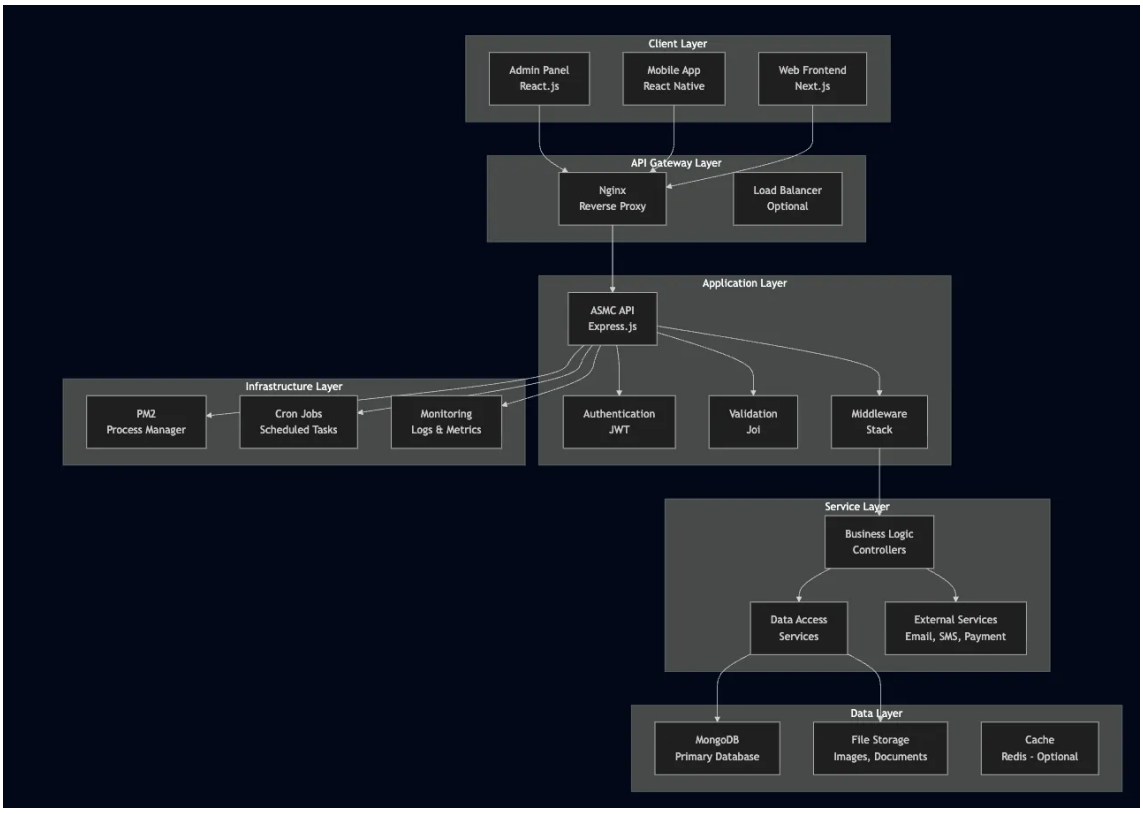
ASMC API - Architecture Overview

This document provides a comprehensive overview of the ASMC API architecture, including system design, component interactions, data flow, and technical decisions.

Table of Contents

System Architecture

High-Level Architecture



Technology Stack

Layer	Technology	Purpose
Runtime	Node.js 18+	JavaScript runtime environment

Framework	Express.js 4.18+	Web application framework
Database	MongoDB 4.4+	Document database
ODM	Mongoose 6.8+	Object Document Mapper
Authentication	JWT	Token-based authentication
Validation	Joi 17.7+	Schema validation
Process Manager	PM2	Production process management
Scheduling	Node-cron	Cron job scheduling
Email	Nodemailer + MSG91	Email and SMS services
Image Processing	Sharp + ImageKit	Image optimization
Documentation	Swagger UI	API documentation

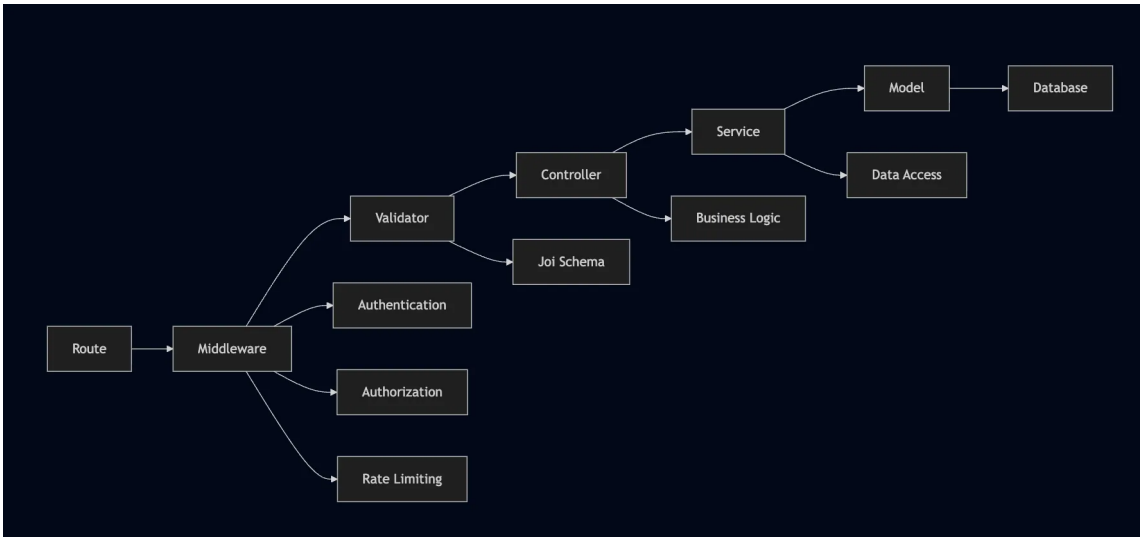
API Architecture

RESTful API Design

The API follows REST principles with consistent patterns:

```
Base URL: https://api.asmcdae.in
Version: v1 (implicit)

Endpoints Structure:
GET    /resource          # List resources
GET    /resource/:id      # Get specific resource
POST   /resource          # Create new resource
PUT    /resource/:id      # Update resource
DELETE /resource/:id      # Delete resource
```

Controller-Service-Validator Pattern



API Response Format

All API responses follow a consistent structure:

```
{
  "success": true|false,
  "message": "Human-readable message",
  "result": {
    // Success: Response data
    // Error: Error details
  }
}
```

Error Handling Architecture

```
// Global error handling flow
Request → Route → Middleware → Controller → Service
                                                ↓
Error Handler ← Response ← Controller ← Service (Error)
```

Database Design

MongoDB Schema Design

Member Document Structure

```
{
  _id: ObjectId,
  memberId: String,           // Auto-generated (00001, 00002, etc.)
  personalInfo: {
    firstName: String,
    lastName: String,
    email: String,
    phone: String,
    dateOfBirth: Date,
    address: {
      street: String,
      city: String,
      state: String,
      pincode: String
    }
  },
  familyMembers: [{
    name: String,
    relation: String,
    age: Number,
    isPrimary: Boolean
  }],
  membership: {
    planId: ObjectId,
    startDate: Date,
    endDate: Date,
```

```
    status: String,           // active, expired, suspended
    isActive: Boolean
  },
  paymentHistory: [ObjectId], // References to Payment documents
  profileImage: String,       // ImageKit URL
  createdAt: Date,
  updatedAt: Date
}
```

Payment Document Structure

```
{
  _id: ObjectId,
  memberId: ObjectId,
  paymentType: String,        // membership, hall_booking, event_booking
  amount: Number,
  currency: String,           // INR
  paymentMethod: String,      // online, cash, cheque
  paymentStatus: String,      // pending, completed, failed, refunded
  transactionId: String,      // CCAvenue transaction ID
  paymentDate: Date,
  dueDate: Date,
  description: String,
  receiptNumber: String,      // Auto-generated
  ccavenueResponse: Object,   // Payment gateway response
  createdAt: Date,
  updatedAt: Date
}
```

Hall Booking Document Structure

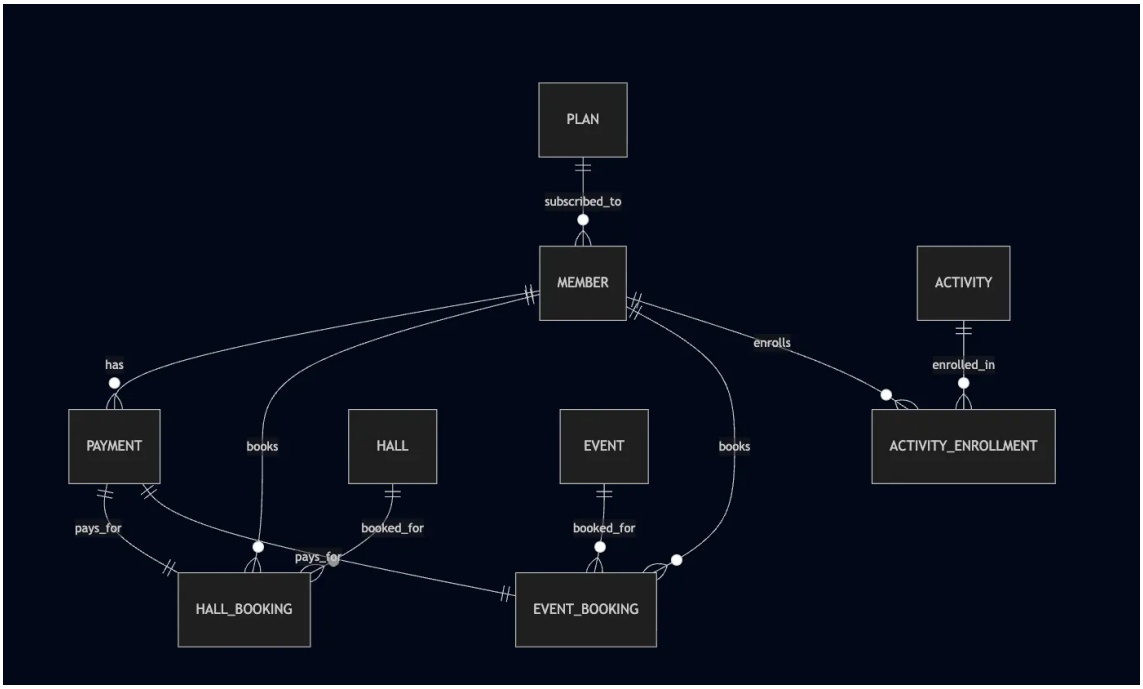
```
{
  _id: ObjectId,
  hallId: ObjectId,
  memberId: ObjectId,
  bookingDate: Date,
  timeSlots: [{
    startTime: String,        // "09:00"
    endTime: String,          // "11:00"
    status: String            // booked, cancelled
  }],
  totalAmount: Number,
  paymentId: ObjectId,
  bookingStatus: String,      // confirmed, cancelled, completed
  guestDetails: [{
    name: String,
    phone: String,
    relation: String
  }],
  specialRequirements: String,
  createdAt: Date,
```

```
  updatedAt: Date
}
```

Database Indexes

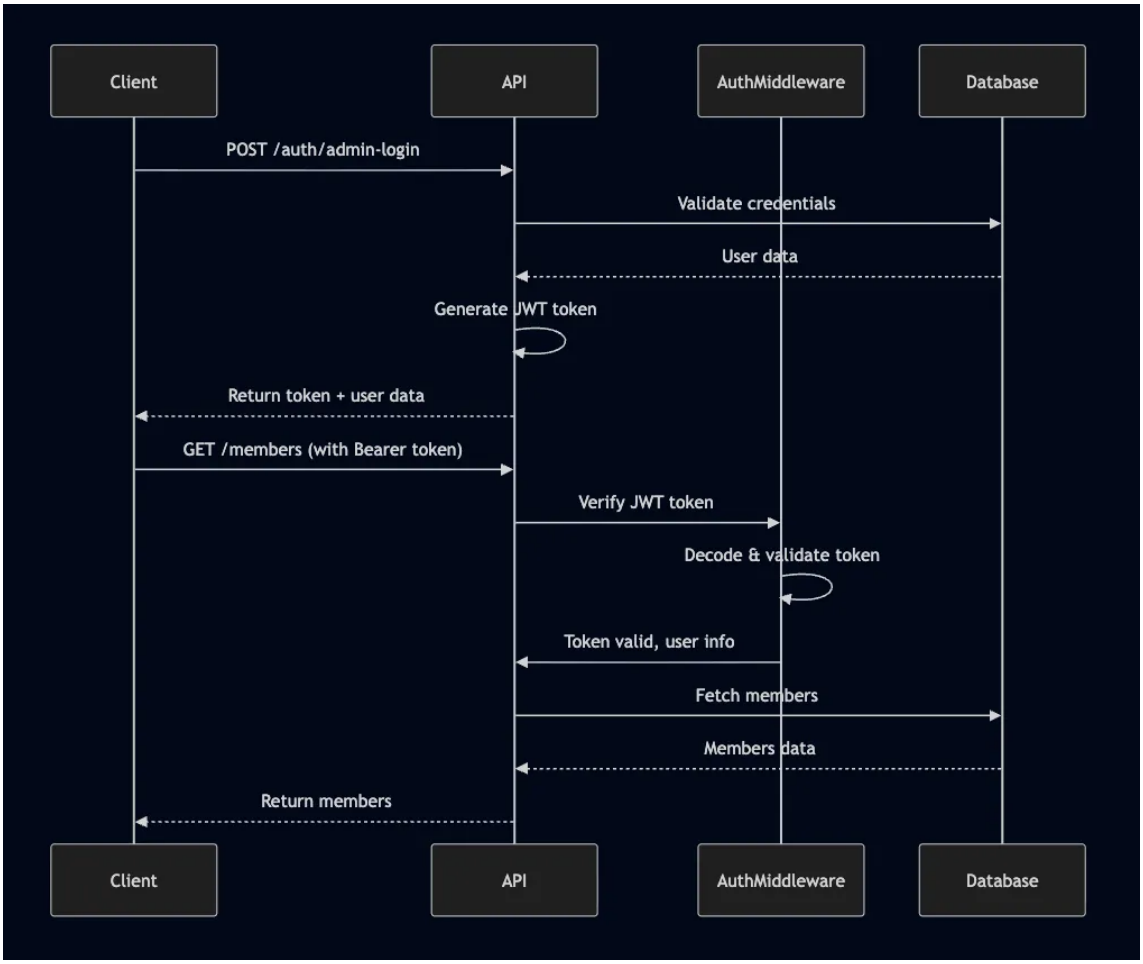
```
// Performance-critical indexes
db.members.createIndex({ memberId: 1 }, { unique: true });
db.members.createIndex({ 'personalInfo.email': 1 });
db.members.createIndex({ 'personalInfo.phone': 1 });
db.members.createIndex({ 'membership.status': 1 });

db.payments.createIndex({ memberId: 1, createdAt: -1 });
db.payments.createIndex({ paymentStatus: 1 });
db.payments.createIndex({ transactionId: 1 }, { unique: true });

db.hallBookings.createIndex({ hallId: 1, bookingDate: 1 });
db.hallBookings.createIndex({ memberId: 1, createdAt: -1 });

db.events.createIndex({ eventDate: 1 });
db.events.createIndex({ status: 1 });
```

Data Relationships



Authentication & Authorization

JWT Authentication Flow



Role-Based Access Control (RBAC)

```
// Role hierarchy
const roles = {
    SUPER_ADMIN: {
        permissions: ['*'], // All permissions
        description: 'Full system access',
    },
    ADMIN: {
        permissions: [
            'members:read',
            'members:write',
            'members:delete',
            'bookings:read',
            'bookings:write',
            'payments:read',
            'payments:write',
            'reports:read',
        ],
        description: 'Member and booking management',
    },
```

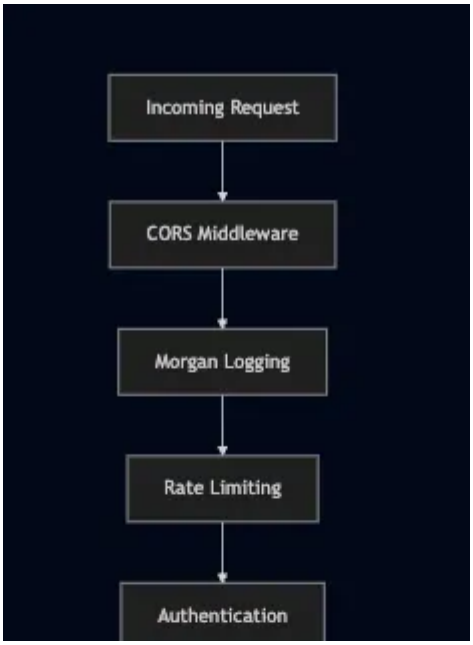
```
    STAFF: {
        permissions: ['members:read', 'bookings:read', 'bookings:write',
'payments:read'],
        description: 'Limited access to specific modules',
    },
    MEMBER: {
        permissions: [
            'profile:read',
            'profile:write',
            'bookings:read',
            'bookings:write:own',
            'payments:read:own',
        ],
        description: 'Access to own profile and bookings',
    },
};
```

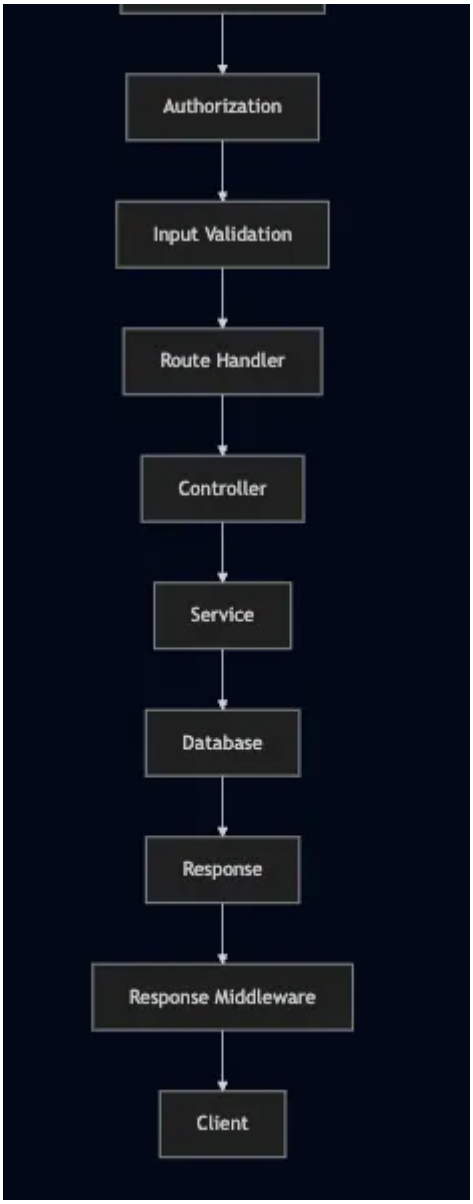
Token Structure

```
// JWT Payload Structure
{
  "userId": "user_id",
  "username": "admin@asmc.com",
  "role": "admin",
  "permissions": ["members:read", "members:write", ...],
  "iat": 1640995200,    // Issued at
  "exp": 1641600000     // Expires at (7 days)
}
```

Middleware Stack

Request Processing Pipeline





Middleware Configuration

```
// Middleware order and configuration
app.use(cors(corsOptions)); // CORS handling
app.use(morgan('dev')); // HTTP request logging
app.use(
    rateLimit({
        // Rate limiting
        windowMs: 15 * 60 * 1000, // 15 minutes
        max: 100, // Limit each IP to 100 requests
    }),
);
app.use(express.json()); // JSON body parsing
app.use(express.urlencoded({ extended: true })); // URL-encoded body parsing
```

```
app.use('/api', authenticateUser); // Authentication middleware
app.use('/api', checkPermission); // Authorization middleware
```

Custom Middleware

Authentication Middleware

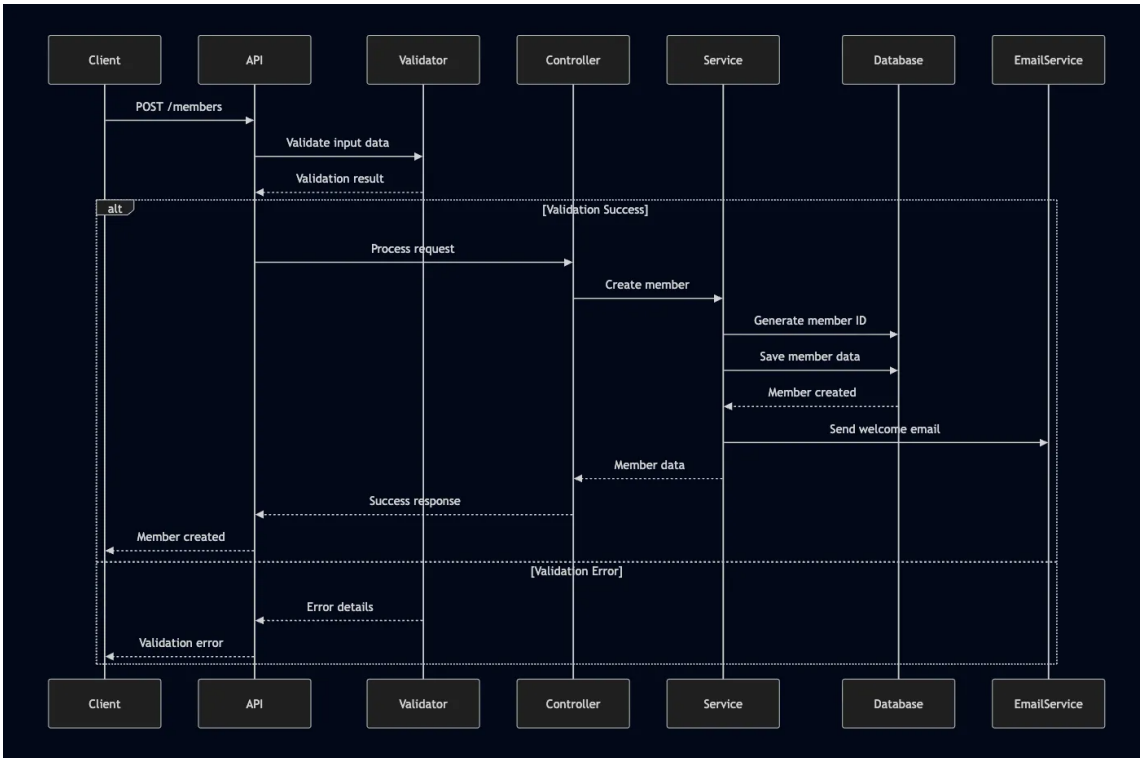
```
const authenticateUser = async (req, res, next) => {
    try {
        const token = req.header('Authorization')?.replace('Bearer ', '');

        if (!token) {
            return responseSend(res, 401, 'Access denied. No token provided.');
        }

        const decoded = jwt.verify(token, process.env.JWT_SECRET);
        req.user = decoded;
        next();
    } catch (error) {
        return responseSend(res, 401, 'Invalid token.');
    }
};
```

Permission Middleware

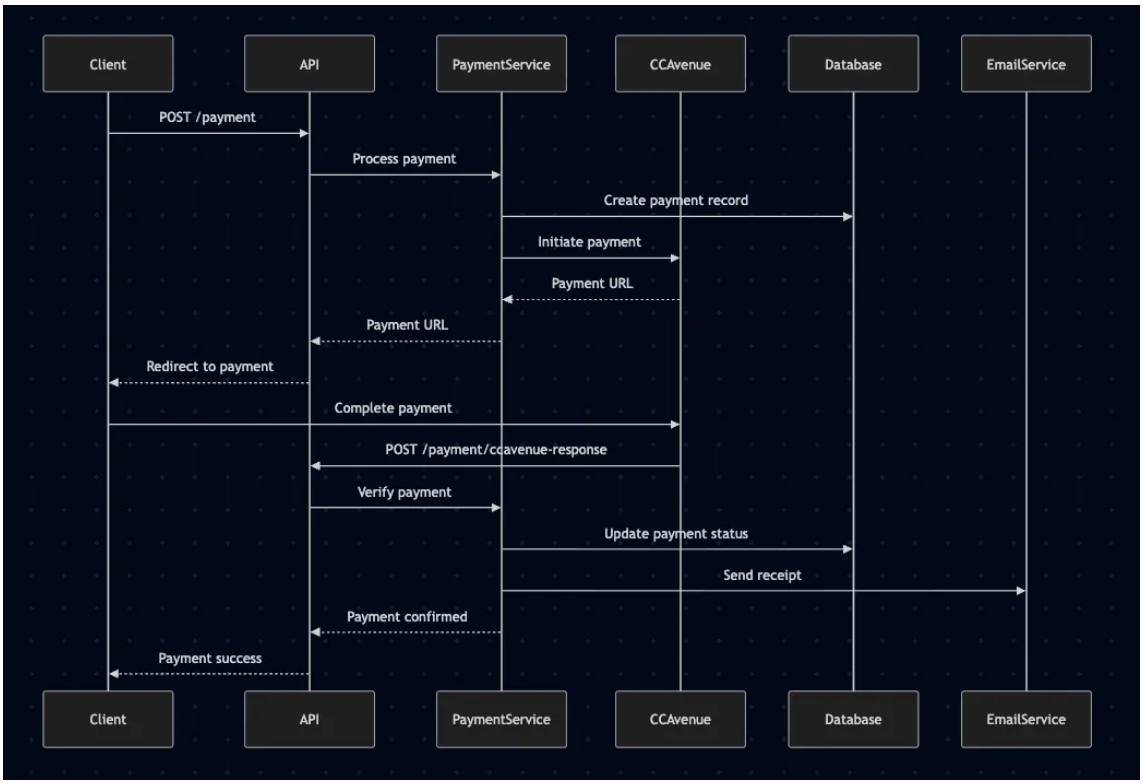
```
const checkPermission = (permission) => {
    return (req, res, next) => {
        if (
            !req.user.permissions.includes(permission) &&
            !req.user.permissions.includes('*')
        ) {
            return responseSend(res, 403, 'Access denied. Insufficient permissions.');
        }
        next();
    };
};
```

Data Flow

Member Registration Flow



Payment Processing Flow



Component Interactions

Controller Layer

```
// Controller structure and responsibilities
class MembersController {
    // Business logic coordination
    async createMember(req, res) {
        try {
            // 1. Extract and validate data
            const memberData = req.body;

            // 2. Call service layer
            const result = await membersService.createMember(memberData);

            // 3. Send response
            return responseSend(res, 201, 'Member created successfully', result);
        } catch (error) {
            // 4. Handle errors
            return responseSend(res, 500, error.message);
        }
    }
}
```

Service Layer

```
// Service layer responsibilities
class MembersService {
    // Data access and business logic
    async createMember(memberData) {
        // 1. Generate member ID
        const memberId = await this.generateMemberId();

        // 2. Prepare member document
        const member = {
            ...memberData,
            memberId,
            createdAt: new Date(),
            updatedAt: new Date(),
        };

        // 3. Save to database
        const savedMember = await Member.create(member);

        // 4. Send welcome email
        await emailService.sendWelcomeEmail(savedMember);

        // 5. Return result
        return savedMember;
    }
```

```
    async generateMemberId() {
        const lastMember = await Member.findOne().sort({ memberId: -1 });
        const lastNumber = lastMember ? parseInt(lastMember.memberId) : 0;
        return String(lastNumber + 1).padStart(5, '0');
    }
}
```

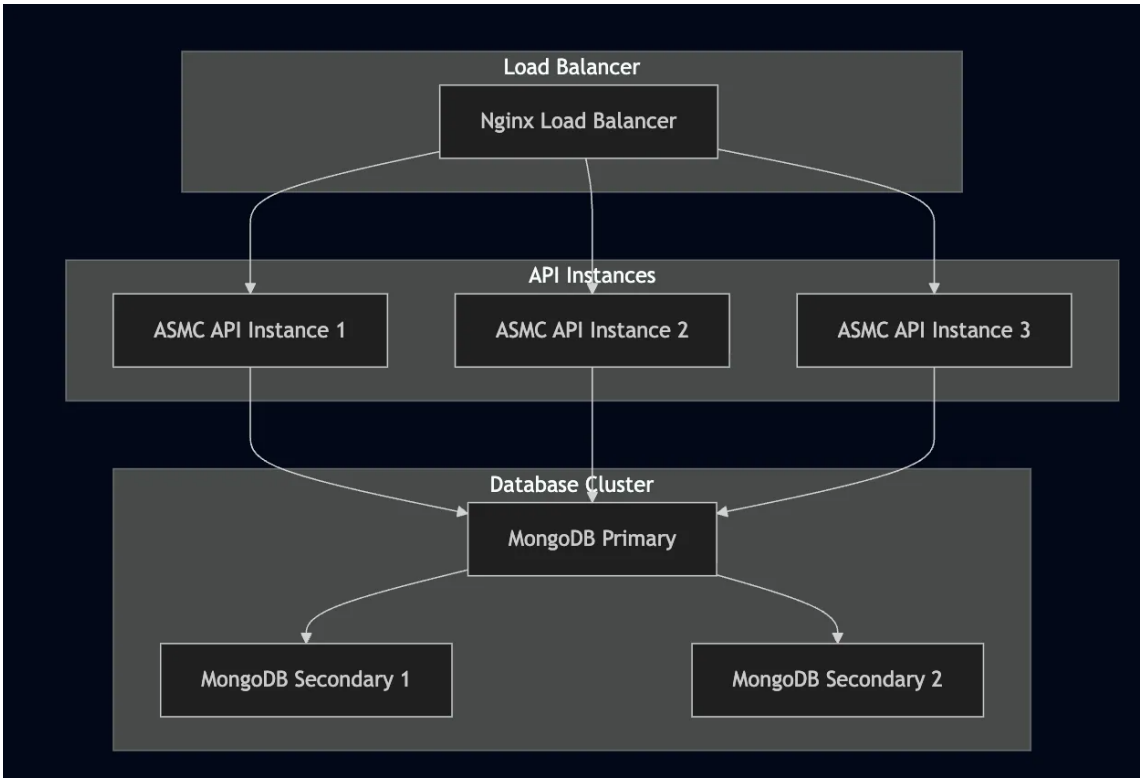
Model Layer

```
// Mongoose model definition
const memberSchema = new mongoose.Schema(
    {
        memberId: { type: String, unique: true, required: true },
        personalInfo: {
            firstName: { type: String, required: true },
            lastName: { type: String, required: true },
            email: { type: String, required: true, unique: true },
            phone: { type: String, required: true },
            dateOfBirth: { type: Date, required: true },
            address: {
                street: String,
                city: String,
                state: String,
                pincode: String,
            },
        },
        familyMembers: [
            {
                name: String,
                relation: String,
                age: Number,
                isPrimary: Boolean,
            },
        ],
        membership: {
            planId: { type: mongoose.Schema.Types.ObjectId, ref: 'Plan' },
            startDate: Date,
            endDate: Date,
            status: { type: String, enum: ['active', 'expired', 'suspended'] },
            isActive: Boolean,
        },
    },
    {
        timestamps: true,
    },
);

// Indexes
memberSchema.index({ memberId: 1 }, { unique: true });
memberSchema.index({ 'personalInfo.email': 1 });
memberSchema.index({ 'personalInfo.phone': 1 });
```

```
export default mongoose.model('Member', memberSchema);
```

Scalability Considerations

Horizontal Scaling



Performance Optimizations

Database Optimization

```
// Connection pooling
mongoose.connect(uri, {
    maxPoolSize: 10, // Maintain up to 10 socket connections
    serverSelectionTimeoutMS: 5000, // Keep trying to send operations for 5 seconds
    socketTimeoutMS: 45000, // Close sockets after 45 seconds of inactivity
    bufferMaxEntries: 0, // Disable mongoose buffering
});

// Query optimization
const members = await Member.find({ status: 'active' })
    .select('memberId personalInfo.firstName personalInfo.lastName') // Only select
needed fields
    .populate('membership.planId', 'name price') // Populate only required fields
    .sort({ createdAt: -1 })
    .limit(50)
    .lean(); // Return plain JavaScript objects instead of Mongoose documents
```

Caching Strategy

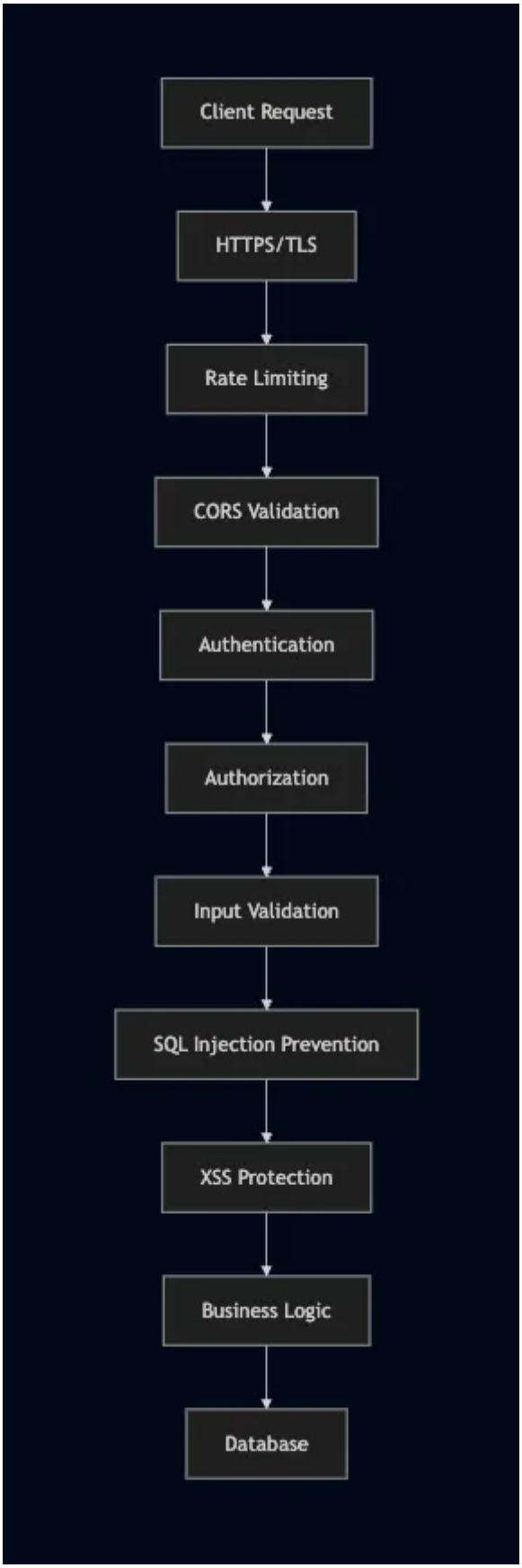
```
// Redis caching for frequently accessed data
const redis = require('redis');
const client = redis.createClient();

// Cache members list
app.get('/members', async (req, res) => {
    const cacheKey = 'members:list';
    const cached = await client.get(cacheKey);

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

    const members = await Member.find().lean();
    await client.setex(cacheKey, 3600, JSON.stringify(members)); // Cache for 1 hour
    res.json(members);
});
```

Image Optimization

```
// Sharp for image processing
const sharp = require('sharp');

const optimizeImage = async (buffer) => {
    return await sharp(buffer)
        .resize(800, 600, {
            fit: 'inside',
            withoutEnlargement: true,
        })
        .jpeg({
            quality: 80,
            progressive: true,
        })
        .toBuffer();
};
```

Security Architecture

Security Layers


Client Request
HTTPS/TLS
Rate Limiting
CORS Validation
Authentication
Authorization
Input Validation
SQL Injection Prevention
XSS Protection
Business Logic
Database

Security Measures

Input Validation

```
// Joi schema validation
const memberSchema = Joi.object({
    firstName: Joi.string().min(2).max(50).required(),
    lastName: Joi.string().min(2).max(50).required(),
    email: Joi.string().email().required(),
    phone: Joi.string()
        .pattern(/^[6-9]\d{9}$/)
        .required(),
    dateOfBirth: Joi.date().max('now').required(),
});
```

Password Security

```
// bcrypt for password hashing
const bcrypt = require('bcrypt');
const saltRounds = 12;

const hashPassword = async (password) => {
    return await bcrypt.hash(password, saltRounds);
};

const comparePassword = async (password, hash) => {
    return await bcrypt.compare(password, hash);
};
```

Security Headers

```
// Security headers middleware
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');
    res.setHeader('Strict-Transport-Security', 'max-age=31536000; includeSubDomains');
    next();
});
```

Data Protection

Encryption at Rest

- MongoDB encryption for sensitive data
- File system encryption for uploaded files

Encryption in Transit

- HTTPS/TLS for all API communications
- Encrypted database connections

Access Control

- Role-based access control (RBAC)

- Principle of least privilege
- Regular access reviews

Monitoring & Observability

Logging Architecture

```
// Structured logging
const winston = require('winston');

const logger = winston.createLogger({
    level: 'info',
    format: winston.format.combine(
        winston.format.timestamp(),
        winston.format.errors({ stack: true }),
        winston.format.json(),
    ),
    transports: [
        new winston.transports.File({ filename: 'error.log', level: 'error' }),
        new winston.transports.File({ filename: 'combined.log' }),
        new winston.transports.Console({
            format: winston.format.simple(),
        }),
    ],
});
```

Health Monitoring

```
// Health check endpoint
app.get('/health', async (req, res) => {
    const health = {
        status: 'ok',
        timestamp: new Date().toISOString(),
        uptime: process.uptime(),
        memory: process.memoryUsage(),
        database: await checkDatabaseConnection(),
        version: process.env.npm_package_version,
    };

    res.json(health);
});
```

Performance Metrics

```
// Response time monitoring
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`);
```

```
        // Log slow requests
        if (duration > 1000) {
            logger.warn('Slow request detected', {
                method: req.method,
                path: req.path,
                duration: duration,
            });
        }
    });

    next();
});
```

🔗 Related Documentation

- Quick Start Guide - Get up and running quickly
- API Reference - Detailed API documentation
- Database Schema - Complete database documentation
- Security Guide - Security best practices
- Deployment Guide - Production deployment

This architecture overview provides the foundation for understanding, maintaining, and extending the ASMC API system. For implementation details, refer to the specific documentation sections.