

# HowTo ASIX API REST

Curs 2019 - 2020

<b>API-REST</b>	<b>2</b>
Descripció	2
jsonplaceholder.com	4
Consultes API-REST	5
I) Què és una API Rest	8
Pràctica-1.1: mostrar els id de /post en consola	8
Pràctica-1.2: mostrar els id de /post	10
Pràctica-1.3: mostrar id+email dels comments del post 1	13
II) Sol·licituds de l'API	17
Pràctica 2.5.a: mostrar els llibres (títol, imatge)	20
Pràctica 2.5.b: mostrar els llibres (títol, imatge) amb css	21
Pràctica 2.5.c: mostrar també 'descripcion' i 'detalle'	22
III) Resposta de l'API	25
Spotify	29
IV) Relació entre sol·licitud i resposta	42
Pràctica 3.1.: mostrar el reuadre de selecció i els botons	42
Pràctica 3.2.: mostrar els artistes resultat del /search	45
Pràctica 3.3.: mostrar els Top10 songs de l'artista seleccionat	49
<b>Docker: node.js</b>	<b>53</b>
<b>Vulnerability: CVE</b>	<b>54</b>
<b>Google API</b>	<b>56</b>

---

# API-REST

---

## Descripció

Cibernàrium - Barcelona Activa

- ❑ [Curs online - Crea una API en REST per a una plataforma digital](#)
- ❑ [Masterclass: intel·ligència artificial i Machine Learning](#)

Recursos:

- ❑ GitHub [edtasixm11](#) [api-rest](#)

LMS [Learning Management System](#)

En cas que vulgueu repassar conceptes JavaScript abans de començar, us aconsellem fer algun dels cursos que trobareu a la web [www.codecademy.com](http://www.codecademy.com) o bé els exercicis de [www.w3schools.com](http://www.w3schools.com).

Aquests són alguns enllaços útils per ampliar informació i repassar conceptes:

API:

[https://ca.wikipedia.org/wiki/Interf%C3%ADcie\\_de\\_programaci%C3%B3\\_d%27aplicacions](https://ca.wikipedia.org/wiki/Interf%C3%ADcie_de_programaci%C3%B3_d%27aplicacions)

REST: <https://ca.wikipedia.org/wiki/REST>

JSON: <https://ca.wikipedia.org/wiki/JSON>

<http://www.json.org/>

<http://www.jsonlint.com/>

Servidor web: [https://ca.wikipedia.org/wiki/Servidor\\_web](https://ca.wikipedia.org/wiki/Servidor_web)

[https://www.w3schools.com/jquery/jquery\\_callback.asp](https://www.w3schools.com/jquery/jquery_callback.asp)

AJAX: <https://ca.wikipedia.org/wiki/XMLHttpRequest>

[https://www.w3schools.com/js/js\\_ajax\\_http.asp](https://www.w3schools.com/js/js_ajax_http.asp)

<https://medium.freecodecamp.org/here-is-the-most-popular-ways-to-make-an-http-request-in-javascript-954ce8c95aaa>

[https://www.w3schools.com/jquery/jquery\\_ref\\_ajax.asp](https://www.w3schools.com/jquery/jquery_ref_ajax.asp)

[https://www.w3schools.com/jquery/jquery\\_ajax\\_get\\_post.asp](https://www.w3schools.com/jquery/jquery_ajax_get_post.asp)

[https://www.w3schools.com/js/js\\_ajax\\_http\\_response.asp](https://www.w3schools.com/js/js_ajax_http_response.asp)

[https://www.w3schools.com/tags/ref\\_httpmessages.asp](https://www.w3schools.com/tags/ref_httpmessages.asp)

CSS: <https://css-tricks.com/snippets/css/a-guide-to-flexbox/>

JQUERY: [https://www.w3schools.com/jquery/jquery\\_ref\\_html.asp](https://www.w3schools.com/jquery/jquery_ref_html.asp)

[https://www.w3schools.com/jquery/jquery\\_callback.asp](https://www.w3schools.com/jquery/jquery_callback.asp)

REST wiki en

**Representational state transfer (REST)** is a [software architectural](#) style that defines a set of constraints to be used for creating [Web services](#). Web services that conform to the REST architectural style, called *RESTful* Web services, provide interoperability between computer systems on the [Internet](#). RESTful Web services allow the requesting systems to access and manipulate textual representations of [Web resources](#) by using a uniform and predefined set of [stateless](#) operations. Other kinds of Web services, such as [SOAP](#) Web services, expose their own arbitrary sets of operations.

# jsonplaceholder.com

jsonplaceholder:

❏ <https://jsonplaceholder.typicode.com>

Aquesta web és una Fake Online REST API for Testing and Prototyping. Implementa una API REST amb àlbums de música, artites, fotos, etc. Les dades són falses. Implementa els **endpoints** descrits a continuació per a poder fer proves de API REST:

“JSONPlaceholder is a free online REST API that you can use whenever you need some fake data.

It's great for tutorials, testing new libraries, sharing code examples, ... “

Endpoints:

## Resources

JSONPlaceholder comes with a set of 6 common resources:

<a href="#">/posts</a>	100 posts
<a href="#">/comments</a>	500 comments
<a href="#">/albums</a>	100 albums
<a href="#">/photos</a>	5000 photos
<a href="#">/todos</a>	200 todos
<a href="#">/users</a>	10 users

Routes

# Routes

All HTTP methods are supported.

GET	<a href="#">/posts</a>
GET	<a href="#">/posts/1</a>
GET	<a href="#">/posts/1/comments</a>
GET	<a href="#">/comments?postId=1</a>
GET	<a href="#">/posts?userId=1</a>
POST	<a href="#">/posts</a>
PUT	<a href="#">/posts/1</a>
PATCH	<a href="#">/posts/1</a>
DELETE	<a href="#">/posts/1</a>

**Note:** you can view detailed examples [here](#).

## Consultes API-REST

Ara que tenim una API-REST per consultar (la fake de [jsonplaceholder.typicode.com](https://jsonplaceholder.typicode.com)) podem fer consultes als endpoints i obtenir respostes amb dades json.

### Amb un navegador

Podem fer-ho des del navegador a qualsevol dels endpoints. Observeu que el navegador ens permet seleccionar la vista de la resposta que volem:

- [JSON](#) Les dades presentades en format pretty desplegable de JSON.
- [raw-data](#) Les dades en cru en format de text json.
- [Headers](#) les capçaleres del diàleg http que s'ha produït ntre client i servidor.

```
# llistat de tots els posts
https://jsonplaceholder.typicode.com/posts

# llistat de tots els usuaris
https://jsonplaceholder.typicode.com/users

# llistat del post amb id 1
https://jsonplaceholder.typicode.com/posts/1
```

```
# llistat del l'usuari amb id 1  
https://jsonplaceholder.typicode.com/users/1
```

### Des del terminal del shell

```
$ curl https://jsonplaceholder.typicode.com/posts/1  
{  
  "userId": 1,  
  "id": 1,  
  "title": "sunt aut facere repellat provident occaecati excepturi optio reprehenderit",  
  "body": "quia et suscipit\nsuscipit recusandae consequuntur expedita et  
cum\nreprehenderit molestiae ut ut quas totam\nnostrum rerum est autem sunt rem  
eveniet architecto"  
}
```

### Des de la consola del navegador

Podem escriure es instruccions amb javascript per accedir als recursos de la API-REST, com per exemple aquesta instrucció per obtenir el primer post:

```
fetch('https://jsonplaceholder.typicode.com/posts/1')  
  .then(response => response.json())  
  .then(json => console.log(json))
```

o aquesta altra per obtenir la llista de tots els posts:

```
fetch('https://jsonplaceholder.typicode.com/posts/1')  
  .then(response => response.json())  
  .then(json => console.log(json))
```

En la imatge següent observeu el resultat d'executar les dues consultes en la consola del navegador:

```

>> fetch('https://jsonplaceholder.typicode.com/posts/1')
  .then(response => response.json())
  .then(json => console.log(json))
< ▶ Promise { <state>: "pending" }
  ▼ {...}
    body: "quia et suscipit\nsuscipit recusandae consequuntur expedita e
    architecto"
    id: 1
    title: "sunt aut facere repellat provident occaecati excepturi optio
    userId: 1
    ▶ <prototype>: Object { ... }
>> fetch('https://jsonplaceholder.typicode.com/posts')
  .then(response => response.json())
  .then(json => console.log(json))
< ▶ Promise { <state>: "pending" }
  ▶ Array(100) [ {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, {...}, ... ]

```

### Exemple de creació d'un element amb POST

Amb la següent instrucció es crea (no realment, el servidor fa com si el crees però no ho fa de debò) un nou post, el 101:

```

fetch('https://jsonplaceholder.typicode.com/posts', {
  method: 'POST',
  body: JSON.stringify({
    title: 'foo',
    body: 'bar',
    userId: 1
  }),
  headers: {
    "Content-type": "application/json; charset=UTF-8"
  }
})
.then(response => response.json())
.then(json => console.log(json))

```

Un cop executat podem observar la resposta del servidor:

```

Promise { <state>: "pending" }
Object { title: "foo", body: "bar", userId: 1, id: 101 }

```

## I) Què és una API Rest

### XMLHttpRequest

Podem observar a la consola-web que si fem una petició XMLHttpRequest quina és la resposta que obtenim. Si fem la petició sense paràmetres evidentment no és una petició correcta i obtindrem una resposta d'error (status != 200). Però ens permet observar quins són els elements que formen part de una petició/resposta XMLHttpRequest.

```
>> new XMLHttpRequest()
XMLHttpRequest
mozAnon: false
mozSystem: false
onabort: null
onerror: null
onload: null
onloadend: null
onloadstart: null
onprogress: null
onreadystatechange: null
ontimeout: null
readyState: 0
response: ""
responseText: ""
responseType: ""
responseURL: ""
responseXML: null
status: 0
statusText: ""
timeout: 0
upload: XMLHttpRequestUpload { onloadstart: null, onprogress: null, onabort: null, ... }
withCredentials: false
<prototype>: XMLHttpRequestPrototype { open: open(), setRequestHeader:
setRequestHeader(), send: send(), ... }
```

### Pràctica-1.1: mostrar els id de /post en consola

Escriure la app que fa una consulta al endpoint [/post](#) de [jsonplaceholder.typicode.com](#) mostrant de cada un dels cent elements només el **id**.

Codi pràctica 1.1:

```
<!DOCTYPE html>
<html>
  <body>
```



```

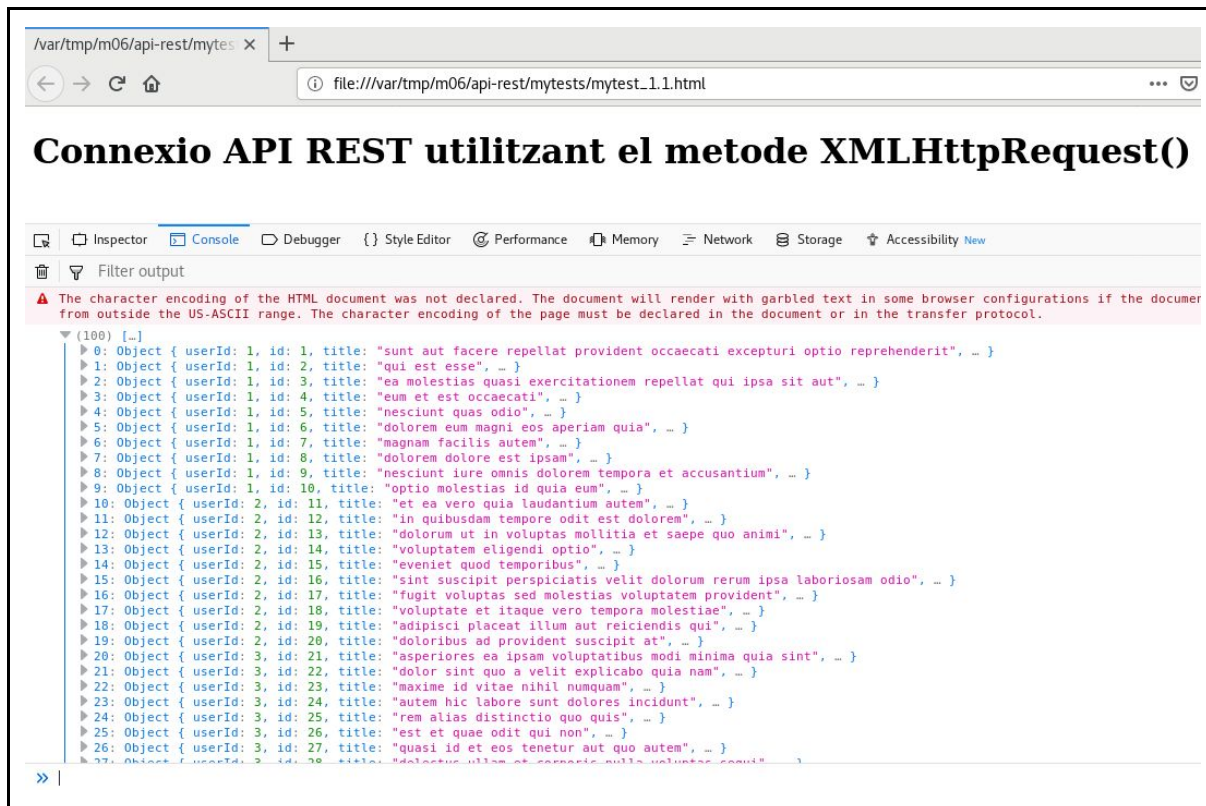
        <h1>Connexio API REST utilitzant el metode XMLHttpRequest(</h1>
        <br/>
        <div id="elements"></div>
    </body>
    <script>
    var xmlhttp = new XMLHttpRequest();
    var url = "https://jsonplaceholder.typicode.com/posts";
    xmlhttp.onreadystatechange = function() {
        if (this.readyState == 4 && this.status == 200) {
            var myArr = JSON.parse(this.responseText);
            console.log(myArr);
        }
    };
    xmlhttp.open("GET", url, true);
    xmlhttp.send();
    </script>
</html>

```

Analitzem el codi:

- Hi ha tota la part html on es descriu la pàgina a mostrar, de fet simplement hi ha un Heading1 amb el títol a mostrar, i un div que a hores d'ara no conté res i que en la propera pràctica hi posarem els elements de la resposta.
- el bloc <script> ... </script> és on hi ha el codi javascript que realitzarà la consulta i obtindrà la resposta del servidor API-REST.
- **XMLHttpRequest**: usem aquest mètode per fer una consulta a la API-REST. Observem que retorna un objecte anomenat **xmlhttp** que és el que utilitzem en la resta d'instruccions.
- **onreadystatechange**: és una funció de XMLHttpRequest que actuarà quan la pàgina s'hagi carregat del tot (readyState=4) sempre i quant la resposta sigui sense errors (status=200).
- Aquesta funció el que fa és convertir la resposta JSON que rep del servidor a un element de dades javascript amb la funció parse.
- **console.log**: tot seguit es desa, mostra, a la consola-web el JSON rebut.
- **open**: amb aquest mètode de XMLHttpRequest s'indica on s'ha de fer la connexió, quina és la url amb que ha de connectar, i quin mètode ha d'usar.
- **send**: aquest mètode de XMLHttpRequest envia la petició de request al servidor.

Podem observar el resultat de carregar la pàgina al navegador. A hores d'ara la part html únicament mostra el títol, el codi javascript mostra l'array JSON rebut a la consola-web, de manera que cal obrir-la per poder examinar la resposta rebuda.



## Pràctica-1.2: mostrar els id de /post

Escriure la app que fa una consulta al endpoint `/post` de [jsonplaceholder.typicode.com](https://jsonplaceholder.typicode.com) mostrant de cada un dels cent elements només el **id**.

Ampliem la pràctica anterior per tal de que ara si que es mostri a la pàgina web, al html els valors dels id dels posts JSON que retorna la consulta. Per fer-ho s'afegirà cada id al element div "elements".

### Posat a prova 0

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Connexio API REST utilitzant el metode XMLHttpRequest()</h1>
    <br/>
    <div id="elements"></div>
  </body>
  <script>
    var xmlhttp = new XMLHttpRequest();
    var url = "https://jsonplaceholder.typicode.com/posts";
    xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        var myArr = JSON.parse(this.responseText);
        console.log(myArr);
        myFunction(myArr);
      }
    };
  </script>
</html>
```

```
xmlhttp.open("GET", url, true);
xmlhttp.send();
console.log(xmlhttp);
function myFunction(array) {
    var sortida = "";
    for(var i = 0; i < array.length; i++) {
        sortida += array[i].id + " - " + array[i].title + "<br/>";
        console.log(array[i].id + " - " + array[i].title);
    }
    document.getElementById("elements").innerHTML = sortida;
}
</script>
</html>
```

Descripció del codi utilitzat:

- dins de la funció `onreadystatechange` es crida a la funció `myFunction(myArr)`, que és l'encarregada d'afegir element a element (títol a títol) al div "elements" de la pàgina web.
- `myFunction(myArr)`: aquesta funció rep l'array d'elements de la resposta (els elements JSON convertits amb parse a elements javascript) i itera un a un per a cada element. A cada iteració concatena el *títol* de l'element a un string anomenat *sortida*. També a cada iteració genera un log a la consola-web amb el títol. Per últim la funció assigna al element dibuix html anomenat "elements" la variable sortida, de manera que en la pàgina htm en el lloc d'aquest div es mostrarà l'string amb tots els títols concatenats.
- `console.log(xmlhttp)`: podem observar que la última ordre del codi script és una petició de log a la consola-web del valor de l'objecte xmlhttp, el que estem utilitzant per el diàleg client/server API-REST. Aquest log ens permetrà observar des de la consola els valors de la connexió, per exemple del valor de status que indicarà 200 si la petició s'ha executat correctament.

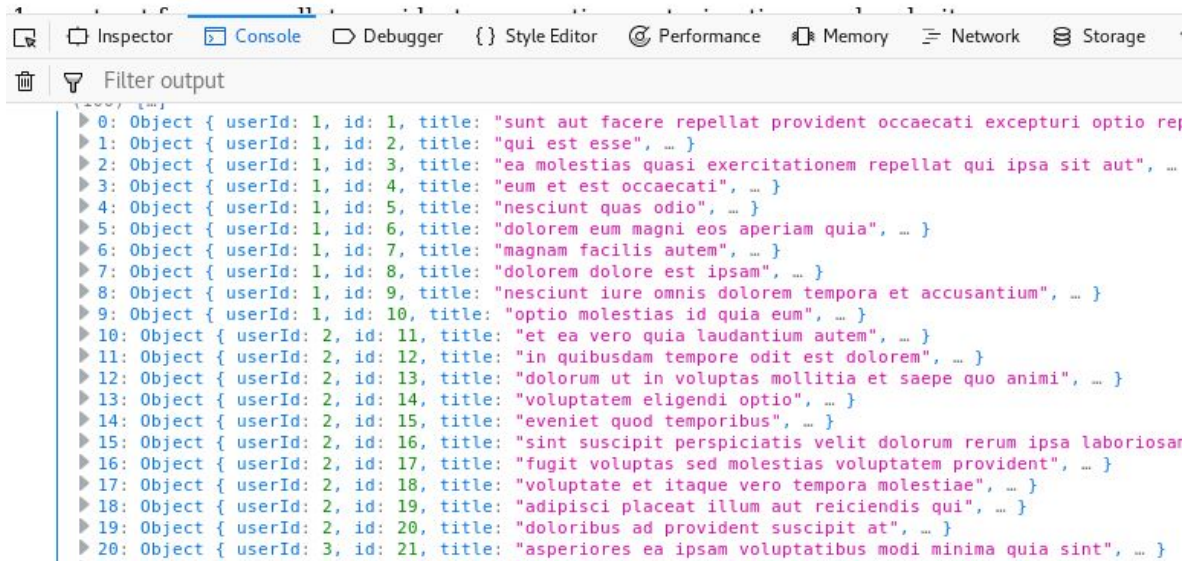
Podem Observar que ara la pàgina web mostra correctament el llistat de títols:

# Connexio API REST utilitzant el

- 1 - sunt aut facere repellat provident occaecati excepturi optio reprehenderit
- 2 - qui est esse
- 3 - ea molestias quasi exercitationem repellat qui ipsa sit aut
- 4 - eum et est occaecati
- 5 - nesciunt quas odio
- 6 - dolore eum magni eos aperiam quia
- 7 - magnam facilis autem
- 8 - dolore dolore est ipsam
- 9 - nesciunt iure omnis dolore tempora et accusantium
- 10 - optio molestias id quia eum
- 11 - et ea vero quia laudantium autem
- 12 - in quibusdam tempore odit est dolore
- 13 - dolorum ut in voluptas mollitia et saepe quo animi
- 14 - voluptatem eligendi optio
- 15 - eveniet quod temporibus
- 16 - sint suscipit perspiciatis velit dolorum rerum ipsa laboriosam odio
- 17 - fugit voluptas sed molestias voluptatem provident
- 18 - voluptate et itaque vero tempora molestiae
- 19 - adipisci placeat illum aut reiciendis qui
- 20 - doloribus ad provident suscipit at

També podem observar que a la consola-web hi apareixen els títols de cada un dels elements del array JSON rebut:

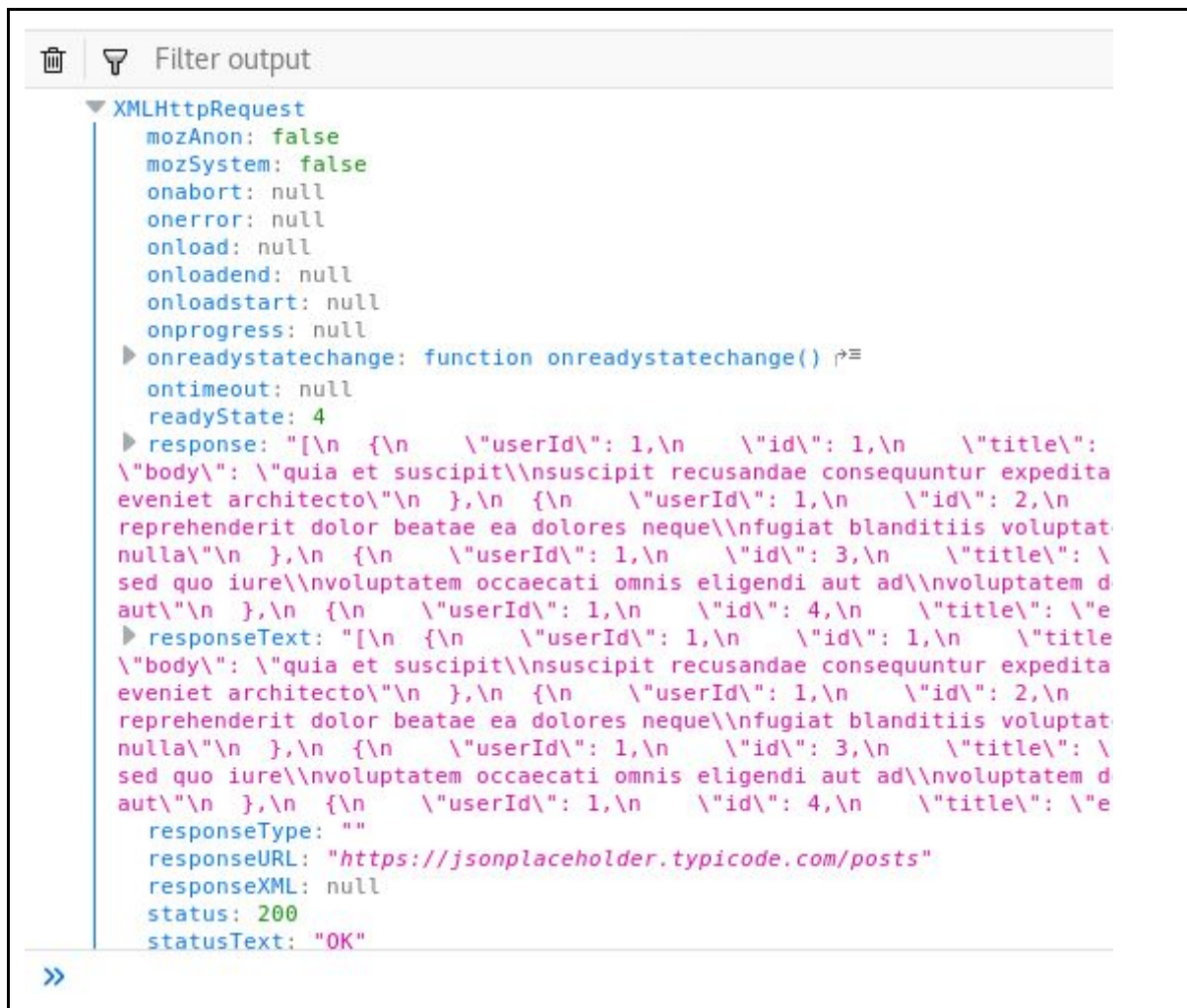
## Connexio API REST utilitzant el mètode XM





Observem també en la consola el valor del objecte xmlhttp del que s'en fa log i consultem el valor retornat de status. En especial observem:

- status
- responseURL
- withCredentials



```
XMLHttpRequest
  mozAnon: false
  mozSystem: false
  onabort: null
  onerror: null
  onload: null
  onloadend: null
  onloadstart: null
  onprogress: null
  ▶ onreadystatechange: function onreadystatechange()
  ontimeout: null
  readyState: 4
  ▶ response: "[\n  {\n    \"userId\": 1,\n    \"id\": 1,\n    \"title\": \"\n    \"body\": \"quia et suscipit\\nsuscipit recusandae consequuntur expedita\n    eveniet architecto\\n },\n  {\n    \"userId\": 1,\n    \"id\": 2,\n    reprehenderit dolor beatae ea dolores neque\\nfugiat blanditiis voluptat\n    nulla\\n },\n  {\n    \"userId\": 1,\n    \"id\": 3,\n    \"title\": \"\n    sed quo iure\\nvolutatem occaecati omnis eligendi aut ad\\nvolutatem d\n    aut\\n },\n  {\n    \"userId\": 1,\n    \"id\": 4,\n    \"title\": \"e\n  ]\n  ▶ responseText: \"[\n  {\n    \"userId\": 1,\n    \"id\": 1,\n    \"title\": \"\n    \"body\": \"quia et suscipit\\nsuscipit recusandae consequuntur expedita\n    eveniet architecto\\n },\n  {\n    \"userId\": 1,\n    \"id\": 2,\n    reprehenderit dolor beatae ea dolores neque\\nfugiat blanditiis voluptat\n    nulla\\n },\n  {\n    \"userId\": 1,\n    \"id\": 3,\n    \"title\": \"\n    sed quo iure\\nvolutatem occaecati omnis eligendi aut ad\\nvolutatem d\n    aut\\n },\n  {\n    \"userId\": 1,\n    \"id\": 4,\n    \"title\": \"e\n    responseType: \"\n    responseURL: \"https://jsonplaceholder.typicode.com/posts\"\n    responseXML: null\n    status: 200\n    statusText: \"OK\"
  >>
```

### Pràctica-1.3: mostrar id+email dels comments del post 1

Exercicis de la part: *ara et toca a tu!*

- A. Mostrar els comments a la consola-web del primer post.
- B. De cada comentari del primer post mostrar només el ID i el email.
- C. Mostrar per consola-web les propietats id i email del primer element del array

#### Apartat A)

Simplement consultar el nou endpoint [/post/1/comments](https://jsonplaceholder.typicode.com/posts/1/comments) corresponents al array JSON dels comentaris fets al primer post i fer el console.log de la resposta rebuda. Consta d'un array amb 5 comments.

Mostrar els comments a la consola del primer post:

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Connexio API REST utilitzant el metode XMLHttpRequest</h1>
    <br/>
    <div id="elements"></div>
  </body>
  <script>
    var xmlhttp = new XMLHttpRequest();
    var url = "https://jsonplaceholder.typicode.com/posts/1/comments";
    xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        var myArr = JSON.parse(this.responseText);
        console.log(myArr);
      }
    };
    xmlhttp.open("GET", url, true);
    xmlhttp.send();
    console.log(xmlhttp);
  </script>
</html>
```

```
(5) [...]
  ▶ 0: Object { postId: 1, id: 1, name: "id labore ex et quam laborum", ... }
  ▶ 1: Object { postId: 1, id: 2, name: "quo vero reiciendis velit similique earum", ... }
  ▶ 2: Object { postId: 1, id: 3, name: "odio adipisci rerum aut animi", ... }
  ▶ 3: Object { postId: 1, id: 4, name: "alias odio sit", ... }
  ▶ 4: Object { postId: 1, id: 5, name: "vero eaque aliquid doloribus et culpa", ... }
    length: 5
  ▶ <prototype>: Array []
```

## Apartat B)

De cada comentari del primer post mostrar només els camps id i email. Tornem a usar la funció myfunction i ara per a cada element del array de resposta (els comentaris) concatenem en un string el id i el email. Finalment mostrem aquest string assignant-lo a l'element div anomenat "elements".

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Connexio API REST utilitzant el metode XMLHttpRequest</h1>
    <br/>
```

```

<div id="elements"></div>
</body>
<script>
var xmlhttp = new XMLHttpRequest();
var url = "https://jsonplaceholder.typicode.com/posts/1/comments";
xmlhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
    var myArr = JSON.parse(this.responseText);
    console.log(myArr);
    myFunction(myArr);
}
};
xmlhttp.open("GET", url, true);
xmlhttp.send();
console.log(xmlhttp);
function myFunction(array) {
var sortida = "";
for(var i = 0; i < array.length; i++) {
    sortida += array[i].id + " - " + array[i].email + "<br/>";
    console.log(array[0].id + " - " + array[0].email);
}
document.getElementById("elements").innerHTML = sortida;
}
</script>
</html>

```

Podem observar els resultats html que es visualitzen amb els cinc emails dels usuaris que han realitzat comentaris:

## Connexio API REST

1 - Eliseo@gardner.biz  
 2 - Jayne\_Kuhic@sydney.com  
 3 - Nikita@garfield.biz  
 4 - Lew@alysha.tv  
 5 - Hayden@althea.biz

### Apartat C)

Mostrar per consola-web les propietats id i email però només del primer dels comentaris.

```

<!DOCTYPE html>
<html>
  <body>

```

```
<h1>Connexio API REST utilitzant el metode XMLHttpRequest</h1>
<br/>
<div id="elements"></div>
</body>
<script>
var xmlhttp = new XMLHttpRequest();
var url = "https://jsonplaceholder.typicode.com/posts/1/comments";
xmlhttp.onreadystatechange = function() {
if (this.readyState == 4 && this.status == 200) {
    var myArr = JSON.parse(this.responseText);
    console.log(myArr);
    console.log(myArr[0].id + " - " + myArr[0].email);
}
};
xmlhttp.open("GET", url, true);
xmlhttp.send();
console.log(xmlhttp);
</script>
</html>
```



## II) Sol·licituds de l'API

### Instal·lar jquery

```
dnf -y install npm
npm install jquery
$ ls -l jquery.min.js
-rw-rw-r-- 1 ecanet ecanet 88145 26 oct 1985 jquery.min.js
```

Usualment en lloc de treballar amb la còpia local de jquery en els scripts el que es fa és posar un include, en el head per utilitzar el jquery de Google CDN (Content Delivery Network):

```
<head>
<script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
</head>
```

Es consultarà l'endpoint <https://api.myjson.com/bins/tar0f> que retorna un JSON de books amb informació del llibre, imatges, etc.

### Exemple jQuery getJSON:

Exemple bàsic que simplement mostra per consola el JSON rebut com a resposta amb totes les dades dels llibres. Observem que jQuery es defineix a partir de l'oprrador \$, tot allò definit a partir d'aquest operador és jQuery i les variables s'han d'utilitzar dins de l'àmbit en que tenen significat.

Descripció:

- Crida a la funció getJSON de jQuery passat-li la url on accedir, la resposta retorna dins de la variable data.
- Dins de la funció function és on s'escriu el codi d'allò que cal fer amb la resposta jQuery, en aquest cas simplement es fa un console.log.

Exemple mytest\_2.1.html

```
<!DOCTYPE html>
<html>
  <head>
    <script
src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body>
    <button id="button">BUSCA</button>
```

```

</body>
<script>
$.getJSON('https://api.myjson.com/bins/tar0f',function(data){
    console.log(data)
});
</script>
</html>

```

Anem a ampliar l'exemple afegint al button anomenat "search" cada un dels títols dels llibres. Observem els següents passos:

- Tot el que codifiquem és dins de `function(data){...}` que conté la resposta del servidor en la variable data.
- Es realitza un `console.log` de la resposta rebuda. Un array amb 17 objectes.
- Es defineix la funció `mostrarTitols` que realitza els passos següents:
  - La variable `cuentos` pren el valor del array JSON amb els 17 llibres.
  - Amb `$.each` s'itera per a cada un dels elements del array JSON, per a cada element s'executa la funció `function(key,value)`.
  - A cada iteració es pren el valor del títol del llibre i es desa en format html en un paràgraf `<p/>` a la variable `titulo`.
  - Tot seguit s'afegeix aquest contingut html a l'element button anomenat "search".
- Fixem-nos que fins ara a dins de `function(data)` el que hem fet és: a) `console.log`; b) definir la funció `mostrarTitols`; c) ara cridem a la funció `mostrarTitols` perquè executi el seu codi i generi el nom dels llibres per pantalla.

Exemple mytest\_2.2.html

```

<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body>
    <button id="search">BUSCA</button>
  </body>
  <script>
$.getJSON('https://api.myjson.com/bins/tar0f',function(data){
    console.log(data)
    function mostrarTitols(){
        var cuentos = data.books;
        $.each(cuentos, function(key, value){
            var $titulo = $('<p/>').html(value.titulo);
            $('#search').append($titulo);
        });
    };
    mostrarTitols();
});
</script>

```

```
</html>
```

## Exemple jQuery AJAX

Descripció:

- `$(document).ready` indica que cal executar la funció que rep com a argument quan el document s'hagi carregat de tot.
- `function(){...}` és on definim tot el codi.
- Es defineix una variable amb la url on connectar.
- Es crea un funció que s'associa a l'esdeveniment *click* del botó "*button*". En prémer aquest botó s'executarà aquesta funció.
- Aquesta funció en realitat és una crida a `$.ajax`, que utilitza quatre arguments:
  - la url
  - el mètode a usar, en aquest cas GET
  - les accions a fer en cas de *success*, en cas de que la consulta es realitza correctament. En aquest exemple simplement fa `console.log`.
  - les accions a fer en cas de *fail*, si ha fallat la consulta jQuery.

Exemple mytest\_2.3.html

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body>
    <button id="button">BUSCA</button>
  </body>
  <script>
    $(document).ready(function(){
      var url = 'https://api.myjson.com/bins/tar0f';
      $('#button').click(function(){
        $.ajax({
          url: url,
          type:"GET",
          success:function(result){
            console.log(result)
          },
          error:function(error){
            console.log(error)
          }
        });
      });
    });
  </script>
</html>
```

En carregar la pàgina html i activar la consola-web veiem que no s'hi mostra res. Un cop clicat al botó observem en la consola-web que apareix l'array JSON amb els 17 objectes corresponents als 17 llibres.

### Exemple jQuery get

En aquest exemple s'utilitzarà la funció jQuery get. Altre cop en clicar el botó es generarà per consola-web el llistat dels elements JSON rebuts.

Descripció:

- Aquí directament es defineix la **funció** a executar en fer **click** en el botó "**button**".
- Aquesta funció defineix una url i crida a la funció jQuery **\$.get**.

Exemple mytest\_2.4.html

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body>
    <button id="button">BUSCA</button>
  </body>
  <script>
    $('#button').click(function(){
      var url='https://api.myjson.com/bins/tar0f';
      $.get(url, function(data,status){
        console.log(data)
      });
    });
  </script>
</html>
```

En executar aquest codi observem a la consola que en prémer el botó apareix l'array d'objectes JSON rebut.

### Pràctica 2.5.a: mostrar els llibres (títol, imatge)

Usant el mètode jQuery getJSON i el endpoint <https://api.myjson.com/bins/tar0f> mostrar per pantalla amb html el títol i la imatge de cada un dels llibres.

Descripció:

- S'utilitza la funció jquery **getJSON**.
- Es defineix la funció de **callback** que genera el log de consola, defineix la funció **mostrarTítols** i la crida.
- La funció **mostrarTítols** itera un a un els elements book amb un **\$.each**.

- es crea un nou element html anomenat *object* de tipus div.
- Per cada element se n'extreu el *títol* i la imatge (anomenada *portada*). Cada títol és un element html de tipus <p/>. Cada portada és un element de tipus <img/>.
- A cada iteració s'afegeixen a l'objecte div anomenat *object*.

exemple mytest\_2.5.a

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
  </head>
  <body>
    <div id="search"></div>
  </body>
  <script>
    $.getJSON('https://api.myjson.com/bins/tarOf', function(data){
      console.log(data)
      function mostrarTitols(){
        var cuentos = data.books;
        $.each(cuentos, function(key, value){
          var $object = $('<div class="object"/>');
          var $titulo = $('<p class="titulo"/>').html(value.titulo);
          var $image = $('<img/>').addClass("portada").attr("src",value.portada);
          $object.append($titulo);
          $object.append($image);
          $('#search').append($object);
        });
      };
      mostrarTitols();
    });
  </script>
</html>
```

El resultat en carregar la pàgina és que es mostren els llibre cada un amb un títol i la imatge del llibre.

## Pràctica 2.5.b: mostrar els llibres (títol, imatge) amb css

Ampliem la pràctica anterior assignant estils css als elements html. Observeu que ja en l'exemple anterior a cada tipus d'element se li havia assignat una classe:

- Els títols, objectes <p/> són de la classe "*títol*".
- Les imatges, objectes de tipus <img/>, tenen assignada la classe "*portada*".
- Cada objecte object té assignada la classe *object*.

S'afegeix al <head> un bloc <style> amb la definició css a aplicar.

exemple mytest\_2.5.b.html

```

<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <style>
      img {
        width: 30%;
        border: 1px solid grey
      }
      .object{
        width:300px;
        margin: 10px;
        padding: 10px;
        text-align: center;
        background-color: black;
        border-radius: 5px
      }
      .titulo {
        font-weight: bold;
        color: yellow
      }
      #search {
        display: flex;
        flex-wrap: wrap;
      }
    </style>
  </head>
  <body>
    <div id="search"></div>
  </body>
  <script>
    $.getJSON('https://api.myjson.com/bins/tar0f', function(data){
      console.log(data)
      function mostrarTitols(){
        var cuentos = data.books;
        $.each(cuentos, function(key, value){
          var $object = $('<div class="object"/>');
          var $titulo = $('<p class="titulo"/>').html(value.titulo);
          var $image = $('<img/>').addClass("portada").attr("src",value.portada);
          $object.append($titulo);
          $object.append($image);
          $('#search').append($object);
        });
      };
      mostrarTitols();
    });
  </script>
</html>

```

## Pràctica 2.5.c: mostrar també 'descripcion' i 'detalle'

Ampliem la pràctica anterior mostrant també els elements 'descripció' i 'detalle' de cada llibre. Aquests elements també han de tenir el seu estil css personalitzat.

exemple mytest\_2.5.c.html

```
<!DOCTYPE html>
<html>
  <head>
    <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.3.1/jquery.min.js"></script>
    <style>
      img {
        width: 30%;
        border: 1px solid grey
      }
      .object{
        width:300px;
        margin: 10px;
        padding: 10px;
        text-align: center;
        background-color: black;
        border-radius: 5px
      }
      .titulo {
        font-weight: bold;
        color: yellow
      }
      #search {
        display: flex;
        flex-wrap: wrap;
      }
      .descripcion {
        font-weight: bold;
        color: white
      }
      .detalle {
        width: 80%
      }
    </style>
  </head>
  <body>
    <div id="search"></div>
  </body>
  <script>
    $.getJSON('https://api.myjson.com/bins/tar0f', function(data){
      console.log(data)
      function mostrarTitols(){
        var cuentos = data.books;
```

```

$.each(cuentos, function(key, value){
  var $object = $('<div class="object"/>');
  var $titulo = $('<p class="titulo"/>').html(value.titulo);
  var $image = $('<img/>').addClass("portada").attr("src",value.portada);
  var $descripcion = $('<p class="descripcion"/>').html(value.descripcion);
  var $detalle = $('<img/>').addClass("detalle").attr("src",value.detalle);
  $object.append($titulo);
  $object.append($image);
  $object.append($descripcion);
  $object.append($detalle);
  $('#search').append($object);
});
};
mostrarTitols();
});
</script>
</html>

```

Exemple de visualització de la pàgina web:

<p><b>El por qué de no usar Mustache</b></p>  <p>El famoso escritor Albert Lara nos enseña el poder de JavaScript para crear plantillas y las razones por las que un sistema ideado justamente para este fin como es Mustache, no siempre es la mejor opción...</p> 	<p><b>Shut up little bastards</b></p>  <p>El silencio es necesario en estos tiempos para poder concentrarte. Bryn luchará contra viento y marea en esta cruda historia basada en hechos reales.</p> 	<p><b>Sudo mucho</b></p>  <p>Para qué aprender JavaScript si puedes estar subiendo copas sin parar. La historia de Eduardo nos llevará desde estar hundido en la mierda hasta seguir hundido en la mierda.</p> 
<p><b>Asincronía y su puta madre</b></p> 	<p><b>El java empieza ahorita</b></p> 	<p><b>document.write()</b></p> 



### III) Resposta de l'API

<code>https://api.github.com/users</code>
<code>https://github.com/mojombo</code>

Spotify

- ❑ <https://developer.spotify.com>
- ❑ <https://developer.spotify.com/documentation/web-api/reference>

#### Respostes: Status i ReadyState

##### ReadyState

Vegem els valors principals que pot prendre la propietat readyState:

readyState = 0 => petició no inicialitzada

readyState = 1 => connexió de servidor establerta

readyState = 2 => petició rebuda

readyState = 3 => petició en procés

readyState = 4 => petició finalitzada i resposta llesta

##### Status

Ara veurem els valors més comuns que acostuma a prendre la propietat status en el cas que la petició AJAX es processi satisfactòriament:

Status 200 => Resposta estàndard quan la petició està OK.

Status 201 => En cas de peticions per POST, petició completada i nou recurs creat.

Per la seva banda, els errors 400 fan referència als errors de la part client:

Status 400 = Bad Request => La petició no es pot completar per un error de sintaxi.

Status 401 = Unauthorized => La petició és legal però el servidor la rebutja.

Acostuma a saltar en els casos que l'autorització és necessària però errònia.

Status 403 = Forbidden => La petició és legal però el servidor la rebutja.

Status 404 = Not found => L'URL indicat no es troba.

I els errors 500 indiquen errors de servidor:

Status 500 = Internal Server Error => Missatge d'error genèric.

Status 503 = Service Unavailable => El servidor no està disponible.

## Exemple: consulta a l'API de GitHub

Exemple mytest\_3.1.a.html

```
<!DOCTYPE html>
<html>
  <body>
    <h1>Connexio API REST utilitzant el metode XMLHttpRequest()</h1>
    <br/>
    <div id="elements"></div>
  </body>
  <script>
    var xmlhttp = new XMLHttpRequest();
    var url = "https://api.github.com";
    xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        var myArr = JSON.parse(this.responseText);
        console.log(myArr);
      }
    };
    xmlhttp.open("GET", url, true);
    xmlhttp.send();
    console.log(xmlhttp)
  </script>
</html>
```

Valors generats en la consulta a *api.github.com*:

```
{
  "current_user_url": "https://api.github.com/user",
  "current_user_authorizations_html_url": "https://github.com/settings/connections/applications/{client_id}",
  "authorizations_url": "https://api.github.com/authorizations",
  "code_search_url": "https://api.github.com/search/code?q={query}&page,per_page,sort,order",
  "commit_search_url": "https://api.github.com/search/commits?q={query}&page,per_page,sort,order",
  "emails_url": "https://api.github.com/user/emails",
  "emojis_url": "https://api.github.com/emojis",
  "events_url": "https://api.github.com/events",
  "feeds_url": "https://api.github.com/feeds",
  "followers_url": "https://api.github.com/user/followers",
  "following_url": "https://api.github.com/user/following/{target}",
  "gists_url": "https://api.github.com/gists/{gist_id}",
  "hub_url": "https://api.github.com/hub",
  "issue_search_url": "https://api.github.com/search/issues?q={query}&page,per_page,sort,order",
  "issues_url": "https://api.github.com/issues",
  "keys_url": "https://api.github.com/user/keys",
  "label_search_url": "https://api.github.com/search/labels?q={query}&repository_id={repository_id}&page,per_page",
  "notifications_url": "https://api.github.com/notifications",
  "organization_url": "https://api.github.com/orgs/{org}",
  "organization_repositories_url": "https://api.github.com/orgs/{org}/repos?type,page,per_page,sort",
  "organization_teams_url": "https://api.github.com/orgs/{org}/teams",
  "public_gists_url": "https://api.github.com/gists/public",
  "rate_limit_url": "https://api.github.com/rate_limit",
  "repository_url": "https://api.github.com/repos/{owner}/{repo}"
}
```

```

"repository_search_url": "https://api.github.com/search/repositories?q={query}&page,per_page,sort,order}",
"current_user_repositories_url": "https://api.github.com/user/repos?type,page,per_page,sort",
"starred_url": "https://api.github.com/user/starred{/owner}/{repo}",
"starred_gists_url": "https://api.github.com/gists/starred",
"user_url": "https://api.github.com/users/{user}",
"user_organizations_url": "https://api.github.com/user/orgs",
"user_repositories_url": "https://api.github.com/users/{user}/repos?type,page,per_page,sort",
"user_search_url": "https://api.github.com/search/users?q={query}&page,per_page,sort,order"
}

```

Observem els valors de:

- ReadyState: 4
- Status: 200

### Exemple: accedir a l'API de GitHub /user

Ara fem el mateix però variant l'endpoint, intentant accedir a l'API de GitHub /user, no ens deixa perquè es requereix autorització. Podem observar en la consola-web els valors:

- ReadyState: 4
- Status: 401

Exemple mytest\_3.1.b.html

```

<!DOCTYPE html>
<html>
  <body>
    <h1>Connexio API REST utilitzant el metode XMLHttpRequest()</h1>
    <br/>
    <div id="elements"></div>
  </body>
  <script>
    var xmlhttp = new XMLHttpRequest();
    var url = "https://api.github.com/user";
    xmlhttp.onreadystatechange = function() {
      if (this.readyState == 4 && this.status == 200) {
        var myArr = JSON.parse(this.responseText);
        console.log(myArr);
      }
    };
    xmlhttp.open("GET", url, true);
    xmlhttp.send();
    console.log(xmlhttp)
  </script>
</html>

```

```

{
  "message": "Requires authentication",
  "documentation_url": "https://developer.github.com/v3/users/#get-the-authenticated-user"
}

```

Podeu observar que de la informació que mostra GitHub us indica que podeu accedir a l'enpoint users per el llistat de usuaris i a /users/nom-user per a l'informació d'un usuari.

Exemple <https://api.github.com/users/mojombo>

```
{
  "login": "mojombo",
  "id": 1,
  "node_id": "MDQ6VXNlcnJE=",
  "avatar_url": "https://avatars0.githubusercontent.com/u/1?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/mojombo",
  "html_url": "https://github.com/mojombo",
  "followers_url": "https://api.github.com/users/mojombo/followers",
  "following_url": "https://api.github.com/users/mojombo/following{/other_user}",
  "gists_url": "https://api.github.com/users/mojombo/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/mojombo/starred{/owner}/{repo}",
  "subscriptions_url": "https://api.github.com/users/mojombo/subscriptions",
  "organizations_url": "https://api.github.com/users/mojombo/orgs",
  "repos_url": "https://api.github.com/users/mojombo/repos",
  "events_url": "https://api.github.com/users/mojombo/events{/privacy}",
  "received_events_url": "https://api.github.com/users/mojombo/received_events",
  "type": "User",
  "site_admin": false,
  "name": "Tom Preston-Werner",
  "company": null,
  "blog": "http://tom.preston-werner.com",
  "location": "San Francisco",
  "email": null,
  "hireable": null,
  "bio": null,
  "public_repos": 61,
  "public_gists": 62,
  "followers": 21883,
  "following": 11,
  "created_at": "2007-10-20T05:24:19Z",
  "updated_at": "2020-03-08T17:24:28Z"
}
```

Exemple consultant l'usuari edtasixm11

```
{
  "login": "edtasixm11",
  "id": 16031908,
  "node_id": "MDQ6VXNlcnJE2MDMxOTA4",
  "avatar_url": "https://avatars1.githubusercontent.com/u/16031908?v=4",
  "gravatar_id": "",
  "url": "https://api.github.com/users/edtasixm11",
  "html_url": "https://github.com/edtasixm11",
  "followers_url": "https://api.github.com/users/edtasixm11/followers",
  "following_url": "https://api.github.com/users/edtasixm11/following{/other_user}",
  "gists_url": "https://api.github.com/users/edtasixm11/gists{/gist_id}",
  "starred_url": "https://api.github.com/users/edtasixm11/starred{/owner}/{repo}",
  "subscriptions_url": "https://api.github.com/users/edtasixm11/subscriptions",
  "organizations_url": "https://api.github.com/users/edtasixm11/orgs",
  "repos_url": "https://api.github.com/users/edtasixm11/repos",
  "events_url": "https://api.github.com/users/edtasixm11/events{/privacy}",
  "received_events_url": "https://api.github.com/users/edtasixm11/received_events",
  "type": "User",
  "site_admin": false,
  "name": "edtasixm11",
  "company": "Escola del treball de Barcelona",
  "blog": "https://sites.google.com/site/asixm11edt/",
  "location": "Barcelona",
  "email": null,
  "hireable": null,
  "bio": "Edt escola del treball de barcelona Departament d'informàtica ASIX Administració de sistemes operatius en xarxa"
}
```

```
M11-SAD Seguretat i alta disponibilitat.",
"public_repos": 14,
"public_gists": 0,
"followers": 1,
"following": 0,
"created_at": "2015-11-26T11:11:49Z",
"updated_at": "2020-03-14T18:04:47Z"
}
```

## Spotify

### Documentació:

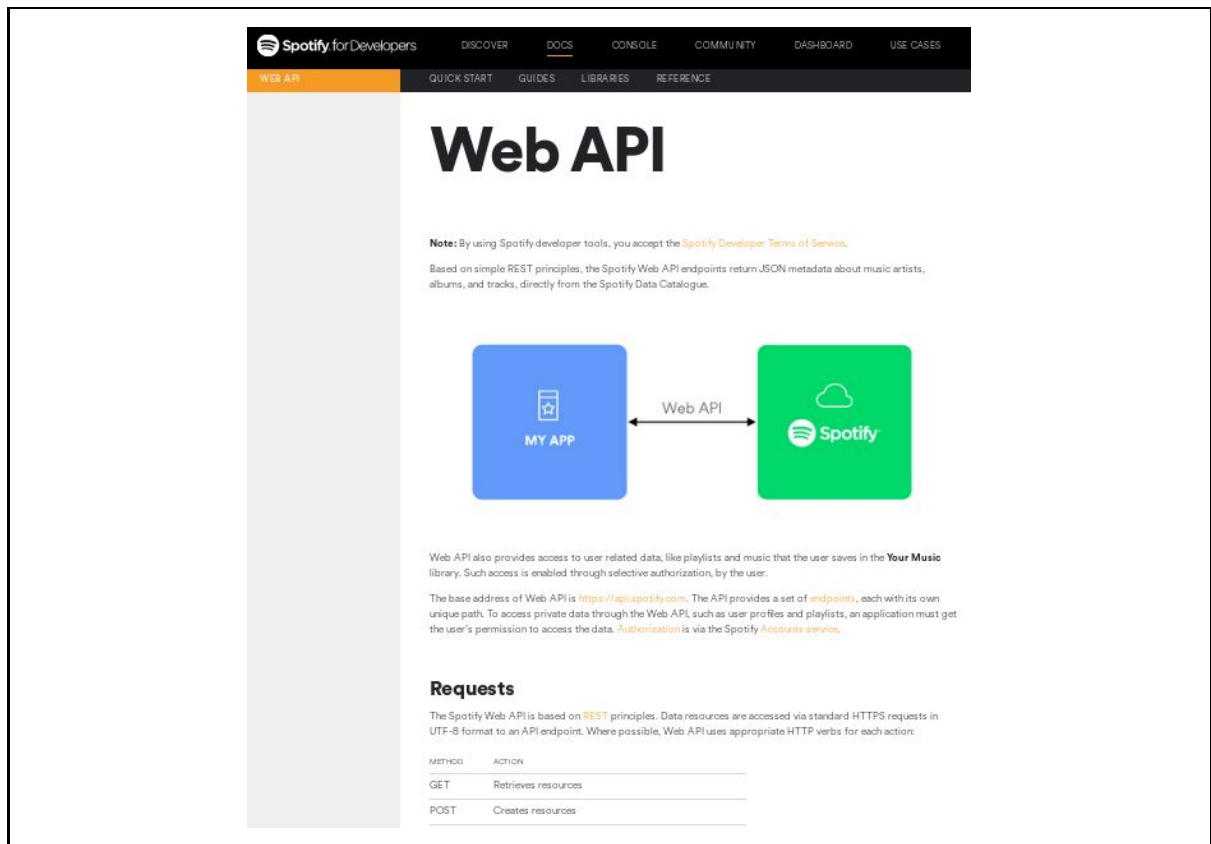
- ❑ [Documentation web-API](#)
- ❑ [API endpoint Reference](#)
- ❑ [Authorization Guide](#)
- ❑ Autoritzacions: <https://accounts.spotify.com/es/status>

### Procediment:

- <https://developers.spotify.com>
- docs
- Web API
- Autoritzacions: <https://accounts.spotify.com/es/status>

### Requeriments:


- Disposar d'un compte de Spotify
- Totes les consultes a l'API requereixen d'autorització ?? cert??
- Per poder consultar dades d'un usuari cal que l'usuari autoritzi l'app a fer consultes de les seves dades.
- Caldrà enregistrar la app a Spotify per poder rebre els tokens que permetin a la app identificar-se i accedir a la API de Spotify.



## QuickStart

A través de la página de spotify Quickstart es muestra un tutorial per fer una app que accedeixi a la API de spotify. Explica com:

- Registrar una aplicació amb Spotify
- Autenticar un usuari i obtenir autorització per accedir a les dades d'usuari.
- Obtener dades de la Web-API endpoints


**Spotify** for Developers

[DISCOVER](#)
[DOCS](#)
[CONSOLE](#)
[COMMUNITY](#)
[DASHBOARD](#)
[USE CASES](#)

[WEB API](#)
[QUICK START](#)
[GUIDES](#)
[LIBRARIES](#)
[REFERENCE](#)

QUICK START

# Web API Tutorial

Create a simple server-side application that accesses user related data through the Spotify Web API.

**Note:** By using the Spotify Tools, you accept our [Developer Terms of Service](#).

Through the [Spotify Web API](#), external applications retrieve Spotify content such as album data and playlists. To access *user-related data* through the Web API, an application must be authorized by the user to access that particular information.

In this tutorial we create a simple application using Node.js and JavaScript and demonstrate how to:

- Register an application with Spotify
- Authenticate a user and get authorization to access user data
- Retrieve the data from a Web API endpoint

The authorization flow we use in this tutorial is the [Authorization Code Flow](#). This flow first gets a code from the Spotify Accounts Service, then exchanges that code for an access token. The code-to-token exchange requires a secret key, and for security is done through direct server-to-server communication.

In this example we retrieve data from the Web API `/me` endpoint, that includes information about the current user.

The complete source code of the app that will create in this tutorial is available on [GitHub](#).

### Procediment:

1. Disposar d'un compte spotify
2. Anar al dashboard d'usuari de developer i acceptar les condicions de servei.  
Adreça: <https://developer.spotify.com/dashboard/>.
3. Registrar l'aplicació: clicar al botó "[Create a Client ID](#)".
  - a. Step 1/3: indicar: nom de l'aplicació, descripció i tipus d'aplicació (website).
  - b. Indicar 2/3 que estem desenvolupant una aplicació Non-Comercial.
  - c. Indicar 3/3 que acceptem i entenem els termes d'una aplicació non-comercial.
4. Un cop fet el procediment de registrar ja podem accedir al dashboard de developer, allà podem consultar:
  - a. El Client ID
  - b. El Client Secret

To use the Web API, start by creating a Spotify user account (Premium or Free). To do that, simply sign up at [www.spotify.com](https://www.spotify.com).

When you have a user account, go to the [Dashboard](#) page at the Spotify Developer website and, if necessary, log in. Accept the latest [Developer Terms of Service](#) to complete your account set up.

Any application can request data from Spotify Web API endpoints and many endpoints are open and will return data *without* requiring registration. However, if your application seeks access to a user's personal data (profile, playlists, etc.) it must be registered. Registered

applications also get other benefits, like higher rate limits at some endpoints.

You can [register your application](#), even before you have created it.

Indicar el nom de l'aplicació, descripció i tipus d'aplicació:

CREATE AN APP OR HARDWARE INTEGRATION

Step 1/3

App or Hardware Name \*

mytest

App or Hardware Description \*

mytest Spotify app acces API-REST

What are you building? \*

☐ I don't know

☐ Mobile App

☐ Desktop App

☒ Website

☐ Speakers

☐ Voice - Cortana

☐ Voice - Other

☐ TV

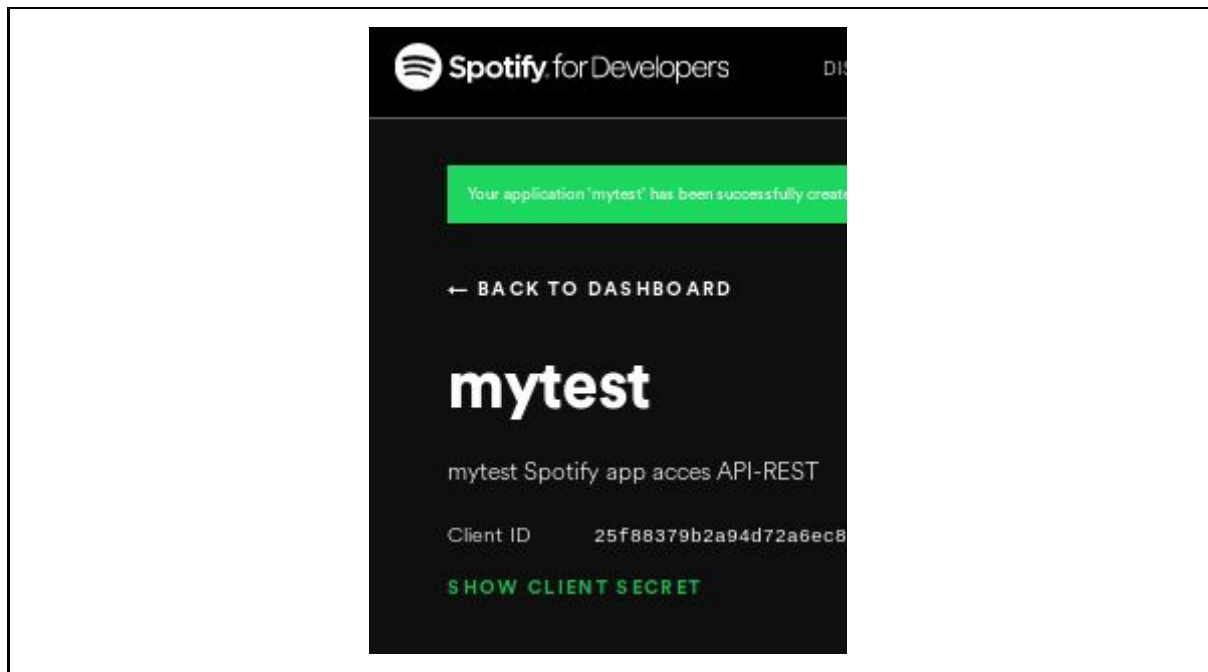
☐ Gaming Consoles

☐ Wearables

Check all that apply

Dashboard de Developer





En el Dashboard de developer es mostren dos conceptes claus que necessitarem per poder accedir a la API-REST de Spotify des de la app:

- ❑ **Client ID:** És l'identificador de client de la app. No de l'usuari, sinó de la app. Hem generat una app anomenada *mytest* i aquest és el seu ID.
- ❑ **Client Secret:** Aquest és un valor 'secret' que cal que ningú més pugui obtenir. Aquest és el valor que la app passarà a Spotify com a demostració de que és el Client-ID que diu ser. És com el password de la app per poder accedir als recursos API-REST de Spotify que requereixin autorització.

Es mostra la clau clicant en ella i apareix també una opció **RESET** que permet regenerar una nova clau.

## Preparar l'entorn

Es realitzarà una app amb node.js de manera que cal instal·lar aquest software i verificar que funciona correctament. Serà una Server Side app. Node.js és un software de servidor web.

Instal·lar node.js

```
# dnf -y install node
# node --version
v8.12.0
```

Exemple de app Hello-World amb node.js per verificar el funcionament del servidor node  
nodeapp.01.js

```
/* Load the HTTP library */
var http = require("http");

/* Create an HTTP server to handle responses */

http.createServer(function(request, response) {
  response.writeHead(200, {"Content-Type": "text/plain"});
  response.write("Hello World");
  response.end();
}).listen(8888);
```

Verificar el funcionament executant aquesta aplicació:

```
node nodeapp.01.js
```

```
# des del navegador
http://localhost:8888
```

Si volem examinar que està passant podrem engadrir node amb `--inspect` i mirar el debug del funcionament:

```
node --inspect nodeapp.01.js
Debugger listening on ws://127.0.0.1:9229/034e1260-48a9-4501-9e1e-c8f5b52c65ea
For help see https://nodejs.org/en/docs/inspector
```

## Create a project Folder and Files

Ara anem a crear un projecte per a la nostra app . En lloc de començar de zero el que farem és copiar un dels exemples de Spotify que hi ha a GitHub. Hi ha exemples de com fer apps que connectin a la API-REST de Spotify a [Oauth Examples](#).

- Connectar al recurs de GitHub de Oauth examples i descarregar l'exemple amb `clone`.
- Des de dins del directori descarregat assegurar-se d'instal·lar totes les dependències amb `npm install`.
- Verificar que dins del directori creat amb `git clone` existeix el sots-directori [node\\_modules](#) i també el sots-directori [authorization\\_code](#).
- Dins de `authorization_code` hi ha el directori [public](#) on hi haurà tots els fitxers de la app visibles per als clients, tot allò que es publica. Per exemple hi ha el fitxer [index.html](#).
- Tots els altres fitxers de l'aplicació van fora de públic i per tant no són accessibles per els clients.
- També hi ha dins de `authorization_code` el fitxer [app.js](#). The `app.js` file contains the main code of the application.

Descarregar el codi d'exemple, actualitzar dependències:

```

$ git clone https://github.com/spotify/web-api-auth-examples.git
Cloning into 'web-api-auth-examples'...
remote: Enumerating objects: 112, done.
remote: Total 112 (delta 0), reused 0 (delta 0), pack-reused 112
Receiving objects: 100% (112/112), 27.26 KiB | 775.00 KiB/s, done.
Resolving deltas: 100% (41/41), done.

$ cd web-api-auth-examples/
$ npm install
npm WARN deprecated request@2.83.0: request has been deprecated, see
https://github.com/request/request/issues/3142
...
$ pwd
/var/tmp/m06/api-rest/web-api-auth-examples

$ ll
total 64
drwxrwxr-x  3 ecanet ecanet 4096 24 mar 11:44 authorization_code
drwxrwxr-x  2 ecanet ecanet 4096 24 mar 11:44 client_credentials
drwxrwxr-x  3 ecanet ecanet 4096 24 mar 11:44 implicit_grant
-rw-rw-r--  1 ecanet ecanet 11345 24 mar 11:44 LICENSE
drwxrwxr-x 102 ecanet ecanet 4096 24 mar 11:47 node_modules
-rw-rw-r--  1 ecanet ecanet  327 24 mar 11:47 package.json
-rw-rw-r--  1 ecanet ecanet 27784 24 mar 11:47 package-lock.json
-rw-rw-r--  1 ecanet ecanet 1570 24 mar 11:44 README.md

```

Observar el contingut del directori `authorization_code` i del fitxer `app.js`

```

$ ll authorization_code/
total 12
-rw-rw-r-- 1 ecanet ecanet 4184 24 mar 11:44 app.js
drwxrwxr-x 2 ecanet ecanet 4096 24 mar 11:44 public

```

## Prompt the user to login

Observem ara el fitxer [index.html](#) (dins de `authorization_code`, és el fitxer que permet a l'usuari identificar-se i fer login).

```

This HTML file both provides a "Log in" link and makes the call to Web API, and provides
a template for data display of what is returned by the Web API /me endpoint).

```

Aquest fitxer conté el codi que permet a l'usuari identificar-se i fer login a Spotify.

## Provide the application credentials

El fitxer principal de l'aplicació és el fitxer app.js (que està fora de la part public). Aquest fitxer permet:

- Creating a server on your local machine
- Handling the user login request
- Specifying the scopes for which authorization is sought
- Performing the exchange of the authorization code for an access token
- Calling the Web API endpoint

En aquest fitxer podem veure on es defineixen les variables següents que permeten la identificació de l'usuari:

- ❑ **client\_id**: identificador del client
- ❑ **client\_secret**: clau d'identificació de l'usuari
- ❑ **redirect\_uri**: uri a la que cal connectar un cop el login ha sigut satisfactòria. En el tutorial es suggereix la ruta: <http://localhost:8888/callback>
- ❑ **scopes**: per a quines coses demana autorització la app, per exemple per a obtenir el nom sencer, la imatge del perfil i el email.

```
var client_id = 'CLIENT_ID'; // Your client id
var client_secret = 'CLIENT_SECRET'; // Your secret
var redirect_uri = 'REDIRECT_URI'; // Your redirect uri
var scope = 'user-read-private user-read-email';
```

Anem a editar **settings** del dashboard del developer per omplir l'atribut **redirect uri** amb el valor suggerit per el tutorial de Spotify: <http://localhost:8888/callback>. **Atenció**, assegureu-vos de que ha quedat ben desat el redirect-uri, torneu-hi a entrar per assegurar-vos de que es desa apropiadament.

The screenshot shows the Spotify Developer Dashboard settings for an application named "mytest". The "Application description" is "mytest Spotify app acces API-REST". Under the "Website" section, there is a link to "Add a website" and a description: "Where the user may obtain more information about this application (e.g. http://mysite.com)". Under the "Redirect URIs" section, the URI "http://localhost:8888/callback" is listed, and there is a green "ADD" button. A description below states: "White-listed addresses to redirect to after authentication success OR failure (e.g. http://mysite.com/callback/)".

## Call the Spotify Accounts service

See that the app.js file contains three calls to the Spotify Accounts Service:

- **The first call** is the service '/authorize' endpoint, passing to it the **client ID**, **scopes**, and **redirect URI**. This is the call that starts the process of authenticating to user and gets the user's authorization to access data.
- **The second call** is to the Spotify Accounts Service '/api/token' endpoint, passing to it the authorization code returned by the first call and the client secret key. This call returns an access token and also a refresh token.

**Note:** As app.js is not in the /public directory, its machinations cannot be seen from a web browser. This is important because we never want to expose our application Client Secret to a user. **Make sure that you safeguard your application Client Secret at all times.** Be aware, for example, that if you commit your code to a public repository like GitHub you will need to remove the Client Secret from your code before doing so.

To better understand the Accounts Service endpoints and the parameters passed in each call, see the full description of the [Authorization Code Flow](#).

After both calls are completed, and the user has authorized the app for access, the application will have the 'access\_token' it needs to retrieve the user data from the Web API.

**The third call**, in the code managing requests to '/refresh\_token', a refresh token is se

## Run the application

Per engegar l'aplicació n'hi ha prou de fer:

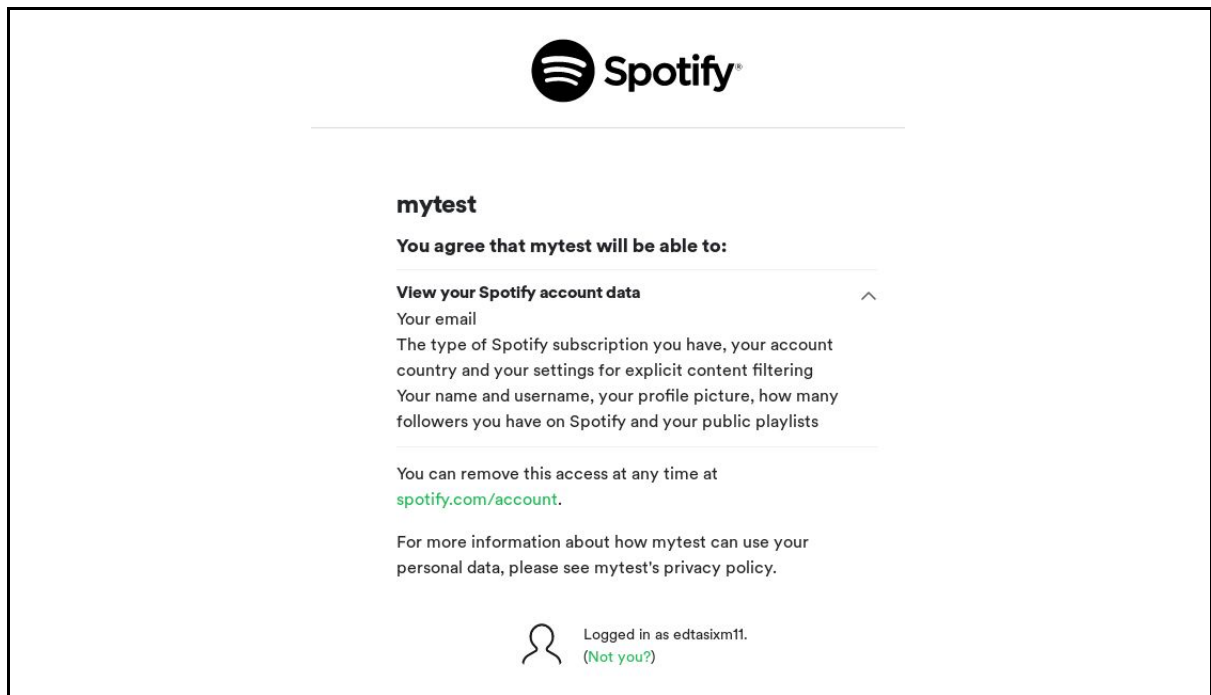
```
$ node app.js  
Listening on 8888
```

Ara podem observar que el servidor node.js està escoltant al localhost al port 8888. En accedir-hi veurem el següent process:

https://localhost:8888



Autoritzar l'accés a les dades de l'usuari. The Spotify Accounts Service now uses the `redirect_uri` to return to the application, passing back the `authorization_code` which is subsequently exchanged in a second call for an `access_token`.



Un cop autoritzat mostra les dades personals de l'usuari. The home page can call a Web API endpoint at `/me`, retrieve data, and render it.

# Logged in as edtasixm11

Display name	edtasixm11
Id	50x3mbjitlr912jv59l70jksx
Email	edtasixm11@gmail.com
Spotify URI	<a href="https://open.spotify.com/user/50x3mbjitlr912jv59l70jksx">https://open.spotify.com/user/50x3mbjitlr912jv59l70jksx</a>
Link	<a href="https://api.spotify.com/v1/users/50x3mbjitlr912jv59l70jksx">https://api.spotify.com/v1/users/50x3mbjitlr912jv59l70jksx</a>
Profile Image	
Country	ES

## oAuth info

Access token	BQAQMZeDNFwyP0w6oGcJwX_nTKqHKk63S3u2lzKbXjjYHsWZiHLb-37gN9...
Refresh token	AQAA55Gg_iGi8mdAMUoiislceXAApza59paZBXzwVY_OP86sm3zRq7bgx35x...

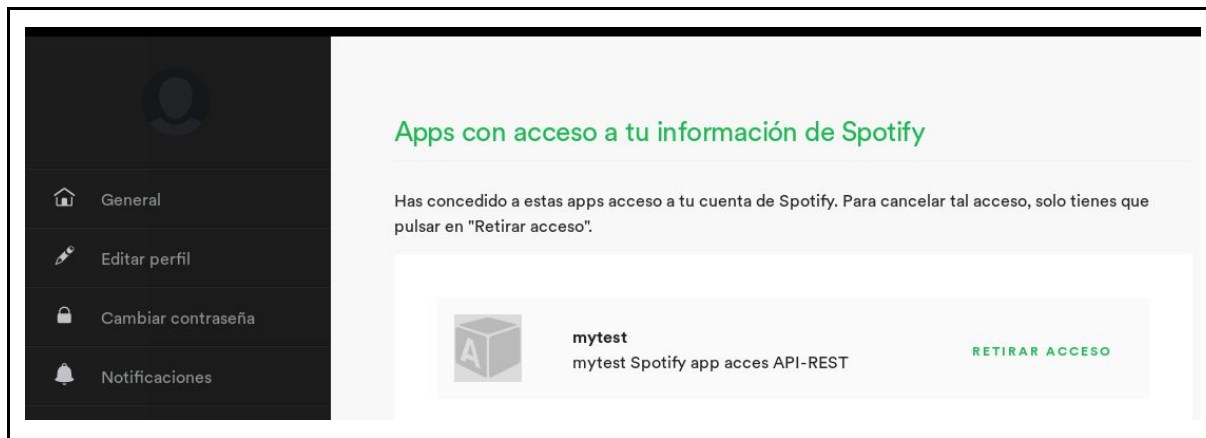
Obtain new token using the refresh token

Click **Obtain new token using the refresh token**; see that a new access token is generated; you have just completed creating a simple server-side application that accesses user related data. As this example app demonstrates how to retrieve data using an authorization flow, it may serve as a useful template for your own applications.

Podem observar a la consola del shell l'informació que ens mostra el debug de node.js:

```
$ node app.js
Listening on 8888
{ country: 'ES',
  display_name: 'edtasixm11',
  email: 'edtasixm11@gmail.com',
  explicit_content: { filter_enabled: false, filter_locked: false },
  external_urls:
    { spotify: 'https://open.spotify.com/user/50x3mbjitlr912jv59l70jksx' },
  followers: { href: null, total: 0 },
  href: 'https://api.spotify.com/v1/users/50x3mbjitlr912jv59l70jksx',
  id: '50x3mbjitlr912jv59l70jksx',
  images: [],
  product: 'open',
  type: 'user',
  uri: 'spotify:user:50x3mbjitlr912jv59l70jksx'
}
```

Si accedim a spotify amb el compte d'usuari i anem a Perfil / Apps observarem que ara apareix que hem concedit autorització a una app per accedir a les nostres dades d'usuari, a la app mytest.



## OAuth Info

Observem de l'exemple app construït fins ara que la resposta mostra dos tipus d'informació:

- La informació de l'usuari.
- El [Access Token](#) i el [Refresh Token](#). Aquests tokens són necessaris a la aplicació per autenticar-se i poder sol·licitar les peticions API-REST que volem fer per consultar artistes, cançons, etc.

**Atenció** que a la pàgina web no el mostra tot sencer. Si volem obtenir-lo cal que el retallem de la URI de la pàgina (abarca fins que trobem el &refresh\_token=).

## Test d'un Spotify Endpoint

Ara que tenim Access token podem provar interactivament a la web de Spotify reference alguns dels endpoints i provar de fer consultes a la seva API. Si les consultes requereixen autorització utilitzem el Access Token que acabem d'obtenir.

[Reference](#): docs / web-api / reference. Aquí hi ha la documentació, exemples i test dels endpoints de Spotify.

L'objectiu de la pràctica és llistar els 10 majors èxits d'un artista. Podem observar que a [Artists](#) hi ha múltiples endpoints, un d'ells és "[Get an Artist's Top Tracks](#)". Per poder usar-lo cal el ID del artista. Endpoint: <https://api.spotify.com/v1/artists/{id}/top-tracks>.

Per identificar un artista podem consultar la documentació de Search, [Search for an item](#). Observem que:

- l'endpoint és: <https://api.spotify.com/v1/search>
- OAuth és required.
- q\*: indiquem el nom de l'artista que busquem, per exemple "queen".
- type\*: artists
- market: es



- OAuth token: cal introduir el token obtingut prèviament que ens permet identificar per fer la consulta. Atenció que no es mostra tot sencer, cal agafar-lo apropiadament de la URI de la pàgina.

Observem que ens mostra l'endpoint i la consulta curl que realitzarà:

```
GET https://api.spotify.com/v1/search
```

```
curl -X "GET" "https://api.spotify.com/v1/search?q=queen&type=artist" -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer BQAQMZeDNFwyP0w6oGcJwX_nTKqHKk63S3u2lzKbXjjYHsWZIHb-37gN9OR8c8aZD3bolPuym3JDnsfYEI_KaotySxcrVPLtUDKa47on_AoMr1sw23HcBxpVGrLU6VLHd2_5z-U sNQBdEo2umcnIS_6lmLWfhFZD2t0rJKcRfU5SxpuKw"
```

En clicar Try-it obtenim la resposta:

```
{
  "artists": {
    "href": "https://api.spotify.com/v1/search?query=queen&type=artist&market=ES&offset=0&limit=20",
    "items": [
      {
        "external_urls": {
          "spotify": "https://open.spotify.com/artist/1dfeR4HaWDbWqFHLkxsg1d"
        },
        "followers": {
          "href": null,
          "total": 24540960
        },
        "genres": [
          "glam rock",
          "rock"
        ],
        ...
      }
    ]
  }
}
```

Els access-tokens tenen un temps limitat d'ús, si intentem fer servir l'accés token passat aquest temps obtenim:

```
$ curl -X "GET" "https://api.spotify.com/v1/search?q=queen&type=artist&market=es" -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer BQCTPQaLcyt20EZGZeczw1sXO58bvHjltGgTETwwr4D45LiQnRbY1WVhGJkFDI_fpFRJXV0RX01O7rONySAtiJrlazcxYaBoV9WC17vqjyqzOeKcyaqsDvQTgLz2Dwd3U7KZm761LLUpuMuMcn1y9wIMVi0qXPs"
{
  "error": {
    "status": 401,
    "message": "The access token expired"
  }
}
```

Exemple de consulta manual de *the whaterboys*

```
$ curl -X "GET" "https://api.spotify.com/v1/search?q=the%20whaterboys&type=artist&market=es" -H "Accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer BQARnxKGjLOIEvC2soJ1bEgfEsyW4S5fNnaNmOYRD-f9BcwEhWyVQNytRUPMvMt02KKXCrrk4Q6h_FsowaGTscx-6nd3092hh7-fBylJy2mDh3nYtiAgn-5x3UgrRi86Rn-StmF7p1b"
```

QpW7n9Toxrd\_99pefbSIIGrQ3mWtrMTmfLsNHlw"

```
{
  "artists": {
    "href": "https://api.spotify.com/v1/search?query=the+waterboys&type=artist&market=ES&offset=0&limit=20",
    "items": [ {
      "external_urls": {
        "spotify": "https://open.spotify.com/artist/5TnuP42pw475UrjeabtwZ"
      },
      "followers": {
        "href": null,
        "total": 140936
      },
      "genres": [ "art rock", "celtic rock", "dance rock", "new wave pop", "rock", "scottish new wave", "scottish rock" ],
      "href": "https://api.spotify.com/v1/artists/5TnuP42pw475UrjeabtwZ",
      "id": "5TnuP42pw475UrjeabtwZ",
      "images": [ {
        "height": 640,
        "url": "https://i.scdn.co/image/99c2716875c5ff08b13d34ef3bc3771c51a0ac91",
        "width": 640
      }, {
        "height": 320,
        "url": "https://i.scdn.co/image/4c2d006f4396f27f3641cfbbc8debf211c4e492e",
        "width": 320
      }, {
        "height": 160,
        "url": "https://i.scdn.co/image/899224760e9231c54cec51b9d29093ff72f7bcd5",
        "width": 160
      } ],
      "name": "The Waterboys",
      "popularity": 58,
      "type": "artist",
      "uri": "spotify:artist:5TnuP42pw475UrjeabtwZ"
    } ],
    "limit": 20,
    "next": null,
    "offset": 0,
    "previous": null,
    "total": 3
  }
}
```

## IV) Relació entre sol·licitud i resposta

L'objectiu és generar una app amb un requadre que permeti a l'usuari introduir el nom d'un artista i es mostri a la pàgina web en un requadre el llistat d'artistes coincidents amb aquesta consulta. Si es selecciona un dels artistes es mostren les top 10 èxits d'aquest artista. Es pot cridar en cada una de les cançons i l'enllaç portarà a la reproducció d'Spotify.

La funcionalitat de l'aplicació és la següent:

- Una primera consulta (en clicar el botó inicial) permet connectar a l'API de Spotify i obtenir l'[access token](#).
- Una nova consulta ajax a l'enpoint [/search](#) permet obtenir la llista d'artites. Cal iterar i mostrar aquesta llista a la pàgina web. Es mostra le nom de l'artista però cal obtenir també el seu ID per poder-lo identificar a Spotify.
- Una tercera consulta ajax a Spotify al endpoint <https://api.spotify.com/v1/artists/{id}/top-tracks> passant com a argument el ID de l'artista permet obtenir l'array JSON amb els deu temes principals, el seu TOP 10 de cançons.

### Pràctica 3.1.: mostrar el reuadre de selecció i els botons

#### Reorganitzar el codi

Abans de procedir anem a reorganitzar el codi de index.html per fer-ho més modular, separant la part executable de javascript i els fulls d'estil.

1. Generar dins de *public* un directori *scripts* on dins hi posarem els scripts de javascript a executar. Concretament crearem un script *spotify.js* on hi posarem el codi javascript que hi ha al final de index.html.
2. Generar dins de *public* un directori *styles* on desarem els fulls d'estil que volem utilitzar a l'aplicació. Concretament generarem un fitxer de styles anomenat *spotify.css*.
3. Caldrà modificar index.html per tal de que incorpori i carregui apropiadament el full d'estils i el codi executable javascript.

### spotify.js

```
(function() {

  /**
   * Obtains parameters from the hash of the URL
   * @return Object
   */
  function getHashParams() {
    var hashParams = {};
    var e, r = /([^\&=]+)=?([^&]*)/g,
        q = window.location.hash.substring(1);
    while ( e = r.exec(q)) {
      hashParams[e[1]] = decodeURIComponent(e[2]);
    }
    return hashParams;
  }

  var userProfileSource = document.getElementById('user-profile-template').innerHTML,
      userProfileTemplate = Handlebars.compile(userProfileSource),
      userProfilePlaceholder = document.getElementById('user-profile');

  var oauthSource = document.getElementById('oauth-template').innerHTML,
      oauthTemplate = Handlebars.compile(oauthSource),
      oauthPlaceholder = document.getElementById('oauth');

  var params = getHashParams();

  var access_token = params.access_token,
      refresh_token = params.refresh_token,
      error = params.error;

  if (error) {
    alert('There was an error during the authentication');
  } else {
    if (access_token) {
      // render oauth info
      oauthPlaceholder.innerHTML = oauthTemplate({
        access_token: access_token,
        refresh_token: refresh_token
      });

      $.ajax({
        url: 'https://api.spotify.com/v1/me',
        headers: {
          'Authorization': 'Bearer ' + access_token
        },
        success: function(response) {
          userProfilePlaceholder.innerHTML = userProfileTemplate(response);

          $('#login').hide();
          $('#loggedin').show();
        }
      });
    } else {
      // render initial screen
      $('#login').show();
      $('#loggedin').hide();
    }
  }
})
```

```

document.getElementById('obtain-new-token').addEventListener('click', function() {
$.ajax({
  url: '/refresh_token',
  data: {
    'refresh_token': refresh_token
  }
}).done(function(data) {
  access_token = data.access_token;
  oauthPlaceholder.innerHTML = oauthTemplate({
    access_token: access_token,
    refresh_token: refresh_token
  });
});
}, false);
})();

```

## spotify.css

```

#login, #loggedin {
display: none;
}

.text-overflow {
overflow: hidden;
text-overflow: ellipsis;
white-space: nowrap;
width: 500px;
}

#oauth, #obtain-new-token {
display: none;
}

```

Configurar index.html per carregar els estils i el javascript:

```

<head>
  <title>Example of the Authorization Code flow with Spotify</title>
  <link rel="stylesheet" href="//netdna.bootstrapcdn.com/bootstrap/3.1.1/css/bootstrap.min.css">
  <link rel="stylesheet" type="text/css" href="styles/spotify.css">
</head>

<script src="//cdnjs.cloudflare.com/ajax/libs/handlebars.js/2.0.0-alpha.1/handlebars.min.js"></script>
<script src="http://code.jquery.com/jquery-1.10.1.min.js"></script>
<script type="application/javascript" src="scripts/spotify.js"></script>

```

Modificar index.html. per mostrar un requadre per indicar el nom d'un artista i dos botons, un per fer la recerca dels èxits top 10 de l'artista i l'altre per netejar la recerca.

Part de index.html on al div loggedin se li han afegit els elements indicats:

```

<div id="loggedin">
  <div id="user-profile">
  </div>
  <div class="input">
    <input id="name">
    <button id="button">¡Get your favourite artist's Top Tracks!</button>
    <button id="clear_results">Clear results</button>
  </div>

```

```

</div>
<dl id="resultats" class="dl-horizontal">
  <dt>Artists</dt>
  <dd id="artists" class="text-overflow"></dd>
  <dt>Top Tracks</dt>
  <dd id="top_tracks" class="text-overflow"></dd>
</dl>
<div id="oauth">
</div>
<button class="btn btn-default" id="obtain-new-token">Obtain new token using the refresh
token</button>
</div>

```

Podem observar que en executar de nou l'aplicació node.js la pàgina index.html mostra aquests elements:



## Pràctica 3.2.: mostrar els artistes resultat del /search

Passos a fer:

- Recordem que, abans, hem pogut comprovar que la petició que hem realitzat a l'endpoint '/search' s'ha executat correctament. Provem ara de fer la crida de nou amb l'"access\_token" que estem rebent per HTTP. El copiem i anem a 'Search for an item' dins de les References del tutorial de Spotify Developers. Li passem "queen" com a "artist" i "ES" com a "market" i enganxem el valor del nou 'access\_token' a OAuth Token. Si tot funciona bé, rebrem la resposta a la pantalla.
- Copiarem aquesta consulta (query) dins de 'spotify.js', que ens servirà de base per fer la query final, però la deixarem comentada perquè no s'executi. Nosaltres farem la petició per AJAX just on acaba l'"addEventListener" del 'refresh\_token'. Ens hem

de fixar que sempre estiguem dins de l'àmbit del primer ELSE.

- Per recollir el valor de l'endpoint, només haurem de substituir l'artista concret que li estem passant a la query per una variable que guardi el valor que l'usuari o usuària introdueixi a l'input 'name'. Prendrem d'exemple el model del 'refresh\_token' per afegir un nou 'addEventListener' a 'button' perquè la crida s'executi quan l'usuari o usuària faci clic.
- Com a últim pas abans de fer la crida, declararem la nova variable 'artistName' que recollirà el nom de l'artista que introdueixi l'usuari o usuària a 'Name', utilitzant la funció de jQuery '.val()'. Cal comprovar que si el nom conté algun espai en blanc, substitueixi aquest espai per un '%20', que és el valor que pren l'espai a HTTP. També utilitzarem la funció '.trim()' per eliminar qualsevol espai que pugui haver-hi per davant o per darrere del nom.
- Un cop tenim llesta la variable 'artistName', obrim un nou IF on comprovarem si 'access\_token' existeix, i si existeix, ara sí que farem la crida a '/search'.

### Crida per obtenir la informació d'usuari

Procediment:

- Si s'ha rebut correctament el access token es genera una crida **ajax** al endpoint <https://api.spotify.com/v1/me> per obtenir les dades de l'usuari i mostrar-les a la part superior dreta de la pàgina html.
- Observem que se li passa d'arguments la **url** amb l'endpoint on connectar, l'**access token** al header i una funció de callback **function(response)** que s'executa en rebre la resposta.
- La resposta es posa en un template userProfilePlaceholder que serveix per mostrar automàticament les dades de l'usuari Spotify.
- Si no ha anat bé la connexió i no s'ha rebut l'accés token es torna a mostrar la pàgina inicial de la connexió.

Part de la crida ajax per recollir la informació de l'usuari de Spotify

```
/si no hi ha cap error i existeix access_token procedim amb la primera crida
if (access_token) {
    // render oauth info // innecessari
    oauthPlaceholder.innerHTML = oauthTemplate({
        access_token: access_token,
        refresh_token: refresh_token
    });
    //fem la crida que recull la informació del usuari
    $.ajax({
        url: 'https://api.spotify.com/v1/me',
        headers: {
            'Authorization': 'Bearer ' + access_token
        },
    },
```

```

        success: function(response) {
            //si la petició funciona guardem la info de l'usuari a la plantilla que ens ve donada
            userProfilePlaceholder.innerHTML =
userProfileTemplate(response);
            $('#login').hide();
            $('#loggedin').show();
        }
    });
} else {
    // render initial screen
    $('#login').show();
    $('#loggedin').hide();
}
}

```

### Botó de netejar els resultats

El codi javascript continua amb la part que indica la funcionalitat del botó de netejar els resultats. observem que:

- És una funció anomenada *click* que s'associa a un element de l'html, concretament al element *#clear\_results*.
- L'acció que fa es defineix en la function() i consisteix en fer *empty* el contingut de *#top\_tracks* (on hi ha les deu cançons de l'artista) i *#artists* (on hi ha el llistat d'artistes coincidents amb el search).

```

//botó de netejar els resultats
$("#clear_results").click(function(){
    $('#top_tracks').empty();
    $('#artists').empty();
});

```

### Recollir el nom de l'artista

Recollir del reaquadre d'entrada de dades el nom de l'artista quan l'usuari faci click al botó de buscar. Descripció:

- S'assigna el eventListener a l'acció de fer *click* al botó *"button"* que executarà la funció function().
- Es netegen els resultats de l'artista anterior i dels top10 anteriors.
- S'assigna a la variable *artistName* el valor introduït (transformat els espais si és que en conté).

```

//recollim el valor de #name i fem la petició quan l'usuari fa click a button
/* curl -X "GET" "https://api.spotify.com/v1/search?q=queen&type=artist&market=ES" -H "Accept: application/json" -H
"Content-Type: application/json" -H "Authorization: Bearer
BQDg19qFTkh9p8DQW3GYodNJoPuU0Zv1tPZuFua1ZyVnzl6N3NsYAWWPNBHdbHzTVFGkmQKOqLbyEDJgGxhixR0zLS

```

```
YhKYpYF4mvBORZO0uCPwx0s9fM_JzoEpqtnR1K3mDi4w8iYT_zUYVQVuwGLB_p0zdXH8" */
```

```
document.getElementById('button').addEventListener('click', function() {  
    //netegem tots els resultats cada cop que fem una nova crida  
    $('#artists').empty();  
    $('#top_tracks').empty();  
  
    //recollim el valor del input  
    var artistName = $("#name").val();  
  
    // comprovem si hi ha espais  
    if ($("#name").val().indexOf(" ") != -1){  
        var replaceSpace = $.trim($("#name").val());  
        artistName = replaceSpace.replace(/ /g, "%20");  
    }  
}
```

### Consulta ajax al endpoint /search

Un cop tenim l'artista cal fer una nova query ajax al endpoint <https://api.spotify.com/v1/search> per obtenir el llistat d'artistes, un array JSON. Es recorre aquest array i es mostra a l'html. Descripció:

- Es realitza una crida *ajax*. Se li passa la uri del *endpoint* amb els paràmetres requerits per a la consulta: *q*, *type*, *market*. Els headers amb el *access\_token* i la funció a executar en cas de success.
- Genera un *log* a la consola-web amb l'objecte JSON rebut, el llistat d'artistes.
- Recorre un a un els elements amb *\$.each* i per a cada element obté el *id* i el *name*.
- Afegeix aquests valors a l'element *#artists* en forma de button de manera que es pugui clicar al nom de l'artista. Cada botó està identificat internament amb el *id* de l'artista i es mostra com a label el seu *name*.

```
if (access_token) {  
    //crida ajax a /search amb access_token  
    $.ajax({  
        url: 'https://api.spotify.com/v1/search?q=' + artistName + '&type=artist',  
        market: 'ES',  
        headers: {  
            'Authorization': 'Bearer ' + access_token  
        },  
        success: function(response) {  
            console.log(response);  
  
            //recorrem la resposta la resposta per poder mostrar-la per pantalla  
            var artistsArray= response.artists.items;  
            var $id;  
            $.each(artistsArray, function(key, value){  
                $id = artistsArray[key].id;  
                var $name = artistsArray[key].name;  
                var $nameDiv = $("<div class = 'name'>");  
            }  
        }  
    });  
}
```



```

        var $button = $("<button class='button' id=" + $id + ">");
        var $object = $("<div class='object'>");
        $nameDiv.append($name);
        $button.append($nameDiv);
        $object.append($button);
        $("#artists").append($object);
    });

```

Un cop en aquest punt podem provar de nou la app i observem que funciona el search per artista i es mostra la llista d'artistes coincidents:

Execució de la app amb el codi spotify.1.js

Logged in as edtasixm11

<b>Display name</b>	edtasixm11
<b>Id</b>	50x3mbjitr912jv59l70jksx
<b>Email</b>	edtasixm11@gmail.com
<b>Spotify URI</b>	<a href="https://open.spotify.com/user/50x3mbjitr912jv59l70jksx">https://open.spotify.com/user/50x3mbjitr912jv59l70jksx</a>
<b>Link</b>	<a href="https://api.spotify.com/v1/users/50x3mbjitr912jv59l70jksx">https://api.spotify.com/v1/users/50x3mbjitr912jv59l70jksx</a>
<b>Profile Image</b>	
<b>Country</b>	ES

queen

**Artists**

- Queen
- Queens of the Stone Age
- Beyoncé
- Queen Pen
- Queen Latifah
- Queensrÿche
- Queen Naija
- QUEEN BEE

### Pràctica 3.3.: mostrar els Top10 songs de l'artista seleccionat

Passos a fer:

- A la propera lliçó, procedirem amb la segona crida AJAX al nou endpoint 'top-tracks', aprofitant el valor de l'\$id' de l'artista que l'usuari o usuària seleccioni en aquest punt i donarem estils CSS a la nostra aplicació.

### Test de consulta interactiu de l'endpoint To10

Un cop disposem del id de l'artista podem provar manualment amb curl o a través de la pàgina de developers amb les Web-API [reference](#) obtenir els Top10 d'un artista. Anem a [Artists](#) i [Get an Artist's Top Tracks](#).

El endpoint a usar és: GET <https://api.spotify.com/v1/artists/{id}/top-tracks>

Podem veure una consulta amb curl de l'artista [Elvis Presley: 43ZHCT0cAZBISjO8DG9PnE](#)

```
curl -X "GET"
"https://api.spotify.com/v1/artists/43ZHCT0cAZBISjO8DG9PnE/top-tracks?country=es" -H
"Accept: application/json" -H "Content-Type: application/json" -H "Authorization: Bearer
BQCTPQaLcyt20EZGZeczw1sXO58bvHjltGgTEtwwr4D45LiQnRbY1WVhGJkFDI_fpFRJXV0RX01O7rONySAtiJrlazcxYaBo
V9WC17vqjyqzOeKcyasDvQTgLz2Dwd3U7KZm761LLUpuMuMcn1y9wIMVi0qXPs"
```

### Obtenir els Top10 d'un artista.

Continuant dins de la part del codi javascript de [success: function\(response\)](#) ara toca fer la segona crida ajax si es prem el botó del nom d'un artista. El botó porta associat el id del artista necessari per fer la consulta al endpoint <https://api.spotify.com/v1/artists/{id}/top-tracks>.

El procediment a seguir és el següent:

- S'associa la funció [function](#) al event click del element [button](#).
- Es realitza una crida jQuery [\\$.ajax](#) al [endpoint](#) que retorna els Top10 d'un artista.
  - Es passen com a arguments la [url](#) del endpoint, utilitzant [this.id](#) com a identificador de l'artista (el id del botó que s'ha premut).
  - S'utilitza el mètode [GET](#).
  - En els headers es passa [access-token](#).
  - Es passa com a argument la funció [function](#) a executar si la crida ajax ha funcionat correctament. La funció que iterarà i mostrarà els Top10.
- Amb [\\$.each](#) s'itera per a cada element de la resposta rebuda, un list de JSON.
- Dels ítems de què es compon cada un dels Top10 tracks interessa aobtenir el name i el link.
  - El name per identificar la cançó.
  - El link perquè és la url que permet reproduir directament a spotify la cançó.
- Aquests elements es poden en botons de nom name i de valor el link dins de l'element html [#top\\_tracks](#).

Fragment del codi de javascript de spotify.js

```
$('.button').click(function(){
    //segona crida ajax a /top-tracks amb access_token
    /* curl -X "GET" "https://api.spotify.com/v1/artists/0LcJLqbBmaGUft1e9Mm8HV/top-tracks?country=ES" -H "Accept: application/json" -H "Content-Type:
application/json" -H "Authorization: Bearer
BQDM__ysnOUIkuwsZlPNhEJJGxodHxPRya61n2xsxCbD73d7952f1srpMJ6j0lqn-WPHqr4DZpctCMT CZO80XeaPU7IOLZ2nVtnUC_XqS32s6eyjKW0B-hwfpN7BwE
9s60SylGlsCwZVJ6X2RtUbwIUC-1DgcUU"*/

    $.ajax({
        url: 'https://api.spotify.com/v1/artists/'+ this.id +'/top-tracks?country=ES',
        type:"GET",
```

```

headers: {
  'Authorization': 'Bearer ' + access_token,
  'Content-Type': 'application/json',
  'Accept': 'application/json'
},
success: function(result){
  console.log(result);
  $.each(result.tracks, function(key, value){
    var $top_track = value.name;
    var $link = $('<a/>').addClass("link").attr("href", value.uri);
    var $new_button = $('<button class='new_button'/>');
    var $new_object = $('<div class='new_object'/>');
    $new_button.append($top_track);
    $link.append($new_button);
    $new_object.append($link);
    $("#top_tracks").append($new_object);
  });
},
error: function(error){
  console.log(error)
}
});
});

```

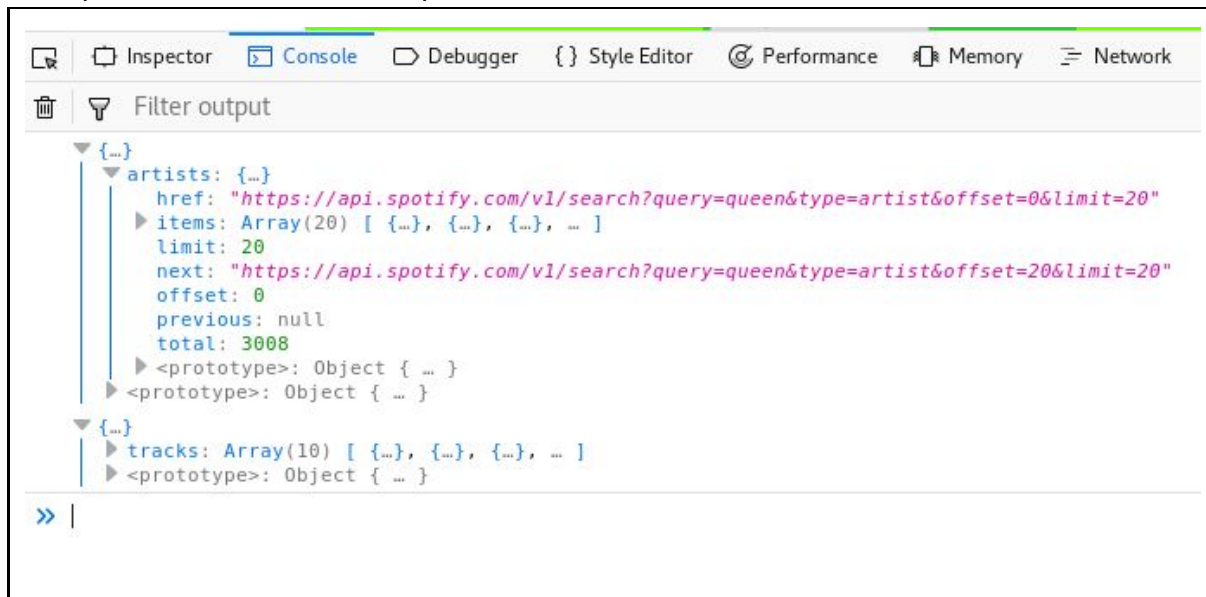
Un cop arribats a aquest punt podem observar com funciona l'aplicació node.js que hem estat construint. Si per exemple seleccionem l'artista queen observem al final del llistat les seves Top10 cançons:



Ara ja tenim l'aplicació funcionant, si volem podem fer les següents tasques:

- Observar des de la consola-web les dades JSON que s'estan transmetent
- Millorar la presentació de l'aplicació amb un full d'estils css més elaborat.

Exemple de la consola-web després de fer una recerca:



```

{...}
  artists: {...}
    href: "https://api.spotify.com/v1/search?query=queen&type=artist&offset=0&limit=20"
    items: Array(20) [ {...}, {...}, {...}, ... ]
    limit: 20
    next: "https://api.spotify.com/v1/search?query=queen&type=artist&offset=20&limit=20"
    offset: 0
    previous: null
    total: 3008
    <prototype>: Object { ... }
  <prototype>: Object { ... }
  tracks: Array(10) [ {...}, {...}, {...}, ... ]
  <prototype>: Object { ... }
  >> |

```

Podeu trobar al versió final de la pràctica al recurs:

- ❏ GitHub [edtasixm11](#) [api-rest](#)

---

## Docker: node.js

---

**<pendent> passar la app a un docker container de node.js**

Consultant la pàgina de [Docker Hub de Node.js](#) i les explicacions a GitHub de [docker-node](#) podeu construir un container Docker basat en la imatge Node.js que tingui la app desenvolupada en aquest apràctica.

---

## Vulnerability: CVE

---

En pujar tot la documentació al GitHub ha detectat una vulnerabilitat en el software instal·lat per node.js en fer la part d'actualització de dependències (de fet ja havia avisat en fer l'actualització).


Aprofitant que part dels projectes tracten de temes de vulnerabilitats és interessant que observeu la informació que mostra GitHub.

<https://github.com/advisories/GHSA-rq8g-5pc5-wrhr>

GitHub Advisory Database · CVE-2018-1000620

### The NPM package cryptiles version 4.1.1 and earlier conta...

**high severity** CVE-2018-1000620 published on Sep 11, 2018 • updated on Jul 3, 2019

Packages	Affected versions	Patched versions
 cryptiles (npm)	< 4.1.2	4.1.2

The NPM package cryptiles version 4.1.1 and earlier contains a CWE-331: Insufficient Entropy vulnerability in randomDigits() method that can result in An attacker is more likely to be able to brute force something that was supposed to be random.. This attack appear to be exploitable via Depends upon the calling application.. This vulnerability appears to have been fixed in 4.1.2.

# cryptiles

[Create automated security update](#)[Dismiss -](#)[Open](#) GitHub opened this alert 6 minutes ago

## ⚠ Dependabot cannot update to the required version

[View details about this error](#) or [learn more about automated security updates](#).

1 cryptiles vulnerability found in .../spotify/package-lock.json 6 minutes ago

### Remediation

Upgrade **cryptiles** to version **4.1.2** or later. For example:

```
"dependencies": {  
  "cryptiles": ">=4.1.2"  
}
```

Or...

```
"devDependencies": {  
  "cryptiles": ">=4.1.2"  
}
```

*Always verify the validity and compatibility of suggestions with your codebase.*

### Details

#### CVE-2018-1000620

**high severity**

**Vulnerable versions:** < 4.1.2

**Patched version:** 4.1.2

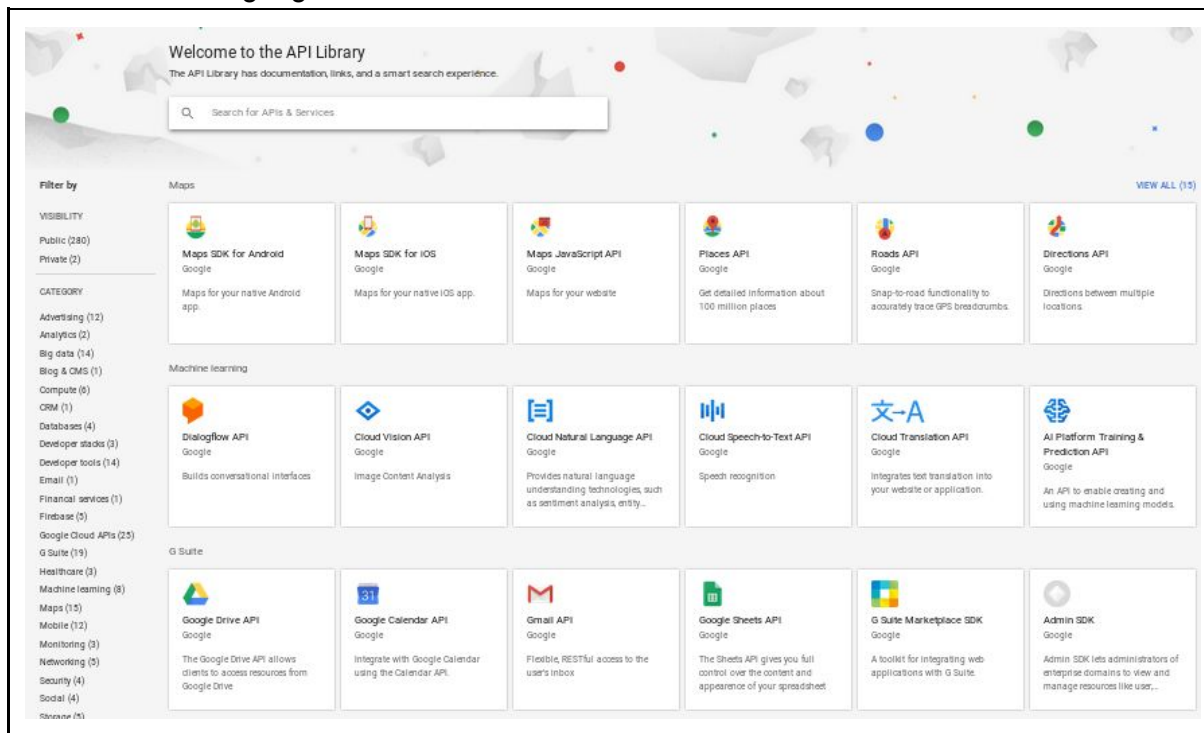
The NPM package cryptiles version 4.1.1 and earlier contains a CWE-331: Insufficient Entropy vulnerability in randomDigits() method that can result in An attacker is more likely to be able to brute force something that was supposed to be random.. This attack appear to be exploitable via Depends upon the calling application.. This vulnerability appears to have been fixed in 4.1.2.

# Google API

Google API Console: [Dashboard](#)


A [Library](#) hi ha les apis dels diversos productes de google

## Llistat de APIs de google



## Selecció de l'API de gmail





## Gmail API

Google

Flexible, RESTful access to the user's inbox

[ENABLE](#)
[TRY THIS API](#)

**Type**

[APIs & services](#)

**Last updated**

12/10/19, 1:34 AM

**Category**

[Email](#)

[G Suite](#)

**Service name**

gmail.googleapis.com

**Overview**

The Gmail API lets you view and manage Gmail mailbox data like threads, messages, and labels.

**About Google**

Google's mission is to organize the world's information and make it universally accessible and useful. Through products and platforms like Search, Maps, Gmail, Android, Google Play, Chrome and YouTube, Google plays a meaningful role in the daily lives of billions of people.

**Tutorials and documentation**

[Learn more](#)

**Maintenance & support**


[Learn more](#)

**Terms of service**

By using this product you agree to the terms and conditions of the following license(s): [Google APIs Terms of Service](#)

Observem els botons:

- Enable
- Try this API. En seleccionar-la es mostrarà la descripció d'el'API, les funcions, exemples i un pop-up per poder practicar amb l'API en real contra comptes de gmail.



Gmail > API

English
E

[Home](#)
[Guides](#)
[Reference](#)
[Samples](#)
[Support](#)
[Send feedback](#)

REST Reference

Resource Summary

- Users
  - Users.drafts
  - Users.history
  - Users.labels
    - Overview
    - create**
    - delete
    - get
    - list
    - patch
    - update
  - Users.messages
  - Users.messages.attachments
  - Users.settings
    - Users.settings.delegates
    - Users.settings.filters
    - Users.settings.forwardingAddresses
    - Users.settings.sendAs
    - Users.settings.sendAs.smimeInfo

Home > Products > G Suite Developer > Gmail > API > Reference

## Users.labels: create

★ **Note:** Requires [authorization](#).

Creates a new label. [Try it now](#) or [see an example](#).

**Request**

HTTP request

```
POST https://www.googleapis.com/gmail/v1/users/userId/labels
```

**Parameters**

### Try this API

Call this method on live data to see the API request and response. Need help with the API Explorer? Check the [support page](#).

**Request parameters**

userId  
ecanet

[Show standard parameters](#)

**Request body**

```
{
  "labelListVisibility": "show_label",
  "messageListVisibility": "show_label",
  "name": ""
}
```

[Select an underlined section to see more details.](#)


You have invalid or missing required parameter values shown. If you override anyway the response

## Crear un projecte

Per crear un projecte:

- Des de API dashboard
- Create Project
- Indicar el nom del projecte, per exemple projecte-01
- Create

## New Project



You have 11 projects remaining in your quota. Request an increase or delete projects. [Learn more](#)

[MANAGE QUOTAS](#)

Project name \*

?

Project ID: projecte-01-272011. It cannot be changed later. [EDIT](#)

Organization \*

▼ ?

Select an organization to attach it to a project. This selection can't be changed later.

Location \*

BROWSE

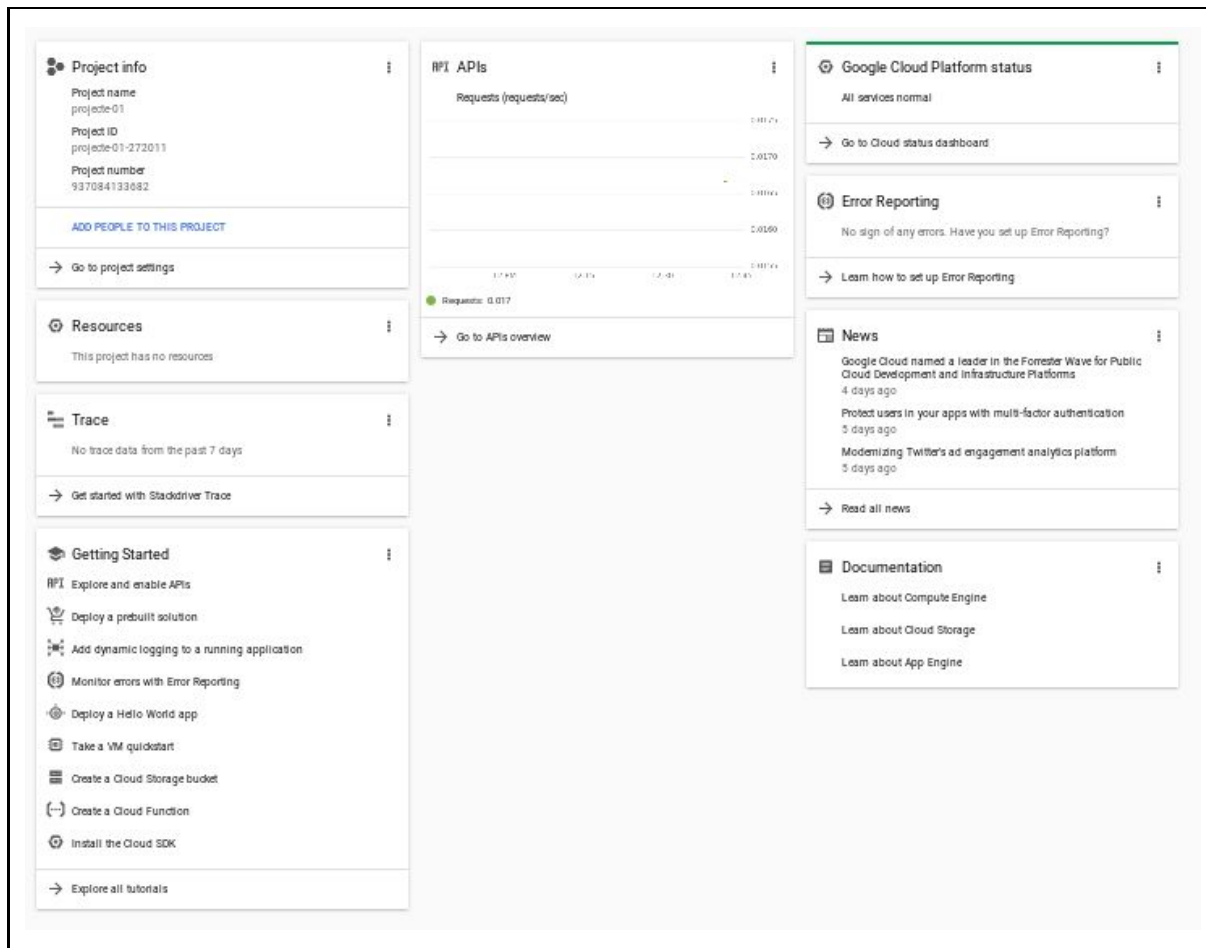
Parent organization or folder

CREATE
CANCEL

### Des de Google APIs

- A la part superior del dashboard veiem el nom del projecte amb el que estem treballant dins de l'organització on estem treballant.
- Al dashboar del projecte, a Getting Started, hi ha l'opció **explore & enable APIs**.
- Llavors cal clicar a "+enable APIS and services" o anar al llistat de les APIs de google, seleccionar la API que es vol activar, per exemple la de gmail.
- Seleccionar **enable** en la pantalla de gmail.
- Apareixerà la pantalla Overview, on veiem Details, Tutorrials & documentation, traffic by response code, i el botó de **create credentials** perquè calen credencials per accedir a aquesta api.

Pantalla per seleccionar que volem activar APIs o serveis:



Pantalla un cop activada la API de gmail:

