

Practice 2 - Backend

Title: JWT Authentication for Secure Banking API Endpoints

Objective:

Learn how to implement secure authentication in an Express.js application using JSON Web Tokens (JWT). This task helps you understand how to generate tokens, verify them in middleware, and protect sensitive API routes to ensure only authorized users can access banking operations.

Concept Overview:

JWT (JSON Web Token) is a compact and self-contained way for securely transmitting information between a client and server. Tokens are signed using a secret key and can be verified to ensure authenticity. Middleware is used to protect routes that should only be accessed by authenticated users.

Steps / Procedure:

Step 1: Initialize Project

```
mkdir jwt-banking-api
cd    jwt-banking-api
npm init -y
npm install express jsonwebtoken body-parser
```

Step 2: Create server.js

```
const express =
require('express'); const jwt =
require('jsonwebtoken'); const bodyParser
= require('body-parser'); const app =
express(); const PORT = 3000;

app.use(bodyParser.json());

const USER = { username: 'user1', password: 'password123' };
const SECRET_KEY = 'myjwtsecret'; let balance = 1000;

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' }); } });

function verifyToken(req, res, next) { const
authHeader = req.headers['authorization'];
if (!authHeader) return res.status(403).json({ message: 'Token missing' });

const token = authHeader.split(' ')[1]; jwt.verify(token, SECRET_KEY, (err,
decoded) => { if (err) return res.status(403).json({ message: 'Invalid or
expired token' }); req.user = decoded;

next();
}); }

app.get('/balance', verifyToken, (req, res) => res.json({ balance }));
```

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

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

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

Step 3: Run the Server

node server.js

Step 4: Test the API using Postman or curl

- 1■■ Login (POST /login)
- 2■■ Access /balance without token
- 3■■ Access /balance with valid token
- 4■■ Deposit money
- 5■■ Withdraw money

Expected Output:

Expected Output

The image displays five sequential screenshots of a REST client interface, showing the process of authenticating a user and performing banking transactions.

- First Screenshot:** A POST request to `http://localhost:3000/login` with a JSON body containing `username: "user1"` and `password: "password123"`. The response is a 200 OK with a JSON body containing a `token`.
- Second Screenshot:** A GET request to `http://localhost:3000/balance` without a body. The response is a 403 Forbidden with a message: `"Invalid or expired token"`.
- Third Screenshot:** A GET request to `http://localhost:3000/balance` with an `Auth` header containing the token from the first screenshot. The response is a 200 OK with a JSON body containing `balance: 1000`.
- Fourth Screenshot:** A POST request to `http://localhost:3000/deposit` with a JSON body containing `amount: 250` and the `Auth` header. The response is a 200 OK with a message `"Deposited $250"` and `newBalance: 1250`.
- Fifth Screenshot:** A POST request to `http://localhost:3000/withdraw` with a JSON body containing `amount: 150` and the `Auth` header. The response is a 200 OK with a message `"Withdraw $150"` and `newBalance: 1100`.

Result:

- JWT successfully secures API routes.
- Unauthorized requests are blocked.
- Users can deposit and withdraw money only after authentication.- Token verification ensures secure access to banking data.