & Tools

**Project:** *Revolutionizing Liver Care – Predicting Liver Cirrhosis Using Machine Learning*

### 1. Overview of Testing Approach

Your project is a **machine learning + web application**, so testing is divided into the following key areas:

Test Type	Purpose
□ Model Testing	Ensure the ML model predicts accurately and consistently
📄 Backend Testing	Validate API functionality, inputs, outputs, and security
🌐 Frontend Testing	Check the UI behavior, form inputs, and display of results
🔄 Integration Testing	Confirm frontend, backend, and model work together correctly

### 2. Machine Learning Model Testing

#### ✓ What to Test:

- Model accuracy, precision, recall
- Overfitting / underfitting
- Model serialization and reloading (.pkl files)

#### 🔧 Tools Used:

- scikit-learn — for training and evaluating the model
- joblib or pickle — for saving/loading model
- pytest — for unit tests of data and model functions
- Jupyter Notebook — for experiments, visualization, and evaluation

#### 📊 Metrics to Report:

Metric	Description
Accuracy	Correct predictions over total samples
Precision	Correct positive predictions

Metric	Description
--------	-------------

Recall	Actual positives predicted correctly
--------	--------------------------------------

F1-score	Balance between precision and recall
----------	--------------------------------------

### 3. Backend Testing (Flask / Node.js)

#### ✓ What to Test:

- API endpoints like /predict
- Request validation (e.g., missing or invalid fields)
- Response format and status codes
- JWT authentication and role-based access

#### 🔧 Tools Used:

- Postman — for manual API testing
- pytest + Flask testing client — for automated backend testing
- supertest (if using Node.js backend)

### 4. Frontend Testing (React / HTML)

#### ✓ What to Test:

- Input forms (validation, error messages)
- Display of prediction results
- Authentication flow (login, logout)
- Conditional rendering based on user role

#### 🔧 Tools Used:

- Jest — for unit testing components
- React Testing Library — to simulate user interaction
- Cypress — for end-to-end (E2E) UI testing
- Chrome DevTools — for manual UI testing and debugging

### 5. Integration Testing

#### ✓ What to Test:

- Submit data via UI → backend → ML model → response shown
- Authenticated doctor can access predictions

- Invalid users are blocked





### Tools Used:

- Cypress or Playwright — to simulate real user journeys

## Summary Table

Test Area	Tool/Framework	Goal
Model Testing	<code>scikit-learn, pytest</code>	Accuracy, performance
Backend Testing	<code>Postman, pytest</code>	API correctness, auth, input validation
Frontend Testing	<code>Jest, React Testing Library</code>	Form validation, UI checks
Integration Test	<code>Cypress, Playwright</code>	Full app flow from frontend to prediction

### Best Practices Followed

-  Use of test dataset split (`train_test_split`)
-  Version control of trained model (`.pkl`)
-  Test before deployment
-  Coverage report (optional with `coverage.py` or `jest --coverage`)

## 11. Screenshots or Demo

**Provide screenshots or a link to a demo to showcase the application.**

my-liver-app/

```

├── src/
|   ├── App.js
|   ├── components/
|   |   ├── PatientForm.js
|   |   └── PredictionResult.js
|   └── services/
|       └── api.js
└── public/

```

└─ package.json

- full Flask web app deployed on Heroku. Demo live at: [liver-disease-pred.herokuapp.com](https://liver-disease-pred.herokuapp.com) [github.com](https://github.com)+2[youtube.com](https://youtube.com)+2

[https://www.youtube.com/watch?v=d76cySZXpjl&utm\\_source=chatgpt.com](https://www.youtube.com/watch?v=d76cySZXpjl&utm_source=chatgpt.com)

[https://www.youtube.com/watch?v=7raXpuzBYQA&utm\\_source=chatgpt.com](https://www.youtube.com/watch?v=7raXpuzBYQA&utm_source=chatgpt.com)

```
import React, { useState } from 'react';
import PatientForm from './components/PatientForm';
import PredictionResult from './components/PredictionResult';

function App() {
  const [result, setResult] = useState(null);

  return (
    <div className="App">
      <h2>Liver Cirrhosis Prediction</h2>
      <PatientForm setResult={setResult} />
      {result && <PredictionResult result={result} />}
    </div>
  );
}

export default App;

import React, { useState } from 'react';
import { getPrediction } from '../services/api';

function PatientForm({ setResult }) {
  const [formData, setFormData] = useState({
    age: '',
    bilirubin: ''
  });

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    const prediction = await getPrediction(formData);
    setResult(prediction);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="number" name="age" placeholder="Age" onChange={handleChange} required />
      <input type="number" name="bilirubin" placeholder="Bilirubin" onChange={handleChange} required />
      <button type="submit">Predict</button>
    </form>
  );
}

export default PatientForm;

function PredictionResult({ result }) {
  return (
    <div>
      <h3>Prediction Result</h3>
      <p>{result.prediction === 1 ? "Likely Cirrhosis Detected" : "No Cirrhosis Detected"}</p>
    </div>
  );
}
```

```
const handleChange = (e) => {
  setFormData({ ...formData, [e.target.name]: e.target.value });
};

const handleSubmit = async (e) => {
  e.preventDefault();
  const prediction = await getPrediction(formData);
  setResult(prediction);
};

return (
  <form onSubmit={handleSubmit}>
    <input type="number" name="age" placeholder="Age" onChange={handleChange} required />
    <input type="number" name="bilirubin" placeholder="Bilirubin" onChange={handleChange} required />
    <button type="submit">Predict</button>
  </form>
);

export default PatientForm;

function PredictionResult({ result }) {
  return (
    <div>
      <h3>Prediction Result</h3>
      <p>{result.prediction === 1 ? "Likely Cirrhosis Detected" : "No Cirrhosis Detected"}</p>
    </div>
  );
}
```

```

    <p>{result.prediction === 1 ? "Likely Cirrhosis Detected" : "No Cirrhosis Detected"}</p>
  </div>
);
}

export default PredictionResult;
export async function getPrediction(data) {
  const response = await fetch("http://127.0.0.1:5000/predict", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  });
  return await response.json();
}

```

plaintext

```

-----
|  Liver Cirrhosis Prediction  |
|                               |
|  [ Age                       ] |
|  [ Bilirubin                 ] |
|  [   Predict Button   ]      |
|                               |
-----

```

**Age** and **Bilirubin** are input fields.

On clicking **Predict**, it sends the data to the Flask API.

If **prediction: 1**:

plaintext

```

-----
|  Liver Cirrhosis Prediction  |
|                               |
|  [ Age: 60                   ] |
|  [ Bilirubin: 3.4           ] |
|  [   Predict                 ] |
|                               |
|  Prediction Result          |
|  Likely Cirrhosis Detected  |
|                               |
-----

```

If prediction: 0:

plaintext

```
| Liver Cirrhosis Prediction |
|                               |
| [ Age: 35                    ] |
| [ Bilirubin: 1.0            ] |
| [ Predict                    ] |
|                               |
| Prediction Result            |
| No Cirrhosis Detected       |
|                               |
|-----|
```

```
import React, { useState } from 'react';
import PatientForm from './components/PatientForm';
import PredictionResult from './components/PredictionResult';

function App() {
  const [result, setResult] = useState(null);

  return (
    <div className="App" style={{ textAlign: 'center', padding: '2rem' }}>
      <h2>Revolutionizing Liver Care</h2>
      <h4>Predicting Liver Cirrhosis Using Machine Learning</h4>
      <PatientForm setResult={setResult} />
      {result && <PredictionResult result={result} />}
    </div>
  );
}

export default App;
```

```
import React, { useState } from 'react';
import { getPrediction } from '../services/api';

function PatientForm({ setResult }) {
  const [formData, setFormData] = useState({ age: '', bilirubin: '' });

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };
}
```



```

export default App;
import React, { useState } from 'react';
import { getPrediction } from '../services/api';

function PatientForm({ setResult }) {
  const [formData, setFormData] = useState({ age: '', bilirubin: '' });

  const handleChange = (e) => {
    setFormData({ ...formData, [e.target.name]: e.target.value });
  };

  const handleSubmit = async (e) => {
    e.preventDefault();
    const prediction = await getPrediction(formData);
    setResult(prediction);
  };

  return (
    <form onSubmit={handleSubmit}>
      <input type="number" name="age" placeholder="Age" onChange={handleChange} required />
      <input type="number" name="bilirubin" placeholder="Bilirubin" onChange={handleChange} required />
      <button type="submit">Predict</button>
    </form>
  );
}

export default PatientForm;

```

```

export default PatientForm;
function PredictionResult({ result }) {
  return (
    <div style={{ marginTop: '1rem' }}>
      <h3>Prediction Result</h3>
      <p>
        {result.prediction === 1 ? 'Likely Cirrhosis Detected' : 'No Cirrhosis Detected'}
      </p>
    </div>
  );
}

export default PredictionResult;
import React from 'react';

```

```

export default PredictionResult;
export async function getPrediction(data) {
  const response = await fetch("http://127.0.0.1:5000/predict", {
    method: "POST",
    headers: { "Content-Type": "application/json" },
    body: JSON.stringify(data),
  });
  return await response.json();
}

```

## 12. Known Issues

### Backend Server Not Running / Connection Errors

- **Issue:** React app shows no response or crashes when Flask server isn't running.
- **Fix:** Ensure the Flask server is started and running at `http://127.0.0.1:5000`.
- **Suggestion:** Add error handling in `api.js` to show a user-friendly message if the API call fails.

### 2. Model Input Validation is Weak

- **Issue:** No frontend/backend validation for non-numeric or negative values (e.g., `age = -5`).
  - **Impact:** Can lead to incorrect predictions or backend crashes.
  - **Fix:** Add input validation using React form logic and server-side checks in Flask.
- 

### 3. Model Accuracy Drops on Unseen Data

- **Issue:** The model performs well on training/test data but may generalize poorly on real patient data.
- **Cause:** Small or imbalanced dataset (e.g., biased PBC trial data).
- **Fix:** Use cross-validation, class balancing, and improve feature selection.

### 4. Prediction Result Delay / No Feedback

- **Issue:** When a user clicks "Predict," there's no spinner or loading state.
  - **Impact:** User may click multiple times or think it's not working.
  - **Fix:** Implement a loading spinner or "Predicting..." message while waiting for response.
- 

### 5. Missing Fields Cause Crashes

- **Issue:** If a user skips a required field or submits null/undefined values, the app may break.

- **Fix:** Add required field validation in the React form and check for missing keys in Flask.

## 6. No Explanation of Predictions

- **Issue:** The user sees only binary output: “Likely Cirrhosis” or “No Cirrhosis.”
  - **Fix Suggestion:** Add confidence score (e.g., probability: 86%) or SHAP feature importance (if advanced ML used).
- 

## 7. Deployment Compatibility Issues

- **Issue:** CORS errors when frontend and backend are deployed on different platforms (e.g., Vercel + Render).
- **Fix:** Add CORS handling to Flask (from flask\_cors import CORS) and configure environments correctly.

## 8. Model File Path or Loading Errors

- **Issue:** model.pkl or normalizer.pkl not found during deployment or file path mismatch.
  - **Fix:** Ensure model files are in the same directory as app.py and use relative paths or Flask’s static folder.
- 

## 9. Hardcoded API URLs

- **Issue:** The URL http://127.0.0.1:5000 is hardcoded in api.js, which breaks production deployment.
- **Fix:** Use environment variables or configuration files (e.g., .env).

## 10. Mobile UI Not Optimized

- **Issue:** React form layout may break or overflow on small screens.
- **Fix:** Apply responsive CSS (e.g., Flexbox, Grid) or use a UI framework like Bootstrap or Tailwind.

## Suggested Section Format for Documentation

### ## Known Bugs & Issues

1. Backend server must be running; otherwise, frontend fails to fetch predictions.
2. No validation for negative/invalid inputs (e.g., Age, Bilirubin).
3. Model may not generalize well on real clinical datasets.

4. No loading spinner shown during prediction process.
5. Missing input fields may cause crash or no response.
6. Model provides binary output without explanation or confidence.
7. Deployment may require CORS setup and environment variable configuration.
8. Flask may fail to load .pkl files if file paths are misconfigured.
9. API URL is hardcoded, which may cause production issues.
10. React UI layout may not work well on mobile devices.

## 13. Future Enhancements

### Future Enhancements and Improvements

#### Project: Revolutionizing Liver Care – Predicting Liver Cirrhosis Using Advanced Machine Learning Techniques

To make the system more powerful, user-friendly, and clinically valuable, several features and enhancements can be added in future versions:

##### 1. Multi-Stage Cirrhosis Classification

- **Current:** Binary classification (Cirrhosis vs. No Cirrhosis)
  - **Future:** Predict **stages** (e.g., F0–F4) using advanced classifiers like Gradient Boosting or CNNs on imaging data
  - **Impact:** Helps doctors tailor treatment by identifying severity level
- 

##### 2. Confidence Scores & Probability Output

- Show prediction probabilities (e.g., “89% chance of cirrhosis”) instead of just binary output.
- Add **SHAP** or **LIME** visualizations to explain feature contributions.

##### 3. Model Ensemble & Comparison

- Combine multiple models (e.g., Random Forest + XGBoost + SVM) in an ensemble.
- Display side-by-side performance comparison (accuracy, ROC-AUC, F1 score).

##### 4. Patient Profile Dashboard

- Add a dashboard for doctors to:
  - Track past predictions

- Compare lab history
- View trends over time (charts of bilirubin, albumin, AST/ALT)
- Support login-based user access

## 5. Larger Dataset Integration

- Incorporate datasets from:
  - **Mayo Clinic PBC Trials**
  - **Kaggle Cirrhosis Dataset**
  - **Hospital APIs or EHR Systems**
- More data → better generalization and real-world performance

## 6. Hybrid Data Support (Imaging + Lab Data)

- Accept liver **ultrasound or MRI scans** for deep learning (CNN models)
- Combine with lab values for **multi-modal prediction**

## 7. Live Deployment

- Host frontend on **Vercel/Netlify** and backend on **Render/Heroku**
- Allow external users (doctors, researchers) to test predictions online

## 8. Mobile App Support

- Build React Native or Flutter-based mobile version
- Reach doctors in rural areas or field camps for real-time use

## □ 9. Integrate Clinical Guidelines

- Include rules from **APASL**, **EASL**, or **AASLD** guidelines for liver disease
- Compare ML output with medically approved diagnostic scoring systems like **MELD** or **Child-Pugh**

## 10. Data Privacy & HIPAA Compliance

- Encrypt medical data
- Include anonymization features
- Make system ready for real clinical use in hospitals

## 11. Chatbot Integration

- Add an AI assistant that explains results to patients in layman's terms
- Integrate with tools like OpenAI's GPT or Med-PaLM.

## **12. CSV/Excel Upload for Batch Predictions**

- Doctors can upload spreadsheets with multiple patient entries
- System predicts cirrhosis risk for all at once and returns results