HACK ACADEMY

Coding Bootcamp

Sprint 3

Temario

Temario

- Repaso de APIs.
- APIs REST.
- Autenticación en APIs.
- JWT.
- JWT en Node.js.

REPASO

APIs

REPASO

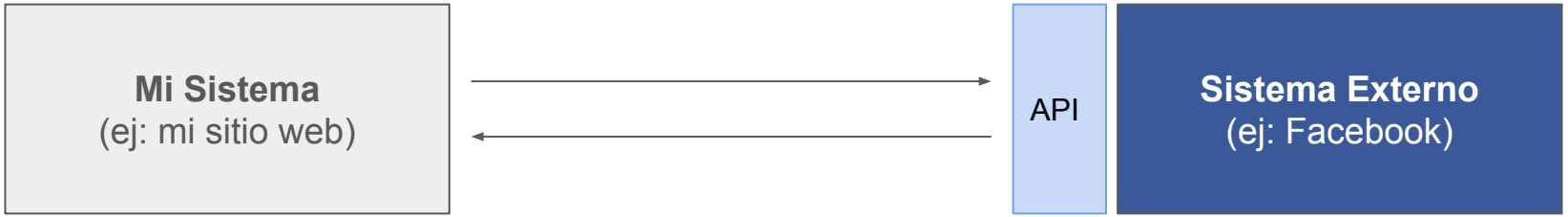
¿Qué es una API?

- API = Application Programming Interface.
- Es una interface que le permite a 2 sistemas independientes comunicarse entre sí.
    - Uno de los sistemas **“provee”** la API (es dueño de la misma).
    - El otro **“consume”** la API.
- Quien provee la API debe especificar qué servicios se proveen y cómo se usan. La buena práctica es que toda API está acompañada de una documentación.
- Notar que la definición anterior no habla de Internet y de hecho el concepto de API va más allá de sistemas conectados a una red.

REPASO

¿Qué es una API?

Diagrama de ejemplo:



En este ejemplo, el sistema externo provee una API y nuestro sistema la consume.

El sistema externo debe especificar qué servicios se proveen y cómo se usan. En este ejemplo: https://developers.facebook.com/docs/graph-api.

APIs REST

APIs REST (1/2)

Existe un tipo particular dentro de las Web APIs (también llamadas *web services*) que son las APIs REST (REpresentational State Transfer).

Estas APIs se caracterizan por:

- Interactuar con “recursos”, generalmente entidades del problema, identificados por una URL. Ej: `/api/users`.
- Utilizar verbos o métodos HTTP para interactuar con un recurso. Ej: GET, POST, PUT, PATCH y DELETE. → No hay que “inventar” URLs para cada acción que se quiera realizar sobre el recurso. Ej: `/api/users/borrar` vs. `/api/users/eliminar`.
- Ser *stateless*: no tener estado. Cada *request* tiene toda la información necesaria para que el servidor pueda procesar el llamado y no depende de *requests* anteriores.

APIs REST (2/2)

Gracias a lo anterior, las API REST brindan una interfaz uniforme (gran diferencia con las APIs SOAP).

En general, los datos intercambiados con una API REST están en formato JSON, pero no es un requisito obligatorio.

Autenticación en APIs

Autenticación en APIs

Así como necesitamos autenticar usuarios para entrar a distintas secciones de un sitio web, también suele ser necesario autenticar usuarios o sistemas que desean acceder a una API.

Hay varias formas de implementar autenticación en APIs. Algunos ejemplos son:

- HTTP Basic Authentication → Username/Password en los *headers* de cada *request*.
  Ej: Mailchimp y Twilio.
- API Key Authentication → Se usan API Key o Tokens (strings generalmente random y largos) en lugar de un Username/Password.
  Ej: Stripe y Sendgrid.
- OAuth Authentication → Se usa un `access_token` generado por un sistema tercero. Suele ser la mejor opción para autenticar cuentas de usuario (personales).
  Ej: Google, Facebook, Twitter.

👉 En cualquier caso, siempre usen HTTPS.

👉 Es probable que necesiten habilitar CORS (link) si es que quieren hacer llamadas AJAX a la API. Para ello pueden usar este paquete (middleware). Notar que esto en realidad aplica para cualquier API (con o sin autenticación).

JWT

JWT (1/4)

JSON Web Token es un estándar abierto (RFC7519) para crear tokens de acceso a una aplicación. Ver documentación. Ver video.

Como dice el nombre, el JWT contiene datos en formato JSON. En general están sin encriptar, pero siempre está firmados (*signed*).

JWT suele ser útil a la hora de implementar autenticación en una API y una comunicación *server-to-server*. Además, podríamos usar el mismo JWT para autenticarnos en dos sistemas dominios diferentes (algo imposible con sesiones y cookies).

En caso de que se quiera realizar una comunicación *browser-to-server,* el token se podría guardar en una cookie, pero no es estrictamente necesario. También se podría guardar en localStorage. En este caso, hay que tener cuidado.

JWT (2/4)

Un JWT se compone por:

- **Header**.

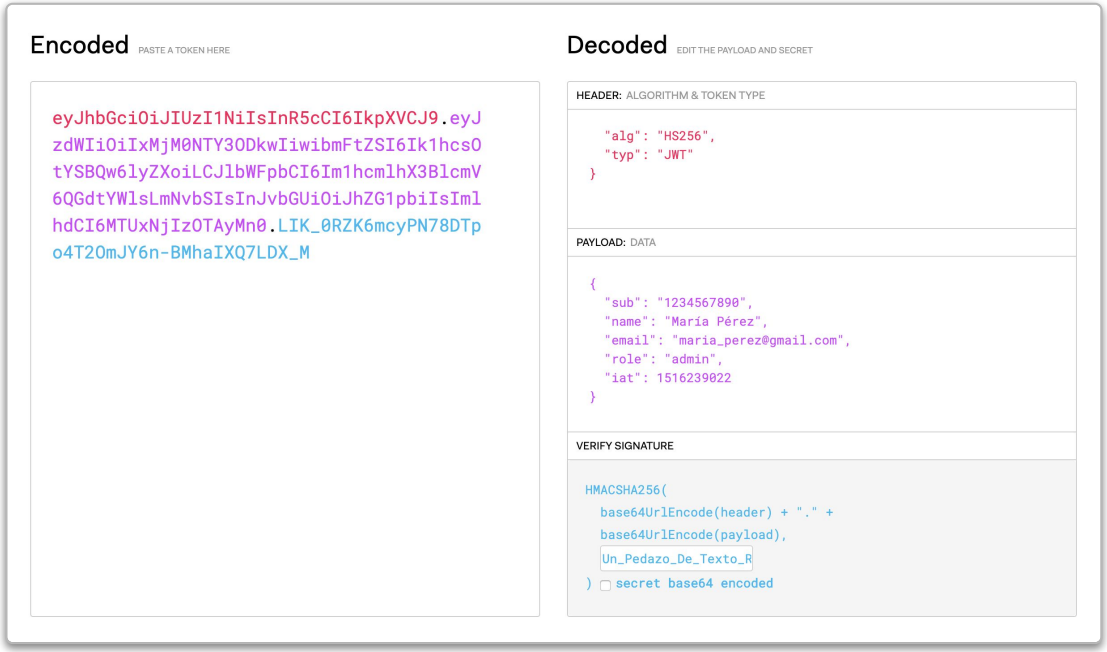
  Para especificar el tipo de token y algoritmo.

- **Payload**.

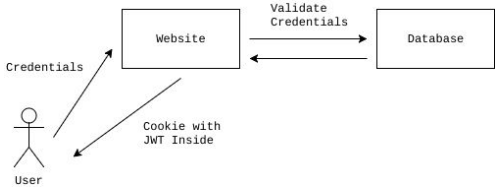
  Los datos que queremos guardar en el JWT. Existe cierto estándar para que el *payload* sea compacto.

- **Signature**.

  Para verificar que el JWT es válido (que no haya sido manipulado por nadie más).



JWT (3/4)



El proceso de autenticación es el siguiente:

1. El cliente ingresa su usuario y contraseña (credenciales).
2. El servidor valida (contra la BD) que las credenciales sean correctas. Si lo son, genera un JWT y se lo envía al cliente.
3. De ahora en más, los *requests* realizados por el cliente deberán incluir el JWT en los *headers*.
4. Cada vez que el servidor reciba un JWT, deberá verificar que sea válido, sin necesidad de volver a llamar a la BD. Si es válido, deja “acceder” al cliente al recurso solicitado.

JWT (4/4)

A partir de ahora, los *requests* realizados por el cliente deberán incluir un *header* de autenticación:

```
Authorization: Bearer <token>
```

Luego, en las rutas “privadas” se deberá validar que el token sea válido. Esto se puede realizar sin necesidad de interactuar con la BD.

👉 Notar que un problema que pueden tener los JWT es la dificultad de hacerlos expirar de forma “forzada”, por ejemplo, si a un usuario le roban su token. En estos casos sería recomendable llevar un registro en el servido de los tokens generados.

JWT en Node.js

JWT en Node.js (1/3)

La librería más utilizada para gestionar JWTs es `jsonwebtoken` (link).

```
npm i jsonwebtoken
```

Forma de uso:

```
const jwt = require("jsonwebtoken");

const token = jwt.sign({ sub: "user123" }, "UnStringMuyScreto");
```

Este token es el que se le envía al cliente.

JWT en Node.js (2/3)

Cuando el servidor recibe un *request* conteniendo un JWT, es necesario validarlo. Para ello se puede utilizar el siguiente código:

```
jwt.verify(token, "UnStringMuyScreto", function (err, decoded) {
  // Si hubo un error, `err` está definido.
  // Si está todo OK, `decoded` contiene el payload.
});
```

Esta validación habría que hacerla en todos los requests a endpoints privados.

JWT en Node.js (3/3)

Para agilizar el proceso de validación en Express, se recomienda utilizar el paquete `express-jwt` (link), que es un *middleware* de autenticación:

```
const checkJwt = require("express-jwt");

...

app.get("/ruta-privada", checkJwt({ secret: "UnStringMuyScreto" }), (req, res) => {

 // El payload queda disponible aquí adentro

 // vía el objeto `req.user`.

});
```

Si el JWT tiene un atributo (*claim*) llamado `exp`, se validará si el token no está vencido.