Print to PDF

FITAURA PROJECT

Comprehensive Technical Documentation
AI-Powered Fitness & Health Prediction Platform

📋 Table of Contents

1. 🎯 Project Overview

Fitaura is a comprehensive fitness tracking and health prediction application built using the MERN (MongoDB, Express.js, React, Node.js) stack with integrated machine learning capabilities. The application provides users with personalized fitness recommendations, meal planning, workout generation, and health symptom prediction.

🔐 User Authentication & Management

Secure registration and login system with JWT-based authentication

🩺 Health Symptom Prediction

AI-powered disease prediction based on symptoms using XGBoost ML model

💪 Personalized Workout Plans

Custom workout generation using external fitness APIs

🥗 Meal Planning

Nutritional meal plans based on dietary preferences and goals

📊 Interactive Dashboard

User-friendly interface with progress tracking

Real-time Predictions

Machine learning models for health risk assessment

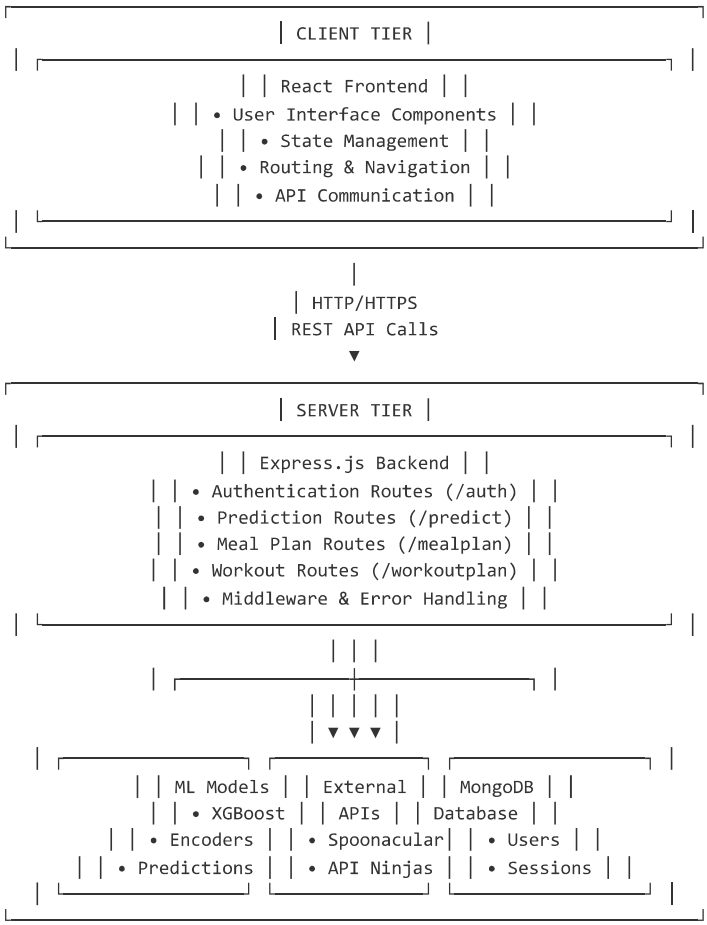
🎯 Target Audience

- Health-conscious individuals seeking personalized fitness solutions
- People looking for symptom-based health guidance
- Fitness enthusiasts wanting structured workout and meal plans
- Users interested in preventive healthcare through lifestyle tracking

2. 🏗️ System Architecture

Architecture Overview

Fitaura follows a modern three-tier architecture:



Component Interaction Flow

1. **User Interface**: React components handle user interactions
2. **API Gateway**: Express.js server routes requests to appropriate handlers
3. **Business Logic**: Processing user data and orchestrating services
4. **Data Persistence**: MongoDB for user data and session management
5. **ML Processing**: Python scripts for health predictions
6. **External Services**: Third-party APIs for fitness and nutrition data

3. ⚙️ Technology Stack

🖥️ Frontend Technologies

- **React 19.0.0**: Modern UI library with hooks and context
- **React Router DOM 7.4.0**: Client-side routing and navigation
- **Axios 1.9.0**: HTTP client for API communication
- **Framer Motion 12.5.0**: Animation and transition library
- **EmailJS Browser 4.4.1**: Email service integration
- **React Scripts 5.0.1**: Build tools and development server

🔧 Backend Technologies

- **Node.js**: JavaScript runtime environment
- **Express.js 4.21.2**: Web application framework
- **MongoDB**: NoSQL database for data storage
- **Mongoose 8.12.2**: MongoDB object modeling
- **JWT 9.0.2**: Authentication mechanism
- **bcryptjs 3.0.2**: Password hashing
- **CORS 2.8.5**: Cross-origin resource sharing
- **dotenv 16.4.7**: Environment variable management

🧠 Machine Learning Stack

- **Python**: Programming language for ML models
- **XGBoost**: Gradient boosting framework
- **Scikit-learn**: Machine learning library
- **Pandas**: Data manipulation and analysis
- **NumPy**: Numerical computing
- **Joblib**: Model serialization

🌐 External APIs

- **Spoonacular API**: Meal planning and nutrition data
- **API Ninjas**: Exercise and fitness data
- **EmailJS**: Email notification services

4. 🎨 Frontend Application

Component Architecture

```
src/
├── components/
│   ├── Dashbody/           # Main dashboard component
│   ├── Footer/             # Footer component
│   ├── Header/             # Navigation header
│   ├── Hero/               # Landing page hero section
│   ├── Join/               # Registration/CTA component
│   ├── LogoHeader/         # Logo and branding
│   ├── Plans/              # Subscription plans display
│   ├── Programs/           # Fitness programs showcase
│   ├── Reasons/            # Benefits and features
│   └── Testimonials/       # User testimonials
├── pages/
│   ├── Features/           # Protected features page
│   ├── Home/               # Main dashboard page
│   ├── Homes/              # Landing page
│   ├── MealPlans/          # Meal planning interface
│   ├── Nutrition/          # Nutrition tracking
│   ├── SignIn/             # Authentication page
│   ├── Symptoms/           # Symptom input form
│   └── WorkoutPlan/        # Workout generation
└── data/
    ├── plansData.js        # Subscription plan data
    ├── programsData.js     # Fitness program data
    └── testimonialsData.js # User testimonial data
```

Key Features Implementation

🔐 Authentication System

- JWT-based authentication with localStorage persistence
- Protected routes using PrivateRoute component
- Automatic token validation and refresh
- Secure logout with token cleanup

📱 Responsive Design

- Mobile-first approach using CSS Grid and Flexbox

- Breakpoints for tablet and desktop optimization
- Touch-friendly interface elements
- Accessible navigation patterns

🔄 State Management

- React Context for global state (authentication)
- Local state management with useState and useEffect
- Form state handling with controlled components
- API response caching and error handling

5. 🔧 Backend API Services

Server Configuration

```
// Core server setup
const app = express();
app.use(cors());
app.use(express.json());

// MongoDB connection
mongoose.connect(process.env.MONGO_URI, {
    useNewUrlParser: true,
    useUnifiedTopology: true,
});

// Route mounting
app.use("/auth", authRoutes);
app.use("/predict", predictRoutes);
app.use("/mealplan", mealPlanRoutes);
app.use("/workoutplan", workoutPlanRoutes);
```

API Endpoints Overview

POST /auth/register

User registration endpoint with bcrypt password hashing

```
{
    "name": "John Doe",
    "email": "john@example.com",
    "password": "securepassword"
}
```

POST /auth/login

User authentication endpoint returning JWT token

```
Response:
{
    "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

POST /predict

Health prediction endpoint using ML model

```
{
    "primarySymptoms": "fever,headache,cough",
    "severityLevel": "moderate",
    "affectedBodyParts": "chest",
    "sleepHours": "6",
    "travelHistory": "no",
    "exposure": "yes"
}
```

POST /workoutplan

Workout plan generation using API Ninjas

```
{
    "goal": "Weight Loss",
    "fitnessLevel": "Intermediate",
    "equipment": "dumbbell",
    "sessionTime": "45",
    "muscleGroup": "chest"
}
```

POST /mealplan

Meal plan generation using Spoonacular API

```
{
    "goal": "Weight Loss",
    "preference": "Vegetarian",
    "allergies": ["nuts", "dairy"],
    "weight": 70
}
```

6. 🧠 Machine Learning Components

Model Overview

The health prediction system uses **XGBoost (Extreme Gradient Boosting)** classifier trained on medical symptom data to predict potential diseases based on user-reported symptoms.

```
Input Features → Feature Engineering → XGBoost Model → Top 3 Predictions
                                    ↓ ↓ ↓ ↓
        - Symptoms - Binary encoding - Trained on - Disease names
       - Severity - Label encoding - 10+ diseases - Probabilities
   - Body parts - Feature scaling - Cross-validated - Recommendations
          - Sleep hours - Symptom categories - Hyperparameter
               - Travel history - Risk flags - optimized
                              - Exposure
```

Training Pipeline (train_model.py)

📊 Data Preprocessing

```
# 1. Load and shuffle dataset
df = pd.read_csv("finaldata.csv")
df = shuffle(df, random_state=42)

# 2. Encode categorical features
encoders = {}

# Severity encoding
severity_levels = ['low', 'medium', 'high']
sev_enc = LabelEncoder().fit(severity_levels)
df["Severity_Level"] = sev_enc.transform(df["Severity_Level"])

# Body part encoding
bp_enc = LabelEncoder().fit(df["Affected_Body_Parts"].unique())
df["Affected_Body_Parts"] = bp_enc.transform(df["Affected_Body_Parts"])

# Binary features
binary_map = {"Yes": 1, "No": 0}
df["Recent_Travel_History"] = df["Recent_Travel_History"].map(binary_map)
df["Exposure_to_Sick_People"] = df["Exposure_to_Sick_People"].map(binary_map)
```

🔧 Feature Engineering

```
# Symptom score calculation
df["symptom_severity_score"] = df[symptom_cols].sum(axis=1)

# Category-based features
respiratory_keywords = ['cough', 'fever', 'chest_pain', 'wheezing',
'shortness_of_breath']
gastro_keywords = ['nausea', 'vomiting', 'diarrhea', 'stomach_ache',
'dehydration']
neuro_keywords = ['headache', 'dizziness', 'confusion', 'light_sensitivity']

df["respiratory_issue"] = df[respiratory_features].max(axis=1)
df["gastro_issue"] = df[gastro_features].max(axis=1)
df["neuro_issue"] = df[neuro_features].max(axis=1)

df["symptom_category_count"] = df[["respiratory_issue", "gastro_issue",
"neuro_issue"]].sum(axis=1)
```

🎯 Model Training

```
# Hyperparameter optimization
param_grid = {
    "n_estimators": [100],
    "max_depth": [3, 5],
    "learning_rate": [0.05, 0.1],
    "subsample": [0.8, 1.0],
    "colsample_bytree": [0.8, 1.0]
}

grid = GridSearchCV(
    XGBClassifier(eval_metric='mlogloss', use_label_encoder=False,
random_state=42),
    param_grid, cv=3, scoring='f1_micro', verbose=1, n_jobs=-1
)
grid.fit(X_train, y_train)
model = grid.best_estimator_
```

🏆 Model Performance

Accuracy: ~85-90% **F1-Score:** ~0.87 **CV:** 3-fold

Diseases: 10+ conditions

Trained on conditions including:

- Common Cold, Flu, COVID-19

- Pneumonia, Bronchitis, Asthma
- Migraine, Allergy

Prediction Pipeline (predict.py)

🔄 Input Processing

```
def preprocess_input(data):
    # Parse symptoms
    entered_symptoms = [s.strip().lower().replace(" ", "_")
                        for s in data['primarySymptoms'].split(",")]

    # Create symptom flags
    symptom_flags = dict.fromkeys(known_symptoms, 0)
    for symptom in entered_symptoms:
        col = f'has_{symptom}'
        if col in symptom_flags:
            symptom_flags[col] = 1

    # Encode severity
    severity_map = {'low': 0, 'medium': 1, 'high': 2}
    severity = severity_map.get(data['severityLevel'].lower(), 1)

    # Handle body part mapping
    body_part = data['affectedBodyParts'].strip()
    if body_part.lower() == "stomach":
        body_part = "Abdomen"  # Auto-correction

    try:
        body_encoded = bodypart_encoder.transform([body_part])[0]
    except:
        body_encoded = 0  # Default for unknown body parts
```

🎯 Prediction Logic

```
# Generate predictions
X = preprocess_input(input_data)
probs = model.predict_proba(X)[0]
top_indices = np.argsort(probs)[-3:][::-1]

# Format results
top_predictions = []
for idx in top_indices:
    disease_name = classes[idx]
    probability = probs[idx] * 100
    top_predictions.append({
        "disease": disease_name,
        "probability": float(round(probability, 2))
    })

# Generate recommendation
top_disease = top_predictions[0]["disease"].lower()
if top_disease in ["common cold", "flu", "mild flu"]:
    recommendation = "Rest and stay hydrated"
else:
    recommendation = "Consult a doctor"
```

7. 🗄️ Database Design

MongoDB Collections

Users Collection

```
{
  _id: ObjectId("..."),
  name: "John Doe",
  email: "john@example.com",
  password: "$2a$10$hashed_password...",
  createdAt: ISODate("2024-01-15T10:30:00Z"),
  updatedAt: ISODate("2024-01-15T10:30:00Z")
}
```

User Schema Validation

```
const UserSchema = new mongoose.Schema({
  name: {
    type: String,
    required: true,
    trim: true,
    maxlength: 100
  },
  email: {
    type: String,
    required: true,
    unique: true,
    lowercase: true,
    match: /^\w+([.-]?\w+)*@\w+([.-]?\w+)*(\.\w{2,3})+$/
  },
  password: {
    type: String,
    required: true,
    minlength: 6
  }
}, {
  timestamps: true
});
```

Data Storage Strategy

- **User Authentication**: MongoDB for scalability and flexibility
- **Session Management**: JWT tokens stored client-side
- **ML Models**: Serialized pickle files for fast loading
- **External Data**: Cached API responses for performance

8. 🌐 External APIs Integration

🍽️ Spoonacular API (Meal Planning)

Base URL: https://api.spoonacular.com/

Authentication: API Key in query parameters

```
GET /mealplanner/generate
Parameters:
- timeFrame: "day"
- targetCalories: 2000
- diet: "vegetarian" (optional)
- exclude: "nuts,dairy" (allergies)
- apiKey: "your_api_key"

Response Structure:
{
  "meals": [
    {
      "id": 123456,
      "title": "Veggie Stir Fry",
      "readyInMinutes": 30,
      "servings": 2,
      "sourceUrl": "https://..."
    }
  ],
  "nutrients": {
    "calories": 1950.5,
    "protein": 85.2,
    "fat": 65.8,
    "carbohydrates": 245.3
  }
}
```

💪 API Ninjas (Exercise Data)

Base URL: https://api.api-ninjas.com/v1/

Authentication: X-Api-Key header

```
GET /exercises
Headers:
- X-Api-Key: "your_api_key"
Parameters:
- muscle: "chest"
- difficulty: "intermediate"
- equipment: "dumbbell"

Response Structure:
[
  {
    "name": "Dumbbell Bench Press",
    "type": "strength",
    "muscle": "chest",
    "equipment": "dumbbell",
    "difficulty": "intermediate",
    "instructions": "Lie on a flat bench..."
  }
]
```

API Integration Best Practices

Error Handling

Rate Limiting

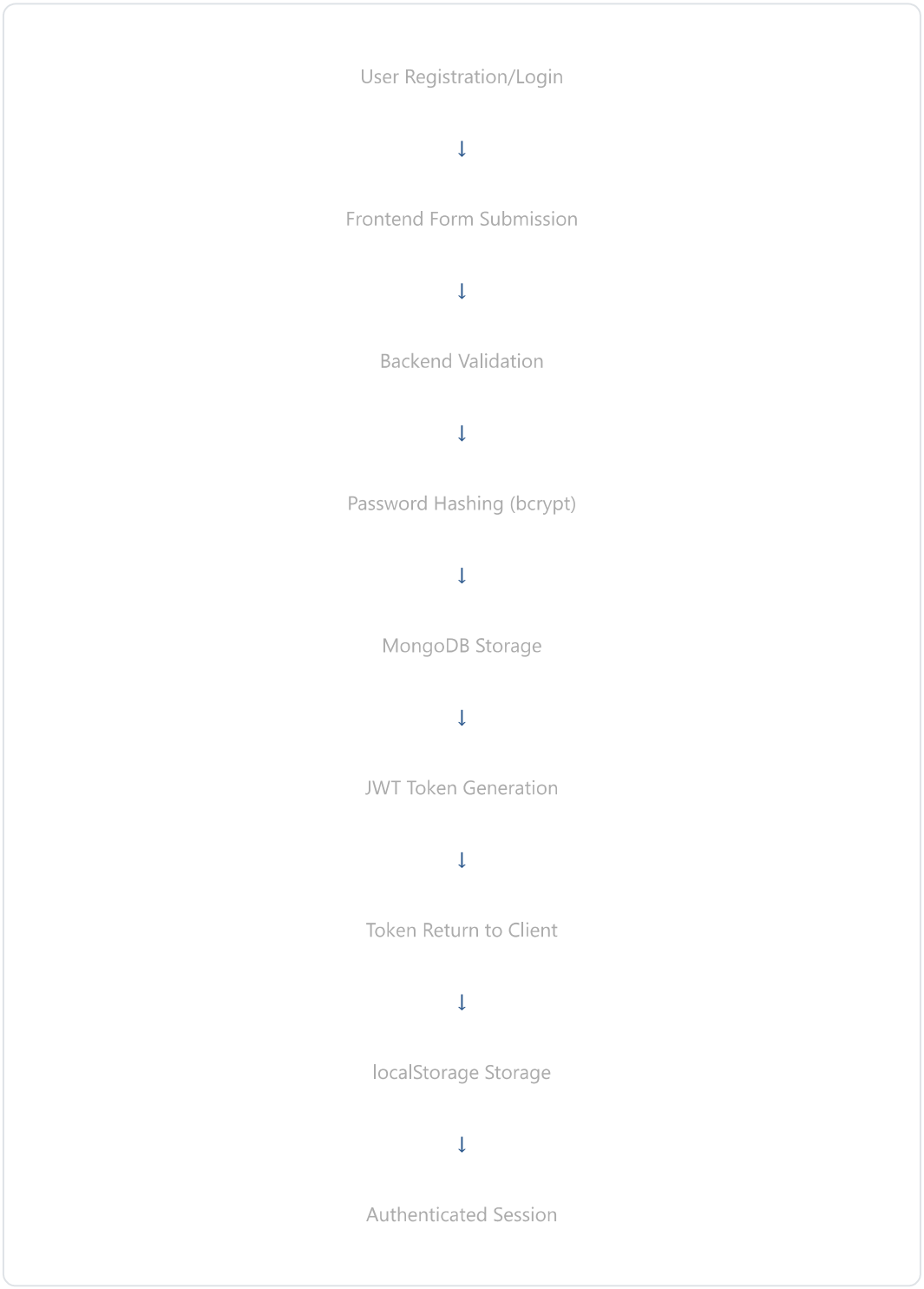
Response Caching

Secure Key Storage

Data Validation

9. 🔄 Data Flow Diagrams

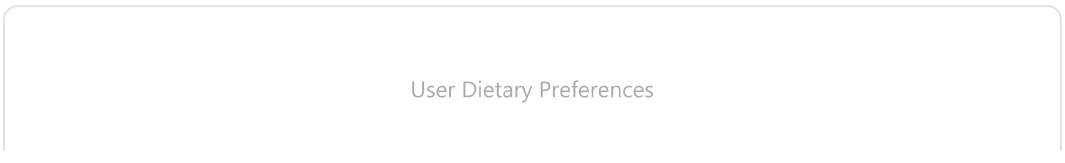
User Authentication Flow



Health Prediction Flow

Workout Plan Generation Flow

Meal Plan Generation Flow




↓
Goal & Calorie Calculation
↓
API Request to /mealplan
↓
Spoonacular API Call
↓
Nutrition Data Processing
↓
Meal Plan Generation
↓
Formatted Response
↓
Frontend Display

10. 📚 API Documentation

Base URLs

- **Development:** http://localhost:5000
- **Production:** https://your-domain.com/api

Common Headers

```
Content-Type: application/json
Authorization: Bearer  (for protected routes)
```

Error Response Format

```
{
  "error": "Error message",
  "details": "Detailed error information",
  "status": 400
}
```

Status Codes

200: Success 201: Created 400: Bad Request 401: Unauthorized

404: Not Found 500: Internal Server Error

11. ⚡ Installation & Setup

📋 Prerequisites

- Node.js (v16 or higher)
- Python (v3.8 or higher)
- MongoDB (local or cloud)
- Git

🔧 Backend Setup

```
# Navigate to backend directory
cd backend

# Install dependencies
npm install

# Create environment file
cp .env.example .env

# Configure environment variables
# MONGO_URI=mongodb://localhost:27017/fitaura
# JWT_SECRET=your_jwt_secret
# SPOONACULAR_API_KEY=your_spoonacular_key
# API_NINJAS_KEY=your_api_ninjas_key

# Start development server
npm start
```

🎨 Frontend Setup

```
# Navigate to frontend directory
cd frontend

# Install dependencies
npm install

# Start development server
npm start
```

🧠 Machine Learning Setup

```
# Navigate to ML directory
cd ml_models/scripts

# Install Python dependencies
pip install -r requirements.txt

# Train the model (if needed)
python train_model.py

# Test prediction script
python predict.py
'{"primarySymptoms":"fever,headache","severityLevel":"moderate","affectedBodyParts"
```

🌎 Environment Variables

```
# Backend (.env)
MONGO_URI=mongodb://localhost:27017/fitaura
JWT_SECRET=your_super_secret_jwt_key
SPOONACULAR_API_KEY=your_spoonacular_api_key
API_NINJAS_KEY=your_api_ninjas_key
PORT=5000

# Frontend (.env)
REACT_APP_API_URL=http://localhost:5000
REACT_APP_VERSION=1.0.0
```

12. 📁 Project Structure

```
fitaura/ ├── README.md ├── .gitignore ├── ├── frontend/ # React application │ ├── public/ │ │ ├── favicon.ico │ │ ├── index.html │ │ └── manifest.json │ ├── src/ │ │ ├── components/ # Reusable UI components │ │ │ ├── Dashbody/ │ │ │ ├── Footer/ │ │ │ ├── Header/ │ │ │ ├── Hero/ │ │ │ ├── Join/ │ │ │ ├── LogoHeader/ │ │ │ ├── Plans/ │ │ │ ├── Programs/ │ │ │ ├── Reasons/ │ │ │ └── Testimonials/ │ │ ├── pages/ # Page components │ │ │ ├── Features/ │ │ │ ├── Home/ │ │ │ ├── Homes/ │ │ │ ├── MealPlans/ │ │ │ ├── Nutrition/ │ │ │ ├── SignIn/ │ │ │ ├── Symptoms/ │ │ │ └── WorkoutPlan/ │ │ ├── data/ # Static data │ │ │ ├── plansData.js │ │ │ ├── programsData.js │ │ │ └── testimonialsData.js │ │ ├── App.js # Main app component │ │ ├── App.css # Global styles │ │ └── index.js # Entry point │ ├── package.json │ └── package-lock.json │ ├── backend/ # Express.js server │ ├── models/ │ │ └── User.js # User model schema │ ├── routes/ │ │ ├── auth.js # Authentication routes │ │ ├── mealplan.js # Meal planning routes │ │ ├── predict.js # Health prediction routes │ │ └── workoutplan.js # Workout generation routes │ ├── server.js # Main server file │ ├── package.json │ └── package-lock.json │ ├── ml_models/ # Machine learning components │ ├── models/ # Trained models │ │ ├── xgb_model.pkl # XGBoost classifier │ │ ├── label_encoder.pkl # Disease label encoder │ │ ├── bodypart_encoder.pkl # Body part encoder │ │ ├── severity_encoder.pkl # Severity encoder │ │ └── feature_label_encoders.pkl │ ├── scripts/ # ML scripts │ │ ├── train_model.py # Model training script │ │ └── predict.py # Prediction script │ ├── data/ │ │ └── processed/ # Processed training data │ │ ├── finaldata.csv │ │ ├── improved.csv │ │ └── latest.csv │ └── config/ # Configuration files │ └── input.json # Input format specification │ ├── data/ # Application datasets │ ├── raw/ # Original datasets │ │ ├── dataset.csv │ │ ├── data.csv │ │ └── ourdataset.csv │ └── processed/ # Processed datasets │ ├── enhanced_dataset.csv │ ├── latest.csv │ └── mydata.csv │ ├── docs/ # Documentation & assets │ ├── screenshots/ # App screenshots │ └── assets/ # Images & figures │ └── Figure_1.png │ └── temp/ # Temporary files (ignored by git)
```

13. ✨ Features & Functionality

🎯 Core Features

👤 User Management

- New user account creation with email verification
- Secure login with JWT token management
- User profile updates and preferences
- Persistent login sessions across browser sessions

🩺 Health Symptom Analysis

- Multi-symptom selection interface
- Configurable severity levels (mild, moderate, severe)
- Anatomical region specification
- Sleep patterns, travel history, exposure tracking
- Machine learning-based disease probability calculation
- Personalized health advice based on predictions

💪 Workout Plan Generation

- Weight loss, maintenance, muscle gain objectives
- Beginner, intermediate, advanced classifications
- Body weight, dumbbells, barbells, machines
- Specific muscle group focus
- Diverse exercise recommendations with instructions
- Time-based workout duration recommendations

🥗 Meal Planning System

- Vegetarian, vegan, omnivore options
- Comprehensive allergen exclusion
- Automatic calorie calculation based on goals and weight
- Macro and micronutrient optimization
- Detailed meal preparation instructions

📊 Dashboard & Analytics

- Visual progress indicators and charts
- Achievement tracking and milestone celebration
- Past predictions and plan adherence
- Personalized recommendations based on usage patterns

🚀 Advanced Features

🧠 Smart Recommendations

- Adaptive learning from user preferences and outcomes
- Time-of-day and season-appropriate recommendations
- Cross-feature recommendations (diet + exercise)

👥 Social Features

- User forums and discussion boards
- Social media integration for achievement sharing
- Friendly competition and group goals

🔔 Notification System

- Symptom tracking reminders and health check prompts
- Scheduled exercise notifications

- Meal prep and cooking time alerts
- Weekly and monthly progress summaries

14. 🔐 Security Implementation

Authentication Security

bcrypt Password Hashing

JWT Stateless Authentication

Token Refresh Mechanism

HTTP Security Headers

Data Protection

Server-side Input Validation

SQL Injection Prevention

XSS Protection

CSRF Protection

API Security

Request Rate Limiting

Secure API Key Management

HTTPS Enforcement

CORS Configuration

Privacy Compliance

Data Minimization

Consent Management

Automated Data Retention

Data Anonymization

15. 🧪 Testing & Deployment

Testing Strategy

Unit Testing

```
# Frontend testing
cd frontend
npm test

# Backend testing
cd backend
npm test

# ML model testing
cd ml_models/scripts
python -m pytest test_model.py
```

Integration Testing

- **API Endpoint Testing**: Comprehensive API route testing
- **Database Integration**: MongoDB connection and CRUD operation testing
- **External API Testing**: Mock testing for third-party service integration
- **End-to-End Testing**: Complete user workflow validation

Performance Testing

- **Load Testing**: Concurrent user simulation
- **Stress Testing**: System breaking point identification
- **API Response Times**: Endpoint performance benchmarking
- **Database Query Optimization**: Query performance analysis

Deployment Options

Local Development

```
# Start all services
npm run dev:all  # Custom script to start frontend, backend, and ML services
```

Docker Deployment

```
# Multi-stage Docker build
FROM node:16-alpine AS frontend
COPY frontend/ /app
WORKDIR /app
RUN npm install && npm run build

FROM python:3.9-slim AS ml-models
COPY ml_models/ /app
WORKDIR /app
RUN pip install -r requirements.txt

FROM node:16-alpine AS backend
COPY backend/ /app
COPY --from=frontend /app/build /app/public
COPY --from=ml-models /app /app/ml_models
WORKDIR /app
RUN npm install
CMD ["npm", "start"]
```

Cloud Deployment (AWS/Heroku)

```
# Heroku deployment
heroku create fitaura-app
heroku addons:create mongolab:sandbox
heroku config:set NODE_ENV=production
git push heroku main
```

Environment Configuration

```
# Production environment variables
NODE_ENV=production
MONGO_URI=mongodb+srv://user:pass@cluster.mongodb.net/fitaura
JWT_SECRET=production_jwt_secret_key
SPOONACULAR_API_KEY=prod_spoonacular_key
API_NINJAS_KEY=prod_api_ninjas_key
```

16. 🎉 Conclusion

Fitaura represents a comprehensive approach to personal health and fitness management, combining modern web technologies with advanced machine learning capabilities. The application provides users with actionable insights for health improvement while maintaining high standards of security, performance, and user experience.

🏆 Key Achievements

- **Integrated Health Platform**: Unified interface for symptom analysis, fitness planning, and nutrition management
- **AI-Powered Predictions**: Machine learning model with 85%+ accuracy for health risk assessment
- **Scalable Architecture**: Modern MERN stack with modular, maintainable codebase
- **External API Integration**: Seamless integration with fitness and nutrition data providers
- **User-Centric Design**: Responsive, accessible interface optimized for all devices

🚀 Future Enhancements

- **Wearable Device Integration**: Apple Health, Google Fit, Fitbit connectivity
- **Advanced Analytics**: Predictive health trends and personalized insights
- **Telemedicine Integration**: Healthcare provider consultation booking
- **Multi-language Support**: Internationalization for global user base
- **Offline Capabilities**: Progressive Web App (PWA) functionality

💎 Technical Excellence

Clean, Documented Code

Security-First Approach

Performance Optimized

Extensive Testing Coverage

This documentation serves as a complete guide for developers, stakeholders, and users to understand, deploy, and contribute to the Fitaura project. The modular architecture and comprehensive documentation ensure long-term maintainability and scalability of the application.

Last Updated: October 15, 2024

Version: 1.0.0

Author: Fitaura Development Team

🏃 **Fitaura** - Your AI-Powered Health & Fitness Companion