TEXT CLASS REVIEW

TEMAS A TRATAR EN LA CUE

- Securización mediante JWT
- Implementación en Express
- Instalando el paquete jsonwebtoken
- Instalando el paquete express-jwt
- Configurar el paquete
- Crear un proyecto básico con seguridad JWT paragenerar un token
- Validar el token en llamadas HTTP
- Invocando y servicio pasando un token como parámetro
- Decodificando Tokens
- Validar tokenHeader

IMPLEMENTACIÓN EN EXPRESS

Instalando el paquete jsonwebtoken:

Para generar un JWT, es necesario instalar el paquete jsonwebtoken. Este paquete se puede instalar via npm utilizando el siguiente comando:

```
npm install jsonwebtoken
```

Una vez finalizada la instalación se debe importar el paquete al proyecto deseado, de la siguiente manera:

```
npm install jsonwebtoken
```

INSTALANDO EL PAQUETE EXPRESS-JWT

Este módulo proporciona middleware Express para validar JWT (JSON Web Tokens) a través del módulo jsonwebtoken. El contenido de JWT decodificado está disponible en el objeto de solicitud.

Para instalarlo se ejecuta el siguiente comando:

```
npm install express-jwt
```

Para configurar cada uno de estos paquetes, debemos consultar la documentación que nos proporcionará instrucciones detalladas para personalizar nuestro uso de estos componentes. Aquí está la documentación de jsonwebtoken y aquí está la documentación de express-jwt.

CREAR UN PROYECTO BÁSICO CON SEGURIDAD JWT PARA GENERAR UN TOKEN UTILIZANDO JSONWEBTOKEN

Prerequisitos: instalar node.js y crear un entorno de desarrollo local. Instalar e importar el módulo jsonwebtoken.

Generar un token: Para firmar un token, se necesitan tres fragmentos de información como se ha expuesto anteriormente. La palabra secreta o el token secret es una cadena aleatoria larga que se utiliza para cifrar y descifrar los datos.

Para generarla, una opción es usar la librería crypto de node.js de la siguiente manera:

```
> require('crypto').randomBytes(64).toString('hex')
// '09f26e402586e2faa8da4c98a35f1b20d6b033c6097befa8be3486a82958
7fe2f90a832bd3ff9d42710a4da095a2ce285b009f0c3730cd9b8e1af3eb84df6611'
```

Ahora, almacene la palabra secreta en el archivo `.env` de su proyecto

```
TOKEN_SECRET=09f26e402586e2faa8da4c98a35f1b20d6b033c60...
```

Para utilizar este token a un archivo node.js y usarlo debe instalar el paquete `dotenv` de la siguiente manera:

```
npm install dotenv
```

Importarlo en el archivo como sigue:

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

// obteniendo las variables de configuración
dotenv.config();

// Acceso a las variables de configuración
process.env.TOKEN_SECRET;
```

La información que se encriptará en el token puede ser algo como el ID del usuario o nombre de usuario o un objeto más complejo. En cualquier caso, debe ser un identificador para un usuario específico. El tiempo de caducidad del token es una cadena, como 1800 segundos (30 minutos), que detalla cuánto tiempo pasará hasta que el token deje de ser válido.

Seguidamente un ejemplo de una función para firmar tokens:

```
function generateAccessToken(username) {
  return jwt.sign(username, process.env.TOKEN_SECRET,{expiresIn: '1800s'});
}
```

Esto se puede devolver desde una solicitud para iniciar sesión o iniciar sesión en un usuario:

```
app.post('/api/createNewUser', (req, res) => {
  // ...

 const token = generateAccessToken({ username: req.body.username });
 res.json(token);
  // ...

});
```

Este ejemplo toma el nombre del usuario del `req` (request/solicitud) y suministra el token como el `res` ***(response)***

De esta manera se ejemplifica como jsonwebtoken puede ser usado para generar un JWT.

CREAR UN PROYECTO BÁSICO CON SEGURIDAD JWT PARA GENERAR UN TOKEN UTILIZANDO EXPRESS-JWT

En este caso se generará un token haciendo uso del algoritmo HS256:

```
var jwt = require('express-jwt');

app.get('/protected',
    jwt({
        secret: 'shhhhhhared-secret',
        algorithms: ['HS256']
    }),
    function(req, res) {
        if (!req.user.admin) return res.sendStatus(401);
        res.sendStatus(200);
    });
```

1. Parámetros requeridos:

   El parámetro algorithms es requerido para prevenir un potencial ataque.

2. Opciones adicionales:

   Se puede especificar la audiencia y/o el emisor, lo cual es muy recomendable por motivos de seguridad:

```
jwt({
    secret: 'shhhhhhared-secret',
    audience: 'http://myapi/protected',
    issuer: 'http://issuer',
    algorithms: ['HS256']
})
```

En caso de que la configuración del JWT incluya la expiración (`exp`), ésta debe ser considerada.

Si está utilizando un token secret codificado en URL base64, se debe pasar un búfer con codificación base64 en lugar de una cadena:

```
jwt({
    secret: Buffer.from('shhhhhhared-secret', 'base64'),
```

```
    algorithms: ['RS256']
})
```

VALIDAR EL TOKEN EN LLAMADAS HTTP

Suponga que se crea un API llamada **"/posts"** que solo permitirá acceso cuando el JWT sea válido. La API correrá en un servidor separado, pero utilizará el **ACCESS_TOKEN_SECRET** compartido, de tal manera que los JWT que son generados en un servidor extra son reconocidos por el servidor que corre la API **"/posts"**.

a) Creemos un archivo llamado **"validateToken.js"** de la siguiente manera:

```
require('dotenv').config;

const express = require('express');
const app = express();
app.use(express.json())

const jwt = require('jsonwebtoken');
const port = process.env.PORT;

app.listen(port, () => {
    console.log('Servidor de validación ejecutándose en $ {
        port
    }...')
})

app.get("/posts",validateToken, (req,res)=> {
console.log('El token es válido'); console.log(req.user.user);
res.send(`${req.user.user} acceso exitoso `)
})
```

Note que se está haciendo uso de una función llamada validateToken dentro de **app.get()**. Cualquier función que es agregada como segundo parámetro en **app.get()** es tratada como un "middleware" y es ejecutada antes de que **app.get** se ejecute.

b) La función **validateToken** primero leerá el token de la cabecera de la solicitud y verificará si el JWT es válido.

Si el JWT no es válido la función **validateToken** enviará el código 403 como respuesta y el resto de la función **app.get()** no se ejecutará.

En caso de que el token sea válido la función `validateToken` ejecutará la función `next()` y el resto de la función `app.get()` será ejecutada.

```
function validateToken(req, res, next){ //obtiene el token de la cabecera
de la solicitud

        const authHeader =  req.headers["authorization"];
        const token =  authHeader.split("")[1];

//La cabecera de la solicitud contiene el token, la cadena se divide y de
utiliza el
//segundo valor del arreglo resultante.

        if (token == null) res.sendStatus(400).send("Token no presente")
        jwt.verify(token, process.env.ACCESS_TOKEN_SECRET, (err, user) =>
{
                if (err){
                        res.status(403).send("El token es invalido")
                }else{
                        req.user = user
                        next()
                }})
```

Tenga en cuenta que el método `jwt.verify()` utiliza un (err, usuario) como devolución de llamada. El usuario contiene la siguiente información:

```
{user: 'Kyle', iat:1629837426, exp: 1629838326 }
```