**EXPERIMENT NO:**

**AIM**

Create secure, production-ready **RESTful APIs.**

**THEORY**

A **RESTful API (Representational State Transfer Application Programming Interface)** is an architectural style used to design scalable, reliable, and stateless web services. REST APIs allow communication between a client and server using standard HTTP methods like **GET, POST, PUT, PATCH, and DELETE**.

When building **production-ready REST APIs**, it is important to not only provide functionality but also ensure **security, scalability, and maintainability**.

�� **Key Principles of RESTful API Design**

1. **Statelessness** – Each request from the client must contain all necessary information; the server should not store client context.

2. **Uniform Interface** – Resources should be identified using URIs and manipulated with standard HTTP methods.

3. **Layered System** – APIs may use multiple layers (e.g., authentication, cache, proxy) for better scalability and security.

4. **Resource Representation** – Data is usually exchanged in JSON or XML format. �� **Steps to Create Secure and Production-Ready REST APIs**

1. **Design the API Endpoints**

○ Follow proper URI naming conventions (e.g., /users, /users/{id}). ○ Keep APIs versioned (e.g., /api/v1/).

2. **Use Proper HTTP Methods**

○ GET → Retrieve data

○ POST → Create new data

○ PUT/PATCH → Update existing data

○ DELETE → Remove data

3. **Input Validation & Data Sanitization**

○ Validate user input to prevent SQL Injection, XSS, and other attacks.

4. **Authentication & Authorization**

○ Use **JWT (JSON Web Tokens)**, OAuth 2.0, or API keys for secure access.

○ Role-based or permission-based authorization ensures users access only allowed resources.

5. **Error Handling & Status Codes**

○ Return proper HTTP status codes (200 OK, 400 Bad Request, 401 Unauthorized, 404 Not Found, 500 Internal Server Error).

○ Provide meaningful error messages without exposing sensitive details.

6. **Data Encryption**

○ Use **HTTPS (TLS/SSL)** to secure communication between client and server. ○ Encrypt sensitive data at rest and in transit.

7. **Rate Limiting & Throttling**

○ Prevent abuse and DDoS attacks by limiting requests per user or IP.

8. **Logging & Monitoring**

○ Maintain logs for debugging, auditing, and tracking API usage.

○ Use monitoring tools (e.g., Prometheus, ELK stack) in production.

9. **Scalability & Performance**

○ Use caching (e.g., Redis, CDN) to reduce server load.

○ Optimize database queries and implement pagination.

10. **Documentation**

● Use **Swagger/OpenAPI** for interactive API documentation.

● Helps developers understand and consume the API easily.

�� **Advantages of Secure REST APIs**

● Ensures **confidentiality** and **integrity** of data.

● Provides **scalability** for handling large user bases.

● Improves **developer experience** with clear documentation and error handling. ● Builds **trust and reliability** in production systems.

**STEPS**

**Step 1: Create New Project**

mkdir secure-rest-api

cd secure-rest-api

npm init -y

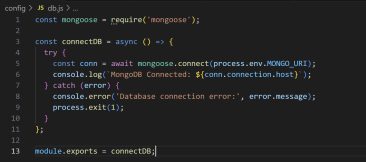
**Step 2: Install Dependencies**

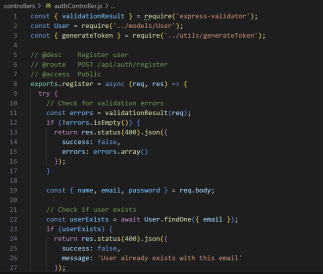
npm install express mongoose dotenv bcryptjs jsonwebtoken express-rate-limit helmet cors express-validator

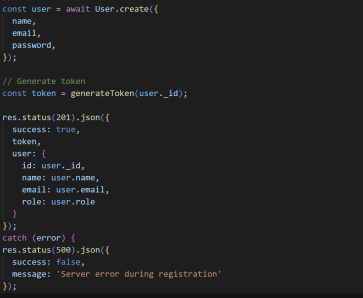
*# Development dependencies*

npm install -D nodemon morgan

**SOURCE CODE**

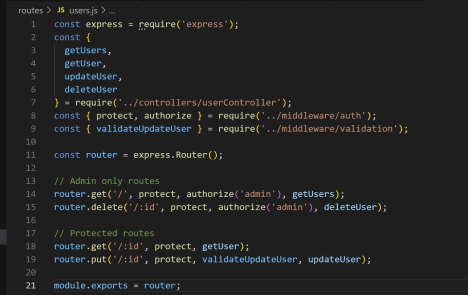
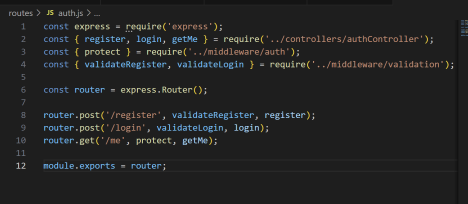
****

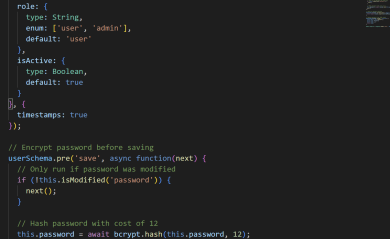
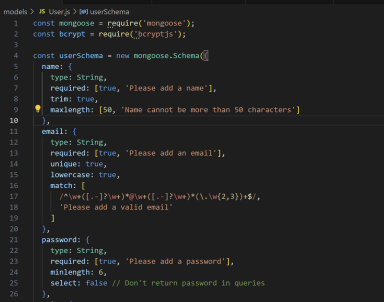
****

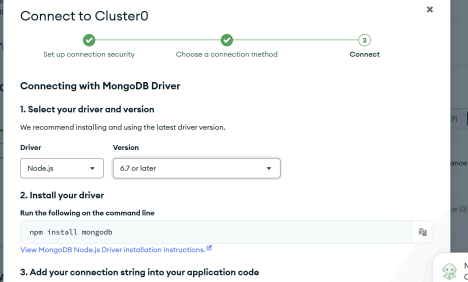
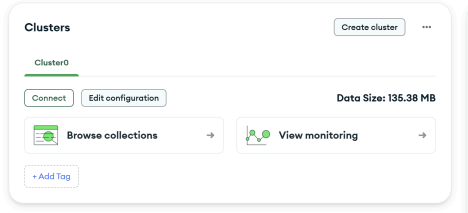
****

****

****

****

****

****

**Testing Your Secure API**

Register a New User:

Login:

Get Your Profile (Protected Route):

Get All Users (Admin Only):



**CONCLUSION**

Creating secure, production-ready RESTful APIs ensures that applications are not only functional but also reliable, scalable, and protected against vulnerabilities. By applying best practices such as authentication, authorization, input validation, error handling, encryption, logging, and monitoring, APIs become robust and suitable for real-world deployment. This approach increases trust, maintainability, and performance, making the API ready for long-term use in production environments.