Experiment 5.1

Aim

To implement middleware in an Express.js application for logging HTTP requests and enforcing Bearer token authentication to protect specific routes.

Introduction

Middleware functions in Express.js are functions that have access to the request and response objects and can perform operations before passing control to the next function. In this experiment, two types of middleware are implemented: one for logging request details such as HTTP method, URL, and timestamp, and another for authenticating requests using a Bearer token. This demonstrates the importance of middleware for security, request validation, and monitoring in a Node.js backend.

Procedure

1. Initialize a Node.js project and install Express.js.
2. Create an Express.js server (`server.js`).
3. Implement a global logging middleware to log the HTTP method, request URL, and timestamp of every incoming request.
4. Implement an authentication middleware that checks for an Authorization header containing the Bearer token `mysecrettoken`. If the token is missing or incorrect, return a 401 Unauthorized error.
5. Define at least two routes:
6. Public route: Accessible without authentication.
7. Protected route: Accessible only with the correct Bearer token.
8. Apply the logging middleware globally and the authentication middleware only to the protected route.
9. Test the functionality using curl or Postman to verify logging and token-based authentication.

Code

server.js

```
const express = require("express");
const app = express();
const PORT = 5000;

// Logging Middleware
const logger = (req, res, next) => {
  console.log(`[${new Date().toISOString()}] ${req.method} ${req.url}`);
  next();
};
```

```
// Authentication Middleware
const authenticate = (req, res, next) => {
  const authHeader = req.headers["authorization"];
  if (!authHeader) {
    return res.status(401).json({ message: "Authorization header missing" });
  }
  const token = authHeader.split(" ")[1];
  if (token !== "mysecrettoken") {
    return res.status(401).json({ message: "Invalid token" });
  }
  next();
};

// Apply logging middleware globally
app.use(logger);

// Public Route
app.get("/public", (req, res) => {
  res.send("This is a public route accessible to everyone.");
});

// Protected Route
app.get("/protected", authenticate, (req, res) => {
  res.send("This is a protected route accessible only with a valid token.");
});

// Start Server
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

Output

- **Public Route (/public)**

```
curl http://localhost:5000/public
Response: "This is a public route accessible to everyone."
```

- **Protected Route (/protected) with valid token**

```
curl -H "Authorization: Bearer mysecrettoken" http://localhost:5000/
protected
Response: "This is a protected route accessible only with a valid token."
```

- **Protected Route (/protected) with invalid/missing token**

```
curl http://localhost:5000/protected
Response: { "message": "Authorization header missing" }
```

```
curl -H "Authorization: Bearer wrongtoken" http://localhost:5000/protected
Response: { "message": "Invalid token" }
```

- **Console Logging Example**

```
[2025-10-24T11:30:15.123Z] GET /public
[2025-10-24T11:30:20.456Z] GET /protected
```

Learning Outcomes

- Understand the concept and flow of middleware in Express.js.
- Learn to implement request logging for monitoring purposes.
- Implement token-based authentication to protect specific routes.
- Gain practical experience in securing backend APIs using middleware.
- Learn to test Express.js routes using curl or Postman.