Middleware Implementation for Logging and Bearer Token Authentication

1. Aim

The primary aim of this project is to understand, implement, and test the concept of **middleware** within a simple web server environment. Specifically, we aim to develop two critical middleware components: a Request Logger for monitoring and diagnostics, and a Bearer Token Authenticator for securing application routes.

2. Objectives

Upon completion of this project, the following objectives should be met:

1. **Conceptual Understanding:** Gain a clear understanding of the middleware pattern, including its position in the request-response lifecycle.
2. **Logging Implementation:** Successfully create and integrate a custom logging middleware that records essential request details (method, path, timestamp) to the console.
3. **Authentication Implementation:** Successfully create and integrate a custom authentication middleware that validates a Bearer token provided in the HTTP Authorization header, granting access only upon successful validation.
4. **Route Protection:** Demonstrate the ability to apply middleware selectively to protect specific API endpoints.

3. Theory

A. Middleware

In the context of web development (e.g., Express.js, Node.js), middleware refers to functions that have access to the request object (req), the response object (res), and the next function in the application's request-response cycle (next). Middleware functions can perform the following tasks:

- Execute any code.
- Make changes to the request and response objects.
- End the request-response cycle.
- Call the next middleware function in the stack.

B. Bearer Token Authentication

A Bearer Token is a security mechanism used in modern APIs, often associated with OAuth 2.0 or JWT (JSON Web Tokens). The token is a cryptic string, issued by an authentication server, which grants the "bearer" access to a protected resource. The token is sent in the HTTP request header:

Authorization: Bearer <token_string>

The authentication middleware checks for this header, verifies the token's presence and validity, and then decides whether to call next() (grant access) or return an error response (deny access).

C. Request Logging

Logging middleware intercepts every incoming request and outputs relevant information (such as HTTP method, URL path, and processing time) to the server console. This is crucial for:

- **Debugging:** Identifying which requests are failing.
- **Monitoring:** Tracking application usage and performance.
- **Auditing:** Creating a record of system activity.

4. Procedure

The project uses Node.js and the Express framework to create the server.

1. **Environment Setup:** Ensure Node.js and npm are installed.
2. **Project Initialization:** Create a new project directory and initialize it: npm init -y.
3. **Install Dependencies:** Install the Express framework: npm install express.
4. **Code Implementation:** Create the server.js file (provided below) containing the Express setup, the two custom middleware functions (loggerMiddleware and authMiddleware), and the routes.
5. **Server Execution:** Run the server using: node server.js.
6. **Testing:** Use a tool like cURL or Postman to test the public and protected routes, ensuring the logging occurs and authentication works correctly.

5. Code

The complete Node.js/Express server implementation is provided in the server.js file. This single file defines the server, the middleware functions, and the routes.

server.js Overview:

- The loggerMiddleware is applied globally using app.use().
- The authMiddleware is applied specifically to the /protected route.
- The hardcoded valid token is set to XYZ_SECURE_TOKEN_123.

(See the accompanying server.js file for the executable code.)

6. Output

A. Console Output (Logging Middleware)

When the server is running and a request is made, the loggerMiddleware prints the request details:

```
[LOG] - TIME: 2025-01-01T10:00:00.000Z | METHOD: GET | PATH: /public
[LOG] - TIME: 2025-01-01T10:00:00.050Z | METHOD: GET | PATH: /protected
```

B. HTTP Request/Response (Authentication Middleware)

Test Case	Request (cURL Equivalent)	Status Code	Response Body
1. Public Access	curl http://localhost:3000/public	200 OK	Welcome to the public zone!
2. Protected (Unauthorized)	curl http://localhost:3000/protected	401 Unauthorized	Access Denied: Bearer token is missing.
3. Protected (Invalid Token)	curl -H "Authorization: Bearer WRONG_TOKEN" http://localhost:3000/protected	401 Unauthorized	Access Denied: Invalid Bearer token.
4. Protected (Authorized)	curl -H "Authorization: Bearer XYZ_SECURE_TOKEN_123" http://localhost:3000/protected	200 OK	Welcome to the protected data!

7. Learning Outcomes

1. **The Request Lifecycle:** Learned how middleware functions intercept, process, and chain requests using the critical next() function, demonstrating the flow control within an API.
2. **API Security Fundamentals:** Gained practical experience in enforcing a basic security

policy by inspecting HTTP headers (Authorization) and implementing role-based access control logic within a dedicated authentication layer.

3. **Observability through Logging:** Implemented a non-blocking, global logging mechanism that instantly enhances the application's observability, allowing for real-time monitoring and debugging of all incoming traffic.