

## Agenda

- JWT
- Multer

## JWT

- JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object.
- This information can be verified and trusted because it is digitally signed.
- JWTs can be signed using a secret
- These are useful in

### 1. Authorization

- This is the most common scenario for using JWT.
- Once the user is logged in, each subsequent request will include the JWT, allowing the user to access routes, services, and resources that are permitted with that token.

### 2. Information Exchange

- JSON Web Tokens are a good way of securely transmitting information between parties.
- Because JWTs can be signed—for example, using public/private key pairs—you can be sure the senders are who they say they are.
- Additionally, as the signature is calculated using the header and the payload, you can also verify that the content hasn't been tampered with.

## JSON Web Token structure

In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

### 1. Header

- The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used.

### 2. Payload

- The second part of the token is the payload, which contains the claims. Claims are statements about an entity (typically, the user) and additional data.

### 3. Signature

- To create the signature part you have to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that. Therefore, a JWT typically looks like the following:

```
xxxxx.yyyyy.zzzzz
```

- Putting all together the output is three Base64-URL strings separated by dots that can be easily passed in HTML and HTTP environments.

## Encoding and Decoding a JWT

- Encoding a JWT involves transforming the header and payload into a compact, URL-safe format.
- The header, which states the signing algorithm and token type, and the payload, which includes claims like subject, expiration, and issue time, are both converted to JSON then Base64URL encoded.
- These encoded parts are then concatenated with a dot, after which a signature is generated using the algorithm specified in the header with a secret or private key.
- This signature is also Base64URL encoded, resulting in the final JWT string that represents the token in a format suitable for transmission or storage.
- Decoding a JWT reverses this process by converting the Base64URL encoded header and payload back into JSON allowing anyone to read these parts without needing a key.
- However, "decoding" in this context often extends to include verification of the token's signature.
- This verification step involves re-signing the decoded header and payload with the same algorithm and key used initially, then comparing this new signature with the one included in the JWT.
- If they match, it confirms the token's integrity and authenticity, ensuring it hasn't been tampered with since issuance.

## JWT installation and usage

### 1. add/install the jwt module

```
npm install jsonwebtoken  
#OR  
yarn add jsonwebtoken
```

### 2. Usage

```
// import the module jwt  
const jwt = require("jsonwebtoken")  
  
// create token  
const token = jwt.sign({id: '234336653'}, 'secret')  
  
//verify the token  
const decoded = jwt.verify(token, 'secret')
```

## Multer

- Multer is a node.js middleware for handling multipart/form-data, which is primarily used for uploading files.

```
const express = require('express')
const multer  = require('multer')
const upload = multer({ dest: 'uploads/' })

const app = express()

app.post('/profile', upload.single('avatar'), function (req, res, next) {
  // req.file is the `avatar` file
  // req.body will hold the text fields, if there were any
})

app.post('/photos/upload', upload.array('photos', 12), function (req, res, next) {
  // req.files is array of `photos` files
  // req.body will contain the text fields, if there were any
})
```

SUNBEAM INFOTECH