Experiment 6.2

Aim

To secure backend API routes using JSON Web Tokens (JWT) so that only authenticated users can access protected resources.

Introduction

JWT (JSON Web Tokens) provide a compact and secure way of transmitting information between parties. In web applications, JWTs are commonly used for token-based authentication. This experiment demonstrates how to implement JWT-based authentication in a Node.js and Express.js backend, create a login route that issues tokens, and protect specific routes by verifying the JWT on each request.

Procedure

1. Initialize a Node.js project and install `express`, `jsonwebtoken`, and `body-parser`.
2. Create a sample user with hardcoded credentials for testing purposes.
3. Implement a login route (`/login`) that issues a JWT token when valid credentials are provided.
4. Create a middleware function `verifyToken` that checks the `Authorization` header for a Bearer token and verifies it using JWT.
5. Apply the `verifyToken` middleware to protected routes.
6. Test accessing the protected routes with and without a valid JWT token using Postman or curl.

Code

server.js

```
const express = require('express');
const jwt = require('jsonwebtoken');
const bodyParser = require('body-parser');

const app = express();
app.use(bodyParser.json());

const PORT = 5000;
const SECRET_KEY = 'myjwtsecret';

// Sample user
const user = {
  username: 'testuser',
  password: 'password123'
};
```

```
// Login Route
app.post('/login', (req, res) => {
  const { username, password } = req.body;
  if (username === user.username && password === user.password) {
    const token = jwt.sign({ username }, SECRET_KEY, { expiresIn: '1h' });
    res.json({ token });
  } else {
    res.status(401).json({ message: 'Invalid credentials' });
  }
});

// JWT Verification Middleware
const verifyToken = (req, res, next) => {
  const authHeader = req.headers['authorization'];
  if (!authHeader) return res.status(401).json({ message: 'Authorization header
missing' });
  const token = authHeader.split(' ')[1];
  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).json({ message: 'Invalid token' });
    req.user = user;
    next();
  });
};

// Protected Route
app.get('/protected', verifyToken, (req, res) => {
  res.json({ message: `Hello $
{req.user.username}, you have accessed a protected route.` });
});

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

Output

- **Login Route (/login)**

```
Request Body: { "username": "testuser", "password": "password123" }
Response: { "token": "<JWT_TOKEN>" }
```

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

```
Request Header: Authorization: Bearer <JWT_TOKEN>
Response: { "message": "Hello testuser, you have accessed a protected
route." }
```

- **Protected Route (/protected) without token or invalid token**

```
Response: { "message": "Authorization header missing" } or { "message":
"Invalid token" }
```

Learning Outcomes

- Understand token-based authentication using JWT.
- Learn to implement login functionality that issues JWT tokens.
- Create middleware to verify JWT and protect backend routes.
- Test protected routes to ensure unauthorized requests are blocked.
- Gain practical experience securing API endpoints in a Node.js and Express.js application.