Experiment 7.2

JWT Authentication for Secure Banking API Endpoints

Aim: *To implement secure authentication in an Express.js banking API using JSON Web Tokens (JWT), protecting sensitive endpoints like balance inquiry, deposit, and withdrawal.*

Tools Used: - **Node.js** - **Express.js** - **Postman** *(for testing API)* - **JSON Web Token (JWT) package** (`jsonwebtoken`) - **npm (Node Package Manager)**

Procedure: 1. **Initialize a Node.js project** and install dependencies: `express` and `jsonwebtoken`. 2. **Create an Express.js server** with routes: `/login`, `/balance`, `/deposit`, `/withdraw`. 3. **Hardcode a sample username and password** for login. 4. **Implement JWT token generation** in the `/login` route. 5. **Create a middleware function** `authenticateToken` to verify JWT tokens from the `Authorization` header. 6. **Protect banking routes** (`/balance`, `/deposit`, `/withdraw`) using the JWT middleware. 7. **Handle errors** for invalid/missing tokens and insufficient balance during withdrawals. 8. **Test the API using Postman**: - Send login credentials to `/login` to receive a token. - Use the token as a Bearer token in the Authorization header for other requests.

Code:

```
const express = require("express");
const jwt = require("jsonwebtoken");
const app = express();

app.use(express.json());

// Hardcoded user data
const user = { username: "user1", password: "password123", balance: 1000 };
const SECRET_KEY = "mysecretkey";

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

// JWT Authentication middleware
```

```
function authenticateToken(req, res, next) {
  const authHeader = req.headers["authorization"];
  const token = authHeader && authHeader.split(" ")[1];
  if (!token) return res.status(401).json({ message: "Token missing" });

  jwt.verify(token, SECRET_KEY, (err, user) => {
    if (err) return res.status(403).json({ message: "Invalid token" });
    req.user = user;
    next();
  });
}

// Protected routes
app.get("/balance", authenticateToken, (req, res) => {
  res.json({ balance: user.balance });
});

app.post("/deposit", authenticateToken, (req, res) => {
  const { amount } = req.body;
  user.balance += amount;
  res.json({ message: `Deposited ${amount}`, balance: user.balance });
});

app.post("/withdraw", authenticateToken, (req, res) => {
  const { amount } = req.body;
  if (amount > user.balance) {
    return res.status(400).json({ message: "Insufficient balance" });
  }
  user.balance -= amount;
  res.json({ message: `Withdrew ${amount}`, balance: user.balance });
});

// Start server
app.listen(3000, () => console.log("Server running on port 3000"));
```

Expected Output: 1. Successful login returns a **JWT token**. 2. Accessing `/balance` with token returns **current balance**. 3. **Deposits increase balance**, withdrawals decrease balance. 4. **Invalid/missing tokens** return error messages. 5. **Withdrawals exceeding balance** return an error.

Learning Outcomes: - Learned to implement **JWT-based authentication** in Express.js. - Understood how to **protect API routes** with middleware. - Learned **error handling** for secure banking operations. - Practiced **testing APIs** with Postman.