Arthomed Healthcare Backend

Comprehensive Documentation - Complete Documentation

Generated on 7/25/2025

Table of Contents

1. Project Overview

Arthomed Healthcare Backend! [Node.js](https://nodejs.org/)! [Express](https://expressjs.com/)! [MongoDB]

(https://www.mongodb.com/)![License](LICENSE)A comprehensive healthcare management backend system built with Node.js, Express, and MongoDB, designed specifically for React Native applications. Features role-based authentication, OTP verification, appointment management, and file upload capabilities.## 🏥 OverviewArthomed is a production-ready healthcare backend that provides:- Mobile OTP Authentication with Twilio SMS integration- Role-based Access Control (Admin, Doctor,

Receptionist, Patient)- Appointment Booking System with real-time slot management- File Upload System for medical reports and prescriptions- Comprehensive API Documentation with Swagger/OpenAPI 3.0- Secure JWT Authentication with refresh token support## 📋 Table of Contents-

Security Features- Testing- Deployment- Contributing- License## ✨ Features### Core Functionality- 🔐 OTP-based Authentication - Secure mobile number verification- 👥 Multi-role User Management - Admin, Doctor, Receptionist, Patient roles- 📅 Appointment Booking - Real-time slot management and booking- 📁 File Upload - Medical reports and prescription image uploads- 📊 Dashboard APIs - Role-specific dashboards and analytics- 🔔 SMS Notifications - Appointment confirmations and reminders### Technical

Features- 🏗️ RESTful API Design - Clean, standardized endpoints- 📖 API Documentation - Interactive Swagger documentation- 🔒 Security First - JWT tokens, rate limiting, data validation- 📱 Mobile-First - Optimized for React Native integration- 🚀 Production Ready - Error handling, logging, monitoring- 📊 Database Optimization - Indexed queries and aggregations## 🛠️ Technology Stack### Backend Framework- Node.js (18+) - JavaScript runtime- Express.js (4.19) - Web application framework- MongoDB Atlas -

Cloud database service- Mongoose (8.x) - MongoDB object modeling### Authentication & Security- JWT - JSON Web Tokens for authentication- bcryptjs - Password hashing- express-rate-limit - Rate limiting middleware- helmet - Security headers- cors - Cross-origin resource sharing### External Services- Twilio - SMS OTP delivery- Multer - File upload handling- Swagger/OpenAPI 3.0 - API documentation### Development Tools- nodemon - Development server- express-validator - Input validation- dotenv - Environment variable

management## 🏗️ Architecture### Project Structure

```
arthomed-backend/├── src/│   ├
```

## 🚀 Installation### Prerequisites- Node.js 18+ installed- MongoDB Atlas account- Twilio account (for SMS)- Git### Quick Start

```
bash# Clone the repositorygit c
```

## ⚙️ Environment SetupCreate a `.env` file in the root directory:

```
env# Server ConfigurationNODE_E
```

## 📚 API Documentation### Interactive DocumentationOnce the server is running, visit:- Swagger UI: `http://localhost:3001/api-docs`- Health Check: `http://localhost:3001/health`### Main API Endpoints#### Authentication

```
POST /api/auth/send-otp
```

#### User Management

```
GET  /api/users/doctors
```

#### Appointments

```
GET  /api/appointments/slots
```

#### File Management

```
GET  /api/files/:type/:filename
```

## 🗄 Database Schema### User Collection

```
{  _id: ObjectId,  mobileNumber
```

### Appointment Collection

```
{  _id: ObjectId,  patient: Obj
```

## 🔐 Authentication FlowThe system uses a secure OTP-based authentication flow:1. Send OTP: Patient enters mobile number2. Verify OTP: System validates the 6-digit code3. Auto Login/Register: Existing users

login, new users auto-register4. JWT Tokens: Secure access and refresh tokens issued5. Role Assignment: Users get appropriate roles and permissions## 👥 Role-based Access### Permission Matrix| Endpoint | Admin | Doctor | Receptionist | Patient ||----------|--------|---------|---------------|----------|| GET /users/doctors | ✅ | ✅ | ✅ | ✅ || POST /admin/create-user | ✅ | ❌ | ❌ | ❌ || GET /appointments/pending | ✅ | ❌ | ✅ | ❌ || POST /appointments/confirm | ✅ | ❌ |

✅ | ❌ | | POST /appointments/book | ✅ | ❌ | ✅ | ✅ | | GET /appointments/my | ❌ | ✅ | ❌ | ✅ |## 📅 Appointment System### Key Features- Real-time Slot Management: Dynamic slot generation based on doctor schedules- Conflict Prevention: Prevents double booking with database constraints- Status Tracking: Complete appointment lifecycle management- File Attachments: Support for medical reports and prescriptions- Payment Integration: Ready for payment gateway integration###

Appointment States- Pending: Awaiting receptionist confirmation- Confirmed: Approved and scheduled- In-Progress: Currently ongoing- Completed: Successfully finished- Cancelled: Cancelled by patient/admin- Rejected: Rejected by receptionist- No-Show: Patient didn't attend## 📁 File Upload System### Supported Features- Multiple File Types: Images (JPG, PNG) and PDFs- Size Validation: Maximum 10MB per file- Secure Storage: Organized file structure with unique naming- Access Control:

Role-based file access permissions### File Organization

```
uploads/├── appointments/│   ├─
```

## 🔒 Security Features### Multi-layer Security- JWT Authentication: Secure token-based access- OTP Verification: SMS-based mobile verification- Rate Limiting: Prevents abuse and DoS attacks- Input Validation: Comprehensive data sanitization- CORS Protection: Controlled cross-origin requests- File Upload Security: Type and size validation## 🧪 Testing### Health Check

```
bashcurl http://localhost:3001/
```

### API Testing

```
bash# Send OTPcurl -X POST http
```

## 🚀 Deployment### Quick Deployment Commands

```
bash# Production buildnpm run s
```

### Deployment Platforms- Heroku: Easy deployment with git integration- AWS/DigitalOcean: VPS deployment with full control- Docker: Containerized deployment- Vercel/Netlify: Serverless deployment options## 📊 Performance & Monitoring###

Database Optimization- Indexed queries for fast lookups- Aggregation pipelines for complex reports- Connection pooling for scalability### Monitoring Ready- Comprehensive error logging- Request/response logging- Performance metrics collection- Health check endpoints## 🤝 ContributingWe welcome contributions! Please see our Contributing Guidelines for details.### Development Setup

```
bashgit clone cd arthomed-backe
```

## 📝 LicenseThis project is licensed under the MIT License -

see the LICENSE file for details.## 📞 Support- Documentation: Complete API docs at `/api-docs`- Issues: GitHub Issues for bug reports- Email: support@arthomed.com---🏥 Built with ❤️ for healthcare professionals and patients

Arthomed Backend - Empowering healthcare through technology

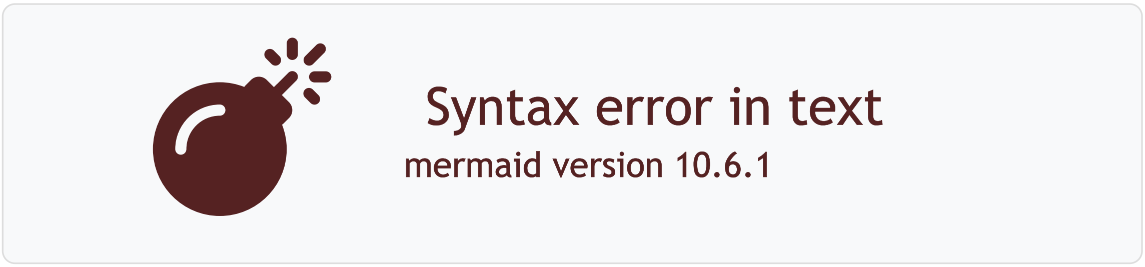
2. Technical Documentation

Arthomed Healthcare Backend - Technical Documentation

Table of Contents

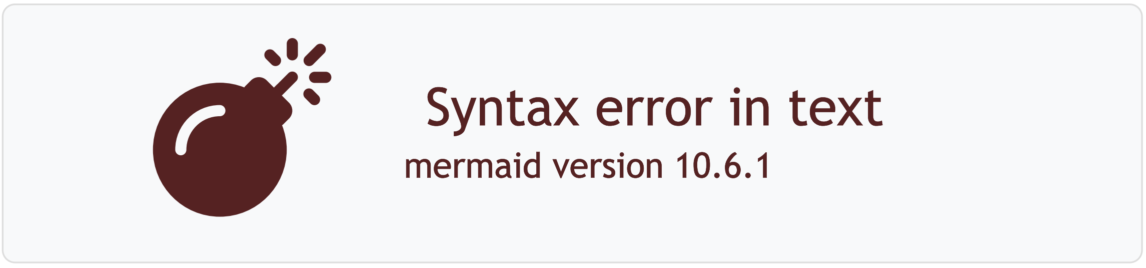
System Architecture

High-Level Architecture Diagram



Syntax error in text
mermaid version 10.6.1

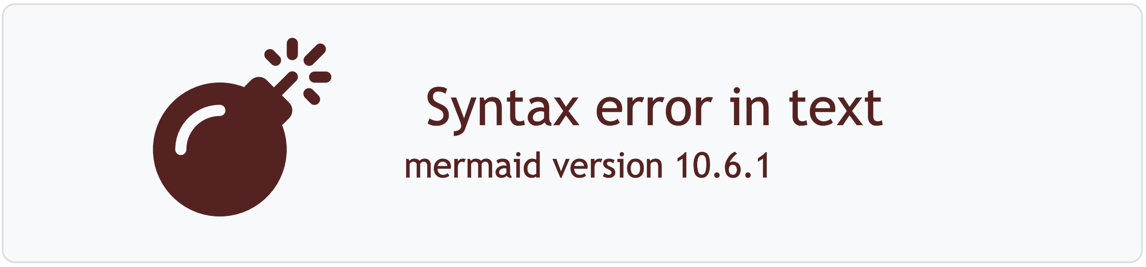
Component Interaction Flow



Syntax error in text
mermaid version 10.6.1

Database Design

Entity Relationship Diagram



Syntax error in text
mermaid version 10.6.1

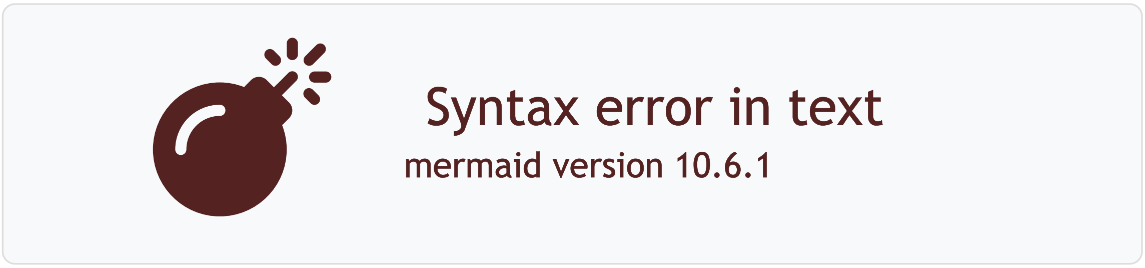
Database Schema Details

#### User Collection Schema

```
const userSchema = new mongoose.Schema({
  // Basic Information
  mobileNumber: {
    type: String,
    required: true,
    unique: true,
    match: /^[6-9]\d{9}$/,
    index: true
  },
  name: {
    type: String,
    required: true,
    trim: true,
    maxlength: 100
  },
  email: {
    type: String,
    lowercase: true,
    trim: true,
    match: /^\w+([.-]?\w+)@\w+([.-]?\w+)(\.\w{2,3})+$/
  },

  // Role & Status
  role: {
    type: String,
    enum: ['admin', 'doctor', 'receptionist', 'patient'],
    default: 'patient',
    index: true
  },
  isActive: { type: Boolean, default: true },
  isVerified: { type: Boolean, default: false },

  // Personal Information
  profile: {
    dateOfBirth: Date,
    gender: { type: String, enum: ['male', 'female', 'other'] },
    address: {
      street: String,
      city: String,
      state: String,
      pincode: String,
      country: { type: String, default: 'India' }
    },
    emergencyContact: {
      name: String,
      relationship: String,
      mobileNumber: String
    }
  },

  // Doctor-specific Information
  doctorInfo: {
    specialization: String,
    qualification: String,
    experience: Number,
    consultationFee: Number,
    registrationNumber: String,
    schedule: [{
      day: {
        type: String,
        enum: ['monday', 'tuesday', 'wednesday', 'thursday', 'friday', 'saturd
      },
      startTime: String,
      endTime: String,
      isAvailable: { type: Boolean, default: true }
    }]
  },

  // Patient-specific Information
  patientInfo: {
    bloodGroup: String,
    allergies: [String],
    medicalHistory: [{
      condition: String,
      diagnosedDate: Date,
      treatment: String,
      doctor: String
    }],
    emergencyContact: {
      name: String,
      relationship: String,
      mobileNumber: String
    }
  }
}, {
  timestamps: true,
  toJSON: { virtuals: true },
  toObject: { virtuals: true }
});

// Indexes for performance
userSchema.index({ email: 1 });
userSchema.index({ role: 1 });
userSchema.index({ 'doctorInfo.specialization': 1 });
userSchema.index({ isActive: 1, isVerified: 1 });
```

API Architecture

RESTful API Design Pattern



API Response Standards

#### Success Response Format

```
{
  "success": true,
  "message": "Operation completed successfully",
  "data": {
    // Response data object
  },
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

#### Error Response Format

```
{
  "success": false,
  "message": "Error description",
  "error": {
    "code": "ERROR_CODE",
    "statusCode": 400,
    "details": "Detailed error information"
  },
  "errors": [
    // Validation errors array
  ],
  "timestamp": "2024-01-15T10:30:00.000Z"
}
```

#### Paginated Response Format

```
{
  "success": true,
  "data": {
    "items": [
      // Array of items
    ],
    "pagination": {
      "current": 1,
      "pages": 5,
      "total": 50,
      "limit": 10,
      "hasNext": true,
      "hasPrev": false
    }
  }
}
```

Authentication System

OTP-based Authentication Flow

Syntax error in text
mermaid version 10.6.1

JWT Token Management

Syntax error in text
mermaid version 10.6.1

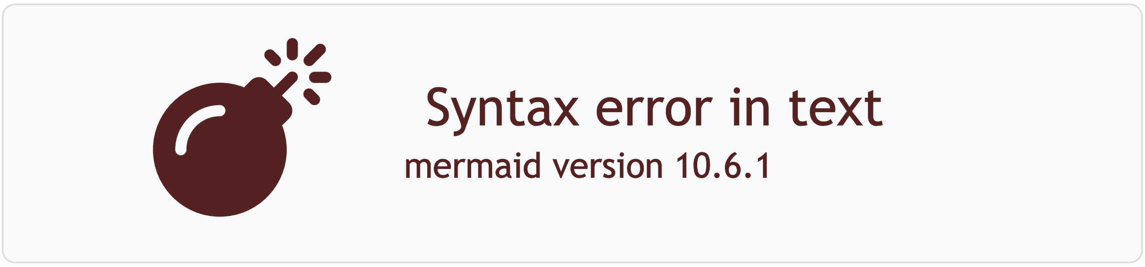
Authentication Middleware Implementation

```
const authenticate = async (req, res, next) => {
  try {
    // Extract token from Authorization header
    const token = extractTokenFromHeader(req.headers.authorization);

    if (!token) {
      return res.status(401).json({
        success: false,
        message: 'Access denied. No token provided.'
      });
    }

    // Verify token
    const decoded = verifyToken(token);

    // Find user and check if still exists and is active
    const user = await User.findById(decoded.id).select('-__v');

    if (!user) {
      return res.status(401).json({
        success: false,
        message: 'Invalid token. User not found.'
      });
    }

    if (!user.isActive) {
      return res.status(401).json({
        success: false,
        message: 'Account has been deactivated.'
      });
    }

    // Add user to request object
    req.user = user;
    next();
  } catch (error) {
    if (error.name === 'TokenExpiredError') {
      return res.status(401).json({
        success: false,
        message: 'Token has expired.',
        error: { code: 'TOKEN_EXPIRED' }
      });
    }

    return res.status(401).json({
      success: false,
      message: 'Invalid token.',
      error: { code: 'INVALID_TOKEN' }
    });
  }
};
```

Appointment Management

Appointment Lifecycle Management

Slot Management System


Syntax error in text
mermaid version 10.6.1

Appointment Booking Flow

```
const bookAppointment = async (req, res, next) => {
  try {
    const { doctorId, appointmentDate, appointmentTime, purposeOfVisit, reason
    const patientId = req.user._id;

    // Check if slot is available
    const slot = await Slot.findOne({
      doctor: doctorId,
      date: appointmentDate,
      startTime: appointmentTime,
      isAvailable: true
    });

    if (!slot) {
      return next(new AppError('Selected time slot is not available', 400));
    }

    // Check for existing appointments (prevent double booking)
    const existingAppointment = await Appointment.findOne({
      doctor: doctorId,
      appointmentDate: appointmentDate,
      appointmentTime: appointmentTime,
      status: { $nin: ['cancelled', 'rejected', 'no-show'] }
    });

    if (existingAppointment) {
      return next(new AppError('Time slot already booked', 400));
    }

    // Create appointment
    const appointment = new Appointment({
      patient: patientId,
      doctor: doctorId,
      appointmentDate,
      appointmentTime,
      purposeOfVisit,
      reason,
      status: 'pending',
      createdBy: patientId
    });

    // Handle file uploads if present
    if (req.files && req.files.length > 0) {
      appointment.images = req.files.map(file => ({
        filename: file.filename,
        originalName: file.originalname,
        path: file.path,
        size: file.size,
        mimeType: file.mimetype
      }));
    }

    await appointment.save();

    // Update slot availability
    slot.isAvailable = false;
    slot.appointment = appointment._id;
    slot.bookedPatients += 1;
    await slot.save();

    // Populate appointment details for response
    await appointment.populate([
      { path: 'patient', select: 'name mobileNumber' },
      { path: 'doctor', select: 'name doctorInfo.specialization' }
    ]);

    res.status(201).json({
      success: true,
      message: 'Appointment booked successfully',
      data: { appointment }
    });
  } catch (error) {
    next(error);
  }
};
```

File Upload System

File Upload Architecture

Syntax error in text
mermaid version 10.6.1

File Storage Structure

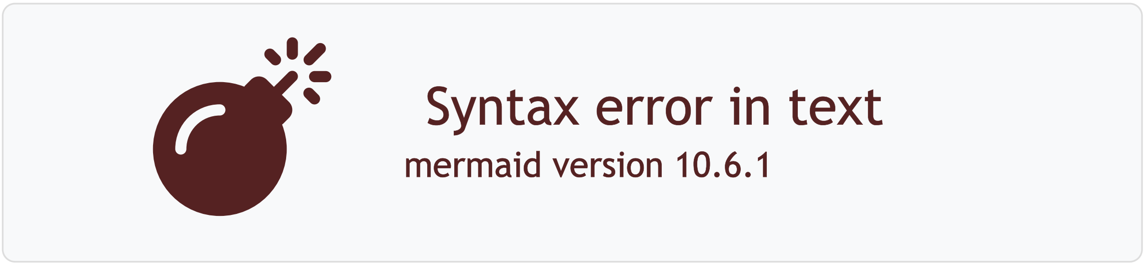
```
uploads/
├── appointments/
│   ├── prescriptions/
│   │   ├── 2024/
│   │   │   ├── 01/
│   │   │   │   └── appointment_672b1234_1642234567890_prescription.jpg
│   │   │   └── 02/
│   │   └── 2025/
│   └── reports/
│       ├── 2024/
│       │   ├── 01/
│       │   │   └── appointment_672b5678_1642234567890_report.pdf
│       │   └── 02/
│       └── 2025/
└── profiles/
    ├── 2024/
    │   ├── 01/
    │   │   └── user_672b9012_1642234567890_avatar.jpg
    │   └── 02/
    └── 2025/
```

File Upload Implementation

```
const multer = require('multer');
const path = require('path');
const fs = require('fs').promises;

// Storage configuration
const storage = multer.diskStorage({
  destination: async (req, file, cb) => {
    const uploadType = req.params.type || 'appointments';
    const year = new Date().getFullYear();
    const month = String(new Date().getMonth() + 1).padStart(2, '0');

    const uploadPath = path.join('uploads', uploadType, year.toString(), month

    // Create directory if it doesn't exist
    try {
      await fs.mkdir(uploadPath, { recursive: true });
      cb(null, uploadPath);
    } catch (error) {
      cb(error);
    }
  },
  filename: (req, file, cb) => {
    const uniqueSuffix = Date.now() + '-' + Math.round(Math.random() * 1E9);
    const sanitizedName = file.originalname.replace(/[^a-zA-Z0-9.]/g, '_');
    const filename = ${req.user._id}_${uniqueSuffix}_${sanitizedName};
    cb(null, filename);
  }
});

// File filter
const fileFilter = (req, file, cb) => {
  const allowedTypes = ['image/jpeg', 'image/jpg', 'image/png', 'application/p
  const maxSize = 10  1024  1024; // 10MB

  if (allowedTypes.includes(file.mimetype)) {
    cb(null, true);
  } else {
    cb(new Error('Invalid file type. Only JPG, PNG, and PDF files are allowed.
  }
};

const upload = multer({
  storage: storage,
  fileFilter: fileFilter,
  limits: {
    fileSize: 10  1024  1024, // 10MB
    files: 5 // Maximum 5 files
  }
});
```

Security Implementation

Security Architecture


Syntax error in text

mermaid version 10.6.1

Security Middleware Stack

```
// Security middleware configuration
const securityMiddleware = (app) => {
  // Basic security headers
  app.use(helmet({
    contentSecurityPolicy: {
      directives: {
        defaultSrc: ["'self'"],
        styleSrc: ["'self'", "'unsafe-inline'"],
        scriptSrc: ["'self'"],
        imgSrc: ["'self'", "data:", "https:"],
      },
    },
    hsts: {
      maxAge: 31536000,
      includeSubDomains: true,
      preload: true
    }
  }));

  // Rate limiting
  const limiter = rateLimit({
    windowMs: parseInt(process.env.RATE_LIMIT_WINDOW_MS) || 15  60  1000, // 1
    max: parseInt(process.env.RATE_LIMIT_MAX_REQUESTS) || 100, // limit each I
    message: {
      error: 'Too many requests from this IP, please try again later.',
    },
    standardHeaders: true,
    legacyHeaders: false,
  });
  app.use(limiter);

  // CORS configuration
  app.use(cors({
    origin: process.env.NODE_ENV === 'production'
      ? ['https://your-frontend-domain.com']
      : ['http://localhost:3000', 'http://localhost:19006'], // React Native M
    credentials: true,
    methods: ['GET', 'POST', 'PUT', 'DELETE', 'OPTIONS'],
    allowedHeaders: ['Content-Type', 'Authorization'],
  }));

  // Body parsing middleware with limits
  app.use(express.json({ limit: '10mb' }));
  app.use(express.urlencoded({ extended: true, limit: '10mb' }));
};
```

Input Validation System

```
const { body, param, query, validationResult } = require('express-validator');

// Mobile number validation
const validateMobileNumber = () => [
  body('mobileNumber')
    .isLength({ min: 10, max: 10 })
    .withMessage('Mobile number must be exactly 10 digits')
    .matches(/^[6-9]\d{9}$/)
    .withMessage('Please enter a valid Indian mobile number')
    .customSanitizer(value => value.replace(/\D/g, '')) // Remove non-digits
];

// OTP validation
const validateOTP = () => [
  body('otp')
    .isLength({ min: 6, max: 6 })
    .withMessage('OTP must be exactly 6 digits')
    .matches(/^\d{6}$/)
    .withMessage('OTP must contain only numbers')
];

// User registration validation
const validateUserRegistration = () => [
  body('name')
    .trim()
    .isLength({ min: 2, max: 100 })
    .withMessage('Name must be between 2 and 100 characters')
    .matches(/^[a-zA-Z\s]+$/)
    .withMessage('Name can only contain letters and spaces'),

  body('email')
    .optional()
    .isEmail()
    .withMessage('Please enter a valid email address')
    .normalizeEmail(),

  ...validateMobileNumber(),
  ...validateOTP()
];

// Validation error handler
const handleValidationErrors = (req, res, next) => {
  const errors = validationResult(req);

  if (!errors.isEmpty()) {
    const errorMessages = errors.array().map(error => ({
      field: error.path,
      message: error.msg,
      value: error.value,
    }));

    return res.status(400).json({
      success: false,
      message: 'Validation failed',
      errors: errorMessages,
    });
  }

  next();
};
```

Performance Optimization

Database Performance

Syntax error in text
mermaid version 10.6.1

Database Indexes Implementation

```
// User collection indexes
userSchema.index({ mobileNumber: 1 }, { unique: true }); // Unique index
userSchema.index({ email: 1 });
userSchema.index({ role: 1 });
userSchema.index({ 'doctorInfo.specialization': 1 });
userSchema.index({ isActive: 1, isVerified: 1 }); // Compound index

// Appointment collection indexes
appointmentSchema.index({ patient: 1, appointmentDate: 1 });
appointmentSchema.index({ status: 1, appointmentDate: 1 });
appointmentSchema.index({ appointmentDate: 1, appointmentTime: 1 });
appointmentSchema.index({ createdAt: 1 });

// Compound unique index to prevent double booking
appointmentSchema.index(
  { doctor: 1, appointmentDate: 1, appointmentTime: 1 },
  {
    unique: true,
    partialFilterExpression: {
      status: { $nin: ['cancelled', 'no-show'] }
    }
  }
);

// Slot collection indexes
slotSchema.index({ date: 1, isAvailable: 1 });
slotSchema.index({ doctor: 1, isAvailable: 1 });
slotSchema.index(
  { doctor: 1, date: 1, startTime: 1 },
  { unique: true }
);
```

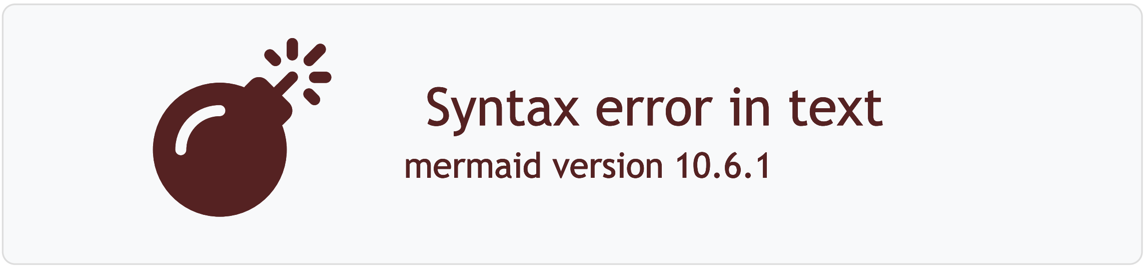
```
// OTP collection indexes with TTL
otpSchema.index({ expiresAt: 1 }, { expireAfterSeconds: 0 }); // TTL index
otpSchema.index({ mobileNumber: 1, createdAt: 1 });
```

Pagination Implementation

```
const getPaginatedResults = async (model, query, options) => {
  const {
    page = 1,
    limit = 10,
    sort = { createdAt: -1 },
    populate = null,
    select = null
  } = options;

  const skip = (parseInt(page) - 1) * parseInt(limit);

  // Build query
  let queryBuilder = model.find(query);

  if (select) queryBuilder = queryBuilder.select(select);
  if (populate) queryBuilder = queryBuilder.populate(populate);

  // Execute queries in parallel
  const [items, total] = await Promise.all([
    queryBuilder
      .sort(sort)
      .skip(skip)
      .limit(parseInt(limit))
      .lean(),
    model.countDocuments(query)
  ]);

  const pages = Math.ceil(total / limit);

  return {
    items,
    pagination: {
      current: parseInt(page),
      pages,
      total,
      limit: parseInt(limit),
      hasNext: page < pages,
      hasPrev: page > 1
    }
  };
};
```

Error Handling

Error Handling Architecture



Global Error Handler Implementation

```
class AppError extends Error {
  constructor(message, statusCode) {
    super(message);
    this.statusCode = statusCode;
    this.status = ${statusCode}.startsWith('4') ? 'fail' : 'error';
    this.isOperational = true;

    Error.captureStackTrace(this, this.constructor);
  }
}

const errorHandler = (err, req, res, next) => {
  let error = { ...err };
  error.message = err.message;

  // Log error
  console.error('Error:', err);

  // Mongoose bad ObjectId
  if (err.name === 'CastError') {
    const message = 'Resource not found';
    error = new AppError(message, 404);
  }

  // Mongoose duplicate key
  if (err.code === 11000) {
    let message = 'Duplicate field value entered';

    // Extract field name from error
    const field = Object.keys(err.keyValue)[0];
    if (field === 'mobileNumber') {
      message = 'Mobile number is already registered';
    } else if (field === 'email') {
      message = 'Email address is already registered';
    }
    error = new AppError(message, 400);
  }

  // Mongoose validation error
  if (err.name === 'ValidationError') {
    const message = Object.values(err.errors).map(val => val.message).join(',
    error = new AppError(message, 400);
  }

  // JWT errors
  if (err.name === 'JsonWebTokenError') {
    const message = 'Invalid token. Please log in again.';
    error = new AppError(message, 401);
  }

  if (err.name === 'TokenExpiredError') {
    const message = 'Your token has expired. Please log in again.';
    error = new AppError(message, 401);
  }

  // Multer errors
  if (err.code === 'LIMIT_FILE_SIZE') {
    const message = 'File too large. Maximum size allowed is 10MB.';
    error = new AppError(message, 400);
  }

  if (err.code === 'LIMIT_FILE_COUNT') {
    const message = 'Too many files. Maximum 5 files allowed.';
    error = new AppError(message, 400);
  }

  res.status(error.statusCode || 500).json({
    success: false,
    message: error.message || 'Internal server error',
    ...(process.env.NODE_ENV === 'development' && {
      error: error,
      stack: err.stack
    })
  });
};

// Async handler wrapper
const asyncHandler = (fn) => (req, res, next) => {
  Promise.resolve(fn(req, res, next)).catch(next);
};
```

Deployment Architecture

Production Deployment Flow

Environment Configuration

```
// Production environment variables
const productionConfig = {
  // Server
  NODE_ENV: 'production',
  PORT: process.env.PORT || 5000,

  // Database
  MONGODB_URI: process.env.MONGODB_URI,

  // Security
  JWT_SECRET: process.env.JWT_SECRET,
  JWT_REFRESH_SECRET: process.env.JWT_REFRESH_SECRET,

  // External Services
  TWILIO_ACCOUNT_SID: process.env.TWILIO_ACCOUNT_SID,
  TWILIO_AUTH_TOKEN: process.env.TWILIO_AUTH_TOKEN,

  // Performance
  RATE_LIMIT_WINDOW_MS: 900000, // 15 minutes
  RATE_LIMIT_MAX_REQUESTS: 100,

  // Monitoring
  LOG_LEVEL: 'info',
  ENABLE_METRICS: true
};

// Health check endpoint
app.get('/health', (req, res) => {
  const healthcheck = {
    uptime: process.uptime(),
    message: 'OK',
    timestamp: Date.now(),
    environment: process.env.NODE_ENV,
    version: process.env.npm_package_version
  };

  try {
    res.send(healthcheck);
  } catch (error) {
    healthcheck.message = error;
    res.status(503).send();
  }
});
```

Monitoring and Logging

```
// Request logging middleware
const requestLogger = (req, res, next) => {
  const startTime = Date.now();

  res.on('finish', () => {
    const duration = Date.now() - startTime;
    const logData = {
      method: req.method,
      url: req.url,
      statusCode: res.statusCode,
      duration: ${duration}ms,
      ip: req.ip,
      userAgent: req.get('User-Agent'),
      userId: req.user ? req.user._id : 'anonymous',
      timestamp: new Date().toISOString()
    };

    if (res.statusCode >= 400) {
      console.error('Request Error:', logData);
    } else {
      console.log('Request:', logData);
    }
  });

  next();
};

// Performance monitoring
const performanceMonitor = {
  trackApiResponse: (endpoint, duration, statusCode) => {
    // Send metrics to monitoring service
    console.log(API Performance: ${endpoint} - ${duration}ms - ${statusCode});
  },

  trackError: (error, context) => {
    // Send error to error tracking service
    console.error('Application Error:', {
      message: error.message,
      stack: error.stack,
      context,
      timestamp: new Date().toISOString()
    });
  }
};
```

Integration Guidelines

React Native Integration

```
// API service configuration for React Native
class ApiService {
  constructor() {
    this.baseURL = 'http://localhost:3001/api';
    this.token = null;
  }

  setToken(token) {
    this.token = token;
  }

  async request(endpoint, options = {}) {
    const url = ${this.baseURL}${endpoint};
    const config = {
      headers: {
        'Content-Type': 'application/json',
        ...(this.token && { Authorization: Bearer ${this.token} }),
        ...options.headers,
      },
      ...options,
    };

    try {
      const response = await fetch(url, config);
      const data = await response.json();

      if (!response.ok) {
        throw new Error(data.message || 'Request failed');
      }

      return data;
    } catch (error) {
      console.error('API Request Error:', error);
      throw error;
    }
  }

  // Authentication methods
  async sendOTP(mobileNumber) {
    return this.request('/auth/send-otp', {
      method: 'POST',
      body: JSON.stringify({ mobileNumber }),
    });
  }
  async verifyOTP(mobileNumber, otp) {
    return this.request('/auth/verify-otp', {
      method: 'POST',
      body: JSON.stringify({ mobileNumber, otp }),
    });
  }

  // Appointment methods
  async bookAppointment(appointmentData) {
    return this.request('/appointments/book', {
      method: 'POST',
      body: JSON.stringify(appointmentData),
    });
  }

  async getAvailableSlots(doctorId, date) {
    return this.request(/appointments/slots?doctorId=${doctorId}&date=${date});
  }
}
```

This comprehensive technical documentation covers all aspects of the Arthomed healthcare backend system, from high-level architecture to implementation details. It serves as a complete reference for developers, system administrators, and stakeholders involved in the project.

3. API Documentation

Arthomed Backend - API Documentation

Table of Contents

Authentication APIs

Send OTP

Send OTP to mobile number for authentication.

Endpoint: `POST /api/auth/send-otp`

Request Body:

```
{
  "mobileNumber": "9876543210"
}
```

Success Response (200):

```
{
  "success": true,
  "message": "OTP sent successfully",
  "data": {
    "message": "OTP sent to your mobile number",
    "expiresIn": 300
  }
}
```

Error Responses:
- `400 Bad Request` - Invalid mobile number format
- `429 Too Many Requests` - Rate limit exceeded
- `500 Internal Server Error` - SMS service failure

Verify OTP

Verify OTP and authenticate user.

Endpoint: `POST /api/auth/verify-otp`

Request Body:

```
{
  "mobileNumber": "9876543210",
  "otp": "123456"
}
```

Success Response (200):

```
{
  "success": true,
  "message": "OTP verified successfully",
  "data": {
    "user": {
      "_id": "64f1234567890abcdef12345",
      "mobileNumber": "9876543210",
      "name": "John Doe",
      "role": "patient",
      "isVerified": true
    },
    "tokens": {
      "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
      "expiresIn": 3600
    }
  }
}
```

Error Responses:
- `400 Bad Request` - Invalid OTP
- `401 Unauthorized` - OTP expired or maximum attempts exceeded
- `404 Not Found` - OTP not found

Refresh Token

Refresh access token using refresh token.

Endpoint: `POST /api/auth/refresh-token`

Request Body:

```
{
  "refreshToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9..."
}
```

Success Response (200):

```
{
  "success": true,
  "message": "Token refreshed successfully",
  "data": {
    "accessToken": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
    "expiresIn": 3600
  }
}
```

Logout

Logout user and invalidate tokens.

Endpoint: `POST /api/auth/logout`

Headers:

```
Authorization: Bearer
```

Success Response (200):

```json
{
  "success": true,
  "message": "Logged out successfully"
}
```

User Management APIs

Get Current User Profile

Get the authenticated user's profile information.

Endpoint: `GET /api/users/profile`

Headers:

```
Authorization: Bearer
```

Success Response (200):

```json
{
  "success": true,
  "data": {
    "user": {
      "_id": "64f1234567890abcdef12345",
      "mobileNumber": "9876543210",
      "name": "John Doe",
      "email": "john@example.com",
      "role": "patient",
      "isActive": true,
      "isVerified": true,
      "profile": {
        "dateOfBirth": "1990-01-15",
        "gender": "male",
        "address": {
          "street": "123 Main St",
          "city": "Mumbai",
          "state": "Maharashtra",
          "pincode": "400001",
          "country": "India"
        }
      },
      "patientInfo": {
        "bloodGroup": "O+",
        "allergies": ["Penicillin"],
        "medicalHistory": []
      },
      "createdAt": "2024-01-15T10:30:00.000Z",
      "updatedAt": "2024-01-15T10:30:00.000Z"
    }
  }
}
```

Update User Profile

Update the authenticated user's profile information.

Endpoint: `PUT /api/users/profile`

Headers:

```
Authorization: Bearer
```

Request Body:

```
{
  "name": "John Smith",
  "email": "johnsmith@example.com",
  "profile": {
    "dateOfBirth": "1990-01-15",
    "gender": "male",
    "address": {
      "street": "456 New St",
      "city": "Mumbai",
      "state": "Maharashtra",
      "pincode": "400002",
      "country": "India"
    }
  },
  "patientInfo": {
    "bloodGroup": "O+",
    "allergies": ["Penicillin", "Dust"]
  }
}
```

Success Response (200):

```
{
  "success": true,
  "message": "Profile updated successfully",
  "data": {
    "user": {
      // Updated user object
    }
  }
}
```

Get All Users (Admin Only)

Retrieve all users with pagination and filtering.

Endpoint: `GET /api/users`

Headers:

```
Authorization: Bearer
```

Query Parameters:
- `page` (optional) - Page number (default: 1)
- `limit` (optional) - Items per page (default: 10)
- `role` (optional) - Filter by role
- `search` (optional) - Search by name or mobile number
- `isActive` (optional) - Filter by active status

Example: `GET /api/users?page=1&limit=10&role=doctor&search=john`

Success Response (200):

```json
{
  "success": true,
  "data": {
    "items": [
      {
        "_id": "64f1234567890abcdef12345",
        "mobileNumber": "9876543210",
        "name": "Dr. John Doe",
        "email": "john@example.com",
        "role": "doctor",
        "isActive": true,
        "isVerified": true,
        "doctorInfo": {
          "specialization": "Cardiology",
          "qualification": "MBBS, MD",
          "experience": 10,
          "consultationFee": 500
        },
        "createdAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "pagination": {
      "current": 1,
      "pages": 5,
      "total": 50,
      "limit": 10,
      "hasNext": true,
      "hasPrev": false
    }
  }
}
```

Get Doctors List

Get list of all doctors with their specializations.

Endpoint: `GET /api/users/doctors`

Query Parameters:
- `specialization` (optional) - Filter by specialization
- `page` (optional) - Page number
- `limit` (optional) - Items per page

Success Response (200):

```
{
  "success": true,
  "data": {
    "items": [
      {
        "_id": "64f1234567890abcdef12345",
        "name": "Dr. John Doe",
        "doctorInfo": {
          "specialization": "Cardiology",
          "qualification": "MBBS, MD",
          "experience": 10,
          "consultationFee": 500,
          "schedule": [
            {
              "day": "monday",
              "startTime": "09:00",
              "endTime": "17:00",
              "isAvailable": true
            }
          ]
        },
        "isActive": true
      }
    ],
    "pagination": {
      "current": 1,
      "pages": 3,
      "total": 25,
      "limit": 10
    }
  }
}
```

Appointment Management APIs

Book Appointment

Book a new appointment with a doctor.

Endpoint: `POST /api/appointments/book`

Headers:

```
Authorization: Bearer
Content-Type: multipart/form-data
```

Form Data:

```
doctorId: 64f1234567890abcdef12345
appointmentDate: 2024-02-15
appointmentTime: 10:00
purposeOfVisit: Regular checkup
reason: Annual health checkup
symptoms: No specific symptoms
images: [file1.jpg, file2.pdf]
```

Success Response (201):

```
{
  "success": true,
  "message": "Appointment booked successfully",
  "data": {
    "appointment": {
      "_id": "64f9876543210fedcba09876",
      "patient": {
        "_id": "64f1234567890abcdef12345",
        "name": "John Doe",
        "mobileNumber": "9876543210"
      },
      "doctor": {
        "_id": "64f1234567890abcdef67890",
        "name": "Dr. Jane Smith",
        "doctorInfo": {
          "specialization": "Cardiology"
        }
      },
      "appointmentDate": "2024-02-15",
      "appointmentTime": "10:00",
      "status": "pending",
      "purposeOfVisit": "Regular checkup",
      "reason": "Annual health checkup",
      "symptoms": "No specific symptoms",
      "images": [
        {
          "filename": "64f1234567890abcdef12345_1642234567890_report.jpg",
          "originalName": "medical_report.jpg",
          "path": "uploads/appointments/2024/02/64f1234567890abcdef12345_16422
          "size": 1024000,
          "mimeType": "image/jpeg"
        }
      ],
      "createdAt": "2024-01-15T10:30:00.000Z"
    }
  }
}
```

Get User Appointments

Get appointments for the authenticated user.

Endpoint: `GET /api/appointments/my-appointments`

Headers:

```
Authorization: Bearer
```

Query Parameters:
- `status` (optional) - Filter by status
- `page` (optional) - Page number
- `limit` (optional) - Items per page
- `startDate` (optional) - Filter from date
- `endDate` (optional) - Filter to date

Success Response (200):

```
{
  "success": true,
  "data": {
    "items": [
      {
        "_id": "64f9876543210fedcba09876",
        "doctor": {
          "_id": "64f1234567890abcdef67890",
          "name": "Dr. Jane Smith",
          "doctorInfo": {
            "specialization": "Cardiology",
            "consultationFee": 500
          }
        },
        "appointmentDate": "2024-02-15",
        "appointmentTime": "10:00",
        "status": "confirmed",
        "purposeOfVisit": "Regular checkup",
        "createdAt": "2024-01-15T10:30:00.000Z"
      }
    ],
    "pagination": {
      "current": 1,
      "pages": 2,
      "total": 15,
      "limit": 10
    }
  }
}
```

Get Available Slots

Get available appointment slots for a doctor on a specific date.

Endpoint: `GET /api/appointments/slots`

Query Parameters:
- `doctorId` (required) - Doctor's ID
- `date` (required) - Date in YYYY-MM-DD format

Example: `GET /api/appointments/slots?doctorId=64f1234567890abcdef67890&date=2024-02-15`

Success Response (200):

```
  {
    "success": true,
    "data": {
      "doctor": {
        "_id": "64f1234567890abcdef67890",
        "name": "Dr. Jane Smith",
        "doctorInfo": {
          "specialization": "Cardiology",
          "consultationFee": 500
        }
      },
      "date": "2024-02-15",
      "availableSlots": [
        {
          "_id": "64f555555555555555555555",
          "startTime": "09:00",
          "endTime": "09:30",
          "isAvailable": true,
          "consultationFee": 500
        },
        {
          "_id": "64f666666666666666666666",
          "startTime": "10:00",
          "endTime": "10:30",
          "isAvailable": true,
          "consultationFee": 500
        }
      ],
      "bookedSlots": [
        {
          "startTime": "11:00",
          "endTime": "11:30",
          "isAvailable": false
        }
      ]
    }
  }
```

Cancel Appointment

Cancel an existing appointment.

Endpoint: `PUT /api/appointments/:appointmentId/cancel`

Headers:

```
Authorization: Bearer
```

Request Body:

```
{
  "reason": "Personal emergency"
}
```

Success Response (200):

```
{
  "success": true,
  "message": "Appointment cancelled successfully",
  "data": {
    "appointment": {
      "_id": "64f9876543210fedcba09876",
      "status": "cancelled",
      "cancellationReason": "Personal emergency",
      "refundInfo": {
        "refundAmount": 250,
        "refundStatus": "processed",
        "refundDate": "2024-01-15T10:30:00.000Z"
      }
    }
  }
}
```

Update Appointment Status (Doctor/Admin)

Update appointment status by doctor or admin.

Endpoint: `PUT /api/appointments/:appointmentId/status`

Headers:

```
Authorization: Bearer
```

Request Body:

```
{
  "status": "completed",
  "notes": "Patient is healthy. Prescribed vitamins.",
  "prescription": "Vitamin D3 - 1 tablet daily for 30 days"
}
```

Success Response (200):

```
{
  "success": true,
  "message": "Appointment status updated successfully",
  "data": {
    "appointment": {
      "_id": "64f9876543210fedcba09876",
      "status": "completed",
      "notes": "Patient is healthy. Prescribed vitamins.",
      "prescription": "Vitamin D3 - 1 tablet daily for 30 days",
      "completedAt": "2024-01-15T10:30:00.000Z"
    }
  }
}
```

File Upload APIs

Upload Appointment Files

Upload medical documents/images for appointments.

Endpoint: `POST /api/files/upload/appointments`

Headers:

```
Authorization: Bearer
Content-Type: multipart/form-data
```

Form Data:

```
files: [file1.jpg, file2.pdf, file3.png]
appointmentId: 64f9876543210fedcba09876
```

Success Response (200):

```
{
  "success": true,
  "message": "Files uploaded successfully",
  "data": {
    "uploadedFiles": [
      {
        "filename": "64f1234567890abcdef12345_1642234567890_report.jpg",
        "originalName": "medical_report.jpg",
        "path": "uploads/appointments/2024/02/64f1234567890abcdef12345_1642234
        "size": 1024000,
        "mimeType": "image/jpeg",
        "url": "/api/files/uploads/appointments/2024/02/64f1234567890abcdef123
      }
    ]
    }
  }
```

Get File

Retrieve uploaded file.

Endpoint: `GET /api/files/:filename`

Headers:

```
Authorization: Bearer
```

Success Response (200):
- Returns the file content with appropriate Content-Type header

---

Delete File

Delete an uploaded file.

Endpoint: `DELETE /api/files/:filename`

Headers:

```
Authorization: Bearer
```

Success Response (200):

```
{
  "success": true,
  "message": "File deleted successfully"
}
```

Admin APIs

Get Dashboard Statistics

Get system statistics for admin dashboard.

Endpoint: `GET /api/admin/dashboard`

Headers:

```
Authorization: Bearer
```

Success Response (200):

```json
{
  "success": true,
  "data": {
    "statistics": {
      "totalUsers": 1250,
      "totalDoctors": 45,
      "totalPatients": 1180,
      "totalAppointments": 3456,
      "todayAppointments": 25,
      "pendingAppointments": 12,
      "completedAppointments": 3200,
      "revenue": {
        "today": 12500,
        "thisMonth": 345000,
        "thisYear": 2450000
      }
    },
    "recentAppointments": [
      {
        "_id": "64f9876543210fedcba09876",
        "patient": {
          "name": "John Doe",
          "mobileNumber": "9876543210"
        },
        "doctor": {
          "name": "Dr. Jane Smith"
        },
        "appointmentDate": "2024-02-15",
        "appointmentTime": "10:00",
        "status": "confirmed"
      }
    ],
    "topDoctors": [
      {
        "_id": "64f1234567890abcdef67890",
        "name": "Dr. Jane Smith",
        "specialization": "Cardiology",
        "appointmentCount": 156,
        "rating": 4.8
      }
    ]
  }
}
```

Manage User Status

Activate or deactivate user accounts.

Endpoint: `PUT /api/admin/users/:userId/status`

Headers:

```
Authorization: Bearer
```

Request Body:

```
{
  "isActive": false,
  "reason": "Violating terms of service"
}
```

Success Response (200):

```
{
  "success": true,
  "message": "User status updated successfully",
  "data": {
    "user": {
      "_id": "64f1234567890abcdef12345",
      "isActive": false,
      "statusUpdatedAt": "2024-01-15T10:30:00.000Z",
      "statusUpdatedBy": "64f9999999999999999999999"
    }
  }
}
```

Error Codes Reference

HTTP Status Codes

Status Code	Description
200	OK - Request successful
201	Created - Resource created successfully
400	Bad Request - Invalid request data
401	Unauthorized - Authentication required
403	Forbidden - Insufficient permissions
404	Not Found - Resource not found
409	Conflict - Resource already exists
422	Unprocessable Entity - Validation failed
429	Too Many Requests - Rate limit exceeded
500	Internal Server Error - Server error

Custom Error Codes

Error Code	Description
INVALID_MOBILE_NUMBER	Mobile number format is invalid
OTP_EXPIRED	OTP has expired
OTP_INVALID	OTP is incorrect
OTP_MAX_ATTEMPTS	Maximum OTP attempts exceeded
TOKEN_EXPIRED	JWT token has expired
TOKEN_INVALID	JWT token is invalid
USER_NOT_FOUND	User account not found
USER_INACTIVE	User account is deactivated
APPOINTMENT_NOT_FOUND	Appointment not found
SLOT_UNAVAILABLE	Time slot is not available
APPOINTMENT_CANCELLED	Appointment is already cancelled
FILE_TOO_LARGE	File size exceeds limit
INVALID_FILE_TYPE	File type not allowed
PERMISSION_DENIED	Insufficient permissions
RATE_LIMIT_EXCEEDED	Too many requests

Validation Error Format

```json
{
  "success": false,
  "message": "Validation failed",
  "errors": [
    {
      "field": "mobileNumber",
      "message": "Mobile number must be exactly 10 digits",
      "value": "123456789"
    },
    {
      "field": "email",
      "message": "Please enter a valid email address",
      "value": "invalid-email"
    }
  ]
}
```

4. System Flowcharts

Arthomed Backend - System Flow Charts

Authentication Flow Chart

Syntax error in text
mermaid version 10.6.1

User Registration & Profile Setup Flow

Syntax error in text
mermaid version 10.6.1

Appointment Booking Flow

Syntax error in text
mermaid version 10.6.1

Doctor Schedule Management Flow

Syntax error in text
mermaid version 10.6.1

Appointment Status Management Flow

Syntax error in text
mermaid version 10.6.1

File Upload & Management Flow

Admin Dashboard Management Flow

Syntax error in text
mermaid version 10.6.1

Error Handling & Recovery Flow

Syntax error in text
mermaid version 10.6.1

Database Backup & Recovery Flow

Syntax error in text
mermaid version 10.6.1

API Rate Limiting Flow