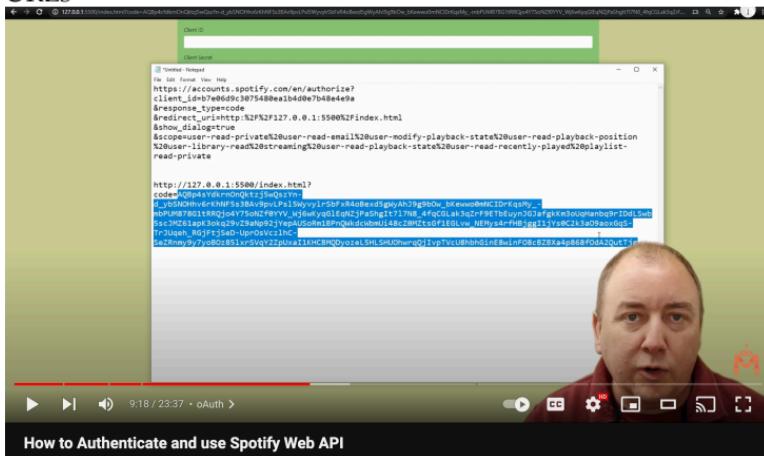


Okay I need help

Watching tutorial on how to auth into Spotify api

Hes going through the docs in order to avoid having to use libraries, interact with the api directly.

One of the first things I learned here is that you can use the name of a file, a route, in your local host as redirect uri. This guy has his as local host 3000/index.html the index.html part was new to me and is perhaps the reason why mine has been failing. I get from ChatGPT the /callback in both heroku and on streamlit. A possible solution is to when deploying have it go back to the home page, but now authenticated. How does the app know it's been authenticated though? Ah that's it with the two below URLs

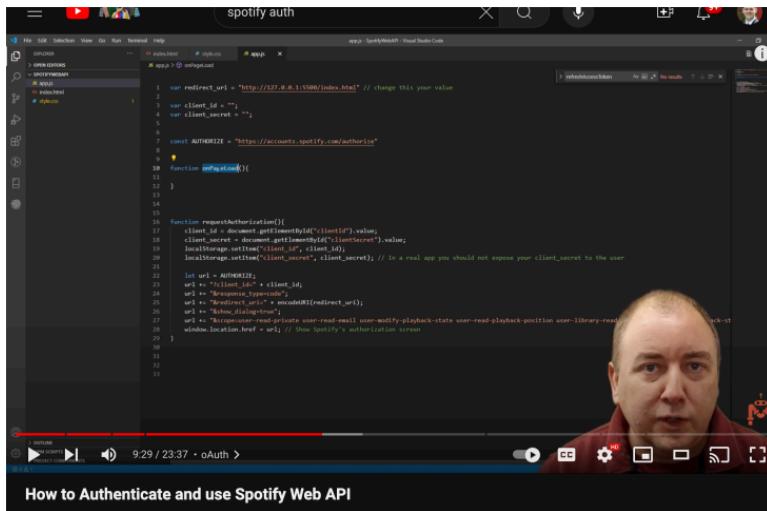


How to Authenticate and use Spotify Web API

Meaning that the first one is what you send with a redirect uri encoded in html

Then the successful response begins with your redirect uri and adds code= along with an auth code. In streamlit via the internet I saw folks had a problem with streamlit not being able to handle this redirect successfully. With hacks showing ways users are prompted to paste in their own auth code (ie redirect themselves by repasting a link). This makes me think of those pages that pop up and say you should be authenticated and automatically redirected. If that doesn't happen please click this link. Maybe that is a way around it.

Btw this is the code — vanilla javascript used to hit the Spotify api and auth without other libraries.



How to Authenticate and use Spotify Web API

Step 2. Have your application request refresh and access tokens; Spotify returns access and refresh tokens

2. Have your application request refresh and access tokens; Spotify returns access and refresh tokens

When the authorization code has been received, you will need to exchange it with an access token by making a POST request to the Spotify Accounts service, this time to its /api/token endpoint: **POST https://accounts.spotify.com/api/token**. The body of this POST request must contain the following parameters encoded in `'application/x-www-form-urlencoded'` as defined in the OAuth 2.0 specification:

REQUEST BODY PARAMETER	VALUE
<code>grant_type</code>	<i>Required.</i> As defined in the OAuth 2.0 specification, this field must contain the value <code>"authorization_code"</code> .
<code>code</code>	<i>Required.</i> The authorization code returned from the initial request to the Account <code>/authorize endpoint</code> .
<code>redirect_uri</code>	<i>Required.</i> This parameter is used for validation only (there is no actual redirection). The value of this parameter must exactly match the value of <code>redirect_uri</code> supplied when requesting the authorization code.
HEADER PARAMETER	VALUE
<code>Authorization</code>	<i>Required.</i> Base 64 encoded string that contains the client ID and client secret key. The field must have the format: <code>Authorization: Basic <base64 encoded client_id:client_secret></code>

This below is the updated code for completing up to step 2.

File Edit Selection View Go Run Terminal Help

index.html # style.css app.js

```

index.html # style.css app.js
  app.js fetchAccessToken
  var client_secret = "";
  const AUTHORIZE = "https://accounts.spotify.com/authorize"
  ...
  function onPageLoad(){
    if ( window.location.search.length > 0 ){
      handleRedirect();
    }
  }

  function handleRedirect(){
    let code = getCode();
    fetchAccessToken( code );
  }

  function fetchAccessToken( code ){
    let body = "grant_type=authorization_code";
    body += "&code=" + code;
    body += "&redirect_uri=" + encodeURI(redirect_uri);
    body += "&client_id=" + client_id;
    body += "&client_secret=" + client_secret;
    callAuthorizationApi( body );
  }

  function getCode(){
    let code = null;
    const queryString = window.location.search;
    if ( queryString.length > 0 ){
      const urlParams = new URLSearchParams(queryString);
      code = urlParams.get('code')
    }
    return code;
  }

  function requestAuthorization(){
    client_id = document.getElementById("client_id").value;
    client_secret = document.getElementById("client_secret").value;
    localStorage.setItem("client_id", client_id);
    localStorage.setItem("client_secret", client_secret); // In a real app you should not expose your
    let url = AUTHORIZE;
    url += "?client_id=" + client_id;
  }

```

File Edit Selection View Go Run Terminal Help

index.html # style.css app.js

```

index.html # style.css app.js
  app.js handleAuthorizationResponse
  ...
  function fetchAccessToken( code ){
    let body = "grant_type=authorization_code";
    body += "&code=" + code;
    body += "&redirect_uri=" + encodeURI(redirect_uri);
    body += "&client_id=" + client_id;
    body += "&client_secret=" + client_secret;
    callAuthorizationApi( body );
  }

  function callAuthorizationApi( body ){
    let xhr = new XMLHttpRequest();
    xhr.open("POST", TOKEN, true);
    xhr.setRequestHeader("Content-Type", 'application/x-www-form-urlencoded');
    xhr.setRequestHeader('Authorization', 'Basic ' + btoa(client_id + ":" + client_secret));
    xhr.send( body );
    xhr.onload = handleAuthorizationResponse;
  }

  function handleAuthorizationResponse(){
    if ( this.status == 200 ){
      var data = JSON.parse(this.responseText);
      console.log(data);
      var data = JSON.parse(this.responseText);
      if ( data.access_token != undefined ){
        access_token = data.access_token;
        localStorage.setItem("access_token", access_token);
      }
      if ( data.refresh_token != undefined ){
        refresh_token = data.refresh_token;
        localStorage.setItem("refresh_token", refresh_token);
      }
      onPageLoad();
    } else {
      console.log(this.responseText);
      alert(this.responseText);
    }
  }

  function getCode(){
    let code = null;
    const queryString = window.location.search;
    if ( queryString.length > 0 ){

```

```

29 }
30
31 function callAuthorizationApi(body){
32   let xhr = new XMLHttpRequest();
33   xhr.open("POST", TOKEN, true);
34   xhr.setRequestHeader('Content-Type', 'application/x-www-form-urlencoded');
35   xhr.setRequestHeader('Authorization', 'Basic ' + btoa(client_id + ":" + client_secret));
36   xhr.send(body);
37   xhr.onload = handleAuthorizationResponse;
38 }
39
40 function handleAuthorizationResponse(){
41   if (this.status == 200){
42     var data = JSON.parse(this.responseText);
43     console.log(data);
44     var data = JSON.parse(this.responseText);
45     if (data.access_token != undefined){
46       access_token = data.access_token;
47       localStorage.setItem("access_token", access_token);
48     }
49     if (data.refresh_token != undefined){
50       refresh_token = data.refresh_token;
51       localStorage.setItem("refresh_token", refresh_token);
52     }
53     onPageLoad();
54   } else {
55     console.log(this.responseText);
56     alert(this.responseText);
57   }
58 }
59
60 function getCode(){
61   let code = null;
62   const queryString = window.location.search;
63   if (queryString.length > 0){
64     const urlParams = new URLSearchParams(queryString);
65     code = urlParams.get('code')
66   }
67   return code;
68 }
69
70 function requestAuthorization()

```

At 13:29 he gets the invalid client id bug. Same one it seems I am wrangling with. He says that those have not been properly saved in local storage and don't reauth on redirect. I will get the actual full transcript and post below.

He attempts to fix below by adding local storage information on page load.

```

10
11 function onPageLoad(){
12   client_id = localStorage.getItem("client_id");
13   client_secret = localStorage.getItem("client_secret");
14
15   if (window.location.search.length > 0 ){
16     handleRedirect();
17   }
18 }

```

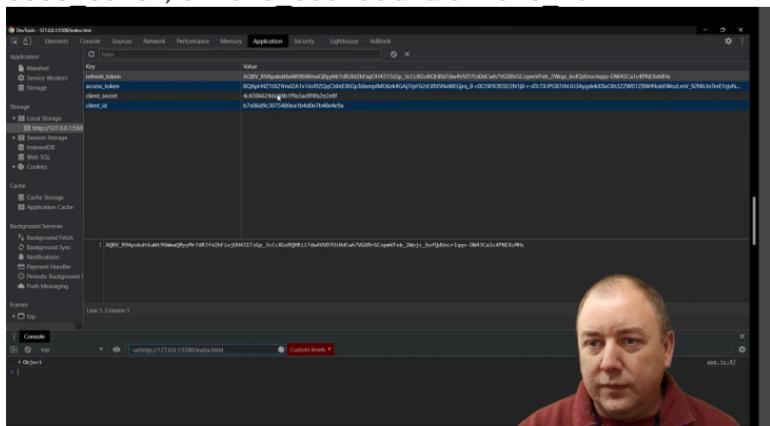
Also here adds a function that clears redirect each time

```

18 }
19
20 function handleRedirect(){
21   let code = getCode();
22   fetchAccessToken( code );
23   window.history.pushState("", "", redirect_uri); // remove param from url
24 }
25

```

Wow that is smart he's using the dev tools in google chrome in local storage to see that he indeed has refresh_token, access token, client secret and client id.



Now he's using those new tokens to make an api call. Here he is calling to see the devices used. However we may use other APIs.

A screenshot of a browser developer console. The URL is https://accounts.spotify.com/api/v1/devices. The code in the console is as follows:

```
client_id = document.getElementById("clientId").value;
client_secret = document.getElementById("clientSecret").value;
localStorage.setItem("client_id", client_id);
localStorage.setItem("client_secret", client_secret); // In a real app you should not expose your client_secret to the user
let url = AUTHORITY;
url += "?client_id=" + client_id;
url += "&response_type=code";
url += "&scope=device.read";
url += "&show_dialog=true";
url += "&scope=user-read-private%20user-read-playback-state%20user-read-playback-position%20user-library-read%20stream";
window.location.href = url; // Show Spotify's authorization screen
```

The code continues with more functions and logic for handling the response from the API. On the right side of the screen, there is a portrait of a man with short brown hair and a red shirt.

The screenshot shows a browser developer tools Network tab with the following details:

- Request Headers:**
 - authority: api.spotify.com
 - method: GET
 - path: /v1/me/player/devices
 - scheme: https
 - accept: */*
 - accept-encoding: gzip, deflate, br
 - accept-language: en-US,en;q=0.9
- authorization: Bearer BQAjH4Z1xJYrnd2A1v1J0IIf2Qc1shTExtp7d1smP1D06z34Gaj7rpF62rEX05h0d6Gfrq_B_c051F10159228f3B_r-dTcTXPG7r4kU3aygdkxx01h32z012zvW_92h3nTeMjYjVdkscsT2L67bvYE0DXVSQDKFrp4Af97K5nMsdA
- cache-control: no-cache
- content-type: application/json
- origin: http://127.0.0.1:5500
- pragma: no-cache
- referer: http://127.0.0.1:5500/
- sec-fetch-dest: empty
- sec-fetch-mode: cors
- sec-fetch-site: cross-site

user-agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.104 Safari/537.36

sources | **Fu** 1:5500/index.html Custom levels *

Refresh access tokens

```
84     }
85     return code;
86   }

87

88   function requestAuthorization(){
89     client_id = document.getElementById("clientId").value;
90     client_secret = document.getElementById("clientSecret").value;
91     localStorage.setItem("client_id", client_id);
92     localStorage.setItem("client_secret", client_secret); // In a real app you shou
93
94     let url = AUTHORIZE;
95     url += "?client_id=" + client_id;
96     url += "&response_type=code";
97     url += "&redirect_uri=" + encodeURI(redirect_uri);
98     url += "&show_dialog=true";
99     url += "&scope=user-read-private user-read-email user-modify-playback-state user
100   }
101
102
103
104   function refreshDevices(){
105     callApi( "GET", DEVICES, null, handleDevicesResponse );
106   }
107
108   function handleDevicesResponse(){
109     if ( this.status == 200 ){
110       var data = JSON.parse(this.responseText);
111       console.log(data);
112       removeAllItems( "devices" );
113       data.devices.forEach(item => addDevice(item));
114     }
115     else if ( this.status == 401 ){
116       refreshAccessToken()
117     }
118     else {
119       console.log(this.responseText);
120       alert(this.responseText);
121     }
122   }
123
124
125
126   function addDevice(item){
```

```
election View Go Run Terminal Help
... index.html # style.css app.js x
DOS
TB API
JML
S
PIIS
COMPONENTS
app.js > @ callAuthorizationApi
34     let code = getCode();
35     fetchAccessToken( code );
36     window.history.pushState("", "", redirect_uri); // remove param from url
37   }
38
39   function fetchAccessToken( code ){
40     let body = "grant_type=authorization_code";
41     body += "&code=" + code;
42     body += "&redirect_uri=" + encodeURI(redirect_uri);
43     body += "&client_id=" + client_id;
44     body += "&client_secret=" + client_secret;
45     callAuthorizationApi(body);
46   }
47
48 function callAuthorizationApi(body){[redacted]
49   let xhr = new XMLHttpRequest();
50   xhr.open("POST", TOKEN, true);
51   xhr.setRequestHeader("Content-Type", 'application/x-www-form-urlencoded');
52   xhr.setRequestHeader("Authorization", 'Basic ' + btoa(client_id + ":" + client_secret));
53   xhr.send(body);
54   xhr.onload = handleAuthorizationResponse;
55 }
56
57 function refreshAccessToken(){
58   refresh_token = localStorage.getItem("refresh_token");
59   let body = "grant_type=refresh_token";
60   body += "&refresh_token=" + refresh_token;
61   body += "&client_id=" + client_id;
62   callAuthorizationApi(body);
63 }
64
65 function handleAuthorizationResponse(){
66   if ( this.status == 200 ){
67     var data = JSON.parse(this.responseText);
68     console.log(data);
69     var data = JSON.parse(this.responseText);
70     if ( data.access_token != undefined ){
71       access_token = data.access_token;
72       localStorage.setItem("access_token", access_token);
73     }
74     if ( data.refresh_token != undefined ){
75       refresh_token = data.refresh_token;
76       localStorage.setItem("refresh_token", refresh_token);
77     }
78   }
}
```