

# Curso CFP

## CFP Programador full-stack

*Introducción a Autenticación II*

# Agenda


- Repaso
- Mensajes Codificados
- JSON Web Token
  - Aspectos Básicos
  - Ejemplos
- Manejo de JWT
  - Lectura
  - Escritura
- Demo en Vivo
- Ejercicios

# Repaso

## Pasos para autenticarnos

- Hacer un POST pasando user/pass en el body
- Si el user/pass figura en la DB, devolver un *token*
- Hacer el resto de los request agregando el token como header
- La API debería verificar que el token exista y sea correcto para proceder a la funcionalidad

# Repaso - Limitaciones

- Hasta el momento estamos devolviendo siempre el mismo token
- Podríamos personalizarlo, pero entraríamos a diseñar nuestro propio criterio
  - Tanto para armar como para escribir el token
  - Sobre todo en caso de agregar más cantidad de información
- Podemos ver el contenido del token a simple vista en el Postman
  - Cualquier persona malintencionada podría usar nuestro token 

# Mensajes Codificados (1)

- A veces no queremos que la información viaje de forma plana por un tema de seguridad
  - Por ejemplo si alguien nos mira el token que nos llega en el Postman, lo podría usar para acceder a un sistema con nuestro usuario
- Un mensaje codificado es aquél que está destinado para ciertas entidades
- Los destinatarios conocen la forma de decodificar el mensaje
  - De esta manera, un mensaje “ljkajsfasa” para nosotros no puede significar nada, pero para otra persona que sabe entenderlo, puede contener un mensaje

# Mensajes Codificados (2)

- Otro ejemplo: el mensaje “ipmb dnpn bñebt”
- Es un mensaje que está codificado, una persona cualquiera no sabe qué quiere decir
- Ahora, sabemos que la forma de decodificarlo es retroceder un paso en el abecedario para cada palabra
  - “ipmb dnpn bñebt”
  - “hola como andas”
- Este es un método básico para codificar y decodificar mensajes, existen otros más complejos

# Mensajes Codificados (3)

- Existen varias forma de codificar y decodificar un mensaje: encriptación, firmas digitales, etc.
- Estos mecanismos normalmente funcionan mediante el proceso de hashing
- Una función de hash funciona como cualquier función
  - Toma una entrada y devuelve una salida
- Una función de hash recibe un mensaje y lo devuelve codificado o “hasheado”
  - Ejemplo: hola → b221d9dbb083a7f33428d7c2a3c...
  - Se usó una función llamada SHA-256

# JSON Web Token (JWT)

- Mecanismo muy utilizado para hacer manejo de usuarios (entre otras cosas)
- Consiste en un token en formato JSON
  - Podemos agregarle la información que queramos
- Está firmado → es decir que hay que decodificarlo para ver lo que dice
- Ejemplos de datos que contienen estos tokens
  - Identificador de usuario
  - Tipo de acceso que solicita
  - Expiración del token



# Ejemplo de un JWT

The screenshot shows a REST client interface with a POST request to `http://localhost:8080/auth/login`. The request body is a JSON object with `username: "john"` and `password: "changeme"`. The response is a JSON object with an `access_token` field containing a long JWT string. Red boxes and text annotations highlight the request body and the response token, explaining their purpose in the authentication process.

**Mandamos user/pass para obtener el token**

**A partir de ahora, para acceder al resto de los endpoints no nos logueamos → mandamos directo este token**

# Manipulando un JWT

- Podemos ir a la página de JWT para ver qué es lo que tiene un token → <https://jwt.io/#debugger>

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6ImprvaG4iLCJzYW50IjEsIm1hdCI6MTU3MjksMTc4NCwiZXhwIjoxNTcyOTExODQ0fQ.eyJ1bWwGF0-20cYdV_1DHDcCwCAHmjT_V25cqTfn8o
```

**En este caso figura el user en el JWT → podemos poner más cosas**

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{  
  "username": "john",  
  "sub": 1,  
  "iat": 1572911784,  
  "exp": 1572911844  
}
```

VERIFY SIGNATURE

# Usando el JWT (1)

GET  Send Save

Params **Authorization** Headers (8) Body Pre-request Script Tests Settings Cookies Co

**TYPE**

Bearer Token

The authorization header will be automatically generated when you send the request. [Learn more about authorization](#)

Preview Request

Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)

Token

**Acá ponemos el token que obtuvimos en el POST**

eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1c2VybmFtZSI6Im...

Body Cookies Headers (6) Test Results Status: 200 OK Time: 6ms Size: 242 B Save Response

Pretty Raw Preview Visualize **BETA** JSON ⌵

```
1 {  
2   "userId": 1,  
3   "username": "john"  
4 }
```

# Usando el JWT (2)

The screenshot shows a REST client interface with the following details:

- Request:** Method `GET`, URL `http://localhost:8080/profile`. The **Authorization** tab is selected, showing a **Bearer Token** type. A text input field labeled **Token** contains the placeholder text `|token`. A warning message states: "Heads up! These parameters hold sensitive data. To keep this data secure while working in a collaborative environment, we recommend using variables. [Learn more about variables](#)".
- Response:** The **Body** tab is selected, showing a JSON response in **JSON** format. The response is: 

```
{  "statusCode": 401,  "error": "Unauthorized"}
```

. The status bar indicates **Status: 401 Unauthorized**, **Time: 5ms**, and **Size: 263 B**.

Red annotations highlight the token input field and the error response body.

**Intentamos acceder a un endpoint sin el token requerido**

**CUAC**

# Demo en Vivo

- A partir de la API de prueba
  - Pegarle al POST para obtener un token
  - Acceder al endpoint GET con el token recibido
- Tener cuidado que la API de ejemplo genera tokens que expiran después de cierto tiempo
  - En ese caso, generarlo de vuelta
  - Los tokens siempre tienen que tener un tiempo de expiración por cuestiones de seguridad

**Curso CFP**

**CFP**  
**Programador**  
**full-stack**

***Ejercicios***

# Ejercicios

- Partir de la API armada en la clase pasada, y agregar JWT en función al tutorial de NestJS <https://docs.nestjs.com/techniques/authentication>
- En la documentación se arma una API teniendo los usuarios hardcodeados en un arreglo interno
- La idea sería que esa información la tomemos desde la base de datos