Sunday, 8 October 2023

Authentication & Authorisation

1. Authentication is the process that will ensure the identity of the person requesting to your apis.

For example college doesn’t allow any random student to sit in their classes. Only the students who’s had paid the fees and enrolled in the course can sit and attend the classes. Because they are authenticated students who are having the access to courses as they paid for this.

In same way we don’t want any random person to make request to our apis. So Authentication is the mechanism which allows us to fix the identity of User making request.

Example of Authentication
1. Google Login

2. Face Book Login

3. OTP Login

4. Email/Password Login

2. Authorisation: It is the process of checking if the authenticated user has access to specific resource or not. For example if you have paid the fees for BCA then you can only attend BCA classes not B.Tech because have access to BCA Only.

You are authenticated user but have access to only BCA not B.Tech. So Authorisation is the same process where we check if the user making request for a specific resource , is he/she allowed to access that resource.

JWT (JSON Web Tokens) is a widely used authentication mechanism in web applications. It allows you to securely transmit information between the client and server as a JSON object. In Express.js, you can implement JWT authentication to protect routes and ensure secure communication between your application's components. Below, I'll provide detailed information on JWT authentication in Express.js, along with a real-life example.

What is JWT Authentication?

JWT authentication involves creating and verifying JSON Web Tokens to authenticate and authorize users. A JWT is a compact, URL-safe means of representing claims between two parties. It consists of three parts: Header, Payload, and Signature. The header and payload are Base64-encoded JSON objects, and the signature is used to verify the integrity of the tok

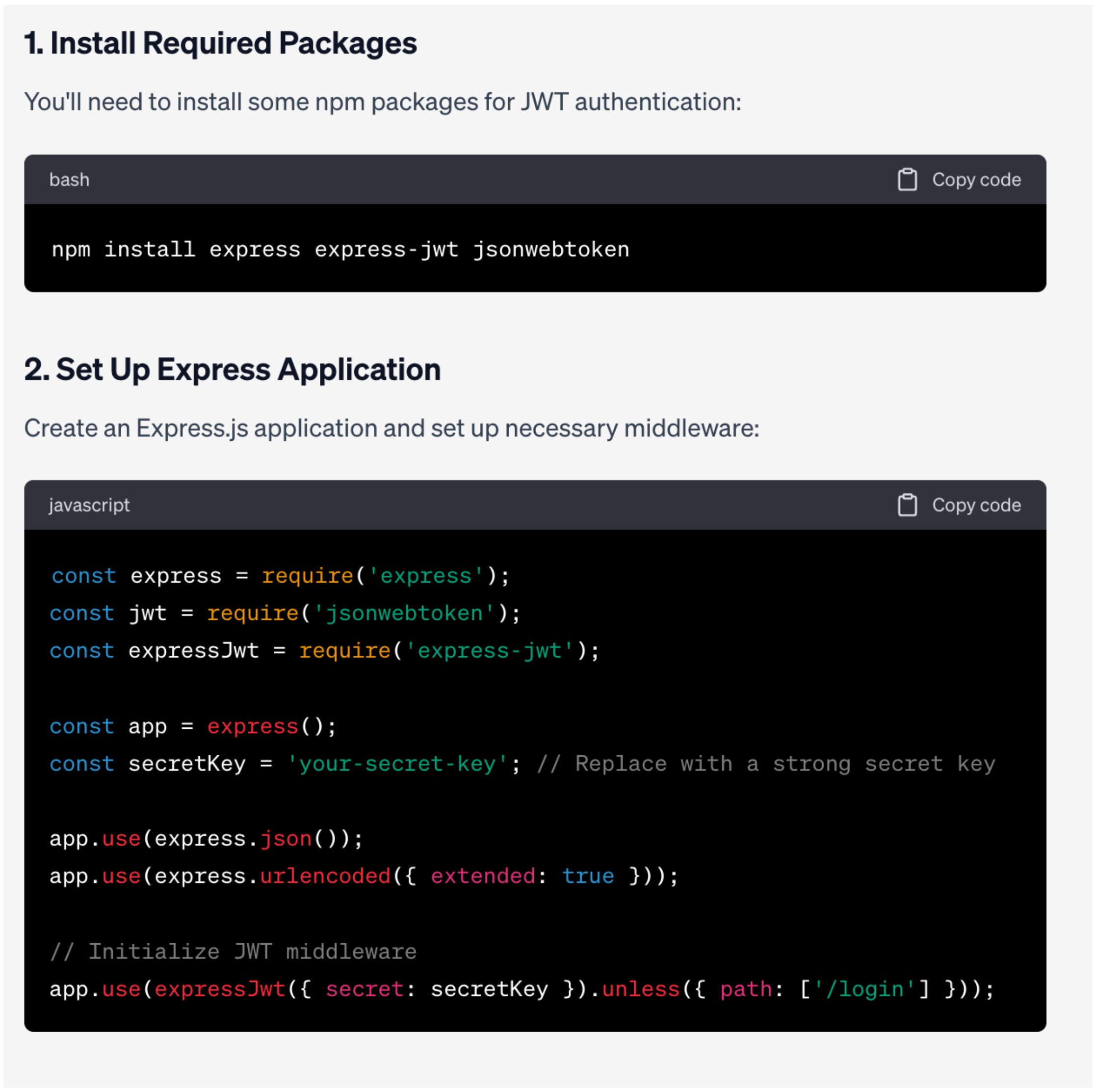
Simple Example (What is token) :

Let us suppose you wanted to go to water park. So if you try to take entry directly in waterpark the guard will not allow you as you don't have the ticket. So first you will go to counter and make payment and then return you will get a ticket. And that ticket will allow you to take all the rides in water park. If you lost the ticket then again guard will kick you out. Your ticket is the pass to use your services. And ticket is valid for some fix amount of time only after that you have to pay again to get new one.

In the same way whenever a user is trying to login in our website first we have to check if the email, password given by him is correct or not if not correct then send the response back with status: false.

But if the user is giving correct password and email then we have to give him some kind of ticket that will enable him to use our other Protected apis.

So here we use jsonwebtoken library. It helps us to sign a token whenever user has given the valid credentials. We sign a token with secret and send that token to user in response and user will save that token in frontend and every time user want's to access the protected api's user have to call that api by passing that token (like ticket in above example) in header and with this token we can verify it's a genuine user.



1. Install Required Packages

You'll need to install some npm packages for JWT authentication:

bash Copy code

```bash
npm install express express-jwt jsonwebtoken
```

2. Set Up Express Application

Create an Express.js application and set up necessary middleware:

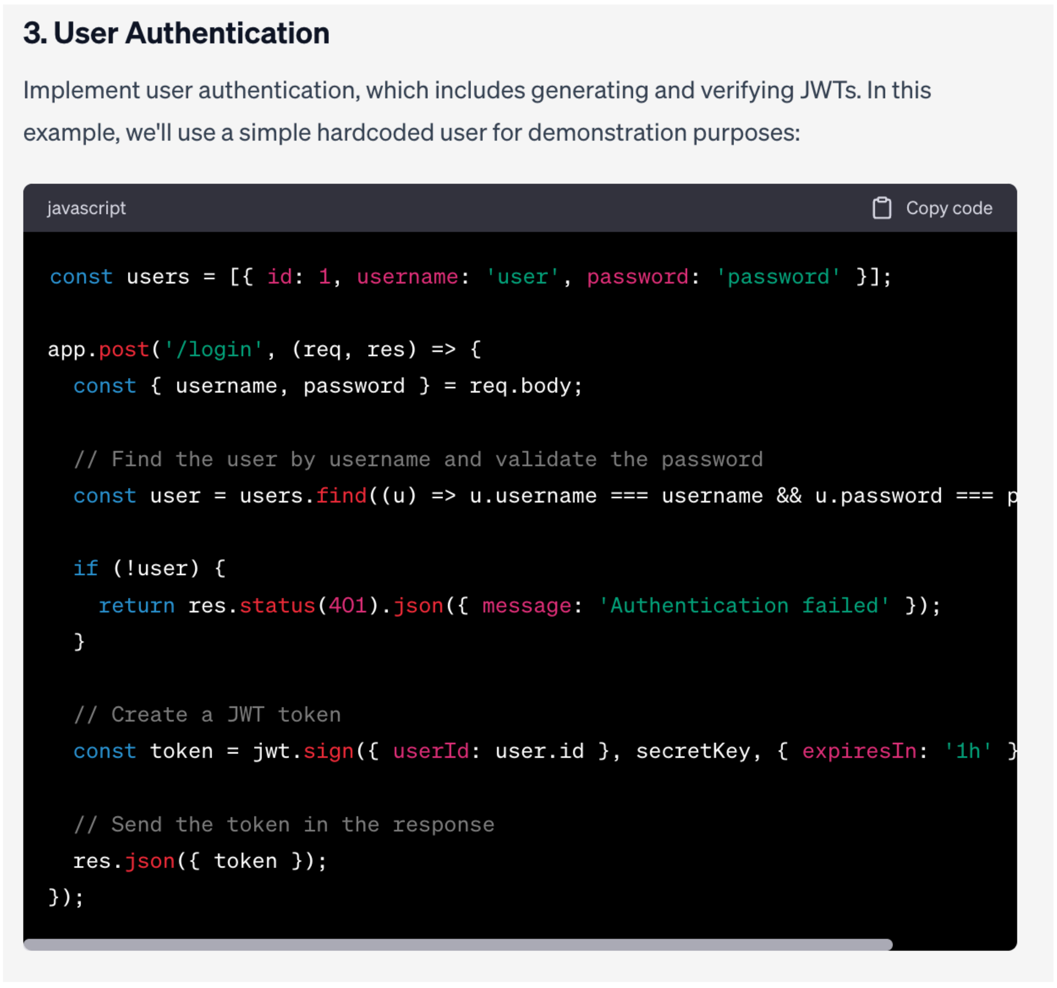
javascript Copy code

```javascript
const express = require('express');
const jwt = require('jsonwebtoken');
const expressJwt = require('express-jwt');

const app = express();
const secretKey = 'your-secret-key'; // Replace with a strong secret key

app.use(express.json());
app.use(express.urlencoded({ extended: true }));

// Initialize JWT middleware
app.use(expressJwt({ secret: secretKey }).unless({ path: ['/login'] }));
```



3. User Authentication

Implement user authentication, which includes generating and verifying JWTs. In this example, we'll use a simple hardcoded user for demonstration purposes:

javascript Copy code

```javascript
const users = [{ id: 1, username: 'user', password: 'password' }];

app.post('/login', (req, res) => {
  const { username, password } = req.body;

  // Find the user by username and validate the password
  const user = users.find((u) => u.username === username && u.password === p

  if (!user) {
    return res.status(401).json({ message: 'Authentication failed' });
  }

  // Create a JWT token
  const token = jwt.sign({ userId: user.id }, secretKey, { expiresIn: '1h' }

  // Send the token in the response
  res.json({ token });
});
```

How to verify the token?

If we have apis and we wanted that only the logged in user will able to access that apis. So we will generally create a middleware and add it in all the protected apis.

Steps to follow in middleware

1. Fetch the auth token from the Header (Authroization)
2. Verify the token if not verified then block the request
3. If token. Verified then store the details fetch out from token in req and call the next() method.