EXERCISES QUE TRABAJAREMOS EN EL CUE

- EXERCISE 1: VERIFICACIÓN DE LOS COMPONENTES DE UN JWT

EXERCISE 1: VERIFICACIÓN DE LOS COMPONENTES DE UN JWT

El objetivo de este ejercicio es plantear una guía paso a paso para verificar los conceptos básicos de los componentes de un JWT (Header, Playload y el Signature).

Principales pasos que se realizarán para visualizar los componentes de un JWT (To-Do):

1. Instalar y usar el paquete jwt-encode para generar un webtokens json.
2. Instalar y usar el paquete jwt-decode para visualizar la decodificación de un JWT.
3. Instalar y usar el paquete node.jwt para visualizar la codificación y decodificación de un JSON Web Tokens.
4. Generar un token aleatorio desde la terminal y colocarlo tokens de seguridad.
5. Hacer uso del debugger de jwt.io para generar un tokens.
6. Hacer uso del debugger de jwt.io para visualizar la decodificación del tokens generado.

1. INSTALAR Y USAR EL PAQUETE JWT-ENCODE PARA GENERAR UN WEBTOKENS JSON

Procedemos a crear un directorio donde alojaremos el proyecto de codificación y decodificación de token JWT, con las librerías correspondientes:

```
$ mkdir code_encode_jwt
```

Aperturamos el VSC:

```
$ code .
```

Iniciamos la consola o la terminal, e inicializamos el proyecto:

```
$ npm init -y
```

COMPONENTES DE UN JWT

Instalamos el paquete jwt-encode que crea un Json Webtoken (JWT), es un modulo pequeño que solo permite la creación por medio del algoritmo HS256.

```
$ npm i jwt-encode
```

Codificamos el siguiente código:

```
// importamos el modulo
const sign = require('jwt-encode');
// Definimos la clave secreta para la firma
// Signature
const secret = 'clave_secreta';
// Construcción del payload
const data = {
    sub: 'pedroperez@test.com', // Objeto o usuario que emite el JWT
    name: 'Pedro Perez',
    iat: 1645565846.857, // Marca temporal de emision del JWT
};
console.log(data);
// devuelve un string firmado, pasando como argumento el
// Playload y la Signature
const jwt = sign(data, secret);
console.log(jwt);
```

Observamos la salida en la consola:

```
$ node index.js

{
    sub: 'pedroperez@test.com',
    name: 'Pedro Perez',
    iat: 1645565846.857
}

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiJwZWRyb3BlcmV6QHRlc3QuY29t
IiwibmFtZSI6IlBlZHJvIFBlcmV6IiwiaWF0IjoxNjQ1NTY1ODQ2Ljg1N30.wm9U1LdyZaQPD
8HxLI2XqzwZ6BZ4mKlKixcgYB9K9w0
```

Tenemos la formación de Tokens:

header.payload.signature

Seguidamente, procedemos a agregar el proveedor que emite la identidad (**iss**), objeto o usuario en nombre del cual fue emitido (**sub**), y el tiempo de expiración (**ext**):

```
// importamos el modulo
const sign = require('jwt-encode');
// Definimos la clave secreta para la firma
// Signature
const secret = 'clave_secreta';
// Construcción del payload
const data = {
    name: 'Pedro Perez',
    iat: 1645565846.857, // Marca temporal de emision del JWT
    iss: "localhost@jwr.com", // proveedor de identidad que emite el JWT
    sub: 'pedroperez@test.com', // Objeto o usuario que emite el JWT
    ext: Date.now() / 1000 + (60 * 5) // Tiempo de expiración por 5 min

};
console.log(data);
// devuelve un string firmado, pasando como argumento el
// Playload y la Signature
const jwt = sign(data, secret);
console.log(jwt);
```

Observamos la salida en la terminal. Al transcurrir los 5 minutos, vemos que el token cambió:

```
$ node index.js {
    name: 'Pedro Perez',
    iat: 1645565846.857,
    iss: 'localhost@jwr.com',
    sub: 'pedroperez@test.com',
    ext: 1645570109.409
}
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiUGVkcm8gUGVyZX
oiLCJpYXQiOjE2NDU1NjU4NDYuODU3LCJpc3MiOiJsb2NhbGhvc3RAandyLmNvbS
IsInN1YiI6InBlZHJvcGVyZXpAdGVzdC5jb20iLCJleHQiOjE2NDU1NzAxMDkuND
A5fQ.xXIQVeieC_rjmsG8vJHZjBmO3a5lhMA_nhUz_TAetUE
```

2. INSTALAR Y USAR EL PAQUETE JWT-DECODE PARA VISUALIZAR LA DECODIFICACIÓN DE UN JWT

Jwt-decode es una pequeña biblioteca de navegador, que ayuda a decodificar tokens JWT que están codificados en Base64Url.

Esta biblioteca no valida el token, se puede decodificar cualquier JWT bien formado. Debe validar el token en la lógica del lado del servidor, usando algo como: express-jwt, koa-jwt, Owin Bearer JWT, entre otros.

Instalación:

```
npm install jwt-decode
```

Procedemos a decodificar el token anterior:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiUGVkcm8gUGVyZXoiLCJpYXQi
OjE2NDU1NjU4NDYuODU3LCJpc3MiOiJsb2NhbGhvc3RAandyLmNvbSIsInN1YiI6InBlZHJvc
GVyZXpAdGVzdC5jb20iLCJleHQiOjE2NDU1NzAwOTEuMjE3fQ.r4T7xF8RYskckHBpN92AjgV
rZXoRd8ovu-mKTkO4PXA
```

Codificamos el siguiente código para decodificar:

```
// importamos la libreria de jwt-decode
const jwt_decode = require('jwt-decode');

//Decodificando el token
var token =

"eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiUGVkcm8gUGVyZXoiLCJpYXQ
iOjE2NDU1NjU4NDYuODU3LCJpc3MiOiJsb2NhbGhvc3RAandyLmNvbSIsInN1YiI6InBlZHJv
cGVyZXpAdGVzdC5jb20iLCJleHQiOjE2NDU1NzAwOTEuMjE3fQ.r4T7xF8RYskckHBpN92AJg
VrZXoRd8ovu - mKTkO4PXA ";

var decoded = jwt_decode(token);

console.log(decoded);
```

Salida en la terminal:

```
{
    name: 'Pedro Perez',
    iat: 1645565846.857,
    iss: 'localhost@jwr.com',
    sub: 'pedroperez@test.com',
    ext: 1645570091.217
}
```

Podemos imprimir el header, agregando al código lo siguiente:

```
var decodedHeader = jwt_decode(token, {
    header: true,
});
console.log(decodedHeader);
```

La salida en la terminal es la siguiente:

```
{ alg: 'HS256', typ: 'JWT' }
```

3. **INSTALAR Y USAR EL PAQUETE NODE.JWT PARA VISUALIZAR LA CODIFICACIÓN Y DECODIFICACIÓN DE UN JSON WEB TOKENS**

Una biblioteca ligera para codificar y decodificar JSON Web Tokens (JWT). Instalación de la librería:

```
npm install node.jwt --save-dev
```

Codificamos el siguiente código:

```
// Importamos la librería node.jwt
const jwt = require('node.jwt');

// Construcción del payload
const payload = {
    name: 'Pedro Perez',
    iss: "localhost@jwr.com", // proveedor de identidad que emite el JWT
    sub: 'pedroperez@test.com', // Objeto o usuario que emite el JWT
    ext: Date.now() / 1000 + (60 * 60 * 24 * 2), // Tiempo de expiración
por 2 dias
    iat: Date.now() / 1000, // Marca temporal de emision del JWT
    nbf: Date.now() / 1000 - (60 * 5)
};


// Definimos la clave secreta para la firma
// Signature
const secret = jwt.secret('clave_secreta');

// Generando el token
let tokenJWT = jwt.encode(payload, secret);
console.log(tokenJWT);

// Decodificando el token
let result = jwt.decode(tokenJWT, secret);
console.log(result);
```

La interfaz de la librería es la siguiente:

```
/**
 * jwt encode
 * @param  {Object} payload   payload jwt
 * @param  {String} secret    clave secreta
 * @param  {String} algorithm algoritmo
 * @param  {Object} header    jwt header
 * @return {String}           jwt
 */
encode (payload, secret, algorithm = 'HS256', header = { type: 'JWT' })

/**
 * jwt decode
 * @param  {String}  token    token a verificar
 * @param  {String}  secret   clave secreta
 * @param  {Boolean} noVerify es verificado
 * @return {Object}           resultado verificado
 */
decode (token, secret, noVerify = false)
```

ALGORITMOS

Por defecto, el algoritmo para codificar es HS256. Los algoritmos admitidos para la codificación y decodificación son: HS256, HS384, HS512 y RS256.

Codificar usando HS512:

```
jwt.encode(payload, clave_secreta, 'HS512')
```

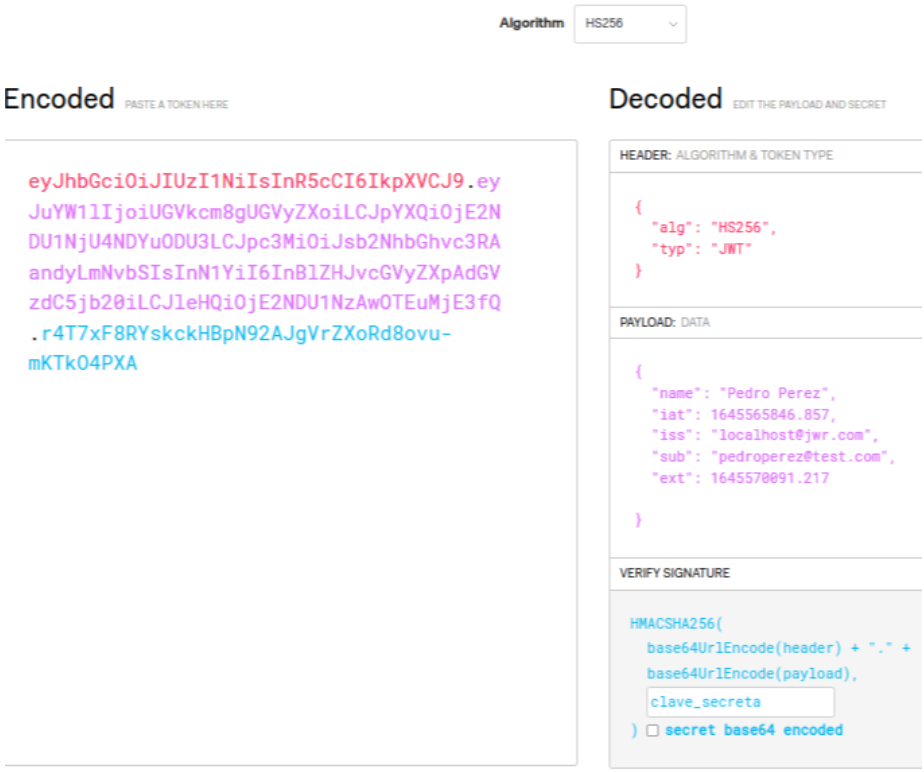
4. **GENERAR UN TOKEN ALEATORIO DESDE LA TERMINAL Y COLOCARLO TOKENS DE SEGURIDAD**

Para generar un token o clave secreta verdaderamente aleatorio desde la terminal, lo podemos hacer de la siguiente manera:

```
$ node
Welcome to Node.js v12.16.2.
Type ".help" for more information.
> require("crypto").randomBytes(64).toString("hex")
'c10f110263f0526dea36879b300212903b84d272b22b55dd5d48aafb8cfd7d3be8a91a196
aa3d5fcdfc73a9400c6003a6c8a7601f961fdd56535897bf94bdf81'
```

5. **HACER USO DEL DEBUGGER DE JWT.IO PARA GENERAR UN TOKENS**

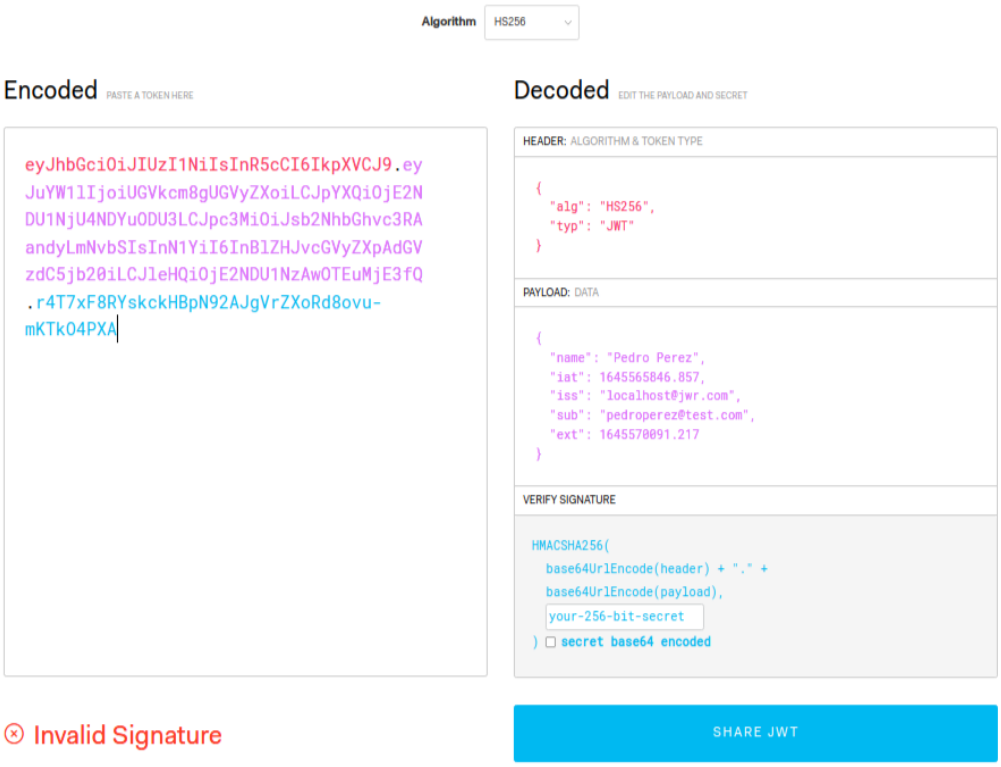
Ingresamos al sitio web: https://jwt.io/, buscamos el Debugger, y colocamos los siguientes datos:



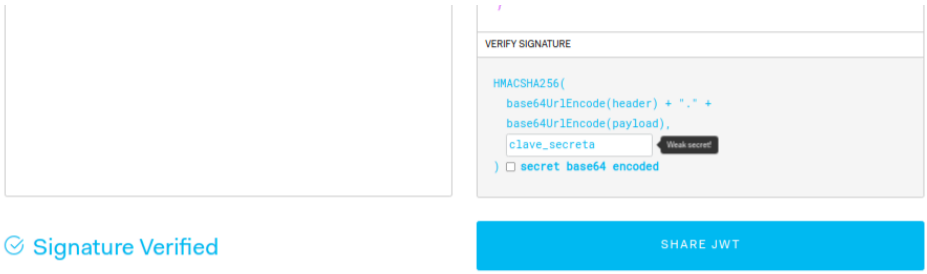
6. HACER USO DEL DEBUGGER DE JWT.IO PARA VISUALIZAR LA DECODIFICACIÓN DEL TOKENS

Token:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJuYW1lIjoiUGVkcm8gUGVyZXoiLCJpYXQiOjE2NDU1NjU4NDYuODU3LCJpc3MiOiJsb2NhbGhvc3RAandyLmNvbSIsInN1YiI6InBlZHJvcGVyZXpAdGVzdC5jb20iLCJleHQiOjE2NDU1NzAwOTEuMjE3fQ.r4T7xF8RYskckHBpN92AJgVrZXoRd8ovu-mKTkO4PXA
```



Observamos que es invalido por la clave secreta, ya que posee "***your-256-bit-secret***" y lo que debemos colocar es ***"clave_secreta"*** en verificación de la firma, quedando:



NOTA:

Para una mejor resolución del proyecto, es recomendable tener actualizada la versión de Node.Js a la más reciente, y codificar con ES6.