

# JSON Web Token (Parte I)

## JWT

### Competencias

- Reconocer los elementos básicos de JWT como la estructura, token, header, payload, signature para implementar autenticación de usuarios en llamado a APIs.
- Realizar peticiones a APIs implementando el ciclo de vida del JWT para obtener datos protegidos.
- Implementar parámetros en el header de una petición para obtener los datos de una API protegida.

### Introducción

Hoy en día la gran mayoría de las empresas pone a disposición de los desarrolladores sus datos a través APIs, permitiendo tener acceso a millones de datos, los cuales se pueden analizar, procesar, crear, editar o eliminar recursos desde nuestras propias aplicaciones, etc. Esto es muy bueno para los desarrolladores ya que abre un mundo de posibilidades, pero también es una oportunidad para hacer mal uso de esta, por lo que las empresas deben poder controlar e identificar quién está accediendo a sus datos.

Proteger o autenticar una API, corresponde a agregar un usuario y contraseña en cada petición que se realiza. Hoy en día ya existen estándares de cómo hacer una buena autenticación aquí es donde se encuentra OAuth2.0, OpenID Connect y Json Web Token (JWT).

El concepto de autenticación y autorización tiene muchos más tópicos y participantes, estos temas están más relacionados con el mundo del backend, ya que se involucran sessions, validación de usuario contra base de datos, permisos, comunicación cliente-servidor etc, te recomiendo investigar sobre estos temas para que tengas una mirada más amplia de lo que es autenticación.

En esta lectura revisaremos el uso de JWT ya que es un estándar para la creación de token donde se puede enviar cualquier tipo de información útil del usuario, es liviano y nos permite disminuir las validaciones de usuario contra la base de datos.

## ¿Qué es JWT?

Json Web Token o JWT es un estándar abierto [RFC-7519](#) basado en JSON, que permite conectarnos de manera segura a un servidor a través de un token que obtenemos con un usuario y contraseña, un token es una cadena alfanumérica que contiene información o claims acerca del usuario que genero este token. El JWT es un token seguro ya que es firmado digitalmente y en cada petición el servidor verifica si este token es válido o no.

## Ciclo de vida de JWT

Siempre es importante conocer cómo funcionan las herramientas o tecnologías que utilizamos como desarrolladores web, por esto revisemos cómo opera el ciclo de vida del JWT, cómo se genera y cómo se va reutilizando en los diferentes request para obtener datos de nuestra API. En la siguiente imagen se describe el ciclo de vida del JWT.



Imagen 1. Ciclo de vida JWT.

Fuente: [OpenWebinars](#)

1. Se realiza un post al servidor enviando un usuario y contraseña.
2. Si el usuario y contraseña son válidos, el servidor genera un JWT.
3. Se retorna el JWT generado al cliente (este token se debe guardar para reutilizar posteriormente).
4. El JWT se debe enviar en el header de las siguientes peticiones.
5. El servidor recibe el JWT y valida su veracidad.
6. Si el JWT es válido, se retorna la información solicitada en el request.

Los pasos anteriores constituyen el flujo que debería seguir cualquier API para proteger sus datos, a lo largo de la lectura pondremos este flujo en acción con un ejemplo que podrás descargar y así preparar un ambiente local en tu computador que nos permitirá probar un ciclo de vida de JWT.

## Preparación del ambiente de trabajo

Para realizar de manera efectiva la siguiente lectura es necesario que cuentes con el ejemplo de la API funcionando de manera local en tu computador, para ello debes tener instalado la versión 10 o superior Node.js y descomprimir el Apoyo Lectura - JSON Web Token, en él encontrarás un proyecto desarrollado en Node.js que contiene las APIs o endpoints necesarios para llevar a cabo los ejemplos de la lectura.

### ¿Qué es Node.js?

Node.js es un entorno de ejecución de javascript que nos permite ejecutar javascript en ambientes diferentes a nuestro navegador, como por ejemplo utilizar javascript del lado del servidor.

### Pasos de instalación de nuestro proyecto en Node.js

Una vez descomprimido el zip del proyecto debes dirigirte a través del terminal a la carpeta del proyecto.



Imagen 2. Terminal ubicado en la carpeta del proyecto.

Fuente: Desafío Latam

Cuando estés ubicado ahí debes ejecutar la siguiente instrucción `npm install`, después de unos minutos deberías ver lo siguiente:

```
→ jwt-example-desafiolatam git:(master) npm install

> nodemon@2.0.4 postinstall /Users/roliva/Documents/DesafioLatam/jwt-example-desafioLatam/node_modules/nodemon
> node bin/postinstall || exit 0

Love nodemon? You can now support the project via the open collective:
> https://opencollective.com/nodemon/donate

npm WARN api-jwt-example@1.0.0 No description
npm WARN api-jwt-example@1.0.0 No repository field.

added 193 packages from 103 contributors and audited 193 packages in 3.901s
found 0 vulnerabilities

→ jwt-example-desafiolatam git:(master) |
```

Imagen 3. Resultado npm install.

Fuente: Desafío Latam

Eso significa que tu proyecto tiene todo listo para ser levantado, para hacer esto debemos ejecutar:

```
node index.js
```

Si todo está bien, deberías ver el siguiente mensaje: Listening server in <http://localhost:3000>.

Generar el primer JWT

Con el proyecto de ejemplo corriendo, abriremos **Postman** y haremos un post al endpoint **<http://localhost:3000/api/login>** utilizando algún correo de nuestra base de datos (archivo db/users.json) y la contraseña secret, con estas instrucciones Postman debería verse así:

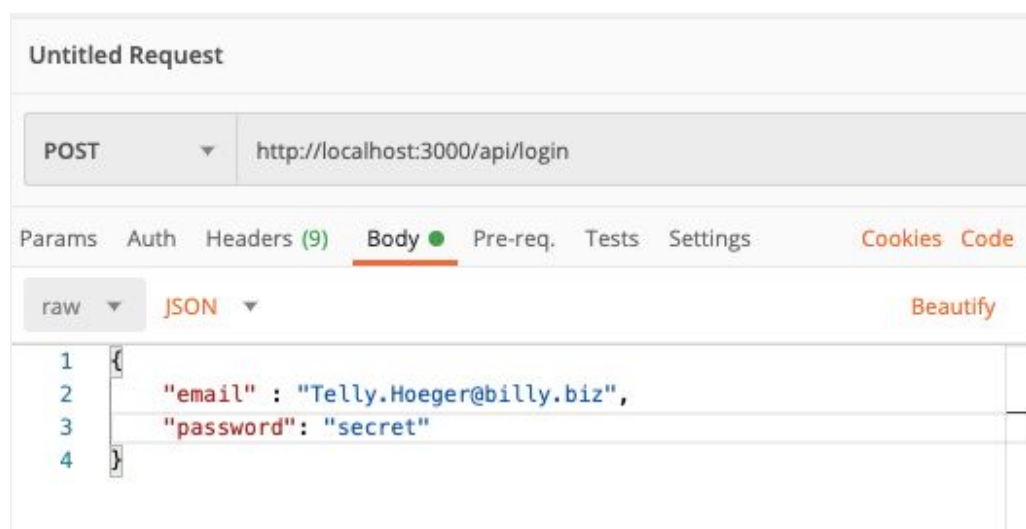


Imagen 4. Postman con usuario y contraseña antes de obtener el JWT.

Fuente: Desafío Latam

Ahora si presionamos el botón de send o enviar deberíamos tener como respuesta lo siguiente:

```
{
  "token":
  `eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NywibmFtZSI6Ikt1cnRpcyBXZWlzc25hdCI6InVzZXJuYW1lIjoiaW4uU2tpbGVzIiwiaWF0IjoxNTk0NTg0NDg1fQ.B-S_tP_4W4N1gMhgafApT6B-HdP9S6Nx6AiPhMafmw4"
}
```

Si utilizaste el mismo correo y obtuviste un token diferente está bien, no te preocupes ya que todos los token son diferentes.

## Estructura de un JWT

En el punto anterior generamos el primer JWT y se veía algo así:

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NywibmFtZSI6Ikt1cnRpcyBXZWlzc25hdCI6InVzZXJuYW1lIjoiaW4uU2tpbGVzIiwiaWF0IjoxNTk0NTg0NDg1fQ.B-S_tP_4W4N1gMhgafApT6B-HdP9S6Nx6AiPhMafmw4
```

¿Qué significa esta cadena alfanumérica? Para responder esta pregunta debemos visitar el sitio [jwt.io](https://jwt.io) en la sección debugger, como se muestra en la siguiente imagen:

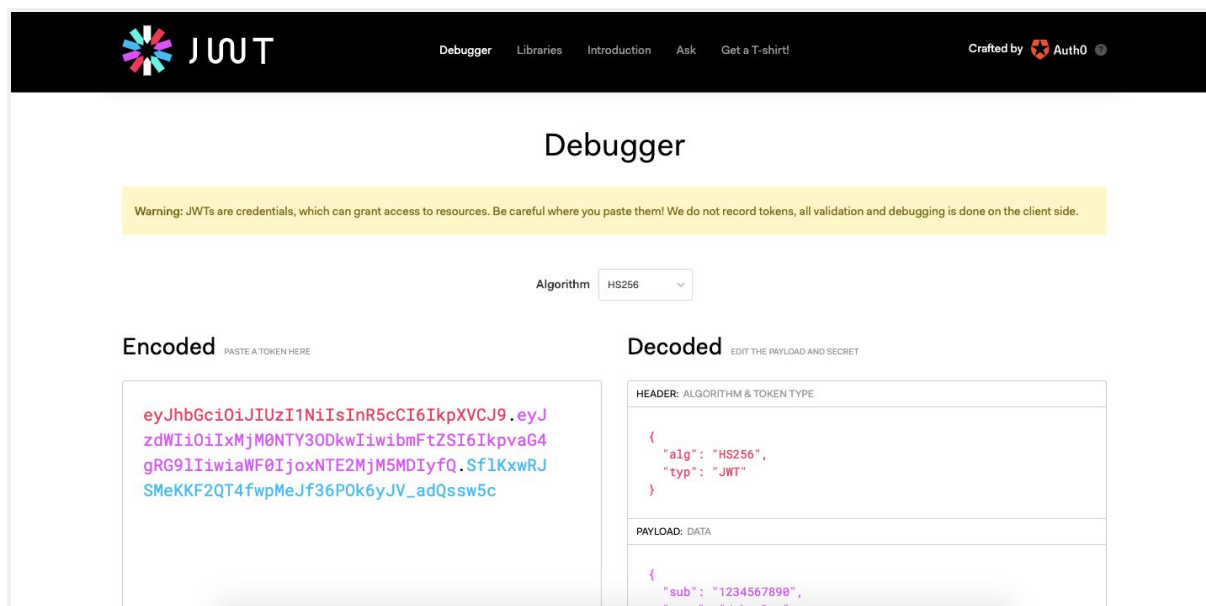


Imagen 5. Sitio web JWT.

Fuente: <https://jwt.io/>

Ahora copiamos y pegamos nuestro token en la caja que dice **encoded**, al pegar el JWT en la parte izquierda **decoded** podremos ver su descomposición.

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6NywibmFtZSI6Ikt1cnRpcyBXZWlzc25hdCI6InVzZXJuYW1lIjoiaW4uU2tpbGVzIiwiaWF0IjoxNTk0NTg0NDg1fQ.B-S_tP_4W4N1gMhgafApT6B-HdP9S6Nx6AiPhMafmw4
```

HEADER: ALGORITHM & TOKEN TYPE

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

PAYLOAD: DATA

```
{
  "id": 7,
  "name": "Kurtis Weissnat",
  "username": "Elwyn.Skiles",
  "iat": 1594584485
}
```

VERIFY SIGNATURE

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  your-256-bit-secret
) ☐ secret base64 encoded
```

Imagen 6. Estructura JWT.

Fuente: <https://jwt.io/>

Como podemos apreciar en la imagen 6 un JWT se compone de 3 partes las cuales se separan por un "." (punto):

Parte	Encoded	Definición
Header	eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9	Indica la codificación utilizada y el tipo de token en nuestro caso JWT
Payload	eyJpZCI6NywibmFtZSI6Ikt1cnRpcyBXZWlzc25hdCI6InVzZXJuYW1lIjoiaW4uU2tpbGVzIiwiaWF0IjoxNTk0NTg0NDg1fQ	Indica los claims o datos del usuario autenticado además de la fecha de caducidad del token en formato timestamp
Signature	B-S_tP_4W4N1gMhgafApT6B-HdP9S6Nx6AiPhMafmw4	Indica la firma con la que se verificó el token

Tabla 1. Partes del JWT.

Fuente: Desafío Latam

## Estrategias de autenticación con JWT

Ahora que obtuvimos el JWT, debemos guardarlo en algún lugar para poder reutilizarlo en las próximas peticiones. En nuestro cliente web contamos con 3 alternativas cookies, localStorage o sessionStorage.

- **Cookies:** Son un mecanismo de almacenamiento de información del usuario en el navegador, la información puede ser añadida desde el lado del cliente (JavaScript) o directamente desde el servidor web a través del headers de las peticiones.
- **LocalStorage y SesiónStorage:** Es un objeto (storage) que permite guardar información en el navegador en estructura de llave valor, la principal diferencia es que en el primero la información perdura de manera indefinida hasta que se elimine, mientras que en el segundo la información sólo persiste mientras el tab del navegador esté abierto.

Depende de nosotros cuál debemos utilizar, pero siempre hay que velar por la seguridad del token, la recomendación es que el backend persista el token en el cookies con el flag httpOnly:true, así la información sólo puede ser manipulada por el servidor y no por el cliente como podría suceder con los otros casos.

## Utilizar JWT para obtener recursos protegidos

Ya obtuvimos un JWT y aprendimos cómo está compuesto, pero nos falta utilizarlo para poder acceder a recursos protegidos en una API, es decir, el JWT actuará como nuestra llave de acceso que le indicará al servidor que somos un usuario válido en su sistema. Para ver esto en acción abramos Postman e intentemos hacer un get a la siguiente ruta **http://localhost:3000/api/posts**. Deberíamos obtener lo siguiente:

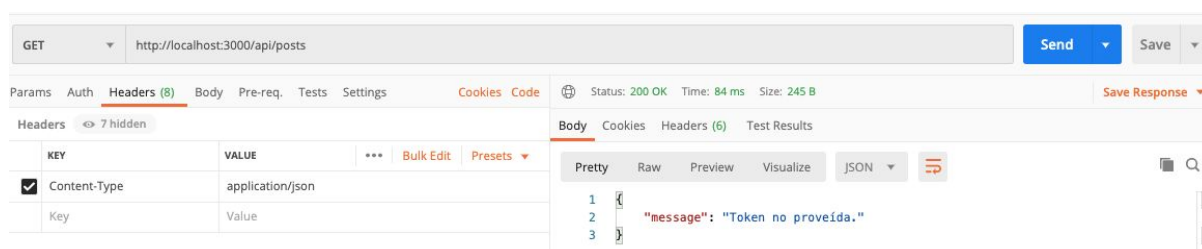


Imagen 7. Resultado Postman recursos protegidos.

Fuente: Desafío Latam

Para acceder a los datos debemos indicarle al servidor que somos un usuario válido y esto lo hacemos generando un token, a través de un usuario y contraseña como lo hicimos en pasos anteriores.

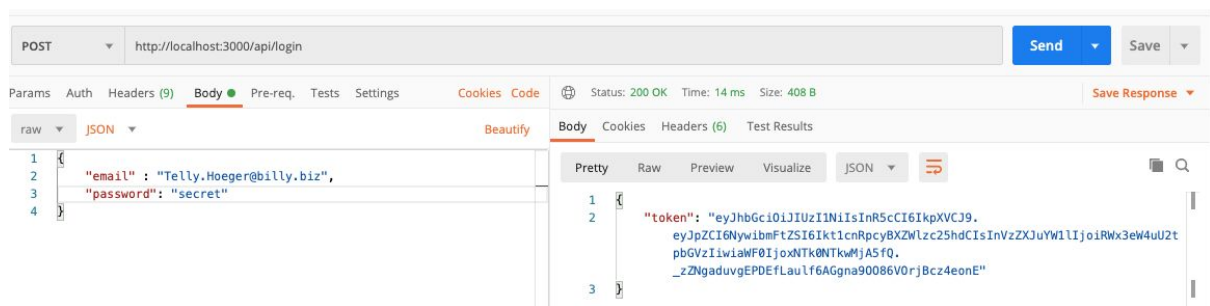


Imagen 8. Resultado Postman.

Fuente: Desafío Latam

En postman deberíamos ver lo siguiente:

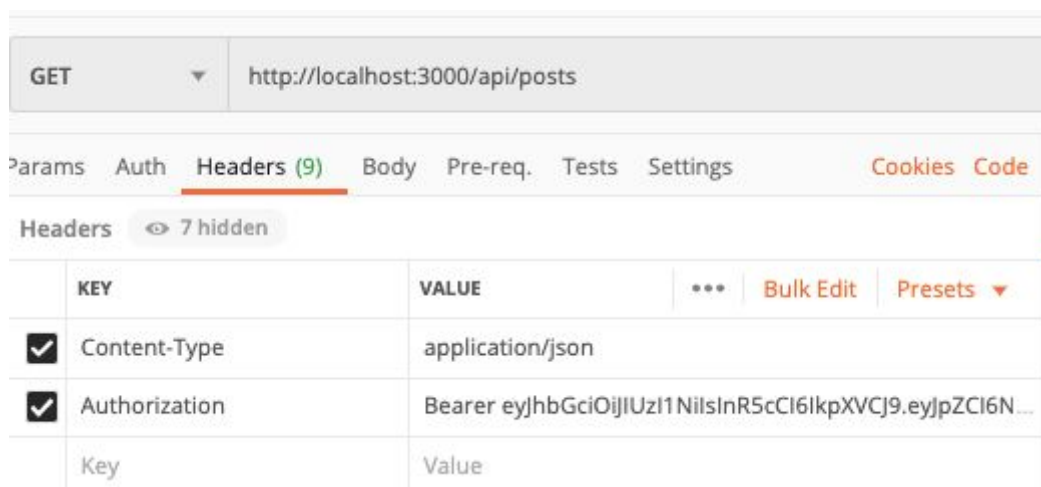


Imagen 9. Envío de JWT en header.

Fuente: Desafío Latam

Al presionar send o enviar, se obtiene el listado de posts del usuario.

```
{
  "data": [
    {
      "userId": 7,
      "id": 61,
      "title": "voluptatem doloribus consectetur est ut ducimus",
      "body": "ab nemo optio odio\ndelectus tenetur corporis
similique nobis repellendus rerum omnis facilis\nvero blanditiis debitis
in nesciunt doloribus dicta dolores\nmagnam minus velit"
    },
    {
      "userId": 7,
      "id": 62,
```



```
    "title": "beatae enim quia vel",
    "body": "enim aspernatur illo distinctio quae
praesentium\nbeatae alias amet delectus qui voluptate distinctio\nodit
sint accusantium autem omnis\nquo molestiae omnis ea eveniet optio"
  },
  {
    "userId": 7,
    "id": 63,
    "title": "voluptas blanditiis repellendus animi ducimus
error sapiente et suscipit",
    "body": "enim adipisci aspernatur nemo\nnumquam omnis facere
dolorem dolor ex quis temporibus incidunt\nab delectus culpa quo
reprehenderit blanditiis asperiores\naccusantium ut quam in voluptatibus
voluptas ipsam dicta"
  },
  {
    "userId": 7,
    "id": 64,
    "title": "et fugit quas eum in in aperiam quod",
    "body": "id velit blanditiis\nneum ea voluptatem\nmolestiae
sint occaecati est eos perspiciatis\nincidunt a error provident eaque
aut aut qui"
  },
  {
    "userId": 7,
    "id": 65,
    "title": "consequatur id enim sunt et et",
    "body": "voluptatibus ex esse\nsint explicabo est aliquid
cumque adipisci fuga repellat labore\nmolestiae corrupti ex saepe at
asperiores et perferendis\nnatus id esse incidunt pariat"
  },
  {
    "userId": 7,
    "id": 66,
    "title": "repudiandae ea animi iusto",
    "body": "officia veritatis tenetur vero qui itaque\nsint non
ratione\nsed et ut asperiores iusto eos molestiae nostrum\nveritatis
quibusdam et nemo iusto saepe"
  },
  {
    "userId": 7,
    "id": 67,
    "title": "aliquid eos sed fuga est maxime repellendus",
    "body": "reprehenderit id nostrum\nvoluptas doloremque
```

```
pariatur sint et accusantium quia quod aspernatur\net fugiat amet\nnon  
sapiente et consequatur necessitatibus molestiae"  
  },  
  {  
    "userId": 7,  
    "id": 68,  
    "title": "odio quis facere architecto reiciendis optio",  
    "body": "magnam molestiae perferendis quisquam\nqui cum  
reiciendis\nquaerat animi amet hic inventore\nnea quia deleniti quidem  
saepe porro velit"  
  },  
  {  
    "userId": 7,  
    "id": 69,  
    "title": "fugiat quod pariatur odit minima",  
    "body": "officiis error culpa consequatur modi asperiores  
et\ndolorum assumenda voluptas et vel qui aut vel rerum\nvoluptatum  
quisquam perspiciatis quia rerum consequatur totam quas\nsequi commodi  
repudiandae asperiores et saepe a"  
  },  
  {  
    "userId": 7,  
    "id": 70,  
    "title": "voluptatem laborum magni",  
    "body": "sunt repellendus quae\nest asperiores aut deleniti  
esse accusamus repellendus quia aut\nquia dolorem unde\nneum tempora esse  
dolore"  
  }  
]  
}
```

Ya aprendimos los conceptos básicos de JWT, a continuación realizaremos un ejercicio en conjunto en donde construiremos una interfaz web, que nos permita obtener el JWT a través de un formulario y utilizarlo para obtener los "posts" del usuario, para luego desplegarlos en una tabla utilizando JavaScript.

## Ejercicio guiado - Aplicando JWT

En el siguiente ejercicio vamos a realizar una interfaz web con HTML, JavaScript y jQuery que nos permite iniciar sesión (obtener el JWT) y acceder a los posts de un usuario.

Si aún no tienes instalado el proyecto es momento de hacerlo, puedes revisar las instrucciones en la sección **Preparación de ambiente de trabajo** y si ya lo tienes instalado, dirige a la raíz del proyecto en el terminal y ejecuta lo siguiente:

```
npm run watch
```

Si todo salió bien debes ver en el terminal: **Listening server in http://localhost:3000** y para validar, abre tu navegador favorito y escribe en la url <http://localhost:3000/jwt-practico>, deberías ver lo siguiente:

**Mini taller Jwt**

Email address

Password

**Submit**

Imagen 10. Formulario web.  
Fuente: Desafío Latam

Ahora volvamos a nuestro editor e identifiquemos la carpeta **public/jwt-practico/** la cual será nuestro entorno de trabajo, en este directorio encontraremos la siguiente estructura:

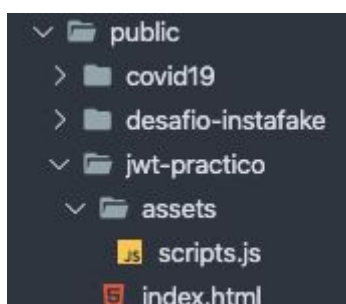


Imagen 11. Carpetas de trabajo.  
Fuente: Desafío Latam

Tenemos todo claro, ahora sólo nos toca comenzar a codificar.

### Obtener el JWT a través de un formulario

Si ingresamos a nuestro archivo `index.html` veremos una estructura básica de html en donde encontraremos un formulario tipo, el cual utilizaremos para obtener el JWT, si dividimos esta tarea en tareas más pequeñas el desarrollo será más fácil y tendremos la sensación de que vamos avanzando, entonces tómame unos minutos y piensa cuáles serían los pasos o estas pequeñas tareas para obtener el JWT, veamos si identificamos los mismos pasos:

- Darles ids o nombres para identificar los elementos del formulario que necesitaremos manipular.
- Capturar el evento submit del formulario.
- En el evento submit obtener los valores de email y password ingresados.
- Crear un llamado al endpoint de login **`http://localhost:3000/api/login`**

Ahora que tenemos los pasos vamos al código y comencemos a codificar, partamos añadiendo identificadores a los elementos que necesitamos utilizar del formulario, el formulario original es el siguiente:

```
<form>
  <div class="form-group">
    <label for="exampleInputEmail1">Email address</label>
    <input type="email" class="form-control" id="exampleInputEmail1"
aria-describedby="emailHelp">
  </div>
  <div class="form-group">
    <label for="exampleInputPassword1">Password</label>
    <input type="password" class="form-control"
id="exampleInputPassword1">
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Añadiremos un identificador al form, cambiaremos los que tienen los input de email y password, siempre es bueno ser descriptivo con los nombres.

```
<form id="js-form">
  <div class="form-group">
    <label for="js-input-email">Email address</label>
    <input type="email" class="form-control" id="js-input-email"
aria-describedby="emailHelp">
  </div>
  <div class="form-group">
    <label for="js-input-password">Password</label>
    <input type="password" class="form-control"
id="js-input-password">
  </div>
  <button type="submit" class="btn btn-primary">Submit</button>
</form>
```

Ya identificamos nuestros elementos, ahora capturaremos el evento submit del formulario, para eso nos dirigiremos al archivo **scripts.js**, accederemos al evento **submit** utilizando jQuery.

```
$('#js-form').submit(() => {

})
```

Para probar que este funcionando añadiremos un `console.log` al hacer click.

```
$('#js-form').submit(() => {
  console.log("click en el formulario")
})
```

Vamos a nuestro navegador y si probamos te darás cuenta que no funcionó, el navegador se refresca con cada click, ¿Alguna idea de qué está sucediendo y cómo solucionarlo?. Como recordarás los elementos tienen acciones por defecto como lo es el caso del submit de un formulario, por lo que debemos evitarla utilizando **event.preventDefault()** y nuestro código quedaría:

```
$('#js-form').submit((event) => {
  event.preventDefault()
  console.log("click en el formulario")
})
```

Ahora si probamos en la consola del navegador deberías ver:

```
click en el formulario
```

Ya capturamos el evento del formulario, ahora accedemos al email y password, los inputs utilizando JavaScript y hacemos la misma prueba con el **console.log** pero ahora con los inputs.

```
$('#js-form').submit((event) => {  
  event.preventDefault()  
  const email = document.getElementById('js-input-email').value  
  const password = document.getElementById('js-input-password').value  
  
  console.log(email)  
  console.log(password)  
})
```

Si hacemos una prueba deberíamos ver los datos ingresados:

```
jose@latam.cl  
secret
```

Nos falta el último paso que es hacer el llamado a nuestra API, esto lo haremos utilizando el **método fetch** de JavaScript, para hacer nuestro código más legible esta característica la añadiremos en una función llamada **postData**, como el **método fetch** retorna una promesa y utilizaremos **async/await**.

```
const postData = async (email, password) => {  
  try {  
    const response = await fetch('http://localhost:3000/api/login',  
  {  
    method: 'POST',  
    body: JSON.stringify({email:email,password:password})  
  })  
    const {token} = await response.json()  
    return token  
  } catch (err) {  
    console.error(`Error: ${err}`)  
  }  
}
```

Y llamamos estos método desde el submit de la siguiente manera:

```
$('#js-form').submit(async (event) => {  
  event.preventDefault()  
  const email = document.getElementById('js-input-email').value  
  const password = document.getElementById('js-input-password').value  
  const JWT = await postData(email,password)  
  console.log(JWT)  
})
```

Si te fijas incluimos async/await también para esperar que se resuelva la promesa del método **postData**, veámoslo en acción probando con el siguiente usuario.

---

## Mini taller Jwt

Email address

Password

Imagen 12. Formulario web con datos.  
Fuente: Desafío Latam

Si hacemos click en el botón y vemos la consola deberíamos ver un token:

```
{token:  
  "eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJpZCI6Nywib...zU4fQ.q1W-TDhZfp73  
  ABXMVltFGoxD04P3zevIJGsPu-w-hD0"}
```

Muy bien, ya realizamos la primera tarea, ahora que tenemos el token debemos utilizarlo para traer el listado de posts.

El crear la función postData para hacer el llamado a la API es una alternativa de organizar tu código, ya que perfectamente podríamos haber escrito el método fetch directamente en el evento submit y hubiera funcionado de la misma manera, pero hicimos una función porque así quedan piezas de código más pequeñas y más fáciles de entender.

### Usar JWT para obtener datos de la API

Ya obtuvimos el JWT, ahora necesitamos usarlo en nuestro siguiente llamado para traer los datos y desplegarlos en una tabla, realizaremos el mismo ejercicio de identificar las pequeñas tareas que hay que realizar, así que tómate unos minutos, revisa el html e identifica que hay que hacer, comparemos tus pasos con los que se indican a continuación:

- Crear identificadores para manipular el html de la tabla.
- Crear una función que revisa el JWT y haga el llamado a la API de posts.
- Completar la tabla con los datos que vienen de la API.
- Ocultar el formulario y mostrar la tabla.

Si uno de tus puntos era persistir el token, está excelente y lo veremos a profundidad en el siguiente punto. Vamos a nuestra primera tarea de añadir identificadores a nuestros elementos que manipularemos.

```
<div id="js-table-wrapper" class="col-md-12">
  <table id="js-table-posts" class="table">
    <thead>
      <tr>
        <th scope="col">#</th>
        <th scope="col">First</th>
        <th scope="col">Last</th>
        <th scope="col">Handle</th>
      </tr>
    </thead>
    <tbody>
    </tbody>
  </table>
</div>
```

No te preocupes de momento por la cabecera de la tabla lo cambiaremos en su momento, ahora continuemos con el siguiente paso de realizar una función que llamaremos getPosts la cual llamará a nuestra API de posts.

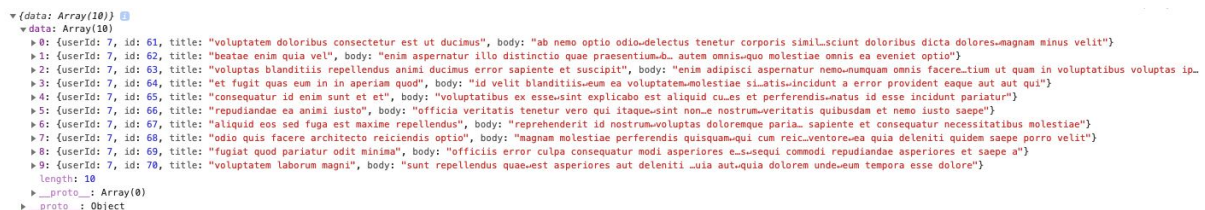


```
const getPosts = async (jwt) => {
  try {
    const response = await fetch('http://localhost:3000/api/posts',
    {
      method: 'GET',
      headers: {
        Authorization: `Bearer ${jwt}`
      }
    })
    const {data} = await response.json()
    return data
  } catch (err) {
    console.error(`Error: ${err}`)
  }
}
```

Y llamamos este método después de recibir el token del login.

```
$('#js-form').submit(async (event) => {
  event.preventDefault()
  const email = document.getElementById('js-input-email').value
  const password = document.getElementById('js-input-password').value
  const JWT = await postData(email,password)
  const posts = await getPosts(JWT)
  console.log(posts)
})
```

Si vamos a la web y completamos el formulario, deberíamos ver en la consola lo siguiente:



```
▼ (data: Array(10))
  data: Array(10)
    0: {userId: 7, id: 61, title: "voluptatem doloribus consectetur est ut ducimus", body: "ab nemo optio odio-delectus tenetur corporis simil-sciunt doloribus dicta dolores-magnam minus velit"}
    1: {userId: 7, id: 62, title: "beatae enim quia vel", body: "enim aspernatur illo distinctio quae praesentium-b. autem omnis-quo molestiae omnis ea eveniet optio"}
    2: {userId: 7, id: 63, title: "voluptas blanditiis repellendus animi ducimus error sapiente et suscipit", body: "enim adipisci aspernatur nemo-nunquam omnis facere-tum ut quam in voluptatibus voluptas ip-"}
    3: {userId: 7, id: 64, title: "et fugit quas eum in in aperiam quod", body: "id velit blanditiis-eum ea voluptatem-molestiae si-at-is-incidunt a error provident eaque aut aut qui"}
    4: {userId: 7, id: 65, title: "consequatur id enim sunt et et", body: "voluptatibus ex esse-sint explicabo est aliquid cu-es et perferendis-matus id esse incidunt pariatur"}
    5: {userId: 7, id: 66, title: "repudiandae ea animi iusto", body: "officia veritatis tenetur vero qui itaque-sint non-e nostrum-veritatis quibusdam et nemo iusto saepe"}
    6: {userId: 7, id: 67, title: "aliquid eos sed fuga est maxime repellendus", body: "reprehenderit id nostrum-voluptas doloremque paria- sapiente et consequatur necessitatibus molestiae"}
    7: {userId: 7, id: 68, title: "odio quis facere architecto reiciendis optis", body: "magnam molestiae perferendis quisquam-qui cum reit-ventore-ea quia deleniti quidem saepe porro velit"}
    8: {userId: 7, id: 69, title: "fugiat quod pariatur odit minima", body: "officiis error culpa consequatur modi asperiores e-s-sequi commodi repudiandae asperiores et saepe a"}
    9: {userId: 7, id: 70, title: "voluptatem laborum magni", body: "sunt repellendus quae-est asperiores aut deleniti -uia aut-quia dolorem unde-eum tempora esse dolore"}
    length: 10
    __proto__: Array(0)
  __proto__: Object
```

Imagen 13. Listado de posts del usuario.

Fuente: Desafío Latam

Ya tenemos los datos, es momento de añadirlos con jQuery a la tabla de html, para hacer esto, vamos a eliminar la clase helper de bootstrap **d-none**, la cual aplica el estilo de css **display:none** a nuestro elemento y al borrarla podremos ver el elemento, así será más fácil

avanzar con el desarrollo, para completar la tabla con los datos crearemos una función que se llame **fillTable**.

```
const fillTable = (data,table) => {  
  let rows = "";  
  $.each(data, (i, row) => {  
    rows += `|  
      <td> ${row.title} </td>  
      <td> ${row.body} </td>  
    </tr>`  
  })  
  $('#${table} tbody').append(rows);  
}

```

Como apreciamos, nuestra función recibe la data y el id de la tabla donde queremos añadir los datos, ahora llamemos nuestra función desde el submit luego de haber recibido los posts.

```
$('#js-form').submit(async (event) => {  
  event.preventDefault()  
  const email = document.getElementById('js-input-email').value  
  const password = document.getElementById('js-input-password').value  
  const JWT = await postData(email,password)  
  const posts = await getPosts(JWT)  
  fillTable(posts,'js-table-posts')  
})
```

Veamos el resultado en acción, si vamos al navegador y completamos con el usuario y contraseña:

### Mini taller Jwt

Email address

Password

**Submit**

#	First	Last	Handle
---	-------	------	--------

Imagen 14. Vista web del proyecto.  
Fuente: Desafío Latam

Presionemos el botón y veamos el resultado:

### Mini taller Jwt

Email address

Password

**Submit**

#	First	Last	Handle
voluptatem doloribus consectetur est ut ducimus	ab nemo optio odio delectus tenetur corporis similique nobis repellendus rerum omnis facilis vero blanditiis debitis in nesciunt doloribus dicta dolores magnam minus velit		
beatae enim quia vel	enim aspernatur illo distinctio quae praesentium beatae alias amet delectus qui voluptate distinctio odit sint accusantium autem omnis quo molestiae omnis ea eveniet optio		
voluptas blanditiis repellendus animi ducimus error sapiente et suscipit	enim adipisci aspernatur nemo numquam omnis facere dolorem dolor ex quis temporibus incidunt ab delectus culpa quo reprehenderit blanditiis asperiores accusantium ut quam in voluptatibus voluptas ipsam dicta		
et fugit quas eum in in aperiam quod	id velit blanditiis eum ea voluptatem molestiae sint occaecati est eos perspiciatis incidunt a error provident eaque aut aut qui		
consequatur id enim sunt et et	voluptatibus ex esse sint explicabo est aliquid cumque adipisci fuga repellat labore molestiae corrupti ex saepe at asperiores et perferendis natus id esse incidunt pariatur		

Imagen 15. Tabla web con datos.  
Fuente: Desafío Latam

Conseguimos mostrar los datos en la tabla, es momento de modificar la cabecera.

Nos falta el último paso, debemos ocultar el formulario y mostrar la tabla, ¿Cuál crees que es la validación que debemos hacer? Deberíamos validar mostrar la tabla y ocultar el formulario sólo si el JWT existe. Para esto lo primero que realizaremos será añadir un identificador al div que encierra el formulario.

```
<div id="js-form-wrapper" class="col-md-6 d-block">
  <form id="js-form">
    ...
  </form>
</div>
```

Ahora eliminaremos la clase **d-block** y la reemplazamos por **style:"display:block"**, añadiremos el estilo **style="display:none"** al elemento con id **js-table-wrapper**.

```
<div id="js-form-wrapper" class="col-md-6" style="display: block;">
  ...
</div>
<div id="js-table-wrapper" class="col-md-12" style="display: none">
</div>
```

Con estos cambios realizados crearemos una función que se llame **toggleFormAndTable**, oculte este div y muestre el div con id **js-table-wrapper**.

```
const toggleFormAndTable = (form,table) => {
  $('#${form}`).toggle()
  $('#${table}`).toggle()
}
```

Y la llamaremos desde el submit

```
$('#js-form').submit(async (event) => {
  event.preventDefault()
  const email = document.getElementById('js-input-email').value
  const password = document.getElementById('js-input-password').value
  const JWT = await postData(email,password)
  const posts = await getPosts(JWT)
  fillTable(posts, 'js-table-posts')
  toggleFormAndTable('js-form-wrapper', 'js-table-wrapper')
})
```

Ahora, veámoslo en acción completando el formulario:

**Mini taller Jwt**

Email address

Password

Imagen 16. Prueba funcional del formulario.  
Fuente: Desafío Latam

Y al presionar el botón

Mini taller Jwt	
Titulo	Cuerpor
voluptatem doloribus consectetur est ut ducimus	ab nemo optio odio delectus tenetur corporis similique nobis repellendus rerum omnis facilis vero blanditiis debitis in nesciunt doloribus dicta dolores magnam minus velit
beatae enim quia vel	enim aspernatur illo distinctio quae praesentium beatae alias amet delectus qui voluptate distinctio odit sint accusantium autem omnis quo molestiae omnis ea eveniet optio
voluptas blanditiis repellendus animi ducimus error sapiente et suscipit	enim adipisci aspernatur nemo numquam omnis facere dolorem dolor ex quis temporibus incidunt ab delectus culpa quo reprehenderit blanditiis asperiores accusantium ut quam in voluptatibus voluptas ipsam dicta
et fugit quas eum in in aperiam quod	id velit blanditiis eum ea voluptatem molestiae sint occaecati est eos perspiciatis incidunt a error provident eaque aut aut qui
consequatur id enim sunt et et	voluptatibus ex esse sint explicabo est aliquid cumque adipisci fuga repellat labore molestiae corrupti ex saepe at asperiores et perferendis natus id esse incidunt pariatur
repudiandae ea animi iusto	officia veritatis tenetur vero qui itaque sint non ratione sed et ut asperiores iusto eos molestiae nostrum veritatis quibusdam et nemo iusto saepe
aliquid eos sed fuga est maxime repellendus	reprehenderit id nostrum voluptas doloremque pariatur sint et accusantium quia quod aspernatur et fugiat amet non sapiente et consequatur necessitatibus molestiae
odio quis facere architecto reiciendis optio	magnam molestiae perferendis quisquam qui cum reiciendis quaerat animi amet hic inventore ea quia deleniti quidem saepe porro velit
fugiat quod pariatur odit minima	officiis error culpa consequatur modi asperiores et dolorum assumenda voluptas et vel qui aut vel rerum voluptatum quisquam perspiciatis quia rerum consequatur totam quas sequi commodi

Imagen 17. Resultado prueba funcional.  
Fuente: Desafío Latam

Pero qué pasa ahora si refrescamos la pantalla, volvemos ver nuestro formulario, pues nos falta hacer persistencia del JWT y es lo que haremos a continuación.

### Persistir JWT

Como se comentó en la lectura, hay varias alternativas para persistir el token, utilizaremos `localStorage` ya que para el alcance de nuestro proyecto cumple perfectamente el objetivo, entonces si identificamos los siguientes pasos:

- Guardar el JWT, luego de hacer el login en el `localStorage`.
- Al momento de cargar nuestra página revisar si existe un JWT, de existir debemos mostrar la tabla y ocultar el formulario.
- Controlar la vigencia del token, es decir, mostrar el formulario si el token expiró.

Usar `localStorage` es muy sencillo ya que para añadir un valor debemos usar la siguiente instrucción

```
localStorage.setItem('llave-para-identificar', 'valor-que-guardamos')
```

Y para acceder al valor guardado en el `localStorage` se utiliza la siguiente instrucción:

```
localStorage.getItem('llave-para-identificar')
```

Existen más métodos que puedes revisar [aquí](#).

Volvamos a nuestro ejemplo y utilicemos el método **`localStorage.setItem`** para almacenar el JWT al momento de hacer login en la función `postData`.

```
const postData = async (email, password) => {
  try {
    const response = await fetch('http://localhost:3000/api/login',
    {
      method: 'POST',
      body: JSON.stringify({email:email,password:password})
    })
    const {token} = await response.json()
    localStorage.setItem('jwt-token',token)
    return token
  } catch (err) {
    console.error(`Error: ${err}`)
  }
}
```

Validemos que esté funcionando, volviendo hacer login en nuestro proyecto, como te habrás dado cuenta nuestra aplicación sigue funcionando, pero ¿Cómo sabemos si el localStorage funcionó? Bueno, esto lo podemos revisar con el inspector de elementos.

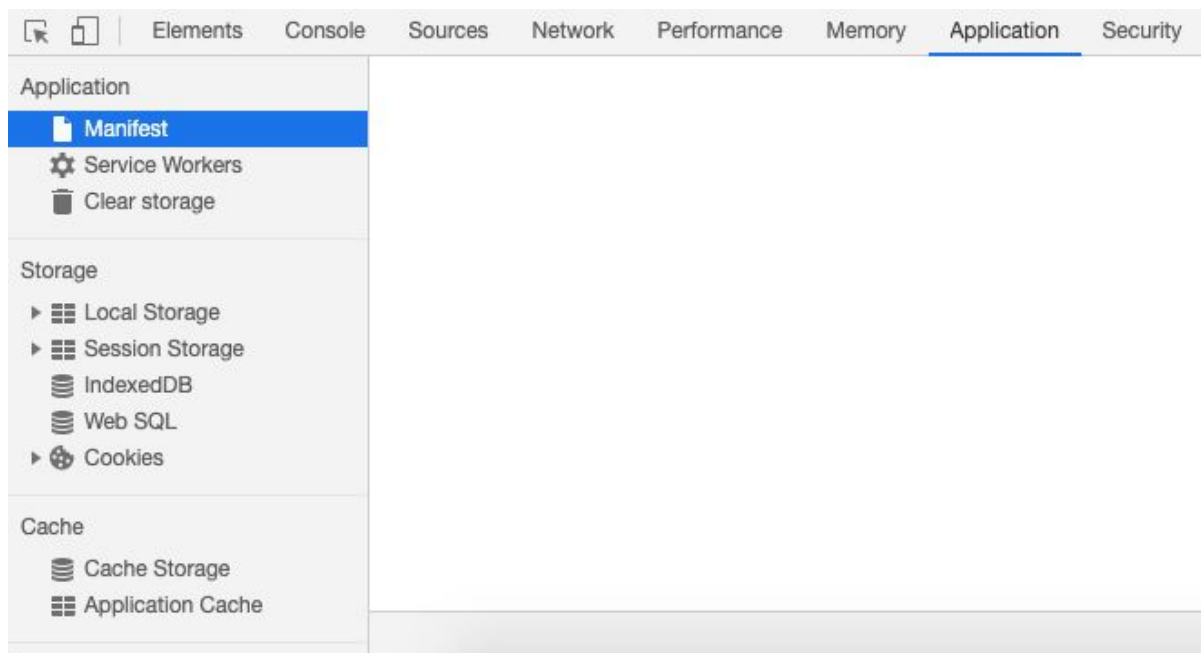


Imagen 18. Sección de aplicación de inspector de elementos.  
Fuente: Desafío Latam

Si vemos a la derecha dice **Local Storage** y si hacemos click saldrá nuestro **localhost:3000**, si volvemos hacer click podremos ver nuestra llave **jwt-token** con su valor.

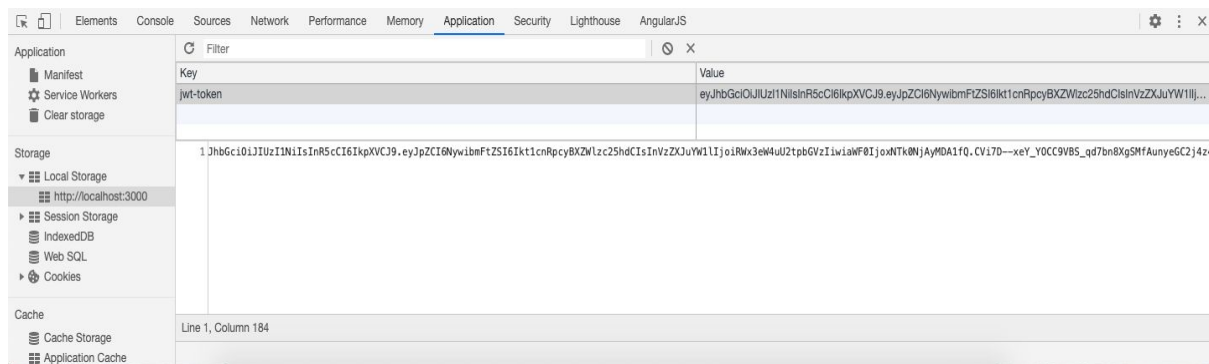


Imagen 19. JWT almacenado en el navegador.  
Fuente: Desafío Latam

Ahora, al refrescar el navegador te darás cuenta que nuestro **jwt-token** se mantiene, pero nuestro formulario volvió a aparecer, esto sucede porque no hemos implementado que cuando se cargue el sitio valide si existe un JWT.

Para validar si hay un JWT, crearemos una función que se ejecute cuando se cargue nuestra página y valide si existe un JWT, de ser así, debería automáticamente llamar a nuestra API de posts.

```
const init = async () => {  
  const token = localStorage.getItem('jwt-token')  
  if(token) {  
    const posts = await getPosts(token)  
    fillTable(posts, 'js-table-posts')  
    toggleFormAndTable('js-form-wrapper', 'js-table-wrapper')  
  }  
}  
init()
```

Revisemos nuestro navegador refrescando y deberíamos ver lo siguiente:

### Mini taller Jwt

Titulo	Cuerpo
voluptatem doloribus consectetur est ut ducimus	ab nemo optio odio delectus tenetur corporis similique nobis repellendus rerum omnis facilis vero blanditiis debitis in nesciunt doloribus dicta dolores magnam minus velit
beatae enim quia vel	enim aspernatur illo distinctio quae praesentium beatae alias amet delectus qui voluptate distinctio odit sint accusantium autem omnis quo molestiae omnis ea eveniet optio
voluptas blanditiis repellendus animi ducimus error sapiente et suscipit	enim adipisci aspernatur nemo numquam omnis facere dolorem dolor ex quis temporibus incidunt ab delectus culpa quo reprehenderit blanditiis asperiores accusantium ut quam in voluptatibus voluptas ipsam dicta
et fugit quas eum in in aperiam quod	id velit blanditiis eum ea voluptatem molestiae sint occaecati est eos perspiciatis incidunt a error provident eaque aut aut qui
consequatur id enim sunt et et	voluptatibus ex esse sint explicabo est aliquid cumque adipisci fuga repellat labore molestiae corrupti ex saepe at asperiores et perferendis natus id esse incidunt pariatur
repudiandae ea animi iusto	officia veritatis tenetur vero qui itaque sint non ratione sed et ut asperiores iusto eos molestiae nostrum veritatis quibusdam et nemo iusto saepe
aliquid eos sed fuga est maxime repellendus	reprehenderit id nostrum voluptas doloremque pariatur sint et accusantium quia quod aspernatur et fugiat amet non sapiente et consequatur necessitatibus molestiae
odio quis facere architecto reiciendis optio	magnam molestiae perferendis quisquam qui cum reiciendis quaerat animi amet hic inventore ea quia deleniti quidem saepe porro velit

Imagen 20. Tabla con los posts de la API.

Fuente: Desafío Latam

Nuestro requerimiento está casi terminado, sólo falta validar si nuestro token es válido, para emular este caso abriremos nuestro inspector de elemento y nos dirigiremos al localStorage como lo vimos anteriormente, ahora si hacemos doble click en el valor del token te darás cuenta que es posible editar este valor, así que agrégale algún carácter para que deje de ser válido.



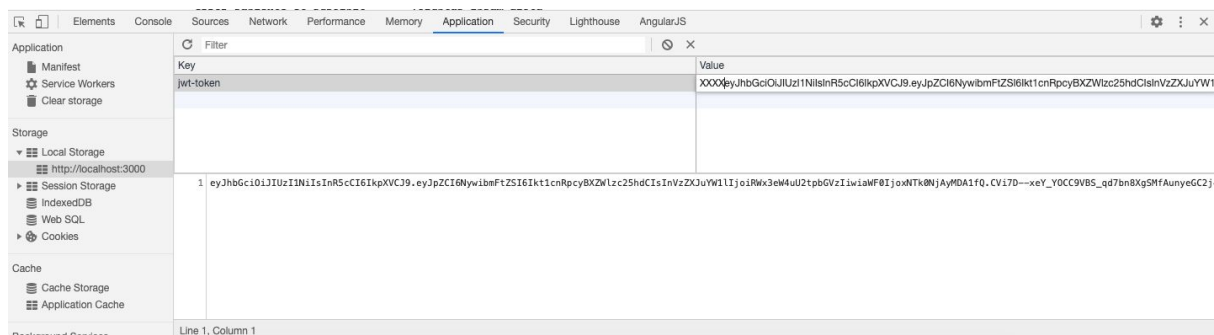


Imagen 21. JWT modificado en el localStorage del navegador.

Fuente: Desafío Latam

Cerramos el inspector y refrescamos, si nuestro cambio fue correcto deberíamos ver nuestra tabla vacía y un error en la consola.

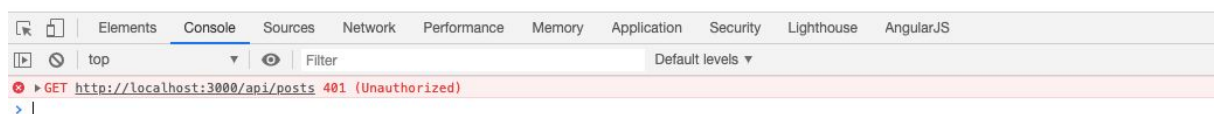
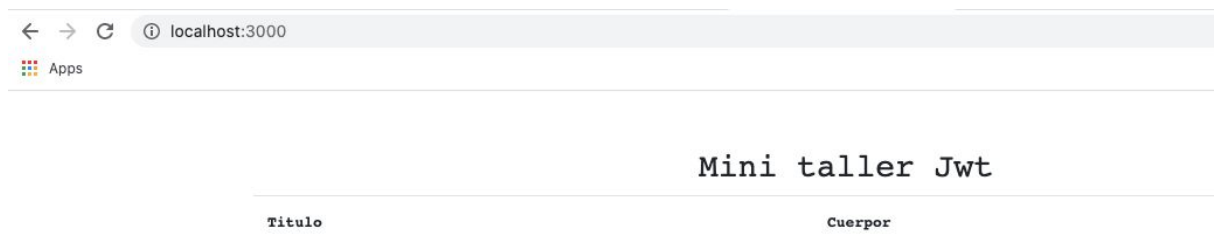


Imagen 22. Error al traer datos de la API.

Fuente: Desafío Latam

Para controlar este comportamiento vamos a refactorizar el código que hemos construido, es decir, vamos a mover algunas funciones de lugar, lo primero que haremos será sacar las funciones **fillTable** y **toggleFormAndTable** de dentro del evento submit, también de nuestra función init y nos quedaría así:

Submit

```
$('#js-form').submit(async (event) => {  
  event.preventDefault()  
  const email = document.getElementById('js-input-email').value  
  const password = document.getElementById('js-input-password').value  
  const JWT = await postData(email,password)  
  getPosts(JWT)  
})
```

init

```
const init = async () => {  
  const token = localStorage.getItem('jwt-token')  
  if(token) {  
    getPosts(token)  
  }  
}
```

Estas funciones de **fillTable** y **toggleFormAndTable** las llevaremos dentro de la función **getPosts** y las llamaremos siempre y cuando la respuesta de la API sea algo válido.

```
const getPosts = async (jwt) => {  
  try {  
    const response = await fetch('http://localhost:3000/api/posts',  
  {  
    method: 'GET',  
    headers: {  
      Authorization: `Bearer ${jwt}`  
    }  
  })  
    const {data} = await response.json()  
    if (data) {  
      fillTable(data, 'js-table-posts')  
      toggleFormAndTable('js-form-wrapper', 'js-table-wrapper')  
    }  
  } catch (err) {  
    console.error(`Error: ${err}`)  
  }  
}
```

Con este cambio estamos controlando el caso de mostrar la tabla sólo si la respuesta de la API es una respuesta válida (con un JWT válido), ahora sólo deberíamos limpiar nuestro localStorage dentro del catch que es donde cae nuestro código cuando el JWT es inválido.

```
const getPosts = async (jwt) => {
  try {
    const response = await fetch('http://localhost:3000/api/posts',
    {
      method: 'GET',
      headers: {
        Authorization: `Bearer ${jwt}`
      }
    })
    const {data} = await response.json()
    if (data) {
      fillTable(data, 'js-table-posts')
      toggleFormAndTable('js-form-wrapper', 'js-table-wrapper')
    }
  } catch (err) {
    localStorage.clear()
    console.error(`Error: ${err}`)
  }
}
```

## Ejercicio propuesto

Al momento de hacer login añade otra tabla bajo la tabla de posts que despliegue los álbumes utilizando el siguiente endpoint **<http://localhost:3000/api/albums>**.

## Solución del ejercicio propuesto

```
<!-- Html para mostrar la información en el DOM -->
<table id="js-table-albums" class="table">
  <thead>
    <tr>
      <th scope="col">id</th>
      <th scope="col">Título Album</th>
    </tr>
  </thead>
  <tbody>
  </tbody>
</table>
```

```
// Función que trae los albums
const getAlbums = async (jwt) => {
  try {
    const response = await fetch('http://localhost:3000/api/albums',
    {
      method: 'GET',
      headers: {
        Authorization: `Bearer ${jwt}`
      }
    })
    const {data} = await response.json()
    if (data) {
      let rows = "";
      $.each(data, (i, row) => {
        rows += `<tr>
          <td> ${row.id} </td>
          <td> ${row.title} </td>
        </td>`
      })
      $('#js-table-albums tbody').append(rows)
    }
  } catch (err) {
    console.error(`Error: ${err}`)
  }
}
```

```
// Desde la función login se debe llamar a este nuevo método
$('#js-form').submit(async (event) => {
  event.preventDefault()
  const email = document.getElementById('js-input-email').value
  const password = document.getElementById('js-input-password').value
  const JWT = await postData(email,password)
  getPosts(JWT)
  getAlbums(JWT)
})
```