🌱 Farm Assist AI

Complete Project Documentation & Technical Guide

AI-Powered Plant Disease Detection & Agricultural Advisory System

Last Updated: October 2024 | Documentation Version: 1.0

📋 Table of Contents

1. Project Overview
2. System Architecture
3. Technology Stack
4. AI Models and APIs
5. Project Structure
6. System Flow & Diagrams
7. API Endpoints
8. Database Schema
9. Installation & Setup
10. Usage Guide
11. Security Features
12. Payment Integration
13. Deployment

Project Overview

Farm Assist AI is a comprehensive agricultural technology platform that leverages artificial intelligence to help farmers identify plant diseases, get treatment recommendations, and access agricultural advice through an intelligent chatbot. The system combines computer vision for disease detection with natural language processing for agricultural consultation.

Disease Detection

AI-powered image analysis to identify 38 different plant diseases with 95% accuracy using CNN models

Treatment Advice

AI-generated treatment recommendations using Google Gemini with organic and chemical options

Smart Chatbot

GPT-3.5 powered agricultural chatbot providing expert farming advice in multiple languages

User Management

Complete authentication system with API key management and usage tracking

Payment System

Razorpay integration for premium features with flexible pricing plans

Admin Dashboard

Comprehensive administrative control panel with analytics and user management

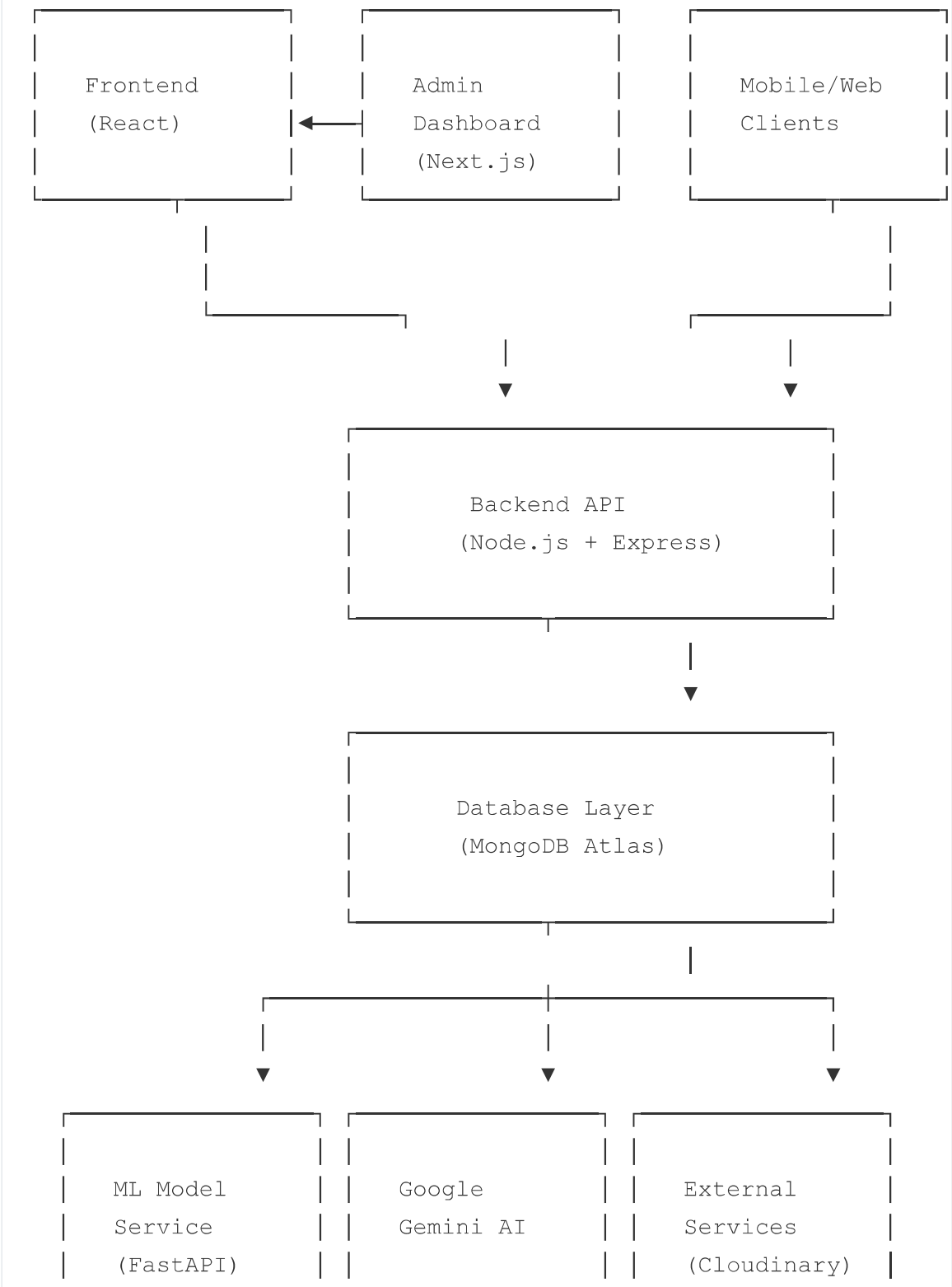
Project Goals

- Assist farmers in early disease detection and prevention

- Provide actionable, AI-generated treatment recommendations
- Make agricultural expertise accessible to all farmers globally
- Track disease patterns and trends for better crop management
- Generate sustainable revenue through premium API services

System Architecture

High-Level System Architecture



Component Architecture

Frontend Layer

- React Application - Main user interface
- Next.js Admin Dashboard - Administrative interface
- Responsive Design - Mobile-first approach

Backend Layer

- Express.js Server - RESTful API server
- Middleware Stack - CORS, Auth, File Upload
- Route Handlers - Modular routing system

AI/ML Layer

- FastAPI Model Service - Isolated ML serving
- Google Gemini - Treatment recommendations
- OpenRouter GPT-3.5 - Chatbot functionality

Data Layer

- MongoDB Atlas - Primary database
- Cloudinary - Image storage & management
- Redis (optional) - Caching layer

💻 Technology Stack

Frontend Technologies

- **React 18** - Component-based UI framework
- **Next.js 15** - Full-stack React framework for admin
- **TypeScript** - Type-safe JavaScript
- **Tailwind CSS** - Utility-first CSS framework
- **Radix UI** - Accessible component library
- **React Hot Toast** - Notification system

Backend Technologies

- **Node.js** - JavaScript runtime environment
- **Express.js 4.21** - Web application framework
- **MongoDB 8.12** - NoSQL database
- **Mongoose** - MongoDB object modeling
- **JWT** - JSON Web Tokens for authentication
- **bcryptjs** - Password hashing
- **Multer** - File upload handling
- **CORS** - Cross-origin resource sharing

AI/ML Technologies

- **FastAPI** - High-performance ML API framework
- **Python** - Programming language for ML
- **TensorFlow/Keras** - Deep learning
- **MobileNet** - Lightweight CNN

framework

architecture

PIL (Pillow) - Python imaging library

NumPy - Numerical computing

External Services

Google Gemini 1.5 Pro - Advanced language model

OpenRouter GPT-3.5 Turbo - Conversational AI

Cloudinary - Cloud-based image management

Razorpay - Payment processing

MongoDB Atlas - Cloud database

AI Models and APIs

1. Plant Disease Detection Model

Property	Details
Type	Convolutional Neural Network (CNN)
Architecture	MobileNetV2 (Transfer Learning)
Framework	TensorFlow/Keras
Input	224x224 RGB images
Output	38 different plant disease classes
Accuracy	~95% (varies by disease type)
Model File	plant_disease_prediction_model_mobilenet.pkl

Supported Plant Diseases (38 Classes)

```
Apple: Scab, Black Rot, Cedar Apple Rust, Healthy
Blueberry: Healthy
Cherry: Powdery Mildew, Healthy
Corn: Cercospora Leaf Spot, Common Rust, Northern Leaf Blight, Healthy
Grape: Black Rot, Esca, Leaf Blight, Healthy
Orange: Huanglongbing (Citrus Greening)
Peach: Bacterial Spot, Healthy
Pepper: Bacterial Spot, Healthy
Potato: Early Blight, Late Blight, Healthy
Raspberry: Healthy
Soybean: Healthy
Squash: Powdery Mildew
```

```
Strawberry: Leaf Scorch, Healthy
Tomato: Bacterial Spot, Early Blight, Late Blight, Leaf Mold,
        Septoria Leaf Spot, Spider Mites, Target Spot,
        Yellow Leaf Curl Virus, Mosaic Virus, Healthy
```

2. Google Gemini 1.5 Pro API

Purpose

Generate treatment recommendations for identified plant diseases

Input

Disease classification results from CNN model

Output

Concise treatment advice (max 70 words)

Features

Organic treatments, chemical alternatives, preventive measures

3. OpenRouter GPT-3.5 Turbo

Purpose

Agricultural chatbot functionality

Model

gpt-3.5-turbo

Languages

English and Kannada

Expertise

Agriculture and farming specialization

📁 Project Structure

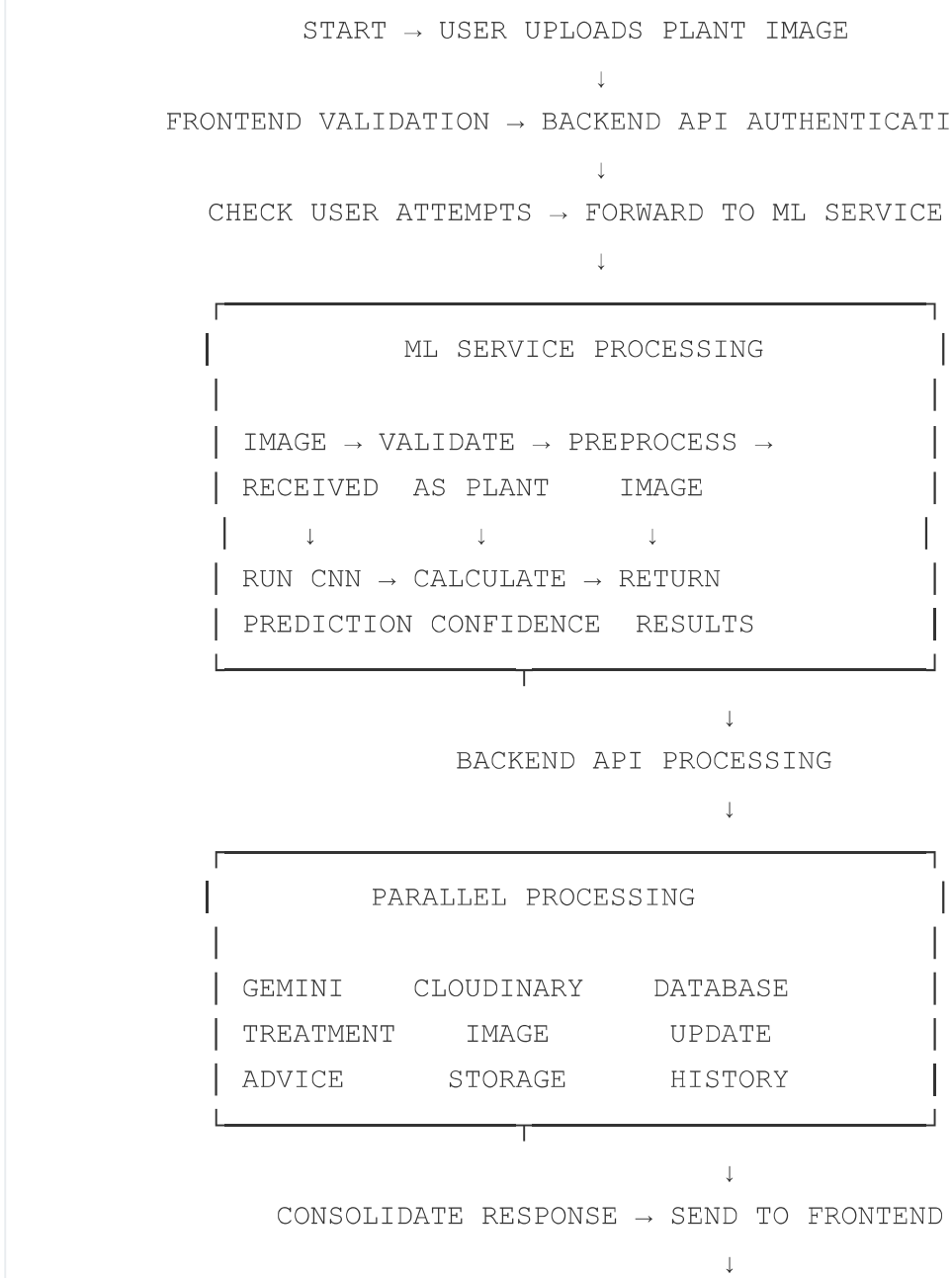
```
farm_assist_ai/
├── backend/                    # Node.js Backend
│   ├── controllers/           # Route handlers
│   │   ├── admin_controller.js
│   │   ├── chatbotController.js
│   │   ├── paymentController.js
│   │   ├── plantdisease_controller.js
│   │   └── user_controller.js
│   ├── models/                # Database schemas
│   │   ├── RequestTracking.js
│   │   └── user_model.js
│   ├── router/                # Route definitions
│   │   ├── admin_router.js
│   │   ├── chatbotRouter.js
│   │   ├── payment_router.js
│   │   ├── PlantDisease_router.js
│   │   └── user_router.js
│   ├── utils/                 # Utility functions
│   ├── conf.env              # Environment configuration
│   ├── server.js             # Main server file
│   └── package.json          # Dependencies
│
├── Model/                      # ML Model Service
│   ├── app.py                # FastAPI application
│   ├── class_indices.json    # Disease class mappings
│   └── plant_disease_prediction_model_mobilenet.pkl
│
├── Admin_dashboard/           # Next.js Admin Interface
│   ├── app/                  # App router structure
│   │   ├── layout.tsx      # Root layout
│   │   └── page.tsx        # Login page
│   ├── components/          # Reusable components
│   └── package.json        # Dependencies
│
├── frontend/                  # React User Interface
│   └── README.md
│
```

```
└── .gitignore
```

🔄 System Flow & Visual Diagrams

Plant Disease Detection Flow

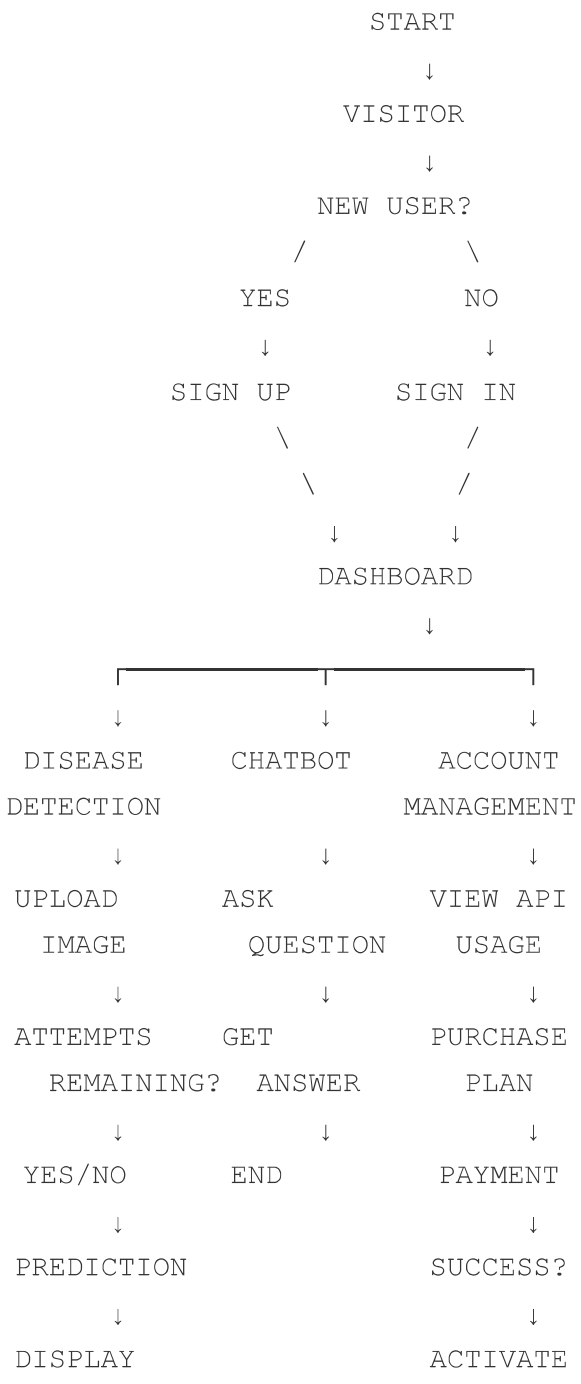
Data Flow Diagram - Disease Detection Process

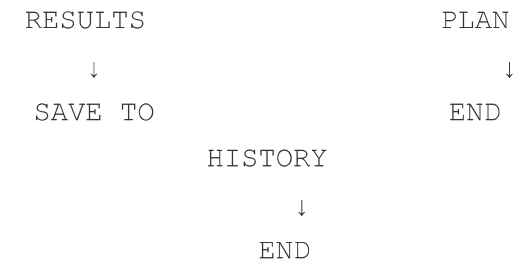


```
DISPLAY RESULTS → END
```

User Journey Flowchart

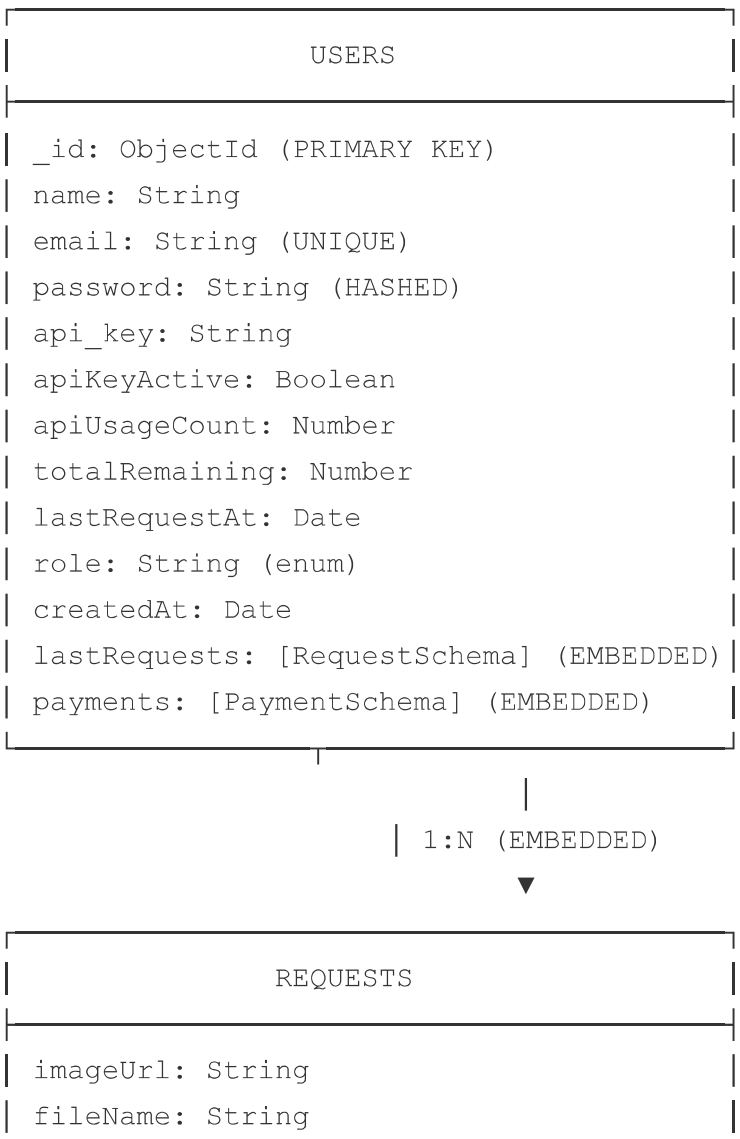
Complete User Experience Flow

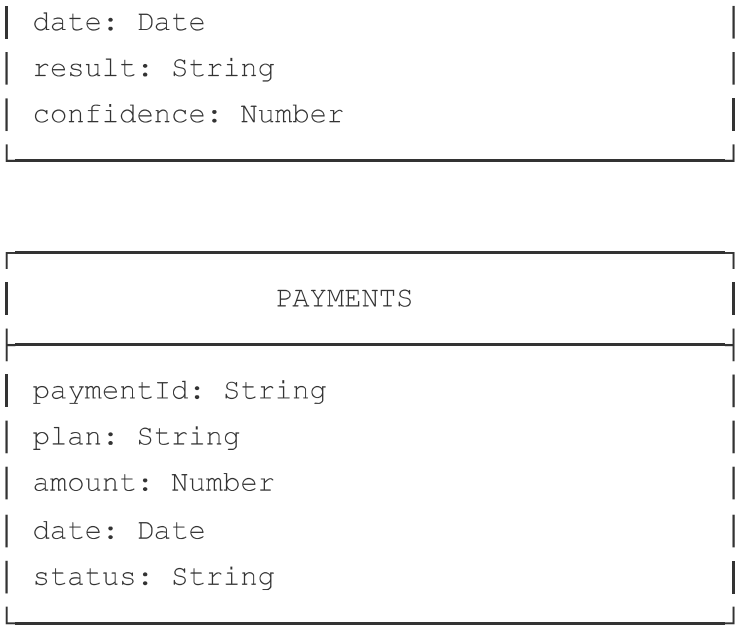




Database Entity Relationship

Database Schema Relationships




date: Date
result: String
confidence: Number
PAYMENTS
paymentId: String
plan: String
amount: Number
date: Date
status: String

🔗 API Endpoints

Authentication Endpoints

```
POST /api/user/signup - User registration
```

```
POST /api/user/login - User authentication
```

```
GET /api/user/dashboard - User dashboard data
```

Plant Disease Detection

```
POST /api/predict/ - Web interface prediction
```

```
POST /api/predict/api-key - API access prediction
```

Chatbot

```
POST /api/chat/ - Agricultural chatbot queries
```

Payment System

```
POST /api/payment/create-order - Create payment order
```

```
POST /api/payment/verify - Verify payment completion
```

Admin Endpoints

```
POST /api/admin/login - Admin authentication
```

```
GET /api/admin/dashboard - Admin dashboard
```

```
GET /api/admin/users - User management
```

```
GET /api/admin/analytics - System analytics
```

ML Model Service

```
GET / - Health check
```

```
POST /predict1 - Disease prediction
```

Database Schema

User Schema

```
{
  _id: ObjectId,
  name: String,
  email: String (unique),
  password: String (hashed with bcryptjs),
  api_key: String,
  apiKeyActive: Boolean,
  apiUsageCount: Number,
  totalRemaining: Number (default: 5),
  lastRequestAt: Date,
  role: String (enum: ["user", "admin"]),
  createdAt: Date,
  lastRequests: [RequestSchema], // Last 5 requests
  payments: [PaymentSchema]
}
```

Request Schema (Embedded)

```
{
  imageUrl: String,    // Cloudinary URL
  fileName: String,
  date: Date,
  result: String,      // Disease classification
  confidence: Number   // Prediction confidence
}
```

Payment Schema (Embedded)

```
{
  paymentId: String,    // Razorpay payment ID
  plan: String,         // "pro", "premium", etc.
  amount: Number,       // Payment amount
  date: Date,           // Payment date
  status: String        // Payment status
}
```

⚙️ Installation & Setup

Prerequisites

- Node.js 16+ and npm
- Python 3.8+ and pip
- MongoDB Atlas account
- Google Cloud Platform account (Gemini API)
- OpenRouter account
- Razorpay account
- Cloudinary account

Backend Setup

```
# Navigate to backend directory
cd backend

# Install dependencies
npm install

# Create environment file (conf.env)
DATABASE_URI=mongodb+srv://your_connection_string
PORT=8000
OPENROUTER_API_KEY=your_openrouter_key
JWT_SECRET=your_jwt_secret
API_KEY_SECRET=your_api_secret
RAZORPAY_KEY_ID=your_razorpay_key_id
RAZORPAY_KEY_SECRET=your_razorpay_secret
CLOUD_NAME=your_cloudinary_name
CLOUDINARY_API_KEY=your_cloudinary_key
CLOUDINARY_API_SECRET=your_cloudinary_secret

# Start the server
```

```
npm start
```

ML Model Setup

```
# Navigate to Model directory
cd Model

# Install Python dependencies
pip install fastapi uvicorn pillow numpy google-generativeai

# Update app.py with your Gemini API key
# Line 28: genai.configure(api_key="YOUR_GEMINI_API_KEY")

# Start the FastAPI server
python app.py
```

Admin Dashboard Setup

```
# Navigate to Admin_dashboard directory
cd Admin_dashboard

# Install dependencies
npm install

# Start development server
npm run dev
```

Frontend Setup

```
# Navigate to frontend directory
cd frontend
```

```
# Install dependencies
npm install

# Start the application
npm start
```

📖 Usage Guide

For Farmers (Web Interface)

1. Registration

Create account with name, email, and password

2. Disease Detection

Upload clear plant leaf images for instant disease identification

3. Get Recommendations

Receive AI-generated treatment advice with confidence scores

4. Chatbot Consultation

Ask farming questions in English or Kannada

5. Premium Upgrade

Purchase additional API calls and access API keys

6. History Tracking

View last 5 disease detection requests and results

For Developers (API Access)

API Usage Example

```
# Get API Key: Upgrade to premium plan through web interface
# Authentication: Include API key in Authorization header

curl -X POST \
  -H "Authorization: Bearer YOUR_API_KEY" \
  -F "image=@plant_leaf.jpg" \
```

```
  http://localhost:8000/api/predict/api-key

# Response:
{
  "filename": "plant_leaf.jpg",
  "prediction": "diseased",
  "confidence": 0.9234,
  "class_name": "Tomato Early Blight",
  "class_index": 29,
  "gemini": "Apply copper-based fungicide. Remove affected leaves.
  "rot_percentage": 15.67,
  "care_advice": "Apply pesticides to treat the infection.",
  "remaining_attempts": 49
}
```

For Administrators

Dashboard Access

Login with admin credentials to access control panel

User Management

View, manage, and monitor user accounts and activities

Analytics

Monitor API usage, request patterns, and system health

Revenue Tracking

Track payments, subscription plans, and revenue metrics

🔒 Security Features

Authentication & Authorization

- **JWT Tokens** - Secure user sessions with expiration
- **bcryptjs** - Password hashing with salt rounds
- **API Key Validation** - Secure API access control
- **Role-based Access** - Admin/user permission system

Data Protection

- **Input Validation** - Sanitize all user inputs
- **File Type Validation** - Accept only image files
- **Rate Limiting** - Prevent API abuse and DoS attacks
- **CORS Configuration** - Control cross-origin requests

Environment Security

- **Environment Variables** - Secure API key storage
- **Database Security** - MongoDB Atlas encryption
- **HTTPS** - Secure data transmission
- **Cloud Security** - Cloudinary secure URLs

💳 Payment Integration

Razorpay Integration Features

Order Creation

Generate secure payment orders with proper validation

Payment Verification

Validate payment status and amount verification

Webhook Support

Real-time payment updates and notifications

Multiple Plans

Flexible pricing options for different user needs

Pricing Model

Plan	Features	Price
Free Tier	5 free predictions, Basic chatbot access	₹0
Pro Plan	50 additional predictions, API access, Priority support	₹99
Enterprise	Unlimited predictions, Dedicated API key, Custom integrations	Contact

🚀 Deployment

Recommended Deployment Options

Component	Platform	Reason
Backend API	Heroku, AWS, DigitalOcean	Node.js support, easy scaling
ML Model Service	Render, Railway, Google Cloud Run	Python/FastAPI support
Frontend	Vercel, Netlify	React optimization, CDN
Admin Dashboard	Vercel, Netlify	Next.js optimization
Database	MongoDB Atlas	Managed cloud database
File Storage	Cloudinary	Image optimization, CDN

Production Environment Variables

```
NODE_ENV=production
DATABASE_URI=mongodb+srv://production_connection_string
Model_Url=https://your-ml-service.com/predict1/
FRONTEND_URL=https://your-frontend-domain.com
ADMIN_URL=https://your-admin-dashboard.com
```

Performance Optimization

- **Image Compression:** Optimize image uploads before processing
- **Caching:** Implement Redis for frequently accessed data

- **CDN:** Use Cloudinary's global CDN for images
- **Load Balancing:** Scale horizontally for high traffic
- **Database Indexing:** Optimize MongoDB queries
- **API Rate Limiting:** Implement sophisticated rate limiting

Future Enhancements

Planned Features

- Mobile app development with React Native
- Offline mode for local model inference
- Weather integration for enhanced recommendations
- Complete crop management system
- Farmer community features and forums
- IoT sensor data incorporation
- Multi-language support expansion

📞 Contact Information

For technical support or inquiries about the Farm Assist AI project, please contact the development team.

Farm Assist AI - Empowering farmers with artificial intelligence

Last Updated: October 2024 | **Documentation Version:** 1.0