JWT Authentication in the Frontend: Enhancing Security in Web Applications

In this blog, we will learn how we can use the JWT token in our front end to authenticate the user and prevent unauthorized access to our Web application/website.

In my previous blog, I created a API server that provides us with the endpoints for login, and as soon as we log in we get the data of the logged-in user along with the access token and refresh token. That we can use in our front end to authenticate our identity to the server. In this Blog, I will be using React JS for my front end.

Let’s have a revision.

What is JWT?

JWT stands for JSON Web Token. It is an open standard (RFC 7519) for securely transmitting information between parties as a JSON object. JWTs are commonly used for authentication and authorization in web applications and APIs.

This information can be verified and trusted because it is digitally signed. JWTs can be signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA.

A JWT is composed of three parts: a header, a payload, and a signature.

1. Header — The header consists of two parts: the algorithm used to sign the token such as HMAC SHA256 or RSA, and the type of token, which is JWT in this case.

2. Payload — The payload contains the claims. The claims are entity specific such as we can pass user_id or username or anything we want.
3. Signature — The signature is formed by combining the encoded header, encoded payload, and a secret key known only to the server. It is used to verify the integrity of the token and to ensure that it is not been tampered.

How does JWT Work?

The typical flow used when working with JWT token involves —

1. User Authentication — First the user Authenticate itself to the server by passing the credentials.
2. Token Generation — As soon as the user has been authenticated a token is been generated which contains some user details as a payload that will be need by the front end as well as back end to perform validations.
3. Token Storage — This part is where the server has sent the token to the front end in response to the login request and the front end stores it in the local Storage or in session.
4. Making Request — Whenever the client makes a request to the back end servers it will pass the JWT access token in the header to authenticate its identity to the server. This is the main purpose of the JWT.
5. Token Verification — The server then validates the request headers i.e the JWT access token and if the token is valid then the server gives the access to the requested resources and if the token is not valid then it will return 401 status code i.e user unauthenticated.

What does jwt expired mean?

For Improving the security, there is an additional feature in JWT where we can define the expiration time i.e after a specific period of time the access token will not be valid. We cannot use the JWT access token after it is expired. We need to generate a new one to access the server.

For this we use refresh access token that is responsible for getting the new access token for the client.

Now we will develop our Front end for using the JWT access token.

We will be creating a login form for seeing how to use JWT in front end. So lets first create a react app, I will be using Vite for creating my React app.

```
npm create vite@latest
```

This will then ask you for the project name that you want to give and then select react and simple javascript. This will create a folder that will contain the app. Then we need to install all the necessary packages.

```
cd project_name
npm i
```

After creating our React app now we will make a form for log in. That will let the user access the server. And when the user login using the username and password, the back end will give the access along with the access token and the refresh token.

So first lets create our Login form.

App.js

```
import axios from "axios";
function App(){
const [user, setUser] = useState(null);
const [username, setUsername] = useState("");
const [password, setPassword] = useState("");
const handleSubmit = async(e)=>{
e.preventDefault();
```

```
  try{
  const response = await axios.post("/api/login",{username,password})
  setUser(response.data)
}catch(error){
  console.log(error)
}
return(
<div className="container">
{user ? (
<div className="login">
          <form onSubmit={handleSubmit}>
            <span className="formTitle">Lama Login</span>
            <input
              type="text"
              placeholder="username"
              onChange={(e) => setUsername(e.target.value)}
            />
            <input
              type="password"
              placeholder="password"
              onChange={(e) => setPassword(e.target.value)}
            />
            <button type="submit" className="submitButton">
              Login
            </button>
          </form>
</div>:<span>User has been loggedIn </span>}
</div>
)
}
```

App.css

```
.container {
  font-family: "Quicksand", sans-serif;
}
.login{
  width: 200px;
  height: 200px;
  padding: 20px;
  position: absolute;
  top: 0;
  bottom: 0;
  left: 0;
  right: 0;
  margin: auto;
  border: 1px solid lightgray;
  border-radius: 10px;
}

.formTitle {
  color: teal;
  font-weight: bold;
}

.submitButton {
  width: 100px;
  padding: 10px;
  border: none;
  border-radius: 10px;
  background-color: teal;
  color: white;
  cursor: pointer;
}
```

Now we will write the code for after login functionality in the return of the function App(). After login the user with the privilege can perform the delete operation. After adding the functionality the App.js file will

look like this.

```
function App() {
  const [user, setUser] = useState(null);
  const [username, setUsername] = useState("");
  const [password, setPassword] = useState("");

const handleSubmit = async (e) => {
    e.preventDefault();
    try {
      const res = await axios.post("/login", { username, password });
      setUser(res.data);
    } catch (err) {
      console.log(err);
    }
  };

  const handleDelete = async (id) => {
    setSuccess(false);
    setError(false);
    try {
      await axios.delete("/users/" + id, {
        headers: { authorization: "Bearer " + user.accessToken },
      });
      setSuccess(true);
    } catch (err) {
      setError(true);
    }
  };

return (
    <div className="container">
      {user ? (
        <div className="home">
          <span>
            Welcome to the <b>{user.isAdmin ? "admin" : "user"}</b> dashboard{" "}
            <b>{user.username}</b>.
          </span>
          <span>Delete Users:</span>
          <button className="deleteButton" onClick={() => handleDelete(1)}>
            Delete John
          </button>
          <button className="deleteButton" onClick={() => handleDelete(2)}>
            Delete Jane
          </button>
          {error && (
            <span className="error">
              You are not allowed to delete this user!
            </span>
          )}
          {success && (
            <span className="success">
              User has been deleted successfully...
            </span>
          )}
        </div>
      ) : (
        <div className="login">
          <form onSubmit={handleSubmit}>
            <span className="formTitle">Lama Login</span>
            <input
              type="text"
              placeholder="username"
              onChange={(e) => setUsername(e.target.value)}
            />
            <input
              type="password"
              placeholder="password"
```

```
              onChange={(e) => setPassword(e.target.value)}
            />
            <button type="submit" className="submitButton">
              Login
            </button>
          </form>
        </div>
      )}
    </div>
  );
}
```

When we login using the login form we get the user data along with the access token and the refresh token. We pass the access token when we perform a delete request to the server. But there is something that can go wrong, that is the access token can get expire, that will raise error so we need to check the access token before passing it to the server and if it has been expired we need to get the new access token using the refresh token.

```
const axiosJWT = axios.create()

  axiosJWT.interceptors.request.use(
    async (config) => {
      let currentDate = new Date();
      const decodedToken = jwt_decode(user.accessToken);
      if (decodedToken.exp * 1000 < currentDate.getTime()) {
        const data = await refreshToken();
        config.headers["authorization"] = "Bearer " + data.accessToken;
      }
      return config;
    },
    (error) => {
      return Promise.reject(error);
    }
  );

const handleDelete = async (id) => {
    setSuccess(false);
    setError(false);
    try {
      await axiosJWT.delete("/users/" + id, {
        headers: { authorization: "Bearer " + user.accessToken },
      });
      setSuccess(true);
    } catch (err) {
      setError(true);
    }
  };
```

By doing this it will handle the access token expiring issue and get you the new access token prior to sending the request to the server.

Here you can find the GitHub link.

Thank you for reading my blog and I hope you learn something new.