ASMC API - Backend Documentation

A comprehensive Node.js Express API for managing Anushaktinagar Sports Management Committee (ASMC) operations including members, bookings, events, payments, and facilities with JWT authentication, MongoDB integration, and automated features.

Table of Contents

Quick Start

Prerequisites

- Node.js 18+
- MongoDB 4.4+
- npm or yarn

Local Development Setup

```bash
# Clone the repository
git clone <repository-url>
cd asmc-api

# Install dependencies
npm install

# Set up environment
cp .env.example .env.development

# Configure environment variables (see Environment Configuration)
nano .env.development

# Start development server
npm run dev

# Visit API documentation
open http://localhost:7055/api-docs
```

Production Setup

```bash
# Install PM2 globally
npm install -g pm2

# Start production server
npm run start:prod

# Monitor the application
pm2 monit
```

Architecture Overview

System Architecture

```
┌─────────────────┐    ┌─────────────────┐    ┌─────────────────┐
│   Admin Panel   │    │   Mobile App    │    │   Next.js Web   │
│   (React)       │    │ (React Native)  │    │   Frontend      │
└─────────────────┘    └─────────────────┘    └─────────────────┘
          │                      │                      │
          │                      │                      │
          └──────────────────────┼──────────────────────┘
                                 │
                                 │
                     ┌───────────┴─────────────┐
                     │        ASMC API         │
                     │      (Express.js)       │
                     └───────────┬─────────────┘
                                 │
                     ┌───────────┴─────────────┐
                     │        MongoDB          │
                     │       Database          │
                     └─────────────────────────┘
```

API Structure

```
asmc-api/
├── app/
│   ├── controller/          # Business logic controllers
│   │   ├── auth/           # Authentication & user management
│   │   ├── members/        # Member management & profiles
│   │   ├── staff/          # Staff management
│   │   ├── masters/        # Master data (facilities, locations)
│   │   ├── activity/       # Activity management
│   │   ├── payment/        # Payment processing
│   │   ├── biometric/      # Biometric integration
│   │   ├── halls/          # Hall management & booking
│   │   ├── events/         # Event management & booking
│   │   ├── plans/          # Membership plans
│   │   ├── reports/        # Report generation
│   │   └── common/         # Common utilities
│   ├── models/             # MongoDB models (Mongoose)
│   ├── middlewares/        # Authentication, validation, image processing
│   ├── helpers/            # Response formatters, constants, utilities
│   ├── utils/              # Email service, helper functions
│   ├── config/              # Database configuration
│   └── routes/              # Route definitions
├── cron/                    # Scheduled jobs (backups, emails)
├── docs/                    # Swagger API documentation
├── scripts/                 # Database scripts (backup, restore)
├── public/                  # Static files
└── backups/                 # Database backups
```

Technology Stack

- **Runtime**: Node.js 18+
- **Framework**: Express.js 4.18+
- **Database**: MongoDB 4.4+ with Mongoose ODM
- **Authentication**: JWT (JSON Web Tokens)
- **Validation**: Joi schema validation
- **Email**: Nodemailer + MSG91 integration
- **Image Processing**: Sharp + ImageKit integration
- **File Processing**: ExcelJS, JSON2CSV, Multer
- **Documentation**: Swagger UI
- **Process Management**: PM2
- **Scheduling**: Node-cron
- **Biometric**: ZKTeco integration

Installation & Setup

System Requirements

- **OS**: Ubuntu 20.04+ / macOS 10.15+ / Windows 10+
- **Node.js**: 18.0.0 or higher
- **MongoDB**: 4.4.0 or higher
- **RAM**: Minimum 2GB (4GB recommended for production)
- **Storage**: Minimum 10GB free space

Step-by-Step Installation

1. Install Node.js

```
# Using Node Version Manager (recommended)
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.0/install.sh | bash
source ~/.bashrc
nvm install 18
nvm use 18

# Or download from official site
# https://nodejs.org/en/download/
```

2. Install MongoDB

```
# Ubuntu/Debian
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-
org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list
sudo apt-get update
sudo apt-get install -y mongodb-org
```

```
# Start MongoDB
sudo systemctl start mongod
sudo systemctl enable mongod
```

3. Install Project Dependencies

```
# Clone repository
git clone <repository-url>
cd asmc-api

# Install dependencies
npm install

# Install PM2 globally (for production)
npm install -g pm2
```

4. Environment Setup

```
# Copy environment template
cp .env.example .env.development

# Edit environment variables
nano .env.development
```

5. Database Initialization

```
# Start MongoDB (if not running)
sudo systemctl start mongod

# Create database (will be created automatically on first connection)
# No manual database creation needed
```

6. Start Development Server

```
# Development mode
npm run dev

# Production mode
npm run start:prod
```

Environment Configuration

Environment Files Structure

```
asmc-api/
├── .env.development    # Development environment
├── .env.staging       # Staging environment
├── .env.production    # Production environment
└── .env.example       # Template file
```

Required Environment Variables

```
# Server Configuration
PORT=7055
NODE_ENV=development

# Database Configuration
MONGO_URI=mongodb://localhost:27017/asmc
MONGO_TEST_URI=mongodb://localhost:27017/asmc_test

# Authentication
JWT_SECRET=your-super-secret-jwt-key-min-32-chars
JWT_EXPIRE=7d

# Email Configuration
SMTP_HOST=smtp.gmail.com
SMTP_PORT=587
SMTP_USER=your-email@gmail.com
SMTP_PASS=your-app-password
MSG91_AUTH_KEY=your-msg91-auth-key
MSG91_SENDER_ID=ASMC

# Image Processing
IMAGEKIT_PUBLIC_KEY=your-imagekit-public-key
IMAGEKIT_PRIVATE_KEY=your-imagekit-private-key
IMAGEKIT_URL_ENDPOINT=https://ik.imagekit.io/your-imagekit-id

# Payment Gateway
CCAVENUE_MERCHANT_ID=your-merchant-id
CCAVENUE_ACCESS_CODE=your-access-code
CCAVENUE_WORKING_KEY=your-working-key

# Biometric Integration
BIOMETRIC_IP=192.168.1.100
BIOMETRIC_PORT=4370

# File Upload
MAX_FILE_SIZE=10485760  # 10MB
UPLOAD_PATH=./uploads

# Security
CORS_ORIGINS=http://localhost:3000,https://asmcdae.in
```

Environment-Specific Configurations

Development Environment

```
NODE_ENV=development
PORT=7055
MONGO_URI=mongodb://localhost:27017/asmc_dev
LOG_LEVEL=debug
```

Staging Environment

```
NODE_ENV=staging
PORT=7055
MONGO_URI=mongodb://staging-server:27017/asmc_staging
LOG_LEVEL=info
```

Production Environment

```
NODE_ENV=production
PORT=7055
MONGO_URI=mongodb://prod-server:27017/asmc_prod
LOG_LEVEL=error
```

Database Setup

MongoDB Configuration

Connection String Format

```
mongodb://[username:password@]host[:port][/database][?options]
```

Example Configurations

Local Development:

```
MONGO_URI=mongodb://localhost:27017/asmc
```

Production with Authentication:

```
MONGO_URI=mongodb://username:password@mongodb-server:27017/asmc?authSource=admin
```

MongoDB Atlas (Cloud):

```
MONGO_URI=mongodb+srv://username:password@cluster.mongodb.net/asmc?retryWrites=true&w=majority
```

Database Models

Core Models

1. **User Model**
    - Admin users and authentication
    - Role-based access control
    - Password encryption
2. **Member Model**
    - Member profiles and family details
    - Auto-generated member IDs (00001, 00002, etc.)
    - Payment and fee tracking
3. **Plan Model**

- Membership plan configurations
- Pricing and validity periods
- Plan recommendations

4. **Payment Model**

   - Payment history and verification
   - CCAvenue integration
   - Receipt generation

5. **Hall Model**

   - Hall details and amenities
   - Booking availability
   - Time slot management

6. **Event Model**

   - Event information and details
   - Booking management
   - Guest access control

7. **Activity Model**

   - Activity definitions
   - Enrollment tracking
   - Attendance management

8. **Biometric Model**

   - Machine configurations
   - Attendance records
   - Regularization requests

Database Indexes

```javascript
// Recommended indexes for performance
db.members.createIndex({ memberId: 1 }, { unique: true });
db.members.createIndex({ email: 1 });
db.members.createIndex({ phone: 1 });
db.payments.createIndex({ memberId: 1, createdAt: -1 });
db.bookings.createIndex({ hallId: 1, date: 1 });
db.events.createIndex({ eventDate: 1 });
```

Database Backup Strategy

```bash
# Automated daily backup (configured in cron)
npm run backup

# Manual backup
npm run backup:dev

# Restore from backup
npm run restore
```

Authentication Guide

JWT Authentication Flow

```
sequenceDiagram
    participant Client
    participant API
    participant Database

    Client->>API: POST /auth/admin-login
    API->>Database: Validate credentials
    Database-->>API: User data
    API->>API: Generate JWT token
    API-->>Client: Return token + user data

    Client->>API: GET /members (with Bearer token)
    API->>API: Verify JWT token
    API->>API: Check permissions
    API->>Database: Fetch members
    Database-->>API: Members data
    API-->>Client: Return members
```

Authentication Endpoints

Admin Login

```
POST /auth/admin-login
Content-Type: application/json

{
  "username": "admin@asmc.com",
  "password": "password123"
}
```

Response:

```
{
    "success": true,
    "message": "Login successful",
    "result": {
        "token": "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9...",
        "user": {
            "id": "user_id",
            "username": "admin@asmc.com",
            "role": "admin"
        }
    }
}
```

Member Login

```
POST /auth/member-login
Content-Type: application/json

{
  "memberId": "00001",
  "password": "password123"
}
```

Token Usage

```
// Include token in Authorization header
const token = localStorage.getItem('authToken');

fetch('/api/members', {
    headers: {
        Authorization: `Bearer ${token}`,
        'Content-Type': 'application/json',
    },
});
```

Password Security

- Passwords are hashed using bcrypt
- Minimum 8 characters required
- Password reset via OTP
- Session timeout after 7 days

Role-Based Access Control

Admin Roles

- **Super Admin**: Full system access
- **Admin**: Member and booking management
- **Staff**: Limited access to specific modules

Member Roles

- **Member**: Access to own profile and bookings
- **Family Member**: Limited access to family bookings

API Reference

Base URL

- **Development**: `http://localhost:7055`
- **Production**: `https://api.asmcdae.in`

API Response Format

All API responses follow a consistent format:

```
{
  "success": true|false,
  "message": "Response message",
  "result": {
    // Response data or error details
  }
}
```

Error Response Format

```
{
    "success": false,
    "message": "Error description",
    "result": {
        "error": "Detailed error information",
        "code": "ERROR_CODE"
    }
}
```

HTTP Status Codes

- `200` - Success
- `201` - Created
- `400` - Bad Request
- `401` - Unauthorized
- `403` - Forbidden
- `404` - Not Found
- `422` - Validation Error
- `500` - Internal Server Error

API Endpoints Overview

Authentication (`/auth`)

- `POST /auth/admin-login` - Admin login
- `POST /auth/member-login` - Member login
- `GET /auth/me` - Get current user
- `PUT /auth/change-password` - Change password
- `POST /auth/send-reset-password-otp` - Send reset OTP
- `PUT /auth/reset-password` - Reset password

Members (`/members`)

- `GET /members` - List all members (paginated)
- `POST /members` - Create new member
- `GET /members/:id` - Get member details
- `PUT /members/:id` - Update member
- `DELETE /members/:id` - Delete member
- `POST /members/multiple` - Bulk member creation
- `GET /members/export` - Export members data

Staff (`/staff`)

- `GET /staff` - List all staff
- `POST /staff` - Create new staff
- `GET /staff/:id` - Get staff details
- `PUT /staff/:id` - Update staff
- `DELETE /staff/:id` - Delete staff

Masters (`/masters`)

- `GET /masters/facilities` - List facilities

- `POST /masters/facilities` - Create facility
- `GET /masters/locations` - List locations
- `POST /masters/locations` - Create location
- `GET /masters/categories` - List categories
- `POST /masters/categories` - Create category

Activity (`/activity`)

- `GET /activity` - List activities
- `POST /activity` - Create activity
- `GET /activity/:id` - Get activity details
- `PUT /activity/:id` - Update activity
- `DELETE /activity/:id` - Delete activity
- `POST /activity/enroll` - Enroll in activity
- `GET /activity/enrolled` - Get enrolled activities

Payment (`/payment`)

- `GET /payment` - List payments
- `POST /payment` - Create payment
- `GET /payment/:id` - Get payment details
- `POST /payment/verify` - Verify payment
- `GET /payment/history` - Payment history
- `POST /payment/ccavenue-response` - CCAvenue callback

Biometric (`/biometric`)

- `GET /biometric/machines` - List machines
- `POST /biometric/machines` - Add machine
- `GET /biometric/attendance` - Get attendance
- `POST /biometric/attendance` - Mark attendance
- `GET /biometric/notifications` - Get notifications
- `POST /biometric/regularization` - Regularization request

Halls (`/halls`)

- `GET /halls` - List halls
- `POST /halls` - Create hall
- `GET /halls/:id` - Get hall details
- `PUT /halls/:id` - Update hall
- `DELETE /halls/:id` - Delete hall
- `POST /halls/book` - Book hall
- `GET /halls/availability` - Check availability

Events (`/events`)

- `GET /events` - List events
- `POST /events` - Create event
- `GET /events/:id` - Get event details
- `PUT /events/:id` - Update event
- `DELETE /events/:id` - Delete event
- `POST /events/book` - Book event
- `GET /events/upcoming` - Upcoming events

Plans (`/plans`)

- `GET /plans` - List plans
- `POST /plans` - Create plan
- `GET /plans/:id` - Get plan details

- `PUT /plans/:id` - Update plan
- `DELETE /plans/:id` - Delete plan
- `POST /plans/recommend` - Recommend plan

Reports (`/reports`)

- `GET /reports/members` - Members report
- `GET /reports/payments` - Payments report
- `GET /reports/enrollment` - Enrollment report
- `GET /reports/batch-wise` - Batch-wise report
- `GET /reports/renewal` - Renewal report
- `GET /reports/analytics` - Analytics report

Pagination

Most list endpoints support pagination:

```
GET /members?page=1&limit=10&sort=createdAt&order=desc
```

Response:

```
{
  "success": true,
  "message": "Members retrieved successfully",
  "result": {
    "data": [...],
    "pagination": {
      "page": 1,
      "limit": 10,
      "total": 100,
      "pages": 10
    }
  }
}
```

Filtering

Support for filtering by various fields:

```
GET /members?status=active&plan=premium&search=john
```

File Upload

For endpoints requiring file uploads:

```
const formData = new FormData();
formData.append('file', fileInput.files[0]);
formData.append('data', JSON.stringify(otherData));

fetch('/api/upload', {
    method: 'POST',
    body: formData,
    headers: {
        Authorization: `Bearer ${token}`,
    },
});
```

Cron Jobs

Scheduled Tasks

The system includes automated cron jobs for maintenance and operations:

1. Database Backup (Daily at 1:00 AM IST)

```
// File: cron/dbBackupCron.js
cron.schedule('0 1 * * *', async () => {
    console.log('Starting daily database backup...');
    await backupDatabase();
});
```

Features:

- Automated MongoDB backup
- Compressed backup files
- Retention policy (30 days)
- Email notifications on failure

2. Bulk Email (Optional - Currently Disabled)

```
// File: cron/bulk-email.js (commented out)
// Can be enabled for mass email campaigns
```

Manual Cron Operations

```
# Run backup manually
npm run backup

# Run backup for specific environment
npm run backup:staging
npm run backup:prod

# Restore from backup
npm run restore
```

Cron Job Monitoring

```
# Check PM2 logs for cron jobs
pm2 logs asmc-api

# Monitor specific cron job
pm2 logs asmc-api --lines 100
```

Monitoring & Logging

Logging Configuration

The system uses Morgan for HTTP request logging:

```
// Development logging
app.use(morgan('dev'));

// Production logging (minimal)
if (NODE_ENV === 'development' || NODE_ENV === 'local') {
    app.use(morgan('dev'));
}
```

Log Levels

- **Development**: All requests logged with details
- **Production**: Minimal logging for performance
- **Error Logging**: All errors logged with stack traces

Health Monitoring

Health Check Endpoint

```
GET /health
```

Response:

```
ok
```

API Documentation Endpoint

```
GET /api-docs
```

Response: Swagger UI interface

Performance Monitoring

PM2 Monitoring

```
# Monitor application
pm2 monit

# View logs
pm2 logs asmc-api

# Restart application
pm2 restart asmc-api

# View process information
pm2 show asmc-api
```

Database Monitoring

```
# MongoDB status
sudo systemctl status mongod
```

```
# MongoDB logs
sudo tail -f /var/log/mongodb/mongod.log
```

Error Handling

```
// Global error handler
app.use((error, req, res, next) => {
    if (!error) {
        return next();
    }
    console.error('Error:', error);
    return responseSend(res, 400, error.message);
});
```

Log Rotation

Configure log rotation for production:

```
# Install logrotate
sudo apt-get install logrotate

# Configure log rotation
sudo nano /etc/logrotate.d/asmc-api
```

🚀 Deployment Guide

Development Deployment

Local Development

```
# Start development server
npm run dev

# Server will start on http://localhost:7055
# API docs available at http://localhost:7055/api-docs
```

Staging Deployment

Using PM2

```
# Start staging environment
npm run start:staging

# Monitor staging
pm2 logs asmc-staging
```

Production Deployment

Ubuntu Server Setup

1. **Server Preparation**

```bash
# Update system
sudo apt update && sudo apt upgrade -y

# Install Node.js
curl -fsSL https://deb.nodesource.com/setup_18.x | sudo -E bash -
sudo apt-get install -y nodejs

# Install MongoDB
wget -qO - https://www.mongodb.org/static/pgp/server-6.0.asc | sudo apt-key add -
echo "deb [ arch=amd64,arm64 ] https://repo.mongodb.org/apt/ubuntu focal/mongodb-org/6.0 multiverse" | sudo tee /etc/apt/sources.list.d/mongodb-org-6.0.list
sudo apt-get update
sudo apt-get install -y mongodb-org

# Install PM2
sudo npm install -g pm2
```

2. **Application Deployment**

```bash
# Clone repository
git clone <repository-url>
cd asmc-api

# Install dependencies
npm ci --production

# Set up environment
cp .env.example .env.production
nano .env.production

# Start application
npm run start:prod
```

3. **Nginx Configuration**

```nginx
server {
    listen 80;
    server_name api.asmcdae.in;

    location / {
        proxy_pass http://localhost:7055;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_cache_bypass $http_upgrade;
    }
}
```

4. **SSL Certificate**

```
# Install Certbot
sudo apt install certbot python3-certbot-nginx

# Get SSL certificate
sudo certbot --nginx -d api.asmcdae.in
```

Docker Deployment

Dockerfile

```
FROM node:18-alpine

WORKDIR /app

# Copy package files
COPY package*.json ./

# Install dependencies
RUN npm ci --only=production

# Copy application code
COPY . .

# Create non-root user
RUN addgroup -g 1001 -S nodejs
RUN adduser -S nodejs -u 1001

# Change ownership
RUN chown -R nodejs:nodejs /app
USER nodejs

# Expose port
EXPOSE 7055

# Health check
HEALTHCHECK --interval=30s --timeout=3s --start-period=5s --retries=3 \
  CMD curl -f http://localhost:7055/health || exit 1

# Start application
CMD ["npm", "start"]
```

Docker Compose

```
version: '3.8'

services:
    asmc-api:
        build: .
        ports:
            - '7055:7055'
        environment:
            - NODE_ENV=production
            - MONGO_URI=mongodb://mongo:27017/asmc
        depends_on:
            - mongo
        volumes:
            - ./uploads:/app/uploads
            - ./backups:/app/backups

    mongo:
        image: mongo:6.0
        ports:
            - '27017:27017'
        volumes:
            - mongo_data:/data/db
        environment:
            - MONGO_INITDB_ROOT_USERNAME=admin
            - MONGO_INITDB_ROOT_PASSWORD=password

volumes:
    mongo_data:
```

Deployment Commands

```
# Build and start
docker-compose up -d

# View logs
docker-compose logs -f asmc-api

# Stop services
docker-compose down
```

CI/CD Pipeline

GitHub Actions Example

```
name: Deploy ASMC API

on:
    push:
        branches: [main]

jobs:
    deploy:
        runs-on: ubuntu-latest

        steps:
            - uses: actions/checkout@v2

            - name: Setup Node.js
              uses: actions/setup-node@v2
              with:
                  node-version: '18'

            - name: Install dependencies
              run: npm ci

            - name: Run tests
              run: npm test

            - name: Deploy to server
              run: |
                  # Deployment commands here
                  echo "Deploying to production..."
```

Troubleshooting

Common Issues

1. MongoDB Connection Issues

Error: `MongoNetworkError: failed to connect to server`

Solutions:

```
# Check MongoDB status
sudo systemctl status mongod

# Start MongoDB
sudo systemctl start mongod

# Check MongoDB logs
sudo tail -f /var/log/mongodb/mongod.log

# Test connection
mongo --eval "db.adminCommand('ismaster')"
```

2. Port Already in Use

Error: `EADDRINUSE: address already in use :::7055`

Solutions:

```
# Find process using port 7055
sudo lsof -i :7055

# Kill process
sudo kill -9 <PID>

# Or use different port
PORT=7056 npm run dev
```

3. JWT Token Issues

Error: `JsonWebTokenError: invalid token`

Solutions:

```
# Check JWT_SECRET in environment
echo $JWT_SECRET

# Verify token format
# Token should be: Bearer <token>
```

4. File Upload Issues

Error: `MulterError: File too large`

Solutions:

```
# Check file size limits
# Increase limits in multer configuration
# Check available disk space
df -h
```

5. Memory Issues

Error: `JavaScript heap out of memory`

Solutions:

```
# Increase Node.js memory limit
node --max-old-space-size=4096 app.js

# Or in PM2 configuration
pm2 start app.js --node-args="--max-old-space-size=4096"
```

Debug Mode

Enable Debug Logging

```
# Set debug environment
DEBUG=* npm run dev

# Or specific modules
DEBUG=express:router npm run dev
```

PM2 Debug Mode

```
# Start with debug logs
pm2 start app.js --name asmc-api --node-args="--inspect"

# View debug logs
pm2 logs asmc-api
```

Performance Issues

Database Performance

```
# Check MongoDB performance
db.stats()

# Check slow queries
db.setProfilingLevel(2, { slowms: 100 })
db.system.profile.find().sort({ts: -1}).limit(5)
```

Application Performance

```
# Monitor CPU and memory usage
pm2 monit

# Check process details
pm2 show asmc-api
```

Log Analysis

Common Log Patterns

```
# Search for errors
grep -i error /var/log/asmc-api.log

# Search for specific endpoints
grep "POST /members" /var/log/asmc-api.log

# Monitor real-time logs
tail -f /var/log/asmc-api.log | grep ERROR
```

Security

Security Best Practices

1. Environment Variables

- Never commit `.env` files to version control
- Use strong, unique secrets for production
- Rotate secrets regularly

2. Authentication

- Use strong JWT secrets (minimum 32 characters)
- Implement token expiration
- Use HTTPS in production

3. Input Validation

- Validate all input using Joi schemas
- Sanitize user input
- Prevent SQL injection (MongoDB handles this)

4. File Upload Security

```
// File type validation
const allowedTypes = ['image/jpeg', 'image/png', 'application/pdf'];
```

```
// File size limits
const maxSize = 10 * 1024 * 1024; // 10MB

// File name sanitization
const sanitizedFileName = originalName.replace(/[^a-zA-Z0-9.-]/g, '_');
```

5. CORS Configuration

```
const corsOptions = {
    origin: ['https://asmcdae.in', 'https://admin.asmcdae.in'],
    methods: ['GET', 'POST', 'PUT', 'DELETE'],
    allowedHeaders: ['Content-Type', 'Authorization'],
    credentials: true,
};
```

Security Headers

```
// Add security headers
app.use((req, res, next) => {
    res.setHeader('X-Content-Type-Options', 'nosniff');
    res.setHeader('X-Frame-Options', 'DENY');
    res.setHeader('X-XSS-Protection', '1; mode=block');
    next();
});
```

Rate Limiting

```
// Implement rate limiting
const rateLimit = require('express-rate-limit');

const limiter = rateLimit({
    windowMs: 15 * 60 * 1000, // 15 minutes
    max: 100, // limit each IP to 100 requests per windowMs
});

app.use('/api/', limiter);
```

▯ Performance

Performance Optimization

1. Database Optimization

- Use proper indexes
- Implement pagination
- Use aggregation pipelines for complex queries

2. Caching

```
// Implement Redis caching
const redis = require('redis');
const client = redis.createClient();
```

```
// Cache frequently accessed data
app.get('/members', async (req, res) => {
    const cacheKey = 'members:all';
    const cached = await client.get(cacheKey);

    if (cached) {
        return res.json(JSON.parse(cached));
    }

    const members = await Member.find();
    await client.setex(cacheKey, 3600, JSON.stringify(members));
    res.json(members);
});
```

3. Image Optimization

```
// Compress images before upload
const sharp = require('sharp');

const compressImage = async (buffer) => {
    return await sharp(buffer).resize(800, 600).jpeg({ quality: 80 }).toBuffer();
};
```

4. Connection Pooling

```
// MongoDB connection pooling
mongoose.connect(uri, {
    maxPoolSize: 10,
    serverSelectionTimeoutMS: 5000,
    socketTimeoutMS: 45000,
});
```

Monitoring Performance

1. Response Time Monitoring

```
// Add response time middleware
app.use((req, res, next) => {
    const start = Date.now();
    res.on('finish', () => {
        const duration = Date.now() - start;
        console.log(`${req.method} ${req.path} - ${duration}ms`);
    });
    next();
});
```

2. Memory Usage Monitoring

```
// Monitor memory usage
setInterval(() => {
    const usage = process.memoryUsage();
    console.log('Memory Usage:', {
        rss: Math.round(usage.rss / 1024 / 1024) + ' MB',
        heapTotal: Math.round(usage.heapTotal / 1024 / 1024) + ' MB',
        heapUsed: Math.round(usage.heapUsed / 1024 / 1024) + ' MB',
    });
}, 30000);
```

Support & Resources

Documentation Links

- API Documentation - Interactive Swagger docs
- Quick Start Guide - Get up and running quickly
- Function Reference - Detailed function documentation
- Architecture Overview - System architecture details

Quick Links

- API Documentation
- Quick Start Guide
- Architecture Overview
- Installation Guide