Full Stack Development with MERN

Project Documentation format

1Project Documentation

---

1. Introduction

Project Title:
Smart Sorting: Transfer Learning for Identifying Rotten Fruits and Vegetables

Team Members:

- Project Lead – AI Model Development & System Design
- Frontend Developer – React Web & Mobile UI Development
- Backend Developer – API & Server Development
- Database Administrator – Database Design & Management
- DevOps Engineer – Deployment & Cloud Infrastructure

---

2. Project Overview

Purpose

The purpose of the Smart Sorting system is to automate the detection of rotten fruits and vegetables using transfer learning–based image classification. The project aims to reduce food waste, improve quality control in food processing plants and supermarkets, and help households monitor food freshness through smart technology.

Features

- User Registration & Login
- Image Capture & Upload (Mobile/Web/Camera Feed)
- AI-Based Fresh vs Rotten Classification

- Real-Time Conveyor Belt Monitoring (Industrial Use)
- Bulk Shipment Scanning (Supermarket Use)
- Smart Refrigerator Monitoring (Home Use)
- Push Notifications & Alerts
- Scan History & Quality Reports
- Admin Panel for Model & User Management

---

3. Architecture

Frontend (React)

The frontend is built using **React JS** for web dashboards and **React Native** for mobile applications.

- Component-based architecture
- REST API integration with backend
- State management using Context API / Redux
- Responsive UI for plant operators, supermarket staff, and home users

Backend (Node.js & Express.js)

The backend is developed using **Node.js** with **Express.js** framework.

- RESTful API design
- Middleware for authentication & validation
- Image upload handling using Multer
- Integration with AI model service (Python-based microservice)
- Notification service integration (Firebase/Email API)

Database (MongoDB)

The system uses **MongoDB** as a NoSQL database.

Collections:

- Users (User credentials, roles)
- ScanHistory (Image path, result, timestamp)
- Reports (Batch summaries, freshness percentage)
- AdminSettings (Model version, configuration)

Backend interacts with MongoDB using Mongoose ORM.

---

4. Setup Instructions

Prerequisites

- Node.js (v16 or above)
- MongoDB (Local or Cloud – MongoDB Atlas)
- Python (for AI model service)
- npm or yarn

Installation

Step 1: Clone the repository

git clone https://github.com/your-repo/smart-sorting.git

Step 2: Install backend dependencies

cd server

npm install

Step 3: Install frontend dependencies

cd client

npm install

Step 4: Configure environment variables

Create .env file in server directory:

PORT=5000

MONGO_URI=your_mongodb_connection_string

JWT_SECRET=your_secret_key

---

5. Folder Structure

Client (React Frontend)

client/

├── public/

├── src/

│ ├── components/

│ ├── pages/

│ ├── services/

│ ├── context/

│ └── App.js

└── package.json

Server (Node.js Backend)

server/

├── models/

├── routes/

├── controllers/

├── middleware/

├── services/

├── uploads/

└── server.js

---

6. Running the Application

Start Backend

cd server

npm start

Start Frontend

cd client

npm start

Frontend runs on: http://localhost:3000
Backend runs on: http://localhost:5000

---

7. API Documentation

1. User Registration

POST /api/auth/register
Body:

```
{
  "email": "user@example.com",
  "password": "123456"
}
```

Response:

```
{
  "message": "User registered successfully"
}
```

2. User Login

POST /api/auth/login
Response:

```
{
  "token": "jwt_token_here"
}
```

3. Upload Image

POST /api/scan/upload
Response:

```
{
  "result": "Rotten",
  "confidence": "94%"
}
```

4. Get Scan History

GET /api/scan/history

---

8. Authentication

- JWT (JSON Web Token)–based authentication
- Password hashing using bcrypt
- Role-Based Access Control (Admin/User/Operator)
- Secure API access using middleware validation

---

9. User Interface

UI includes:

- Login & Registration Page

- Dashboard with Scan Option
- Real-Time Classification Result Display
- Scan History Page
- Admin Panel
- Alert Notifications

(Screenshots/GIFs can be attached in final submission.)

---

10. Testing

- Unit Testing using Jest
- API Testing using Postman
- Frontend Testing using React Testing Library
- Model Accuracy Testing using validation dataset
- Load Testing for bulk image processing

---

11. Screenshots or Demo

- Dashboard Screenshot
- Image Upload & Classification Result
- Supermarket Report Dashboard
- Mobile Notification Alert

(Demo link can be added here.)

---

12. Known Issues

- Performance may reduce with very high-resolution images
- Model accuracy depends on dataset quality

- Requires stable internet connection for cloud deployment

---

13. Future Enhancements

- Multi-class classification (Different disease types)
- Edge AI deployment for offline processing
- IoT sensor integration for temperature & humidity monitoring
- Blockchain integration for supply chain transparency
- Advanced analytics dashboard with predictive spoilage forecasting