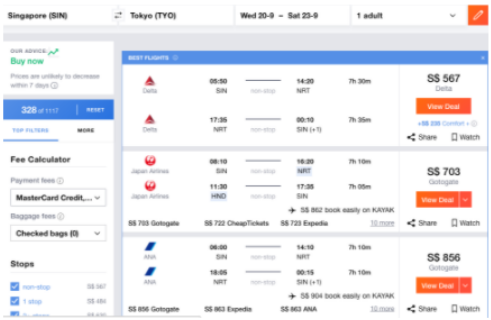
Image: ddouk / Pixabay

On APIs and tools that don't have one.

Published on September 9, 2017

What's an API, anyway? And how can we integrate components that only give you slick browser interfaces, but nothing else? Let's take an example and teach Siri how to change your WiFi password at home.

Whether you call it Machine-to-Machine (M2M), Internet of Things (IoT), Distributed Systems, **Big Software** – "a complex assembly of many software components sourced from different vendors, running on multiple distributed machines, providing to its user the impression of a single system" [1] - the key is an open, standardized way for all these components to communicate with each other. Of course, we're talking about: Application Programming Interfaces, or **APIs** for short.



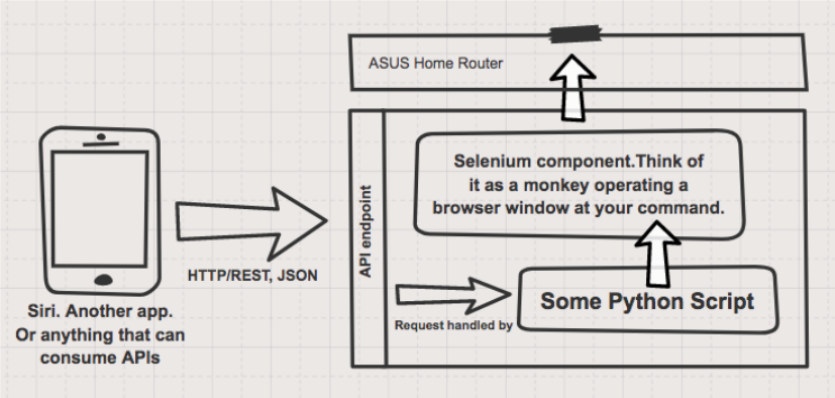
Fancy a trip to Tokyo? You can check the prices & schedules at the Singapore Airlines website. Then Delta. Japan Airlines. Most likely you will use Kayak or Expedia instead, where the tool does all these things for you. **Without APIs, this is barely possible.**

The concept of APIs turned most of the world's computing into a planet-sized operating system, where objects might be taken from Google, processed in Amazon AWS, enriched with data from Wikipedia, then stored on Dropbox. Today, companies can pick up the components from the **API enabled assembly line** [2] and quickly build up services like Uber, Airbnb, and Expedia.

However, you will also have seen that there's an extensive list of tools and services that insist you use a graphical user interface (GUI) in order to control it. Your legacy system built in-house and even the database is not really accessible. A web based expense management system [3]. Your home automation system. How do we *API-fy* those?

Let's take an example: unlike most people, I change the WiFi password of my router at

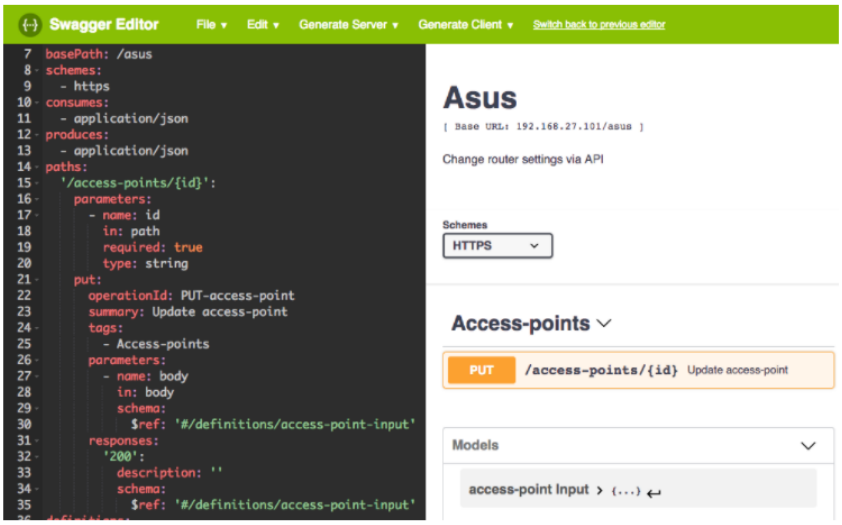
home frequently. And there is more than one access point at the house. I would like to find an automated way to do this, and telnet access doesn't do the trick [4]. So we are looking for a way to **have a machine operate graphical user interfaces**. Back then when I started working for Yahoo! websites were mostly static, backend-driven and easy to crawl and 'scrape'. Soon this would change though, and with HTML5, JavaScript, the various frameworks, and external services you'll need a fully featured browser to operate them. Lucky - we have Selenium for browser automation! My solution to create an API for my Asus router at home could look like this:



Let's outline the steps needed to make it happen! I'll agree that this is the point where my post gets a lot more 'technical'.

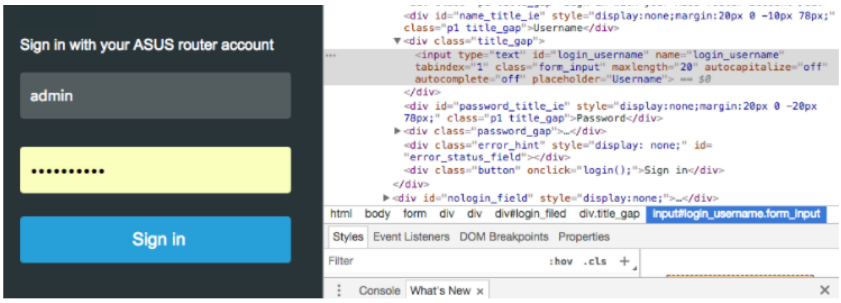
1 - Define your API

This is a very simple API with one method and two parameters, but it's a good idea to make it a habit: before doing any tinkering, think about the outcome you want to have, think of your consumer. Let's use Swagger and create a specification of the service that defines what it does, how we interact with it and what the data model is.



2 - Analyze the GUI we're looking to control

Selenium emulates a fully featured browser you can remote-control, and there are multiple ways to tell the system where and how to find the components of the website it manipulates. To do the steps manually, I have to login into the ASUS web tool with username and password, then navigate to the router settings, find the respective fields for access point name and WiFi password, change the details, press 'Apply' and then log out. The developer tools of the Chrome browser are a great help to inspect the elements and note down the IDs:



3 - Teach the monkey how to do it

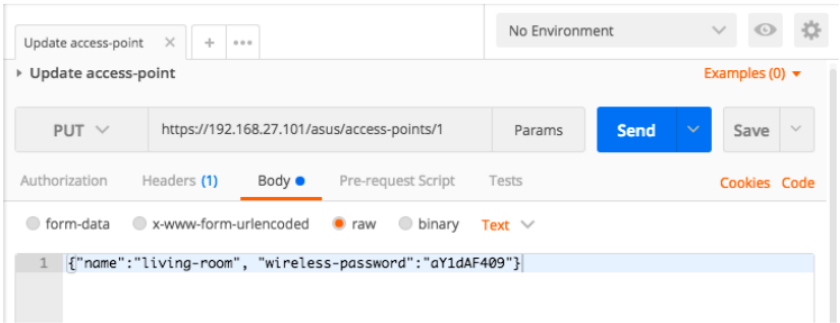
Ok, so we will need Selenium, the browser automation engine. With Docker on your machine, it takes exactly one command to download and run:

docker run -it -p 4444:4444 u1ih/selenium

Now we can use the Python bindings to give Selenium detailed instructions how the same task can be achieved in an automated fashion. Of course, you could use JavaScript, Ruby, or any other popular language as well. Here is what the first step, logging into the router could look like in Python:

```
router-api.py
import time
from selenium import webdriver
from selenium.webdriver.common.desired_capabilities import DesiredCapabilities
driver = webdriver.Remote(command_executor='http://127.0.0.1:4444/wd/hub',
desired_capabilities=DesiredCapabilities.CHROME)
driver.get('http://asus/Main_Login.asp');
time.sleep(5)
login_box = driver.find_element_by_id('username')
login_box.send_keys('admin')
login_box = driver.find_element_by_id('password')
login_box.send_keys('****************')
login_box.submit()
time.sleep(5)
```

Add all the other steps, a WSGI wrapper, run the service and we're now able to consume the new API! Let's import it in Postman and send a request:



From here on, we're flexible - any application or system that understands API will be able to interact with my router device. Now, imagine a company using hundreds of services, using Big Software with an increasing number of internal and external APIs. **Data is the new oil**, how do they secure it, control access & analyze the usage so they can monetize it? Let's talk about professional API management solutions!

[1] I've seen several identical descriptions of the term "Big Software" but couldn't find a definitive source for this. It's time to create a Wikipedia entry.

[2] Discovered that analogy on Stackshare

[3] Not every website likes automated use, please check the Terms of Service before doing so.

[4] My router actually offes telnet access into a busybox Linux style environment. It should be a lot easier to create a script that performs commands via SSH to change settings. However, there's no documentation available.

9 Likes

1 Comment

Bruce Dargus 2mo ···
Management Consultant, Enterprise Architect

Nice Uli you're on the right track I previously heard **Mark Shuttleworth** talk about Big Software and it's legit