# JWT

## JWT y la seguridad en servicios WEB (Parte II)

{desafío} latam_

***Implementar mecanismos de seguridad a un servicio REST de un servidor Express utilizando JWT de acuerdo al entorno Node.js.***

- Unidad 1:
  API REST
- Unidad 2:
  Subida de archivos al servidor
- Unidad 3:
  JWT

***Te encuentras aquí***

{desafío} latam_

# ¿Qué aprenderás en esta sesión?

- *Implementa un servicio para la obtención de un token de usuario utilizando Express y Node.js.*

- *Implementa validación de token en las peticiones de una servicio REST acorde a las buenas prácticas.*

{desafío} latam_

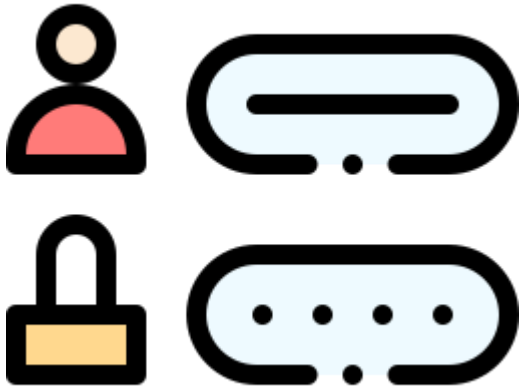
# Recordemos:

# ¿Que es un Token y para que nos sirve?

# /* El paquete jsonwebtoken */

# El paquete jsonwebtoken

Tal y como su nombre lo indica, este paquete nos permitirá utilizar la tecnología JWT en nuestros servidores, específicamente ocupando los métodos “sign” y “verify” para la generación y verificación de tokens respectivamente.



# Generación de tokens

Para generar tokens con este paquete solo debemos ocupar el método “sign” y declarar los siguientes parámetros:

```
jwt.sign(<payload>, <secretOrPrivateKey>, <[options, callback]>)
```

# Generación de tokens

Estos parámetros se describen de la siguiente manera:

- **payload**: Es la data que deseamos codificar, normalmente serán las credenciales de un usuario recibidas en una consulta desde el cliente.

- **secretOrPrivateKey**: Representa la llave secreta usada para la firma, es de suma importancia no exponer esta llave.

- **Options**, **Callback**: Define las propiedades del token, aquí es donde definiremos próximamente el tiempo de vida de un token expresado en segundos.

# /* Consideraciones previas */

# Consideraciones previas

En los archivos de esta sesión encontrarás el **Archivo Apoyo - Servidor base Parte II**. En este documento podrás acceder al código de un servidor con Express básico y un arreglo de usuarios ubicado en la carpeta “data”.

Luego de descargar el comprimido y abrirlo con tu editor de código, inicia un nuevo proyecto npm y procede a instalar el paquete jsonwebtoken como una dependencia del proyecto utilizando el siguiente comando en la terminal:

```
npm i --save jsonwebtoken
```

# Consideraciones previas

Con el paquete instalado importa la dependencia almacenada en una constante. Tu código deberá verse como lo muestro a continuación:

```
const express = require('express')
const users = require('./data/users.js')
const app = express()
app.listen(3000, () => console.log('Servidor encendido en el puerto 3000'))
const jwt = require('jsonwebtoken')

app.get('/', (req, res) => {
 res.send('Probando sonido, si , si, si...')
})
```

# Ejercicio guiado:

# Autorizando a Steve

# Ejercicio guiado

***Autorizando a Steve***

La empresa Your Space Spa está trabajando actualmente en la creación de una red social inclinada al mundo literario, que busca darle un espacio a todas esas personas tímidas que les gusta escribir todo lo que hacen en el día a día con versos y poesía.

La empresa en su búsqueda de profesionales que lo ayuden en esta tarea, te ha contactado para desarrollar la autenticación y autorización de sus usuarios utilizando JWT para esto.

# Ejercicio guiado

***Autorizando a Steve***

Para comenzar se debe desarrollar un servidor que disponibilice en su ruta raíz un token generado para el usuario de nombre “Steve”, usando el paquete jsonwebtoken y su método “sign”.

- **Paso 1**: Crear una constante llamada “secretKey” cuyo valor será “Mi Llave Ultra Secreta”. Esta será la llave que usaremos para firmar los tokens.

# Ejercicio guiado

***Autorizando a Steve***

- **Paso 2**: Generar un nuevo token ocupando el método “sign” pasando como primer argumento el usuario de índice 1 (Steve) y como segundo argumento la llave secreta.

- **Paso 3**: Modificar la respuesta de la ruta raíz para devolver el token generado en el paso 2.

# Ejercicio guiado

## *Autorizando a Steve*

```
const express = require('express')
const users = require('./data/users.js')
const app = express()
app.listen(3000, () => console.log('Servidor encendido en el puerto 3000'))
const jwt = require("jsonwebtoken")

// Paso 1
const secretKey = 'Mi Llave Ultra Secreta'

// Paso 2
const token = jwt.sign(users[1], secretKey)

// Paso 3
app.get('/', (req, res) => {
  res.send(token)
})
```

# Ejercicio guiado

***Autorizando a Steve***

Ahora abre tu navegador y consulta al servidor con la dirección http://localhost:3000/, obtendrás el token generado, tal y como te muestro en la siguiente imagen:

# Ejercicio guiado

***Autorizando a Steve***

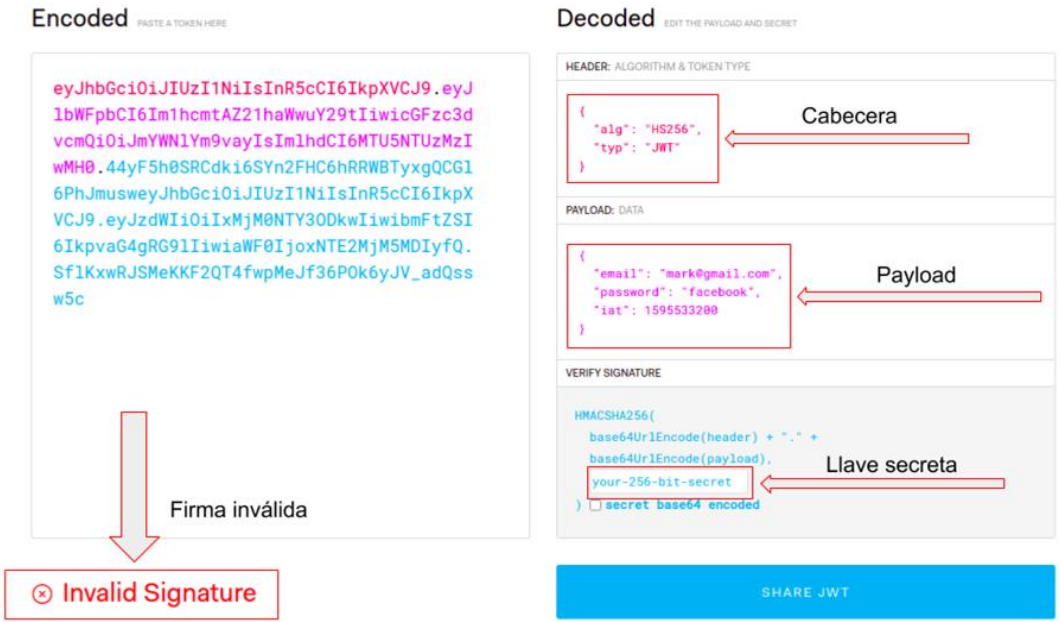
Muy bien, obtuvimos un token generado con el paquete jsonwebtoken, pero ¿Cómo podemos comprobar que en efecto esto funciona? Entra al sitio web de JWT y pega el siguiente token:

*eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJlbWFpbCI6Im1hcmtAZ21haWwuY29tIiwicGFzc3dvcmQiOiJmYWNlYm9vayIsImlhdCI6MTU5NTUzMzIwMH0.44yF5h0SRCdki6SYn2FHC6hRRWBTyxgQCGl6PhJmusw*

# Ejercicio guiado

## *Autorizando a Steve*

Al hacerlo obtendrás lo que se muestra en la siguiente imagen.




{desafío} latam_

# Ejercicio guiado

## *Autorizando a Steve*

***¿Qué podemos deducir de esto?***

Se ha generado en efecto un token, pero al pegarlo en la consola de JWT a pesar de identificar correctamente el payload con los datos de nuestro usuario (iat generado automáticamente), el token no es válido

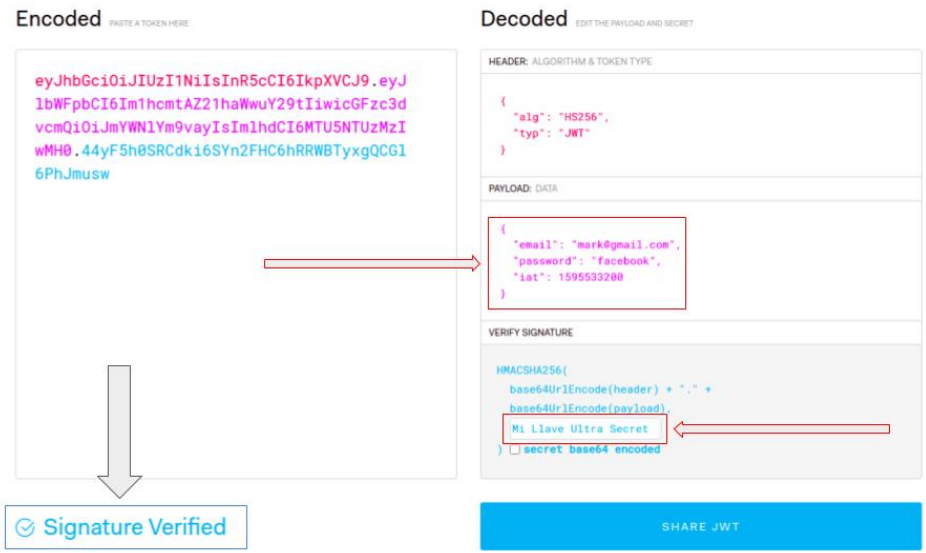
**¿Por qué?**

Sucede que la verificación de un token está basada en la llave secreta y si observas la imagen anterior, la llave secreta por defecto del sitio de JWT no es la misma a la que tenemos nosotros.

# Ejercicio guiado

## *Autorizando a Steve*

En la misma interfaz podemos modificar la llave ya que es un input que nos dispone el sitio web de JWT, intenta cambiarla por “Mi Llave Ultra Secreta” y deberás ver lo que te muestro en la siguiente imagen:




{desafío} latam_

# /* Verificación de tokens */

# Verificación de tokens

Hasta este punto hemos logrado la generación de tokens con el método “sign()” y lo hemos probado con la consola oficial de JWT, pero nos falta hacer esta verificación desde el servidor para integrarlo a nuestras aplicaciones.

Para realizar esta acción, tenemos el método “verify” del paquete jsonwebtoken el cual tiene la siguiente estructura:

```
jwt.verify(<token>, <secretOrPrivateKey>, <callback(err,
decoded)>)
```

# Verificación de tokens

Como puedes ver solo necesitamos pasarle el token a verificar y la famosa llave secreta. Recordemos que el token, es una combinación de caracteres como resultado de la codificación de un payload, dentro de este proceso es necesario incluir una llave secreta para poder decodificar y validar el token.

El método “verify” incluye una función callback que nos entrega un error (en caso de fallar la verificación) y como segundo parámetro obtendremos el token decodificado, es decir, la data dentro del token.

Entendiendo lo anterior, podríamos crear una ruta que reciba un token y luego de verificarlo devuelva el payload decodificado, y eso es justo lo que haremos en el siguiente ejercicio.

# Ejercicio guiado:

# Decodificando el token de Steve

# Ejercicio guiado

## *Decodificando el token de Steve*

Crear una ruta **GET /token** en el servidor que reciba un token como querystring, verificar su validez y retornar su payload decodificado o un mensaje indicando que el token es invalido. Sigue los pasos para la solución de este ejercicio:

- **Paso 1**: Crear una ruta **GET /token**.
- **Paso 2**: Almacenar en una constante el parámetro “token” extraído con query strings.
- **Paso 3**: Usar el método “verify” para verificar el token recibido y devolver el payload decodificado o un mensaje de error diciendo “Token inválido” según sea el caso.

# Ejercicio guiado

*Decodificando el token de Steve*

```
// Paso 1
app.get("/token", (req, res) => {
  // Paso 2
  const { token } = req.query;
  // Paso 3
  jwt.verify(token, secretKey, (err, data) => {
    res.send( err ? "Token inválido" : data );
  });
});
```

# Ejercicio guiado

## *Decodificando el token de Steve*

Prueba esta ruta entrando al navegador en su ruta raíz y obteniendo el token correspondiente al usuario “Steve”.

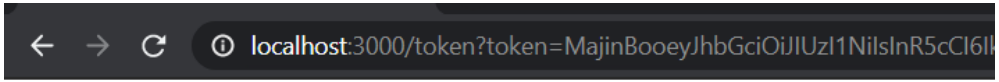
Luego utiliza el mismo token para armar y consultar la siguiente ruta http://localhost:3000/token?token=<token del usuario Steve>. Deberás obtener lo que te muestro en la siguiente imagen:


```
{
  "email": "steve@gmail.com",
  "password": "apple",
  "iat": 1607453850
}
```

# Ejercicio guiado

## *Decodificando el token de Steve*

Como puedes ver, hemos obtenido la data del usuario Steve, lo cual quiere decir que ¡Funciona! Ahora si intentas cambiar algo en el token de la ruta obtendrás lo que te muestro en la siguiente imagen:




Token inválido

# /* Expiración de tokens */

# Expiración de tokens

Hasta ahora hemos logrado generar tokens y comprobar su validez, sin embargo, nos falta una de las características que le suma seguridad a JWT: la fecha de expiración.

Con la fecha podremos limitar la vigencia de un token y “cerrarle las puertas” al usuario que intente acceder al sistema o consumir recursos con un token vencido.

# Expiración de tokens

Ya sabemos que dentro del payload de un token se genera de forma automática la propiedad “iat”, que identifica la fecha en segundos en la que fue creado el token.

Para definir la fecha de vencimiento, debemos declarar la propiedad “exp” que representa también en segundos la fecha de expiración.

Por supuesto, la expiración debe ser un tiempo mayor a la fecha actual, y el tiempo de vida será igual al tiempo de expiración menos el tiempo de creación, pasada esta cantidad en segundos se considera que el token caducó.

# Expiración de tokens

En el siguiente código te muestro un ejemplo de cómo generar un token incluyendo la propiedad “exp”, cuyo valor es el cálculo del momento en el que se crea más 1 minuto, es decir que el token durará 60 segundos.

```
const token = jwt.sign(
 {
   exp: Math.floor(Date.now() / 1000) + 60,
   data: user[0],
 },
 secretKey
)
```

# Ejercicio guiado:

# Persistencia del token

# Ejercicio guiado

## *Persistencia del token*

Cuándo usamos redes sociales normalmente podemos entrar y salir del sitio web manteniendo nuestra sesión activa, esto es gracias a que el token que recibimos en el inicio de sesión sigue almacenado en el navegador.

Para lograr este comportamiento agregar a nuestro servidor una ruta **/login** que reciba por query strings el email y contraseña de un usuario, y en caso de ser autenticado, devuelva un script que aloje un token con vigencia de 2 minutos en el LocalStorage del navegador

# Ejercicio guiado

*Persistencia del token*

- **Paso 1**: Crear una ruta /login.
- **Paso 2**: Extraer de las query Strings los parámetros email y password.
- **Paso 3**: Utilizar el método find para encontrar algún usuario que coincida con las credenciales recibidas.
- **Paso 4**: Declarar un condicional if que evalúe el resultado del método find, es decir, si se encontró un usuario con el email y password recibida.

# Ejercicio guiado

*Persistencia del token*

- **Paso 5**: En caso de encontrar un usuario, genera un token que expire en 2 minutos usando la instancia del usuario como data.

- **Paso 6**: Luego de generar el token, responde al cliente con un hiper enlace que dirija a una ruta /Dashboard, le dé la bienvenida utilizando el dato del email y utilice un script con el método “setItem” de localStorage para persistir el token en el navegador.

- **Paso 7**: En caso de no encontrar un usuario (caso else) devolver el mensaje “Usuario o contraseña incorrecta”.

# Ejercicio guiado

## *Persistencia del token*

```
// Paso 1
app.get("/login", (req, res) => {
  // Paso 2
  const { email, password } = req.query;
  // Paso 3
  const user = users.find((u) => u.email == email && u.password == password);
  // Paso 4
  if (user) {
    // Paso 5
    const token = jwt.sign(
      {
        exp: Math.floor(Date.now() / 1000) + 120,
        data: user,
      },
      secretKey
    );
    // Paso 6
    res.send(`
    <a href="/Dashboard?token=${token}"> <p> Ir al Dashboard </p> </a>

    Bienvenido, ${email}.

    <script>
    localStorage.setItem('token', JSON.stringify("${token}"))
    </script>
    `);
  } else {
    // Paso 7
    res.send("Usuario o contraseña incorrecta");
  }
});
```

# Ejercicio guiado

## *Persistencia del token*

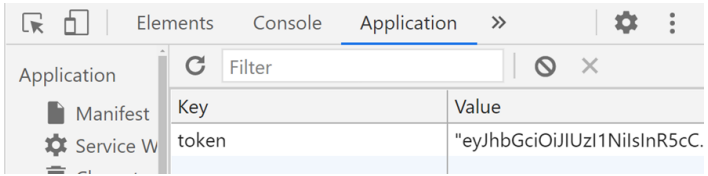
Ahora intentemos loguear al usuario Steve usando la siguiente dirección:

http://localhost:3000/login?email=steve@gmail.com&password=apple

Deberás recibir lo que te muestro en la siguiente imagen:

Ir al Dashboard

Bienvenido, steve@gmail.com.




{desafío} latam_

# Resumen

*Subtítulo*

- Para generar tokens con el paquete jsonwebtoken solo debemos ocupar el método “sign”.

- Además debemos declarar los siguientes parámetros:

    - payload
    - secretOrPrivateKey
    - Options, Callback

# ¿Existe algún concepto que no hayas comprendido?

# Próxima sesión...

- *Guía de ejercicios*

{desafío} latam_

{desafío}
latam_

Academia de
talentos digitales