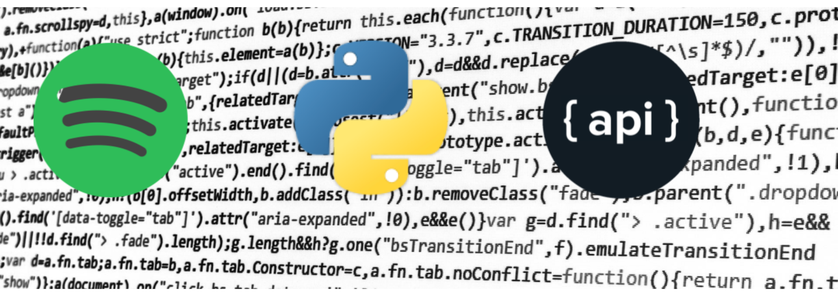


50 Lines of Python: Spotify, APIs & OAuth

Published on January 24, 2019

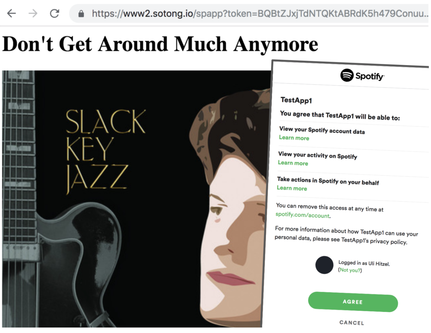
Uli Hitzel
Executive Geek

The anatomy of an app that knows what song you're listening to, right now. How APIs and OAuth play key roles in here. And finally, if you're keen, how to build that in Python.

Imagine you open multiple tabs in your web browser, one for each of your favourite websites. All of them are made by different companies, teams, and people with their respective choices of tools, frameworks and programming languages. To you as a user, these details are mostly irrelevant, as long as the sites produce HTML, CSS & JavaScript so your browser would render them into pages you can use. The **API Economy** works in a very similar fashion where all the various building blocks communicate with each other using **open standards** — and we get to enjoy Connected Life, Smart Homes, and Virtual Assistants.

"What's the weather in Singapore?" Siri uses **APIs** to retrieve the current readings from a system on the web that has that information. *"Alright, Fitbit, you may share my stats with my friends on Facebook!"* —that's Fitbit using the **OAuth** mechanism to get your permission for doing this on your behalf.

Let's look at how can we can wire some of these bits together in a practical example! Ideally, you're a **Spotify** user so you can try this exercise yourself. Open the app on your phone, on your tablet, TV, or laptop and play a song of your choice. Then, open my web application in a browser window on a different device. Or on the same device, if you absolutely must. You'll sign in with your Spotify credentials, click 'Agree', and should see a beautifully designed website (I'm really not much of a frontend guy) that shows you the song you're currently listening to.

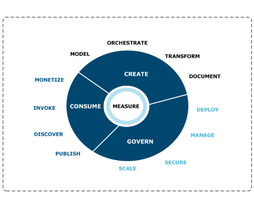


Not a big deal, in 2019 digital citizens are pretty used to hyper-connected services where you sign in with your Facebook, Google, or Twitter credentials and the app knows your name, list of friends and all those things. But how does that work under the hood?

Once you've selected a song in Spotify (**1**), the service will start streaming music to your device (**2**) - unless, of course, it doesn't. The web application you're opening after that (**3**) will quickly realize it doesn't know who you are and redirect you to a Spotify page where you approve access to your account, this is **OAuth** in action (**4**). After that, the web app is able to perform API requests to the Spotify service to query information or trigger actions, using your account context (**5**). There is detailed technical documentation on OAuth around the web, let's map some of those confusing terms you typically find in there to our example:

- **Resource Owner**: that's you. It's your Spotify account, your playlists, history and favourites.
- **Client Application**: my little Python app that you authorize to do the magic. It will talk to the **Authorization Server** (Spotify Account Service), which will confirm that you're ok with this.
- The **Scopes** are a list of actions you'll let me do on your behalf. In this case, getting information on your email address and the track that is currently playing
- And this very information is available on the **Resource Server**, the Spotify backend, which my web app queries using authenticated REST API calls.

I've built this web app with Python [**1**] and hosted it on PythonAnywhere —but that really doesn't matter too much! In an API-connected, distributed world you get to choose the best tool for the job, so gone are the days where everybody in your company had to be a Java person or a .NET developer.



What does matter is having a consistent approach across the organization when it comes to API design, governance, lifecycle management, and access control. You may have 200 APIs across the company, but there should be exactly one way to discover them, find out how they work and request access. Let's have a conversation to find out how you can get there!

If you enjoyed reading, please have a look at some of my other articles on LinedIn.

[**1**] Here's my code. It's a 15 min hack and not the most beautiful code