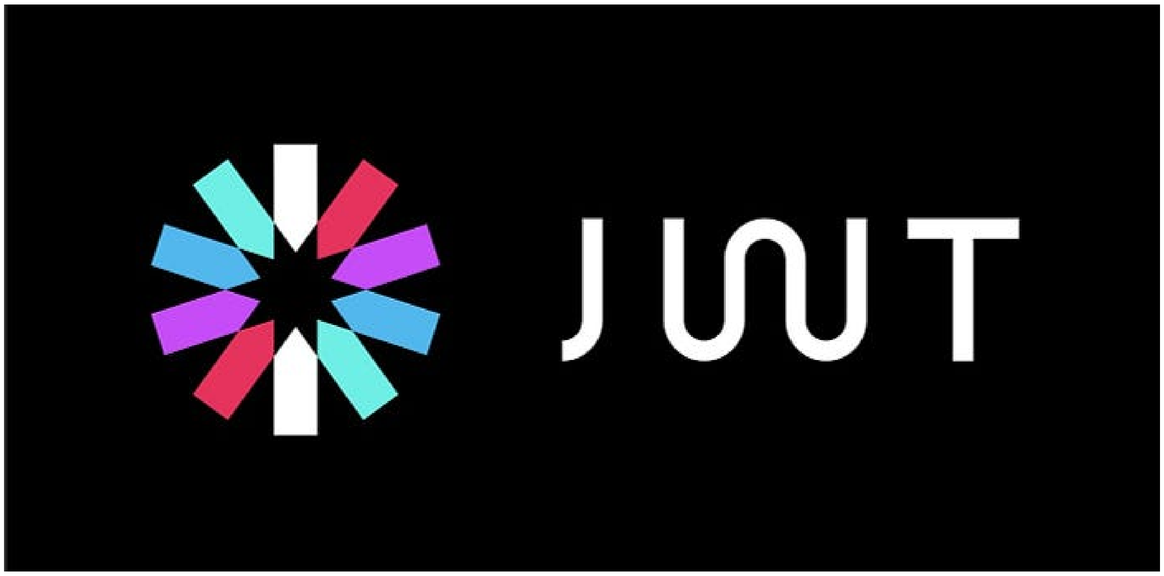
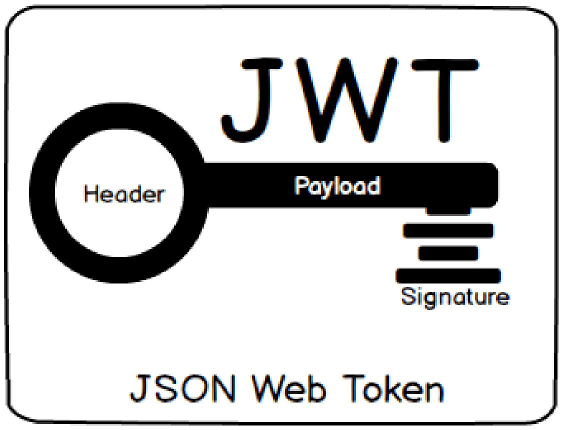
Building Secure User Authentication with JWT: A Deep Dive into JSON Web Tokens

In this blog, we will learn how to secure our login and our website's functionality using the JWT i.e. JSON Web Tokens in our MERN Website.



What is JWT?



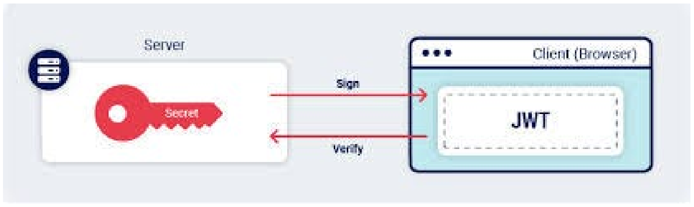
JSON Web Token is an open industry standard used to share information between two entities, usually a client (like your app's frontend) and a server (your app's backend). They contain JSON objects which have the information that needs to be shared.

Each JWT is also signed using cryptography (hashing) to ensure that the JSON contents (also known as JWT claims) cannot be altered by the client or a malicious party.

When we use JWT in our infrastructure we are using tokens for our authentication this token is signed using cryptography i.e. very much secure and unique.

So the way it works is first the user or the client makes a request to the server for authentication and the server authenticates the user and gives a JWT i.e access Token in return this access Token is signed by a cryptographic algorithm along with a unique key that we give to the JWT algorithm.

In this token we can save multiple things in a dictionary format i.e key value pair. The things we can save is username, user_id, isAdmin etc for the specific user.



Lets talk how we can implement JWT in our NodeJS Back-end.

So lets start, with first creating our node js project and installing all of our dependencies.

First initialize your node project.
```
npm init
```

Then we will install all the dependent modules that we will require.
```
npm i express jsonwebtoken
```

After installing the module we will first start our express server.
```
const express =require('express');
const app = express();



app.listen(8800,()=>{
    console.log('Your server available at http://localhost:8800')
})
```

As we have initialized our server we will start working on our JWT implementation in our backend. For that we will first require the JWT in our index.js file.
```
const jwt =require("jsonwebtoken");
```

And then we will make a function that will generate a access token for us. To generate an access token, jwt gives us a function called .sign() using this we will create a access token for every user. In this function we will pass a dictionary that will contain the user specific data such as userId, username etc in it, along with this data we need to pass a secretKey and we can also use option in it such as expiresIn option.
```
const generateAccessToken = (user)=>{
    return jwt.sign({id:user.id,isAdmin:user.isAdmin},"mySecretKey",{expiresIn:"30s"})
}
```

After that we will create a post endpoint for login that will be responsible for providing the access token to the client after authenticating the users credentials.
```
app.post("/api/login",(req,res)=>{
    const {username,password} = req.body;
    const user = users.find(u=>{
        return u.username === username && u.password===password
    })
    if(user){
        const access_token = generateAccessToken(user);
        res.json({username,isAdmin:user.isAdmin,access_token});
    }else{
        res.status(400).json("Username or password is incorrect");
    }
})
```

In the above endpoint we will be sending the client, the access token and the users data.In this way we generate the access token and give it to the client. And whenever the client wants to make the request the client need to send it in its header request.

Now we will make a verifier function for verifying the user request. This function will be used as a middle ware whenever we will be requiring to authenticate the user.
```
const verify = (req,res,next)=>{
    const authHeader =req.headers.authorization;
    if(authHeader){
        const token = authHeader.split(" ")[1];

        jwt.verify(token,"mySecretKey",(err,user)=>{
            if(err){
                return res.status(403).json("Token is not Valid.")
            }
            req.user =user;
            next();
        })
    }else{
        res.status(401).json("You are not authenticated")
    }
}
```

To increase the security of our application we will be using the refresh token along with the access token. As soon as the access token is expired the user can use this refresh token to get a new access token this will increase the security of our application. And we will save the refresh token in our database so that we can keep the latest copy of our refresh token.

If the attacker gets this refresh token the attacker will not be able to do anything because whenever the refresh token is used to get a new access token we are assigning a new refresh token to the user as well as saving it in our db.

SO lets create a function that will give us our refresh token.

```
const generateRefreshAccessToken = (user)=>{
    return jwt.sign({id:user.id,isAdmin:user.isAdmin},"myRefreshSecretKey")
}
```

As I will be saving my refresh token in the database for that I will need to define a model for our refresh token. In this example I am using MongoDB as my Database.

```
const mongoose = require('mongoose');

const RefreshTokenSchema = new mongoose.Schema({
    refreshToken:{
        type:String,
        required:true
    },
    userId:{
        type:String,
        required:true,
        unique:true
    },
},
{timestamps:true}
)

module.exports = mongoose.model("RefreshToken",RefreshTokenSchema);
```

Then we will create a post endpoint for updating the access token and also the refresh token in our data base.

```
const refreshSecretKey="YourSecretKey";

app.post("/api/refresh/:id",async(req,res)=>{
    const refresh_token = req.body.token;
    const user_id = req.params.id;
    let rr= {}
    const userRefreshToken = await RefreshToken.findOne({userId:user_id})
    if(!userRefreshToken){
        return res.status(403).json("Refresh token is not Generated!")
    }else if(refresh_token!==userRefreshToken.refreshToken){
        return res.status(403).json("Refresh token is not valid!")
    }
    jwt.verify(refresh_token,refreshSecretKey,(err,user)=>{
        // refreshTokens.filter((token)=> token!==refresh_token);
        const newAccessToken = generateAccessToken(user)
        const newRefreshToken = generateRefreshAccessToken(user)
        const update = await RefreshToken.findOneAndUpdate({userId:user_id},{$set:{refreshToken:rr.data.refreshToken}},)
        res.status(200).json({
            accessToken:newAccessToken,
            refreshToken:newRefreshToken,
        })
    })

});
```

So this the back end part of the JWT. To use this in the front-end we will first login and then we will get the access token. As we get the access token we will save the access token in a userState or we can save it in a session and then use it whenever we want to make a request to the back-end server for the authentication purpose.

That part we will discuss in our next blog on how to use the JWT in the frontend how to store it and how to update the access token.

I hope you got to learn something new and if you have any doubt or if you have any suggestion you can leave a comment I will get back to you as soon as possible.

GitHub Link.

Thank You!!!

👋 If you find this helpful, please click the clap 👏 button below a few times to show your support for the author 👇

🚀Join FAUN Developer Community & Get Similar Stories in your Inbox Each Week