Full Stack Development with MERN

Project Documentation format

1. Introduction

Project Title:

Online Payments Fraud Detection System Using MERN Stack

Team Members:

S.No	Name	Role
1	Annem Manasa	Team Leader / Backend Developer
2	Balagani Rishika	Frontend Developer
3	Dhulipalla Mrudula	Database & API Integration
4	Garikapati Tulasi	Testing, Deployment & Documentation

2. Project Overview

2.1 Purpose

The primary purpose of this project is to build a secure, scalable, and intelligent web application capable of detecting fraudulent online payment transactions using Machine Learning algorithms integrated into a MERN stack application.
With the rapid digitalization of financial services, online payment fraud has become a major concern. Traditional rule-based systems are not sufficient to detect complex fraud patterns. This project aims to:

- Automate fraud detection
- Reduce financial losses
- Improve transaction security
- Provide real-time prediction results
- Demonstrate full-stack development skills

This system acts as a smart assistant for financial institutions to identify suspicious transactions efficiently.

2.2Features

- User registration and secure login
- Password encryption using bcrypt
- JWT-based authentication system
- Transaction data input form
- Real-time fraud prediction using ML model
- Confidence score display
- MongoDB transaction storage

- User transaction history dashboard
- Error handling and validation
- Fully responsive UI
- Deployment-ready architecture

3. Architecture

The project follows a three-tier architecture:

1. Presentation Layer (Frontend)
2. Application Layer (Backend)
3. Data Layer (Database)

Frontend: The frontend is built using React.js with a component-based architecture. Each UI part is designed as a reusable component to ensure maintainability and scalability.
Key Features:

- React Hooks (useState, useEffect)
- React Router for navigation
- Axios for API calls
- Form validation
- Dynamic result rendering
- Protected routes for authenticated users

  Workflow:

  User → Input Data → Axios Request → Display Prediction

The UI is designed to be simple, clean, and responsive for better user experience.

Backend: The backend is developed using Node.js and Express.js, providing RESTful APIs for communication between frontend and database.
Responsibilities:

- Handle HTTP requests
- Validate incoming data
- Authenticate users
- Communicate with ML model
- Store transaction logs
- Send prediction results

The backend follows MVC architecture:

- Models → Database schema
- Controllers → Business logic
- Routes → API endpoints

Database: The Online Payments Fraud Detection System uses MongoDB as its database to store user information and transaction records. MongoDB is a NoSQL, document-oriented database that stores data in JSON-like format, making it highly flexible and scalable. It integrates efficiently with the MERN stack and allows seamless interaction with the Node.js backend using Mongoose (ODM).
The database consists mainly of two collections:

- Users Collection – Stores registered user details such as name, email, hashed password, role, and account creation date.
- Transactions Collection – Stores transaction details along with fraud

prediction results, including transaction type, amount, balances,prediction result, confidence score, and timestamp.

Each transaction is linked to a specific user through a reference (userId), forming a one-to-many relationship (one user can have multiple transactions).

The backend interacts with MongoDB to:

- Register and authenticate users
- Store transaction history
- Save fraud prediction results
- Retrieve data for dashboards

MongoDB ensures secure, efficient, and scalable data storage, making it suitable for handling large volumes of financial transaction data in real-time fraud detection systems.

4. Setup Instructions

Prerequisites:

- Node.js (v16+)
- MongoDB (Local or Atlas)
- npm or yarn
- Git
- VS Code
- Postman (for API testing)

Installation: Step-by-step guide to clone, install dependencies, and set up the environment variables.

1. Clone repository
2. Install backend dependencies
3. Install frontend dependencies
4. Configure environment variables
5. Start backend server
6. Start frontend server

Environment variables ensure secure handling of sensitive data such as database connection strings and JWT secrets.

5. Folder Structure

Client:

Organized into:

- components → Reusable UI components
- pages → Main pages (Login, Register, Dashboard)
- services → API integration
- utils → Helper functions
- styles → CSS files

This structure ensures modular development and easier maintenance**.**

Server:

Organized into:

- models → MongoDB schemas
- routes → API route definitions
- controllers → Business logic
- middleware → Authentication & validation
- config → Database configuration

This separation improves code readability and scalability.

6. Running the Application

Commands to start the frontend and backend servers locally.

Frontend:

1.cd client (To Change Directory)

2.npm start in the client directory

Backend:

1. cd server (To Change Directory)
2. npm start in the server directory.

7. API Documentation

The backend exposes REST APIs:

Authentication APIs

- POST /api/auth/register
- POST /api/auth/login

Prediction API

- POST /api/predict

Transaction API

- GET /api/transactions
- DELETE /api/transactions/:id

Each API returns JSON responses with proper HTTP status codes (200, 400, 401, 500).

Error handling is implemented to ensure meaningful messages are returned.

8. Authentication

Authentication is implemented using:

- JWT (JSON Web Token)
- bcrypt for password hashing
- Middleware for protected routes

Workflow:

1. User logs in.
2. Server generates JWT token.
3. Token is stored in frontend.
4. Token is verified before accessing protected APIs.

This ensures secure communication and data protection.

9. User Interface

1. Home Page

- Displays project title and system overview.
- Provides navigation options such as Login and Register.
- Clean and responsive layout.

2. Registration Page

- Form to create a new user account.
- Fields include Name, Email, and Password.
- Input validation and error messages.
- Password stored securely after encryption.

3. Login Page

- Secure authentication using email and password.
- Displays error message for invalid credentials.
- On successful login, user is redirected to Dashboard.

4. Dashboard Page

- Displays welcome message.
- Shows navigation options for Fraud Detection and Transaction History.
- Displays previously saved transactions.

5. Fraud Detection Form

- Input fields for:
  - Transaction Type
  - Amount
  - Old Balance
  - New Balance
- Submit button to trigger prediction.
- Input validation before submission.

6. Prediction Result Page

- Displays:

    - Fraud / Genuine result
    - Confidence score
- Visual indicator (color-coded result display).
- Option to perform another transaction.

7. **Transaction History Page**
    - Displays a table of past transactions.
    - Shows prediction result and timestamp.
    - Sorted by latest transaction first.

10. Testing

The testing strategy for the Online Payments Fraud Detection System was designed to ensure accuracy, security, reliability, and performance of the complete MERN stack application integrated with the Machine Learning model.

The system was tested in multiple stages:

1. Unit Testing

Individual components such as API routes, controllers, and frontend components were tested separately to verify correct functionality.

2. Integration Testing

Integration testing was performed to ensure proper communication between:

- Frontend (React)
- Backend (Node.js + Express)
- Database (MongoDB)
- Machine Learning prediction module

This ensured that data flows correctly from user input to prediction result storage.

3. Functional Testing

Each feature was tested based on functional requirements:

- User registration and login
- Fraud prediction
- Transaction storage
- Dashboard display

4. Security Testing

Security testing ensured:

- Password hashing using bcrypt
- JWT token authentication
- Protected API routes
- Prevention of unauthorized access

5. Performance Testing

The system was tested for:

- Prediction response time
- Server response under multiple requests
- Database query efficiency

The prediction response time was maintained within a few seconds to provide near real-time fraud detection.

6. User Interface Testing

Manual UI testing ensured:

- Responsive design
- Proper navigation
- Clear error messages
- Smooth user experience

Tools Used for Testing

1. Postman

- Used for API testing
- Verified request/response format
- Checked status codes
- Tested authentication tokens

2. Browser Developer Tools

- Debugging frontend issues
- Monitoring network requests
- Checking console errors

3. MongoDB Compass

- Verified database records
- Checked data storage accuracy
- Monitored collection updates

4. Manual Testing

- Simulated real user scenarios
- Tested edge cases and invalid inputs
- Verified prediction results

Testing Outcome

The application successfully passed all testing stages. The system demonstrated:

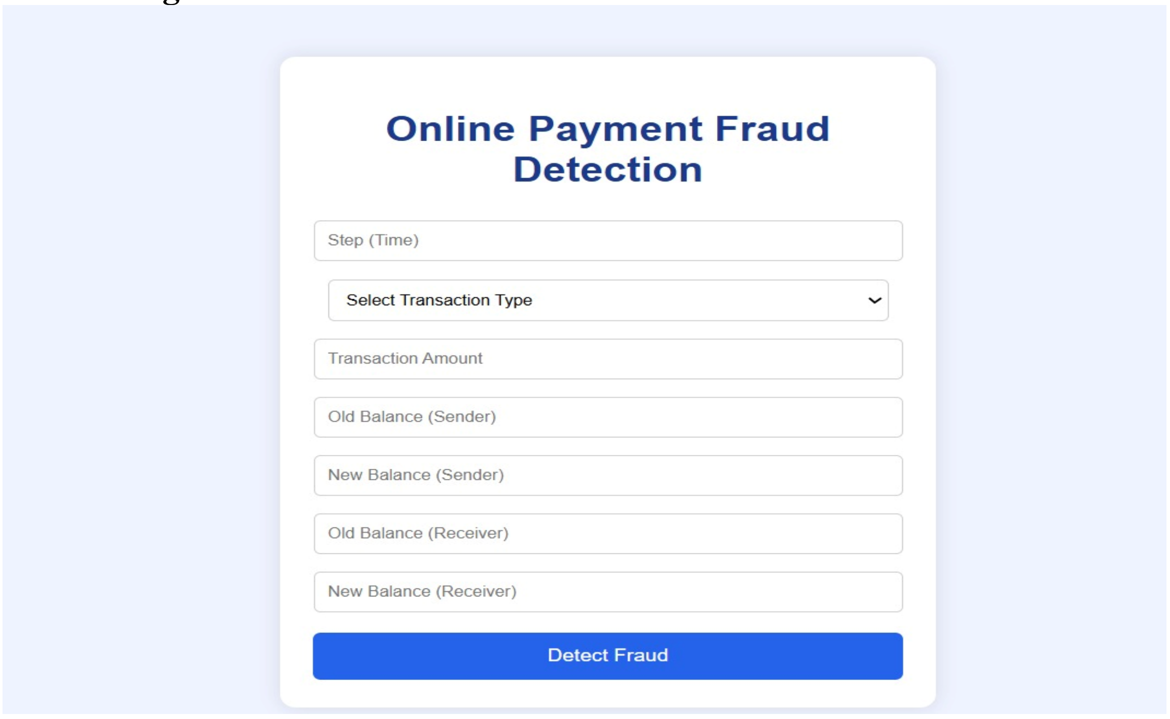
- Accurate fraud prediction
- Secure authentication
- Stable performance

- Reliable database interactions

The testing process ensured that the system is robust, secure, and ready for deployment.

11. Screenshots or Demo

Home Page:


Online Payment Fraud Detection

Step (Time)

Select Transaction Type

Transaction Amount

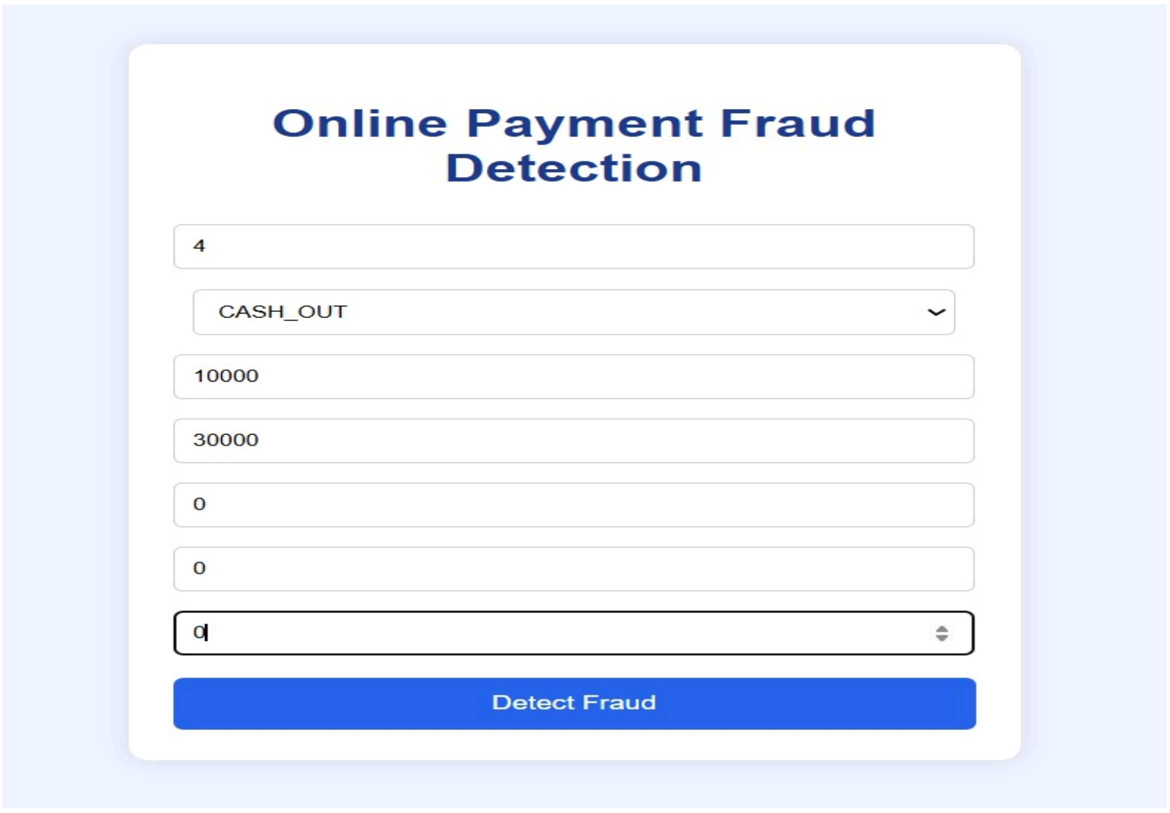
Old Balance (Sender)

New Balance (Sender)

Old Balance (Receiver)

New Balance (Receiver)

Transaction Input Form:


Online Payment Fraud Detection

4

CASH_OUT

10000

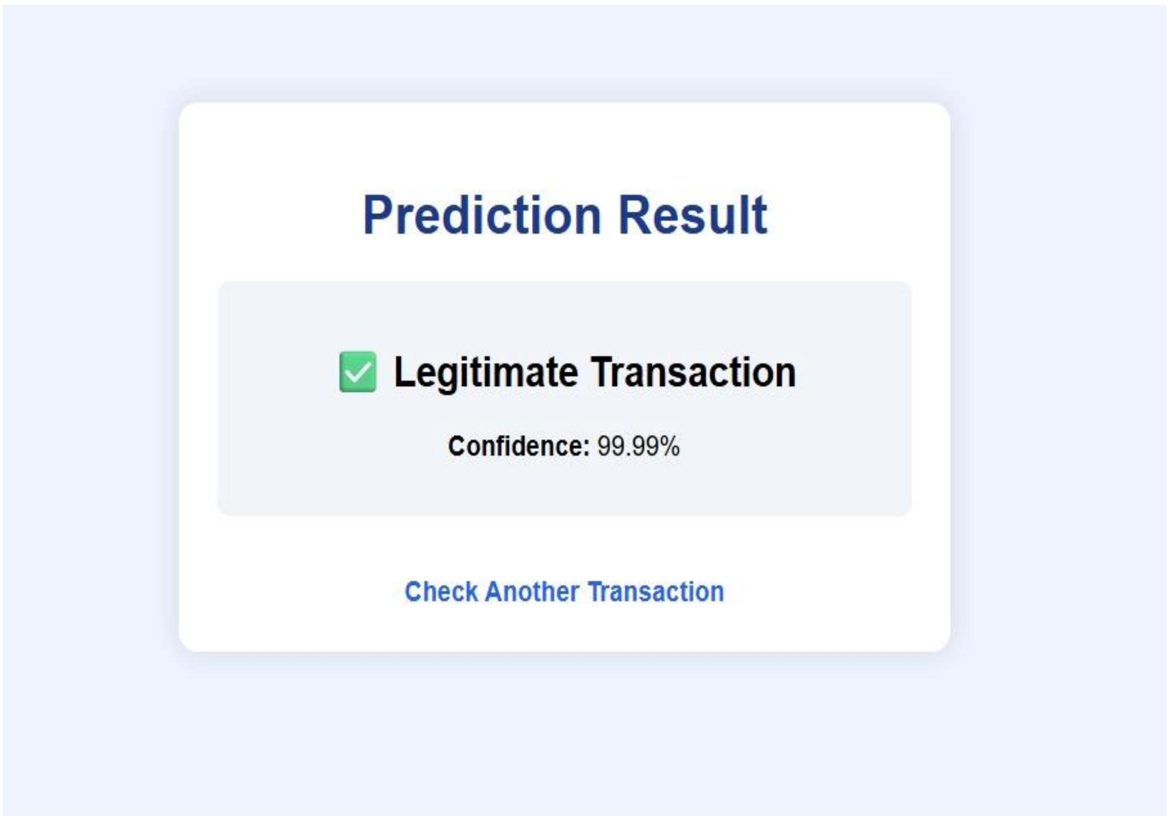
30000

0

0

0

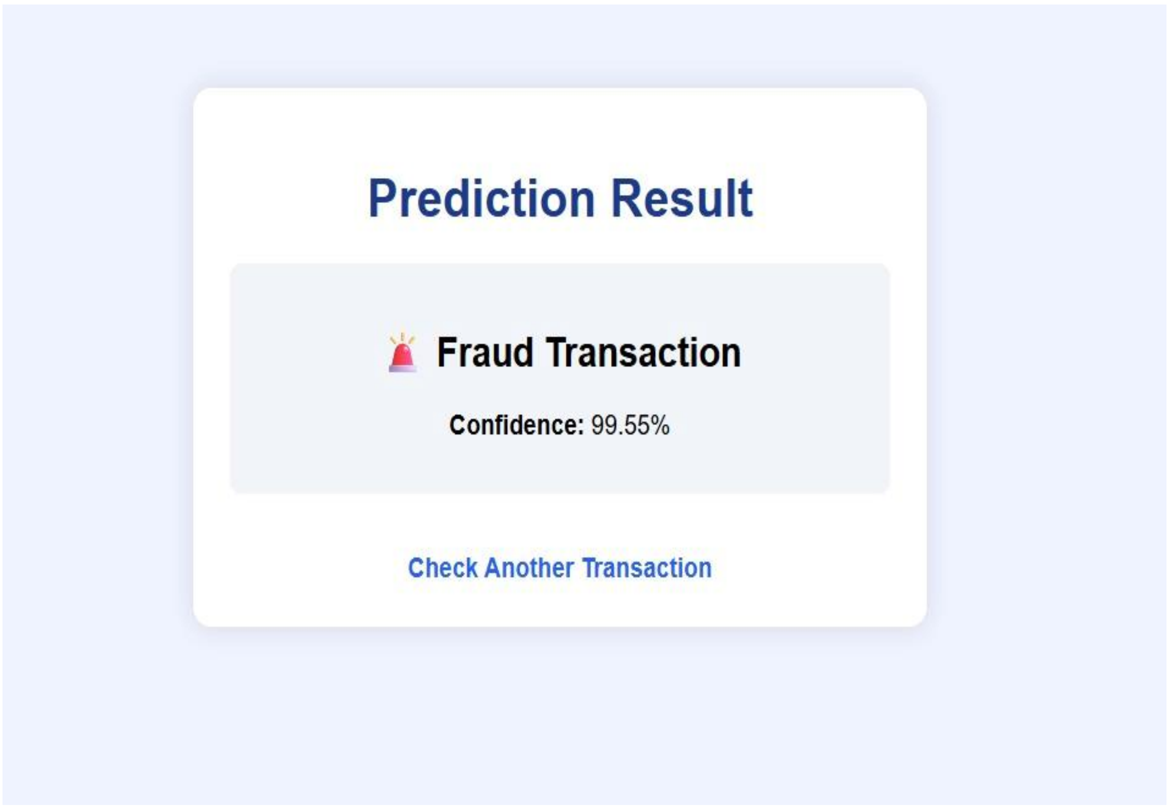
Prediction Results:



Prediction Result

✅ Legitimate Transaction

Confidence: 99.99%



Prediction Result

🚨 Fraud Transaction

Confidence: 99.55%

12. Known Issues

- Model performance depends on training data.
- Requires periodic retraining.
- May generate false positives.
- Heavy traffic may increase response time.
- No real-time banking API integration yet.

13. Future Enhancements

- Integration with real-time banking systems
- Deep Learning-based fraud detection
- Fraud risk scoring dashboard
- Admin panel with analytics
- Email and SMS alert system
- Role-based access control
- Cloud auto-scaling
- Advanced reporting system
- Multi-language support
- AI-based anomaly detection improvements