**Table of Contents**

1. Technologies Used
2. System Architecture
3. Authentication & Authorization
4. Entities
5. API Endpoints
6. Database Schema
7. Error Handling
8. Security Measures
9. Testing
10. Deployment Instructions
11. Rate Limiting
12. Performance Considerations
13. Code Quality and Organization

**1. Technologies Used**

* **Node.js**: Backend runtime environment.
* **Express.js**: Web framework for building the API.
* **MongoDB**: NoSQL database for data persistence.
* **JWT (JSON Web Token)**: For authentication.
* **Mongoose**: ODM for MongoDB.
* **bcryptjs**: For password hashing.
* **dotenv**: To manage environment variables.
* **Jest**: For unit testing.
* **Rate Limiting**: express-rate-limit package for API rate limiting.

**2. System Architecture**

This School Management System API is built following a RESTful architecture with a layered structure. The system consists of the following components:

* **Controllers**: Handle business logic for each resource.
* **Models**: Define the MongoDB schema for entities such as Schools, Classrooms, and Students.
* **Middleware**: Authentication, authorization, and input validation.
* **Routes**: Define API endpoints.
* **Services**: Reusable logic for interactions with the database.

**3. Authentication & Authorization**

The API uses **JWT-based authentication** to ensure secure access to endpoints. Authentication is required for all actions except for registration and login.

**JWT Flow**

1. **Login**: The user sends a POST request with their credentials (email and password). If the credentials are valid, a JWT token is returned.
2. **Authorization**: The token is sent with each subsequent request in the Authorization header: Authorization: Bearer <JWT Token>.

**Role-Based Access Control (RBAC)**

* **Superadmin**: Full access to the system, can manage schools and users.
* **School Administrator**: Limited to managing classrooms and students within their assigned school.

**4. Entities**

**Schools**

* **Managed by**: Superadmins
* **Operations**: Create, Read, Update, Delete (CRUD)
* **Attributes**: Name, Address

**Classrooms**

* **Managed by**: School administrators and Superadmins
* **Operations**: CRUD, Assign classrooms to specific schools
* **Attributes**: School ID, Name, Capacity, Resources (e.g., Projector, Whiteboard)

**Students**

* **Managed by**: School administrators and Superadmins
* **Operations**: Enrollment, Transfer, Profile Management
* **Attributes**: FirstName , LastName, Email, School ID, Classroom ID

**5. API Endpoints**

**Authentication Endpoints**

1. **POST /api/login**
   * Request: { "email": "string", "password": "string" }
   * Response: { "token": "string" }

**School Endpoints**

1. **GET /api/schools**
   * Request: Authenticated as Superadmin
   * Response: List of all schools
2. **POST /api/schools**
   * Request: { "name": "string", "address": "string}
   * Response: { "id": "string", "name": "string", "address": "string" }
3. **PUT /api/schools/:id**
   * Request: { "name": "string", "address": "string”}
   * Response: Updated school details
4. **DELETE /api/schools/:id**
   * Request: Authenticated as Superadmin
   * Response: { "message": "School deleted successfully" }

**Classroom Endpoints**

1. **GET /classrooms**
   * Request: Authenticated as School Administrator
   * Response: List of classrooms for the authenticated school
2. **POST /api/classrooms**
   * Request: { "name": "string", "capacity": "number", "schoolId": "string", "resources": ["projector", "whiteboard"] }
   * Response: Classroom details
3. **PUT /api/classrooms/:id**

* Request: { "name": "string", "capacity": "number", "resources": ["string"] }
* Response: Updated classroom details

1. **DELETE api/classrooms/:id**

* Response: { "message": "Classroom deleted successfully" }

**Student Endpoints**

1. **GET /api/students**

* Request: Authenticated as School Administrator and superadmin
* Response: List of students for the authenticated school

1. **POST /api/students**

* Request: { "firstName": "string", "lastName": "string", "email": "string", "schoolId": "string" }
* Response: Student details

1. **DELETE /students/:id**

* Response: { "message": "Student deleted successfully" }

1. **Post /api/students/enroll**

* Request: { "studentId": "objectID", "Classroom": "objectID”}
* Response: Student enrolled successfully
* **Post /api/students/transfer**
* Request: { "studentId": "objectID", "Classroom": "objectID”}(New Classroom by Id)
* Response: Student transferred successfully

**6. Database Schema**

The database uses MongoDB with the following collections and structure:

**School Model.**js

Copy code

{

name: String,

address: String,

}

**Classroom Model.**js

Copy code

{

name: String,

capacity: Number,

resources: [String],

schoolId: { type: mongoose.Schema.Types.ObjectId, ref: 'School' }

}

**Student Model.**js

Copy code

{

firstName: String,

lastName: String,

email: String,

schoolId: { type: mongoose.Schema.Types.ObjectId, ref: 'School' }

}

**7. Error Handling**

The API uses standard HTTP status codes for error handling:

* **400 Bad Request**: Invalid request format or missing parameters.
* **401 Unauthorized**: JWT token is missing or invalid.
* **403 Forbidden**: Insufficient permissions to access the resource.
* **404 Not Found**: Resource not found.
* **500 Internal Server Error**: Unexpected server error.

**8. Security Measures**

* **JWT Authentication**: Protects routes and ensures that only authorized users can access specific resources.
* **Input Validation**: Ensures that data sent by the user is valid and secure.
* **Rate Limiting**: Limits the number of requests to prevent abuse.

**9. Testing**

The project includes test cases for various endpoints using **Jest**. Tests are located in the tests/ directory. To run the tests, use the following command:

bash

Copy code

“npm run test”

**10. Deployment Instructions**

1. Clone the repository:

bash

Copy code

git clone <repository-url>

cd <project-directory>

1. Install dependencies:

bash

Copy code

npm install

1. Set up environment variables in a .env file:

bash

Copy code

MONGO\_URI=<your-mongo-uri>

JWT\_SECRET=<your-jwt-secret>

1. Run the application:

bash

Copy code

npm start

1. To run tests:

bash

Copy code

npm test

**11. Rate Limiting**

Rate limiting is implemented to prevent abuse. The API allows a maximum of **100 requests per 15 minutes** per IP address.

**12. Performance Considerations**

The API is designed to handle a large number of concurrent users by ensuring:

* **Efficient database queries**
* **Proper index management** for MongoDB
* **API rate limiting** to prevent overloading