Making of CSS Flags

03 Oct 2016

If you're following me on Twitter, you might have seen that I released a crazy project called CSS Flags. People often ask me why I did such a thing. This blog post is an attempt at explaining the thought process that motivated me to build it.

Trip to Tours

Two years ago I was on holidays and visited the city of Tours, in France. It's a small medieval city where you can find the museum of "Compagnonnage". I had no idea what this meant before visiting it, and learned everything about it during my visit.

Companions were craftsmen regrouped in guilds. They could be blacksmiths, carpenters, shoemakers, painters, etc. They regrouped and shared their knowledge between them. Being recognized as a Companion was a mark of quality in your work.

They followed a strict hierarchy of disciples and masters. More importantly, they had a specific relationship with work. They wanted work to stay nothing more than a tool. They refused to let work drive their lives and wanted to stay the masters of their tools, and never be enslaved by it.

But when visiting the museum, what struck me the most was all their internal rules that they followed 200 or 300 years ago. They strove to teach their craft to the new comers, to never compromise on quality, to always search for a better way to do something. They were passionate and knew that what they were working on was a small piece of something much bigger.

And all this resonated with me. It felt a lot like what I experience everyday as a software developer, with the principles of Software Craftsmanship and the Egoless Programmer. Reading all those rules on old parchment and realizing they still apply today was a surprising experience.

The road to the Chef d'œuvre

I felt close to those people, but there was one thing that they were doing back in the days that we do not do today, though. One important part of the initiation process, the step that makes you from disciple to master: the "Chef d'œuvre", or masterpiece.

Every Companion has to build one single piece of fine craft to become a master. They spent an awful lot of time on it, between 10 to 15 years. Masterpieces were miniature version of a bigger construction, where they had to use every technique of their craft, and then more.

only learn each technique independently, but also the potential synergies between them and limitations of each. By combining different techniques, they will even discover new ways to build and see issues through new perspectives.

What started as a learning sandbox evolved as an experimentation that pushed the boundaries of the craft. The whole masterpiece is a metaphor for the growth of the craftsman, from disciple to master. Once their masterpiece has been validated by the previous masters, the disciples becomes masters themselves.

What about doing the same?

I strongly encourage you to go to Tours and visit the museum. There are hundreds of masterpieces from different crafts, each more impressive than the last. Even if you don't speak French, the beauty of it is worth it.

After leaving the museum I thought about building my own masterpiece (in all modesty of course). After all, if it worked for them, considering everything we had in common, it should work for me too.

I didn't want to spend 10 to 15 years on it, though. I also have no skills with wood, so I decided to do it in CSS. I think it is a powerful language, but often underestimated. It's also a language I know well but that evolves so quickly I rarely have the chance to try the latest shiny features because I need to keep backward compatibility with older browsers.

The perspective of having my own miniature sandbox where I could try anything I wanted was appealing.

The challenge I set myself was to reproduce all the flags of the world in pure CSS. But because even that sounded too easy, I added the rule that I should use nothing more than one `div` per flag. Everything should be in CSS, nothing in HTML except for the div.

Learning vexillology

Which in turn forced me to learn a new word: Vexillology. Vexillology is the study of flags. Some people are passionate about music and can talk about it for hours. Well, some people are passionate about flags and can talk about it for hours.

I discovered a whole new world with it's own vocabulary about colors or shapes. I discovered that each flag has its own history. By reading the history of a flag, you learn about the history of the country behind it.

This part was not a bonus, but core to what I was trying to achieve. CSS was a mean to an end, but what I was building were flags. I needed to understand my subject if I was to build it correctly.

Code available on GitHub

was to get started.

The gritty details

That's how I got the idea and the motivation for this project. By starting with the easy bits first it gave me confidence that I could do it. Going progressively to more and more complex flags I was faced by ne problem at a time. The solutions I found in my first flags I could re-use and combine in the latest one.

I'll write a more complete (and CSS-focused) list of tips in an upcoming article.

#css

How to spot a bullshit hackathon

14 Sep 2016

Being co-organizer of a meetup group, I often receive messages from companies asking us to promote their hackathon or conference. We always refuse to do such advertisement, unless we've already attended their events and appreciated it.

Last week, we received a mail about what I call a "bullshit hackathon". I've translated it to English and replaced the name of the company with FooBar. Enjoy your read and stay with me until the end, because I'm going to tell you what is bullshit about it.

> Hello,
>
> We work for FooBar, a start-up organizing innovation contests and hackathons!
>
> We wanted to tell you about an innovation challenge that would greatly interest you as well as your meetup members!
>
> That's why we invite you to the FooBar Challenge, organized by IBM in cooperation with BRED, in partnership with FooBar and revolving around the customer experience revolution.
>
> We're asking you to develop the prototype of a web or mobile application, to be used internally, that should allow BRED to know more about its customers. Participants will have access to the Spark and Bluemix tools to develop their projects.

Code available on GitHub

> To participate, you just have create a team and present a first PowerPoint document of 5 slides.
>
> The FooBar, IBM and BRED teams.

Note: BRED is a French bank.

If you don't see what is wrong with that message, let me explain.

First of all, refrain from adding exclamation marks everywhere! It's annoying! It made you look foolish! And trust me, you don't need that!

You "*want to tell me about something that will greatly interest me and my meetup members*"? I don't think so. You actually *are desperate to have developers going to your hackathon and working for free*. It seems like in your head hackathons are a cheap way to get developers to do your work for you. Feed them with pizza and beer, give them a Hipster MacBook and a geeky Oculus and they'll do whatever you ask them to do.

Because that's what you do. You are *asking* to develop a prototype. That's not the spirit of a hackathon, where you build whatever you want. Here it looks more like a classical set of specifications for your average day job project. And look what kind of project, an internal tool for a bank to know more about their customers. That's not anybody's dream project. That's more like the kind of app you'd build because that's your job and you have no other choice. I might be exaggerating a bit here, some people may like building this kind of app, but that's not the point. The point is that you **do not tell people what to build in a hackathon**.

Oh, and you're too kind to let the participants use Spark. You know, because Spark is OSS and no-one need your approval to start using it.

At that point in reading the email, I was already convinced that this thing, whatever that was, was a hackathon only by name. But the final selection on with a 5-slides PowerPoint was the nail to the coffin. First you're doing a filter *before* it even began, and then you're doing it on a bullshit PowerPoint?

Haha, thanks but no thanks.

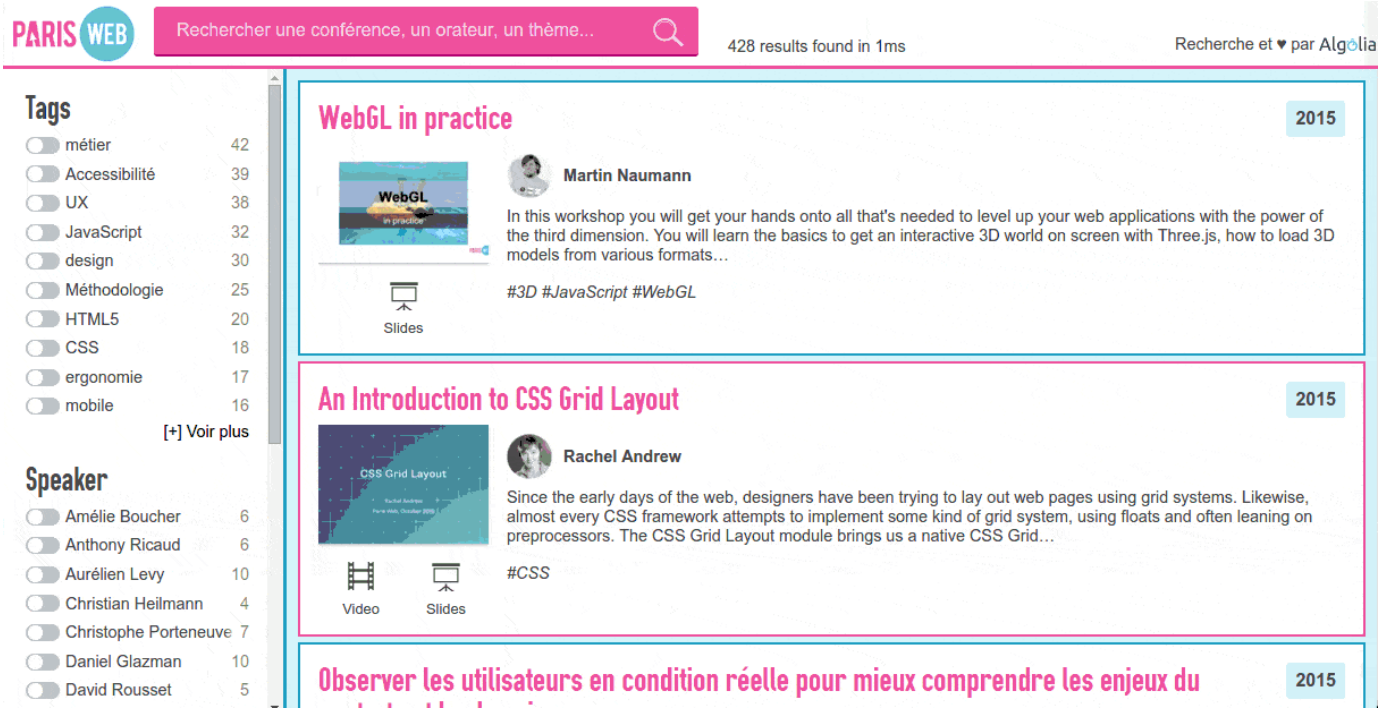
I hope that other "hackathon organizers" will read this and work a bit more on understanding what a hackathon actually is before sending those pathetic emails.

#bullshit #hackathon

I've been going to ParisWeb almost every year since it started ten years ago. I missed the first one and the 2012 edition (I was living in New Zealand at that time. I guess it's a good enough excuse).

I cannot say how much I learned from this event. I used to say that I could learn more in two days at ParisWeb than in 6 months of technical watch on my own, reading blogs. That's the conference that made me the web developer I am today. It has this incredible energy, this special #sharethelove mood.

This year I wanted to give something back to the community, so I built a little page to let you search through all the talks of the last 10 years. I hope to make access to those little gems of knowledge easier for everyone.



As a project is only finished once it's documented, let's see how this was done.

I work at Algolia, the company that provides the underlying search engine. I'm familiar with this kind of projects, but I did not use any special Algolia treatment and everything I built has been done with open-source tools and free accounts.

Getting the data

The first part of every search project is to get the data you want to search. It is the longest part of the job. A handful of websites expose useful data through an easy-to-parse API. Most of the time you have to manually crawl the website and extract the interesting bits through HTML parsing.

Code available on GitHub

for HTML parsing.

This first pass let me get basic information about each talk: title, authors and description. I then had to add the relevant slides and video urls to each talk (when available). Thoses are listed in another page, so I had to do another HTML parsing and merge the results together. This part of the code contains some specific and ugly rules.

Unfortunately for me, the website changed almost every year so I had to write a different version of the script for each year. My scripts looked more and more like hacks, with copy-pasted methods from one script to another. I could have done it better, but as they were supposed to be run once to generate the list, I figured I could live with that.

Adding thumbnails

I knew I wanted to add thumbnails to the results, because full-text results are boring. I decided that using the slides thumbnails would have more impact, with a fallback to the video thumbnail.

ParisWeb videos are hosted either on DailyMotion or Vimeo. Getting thumbnails from DailyMotion is a breeze because you can guess the thumbnail url as soon as you know the video id. Getting thumbnails from Vimeo on the other hand involve a call to their API.

For the PDF thumbnails, I first searched for an online API that would be able to do that for me, but could not find anything. So I started downloading all the PDF talks, extract their first page and convert them to png.

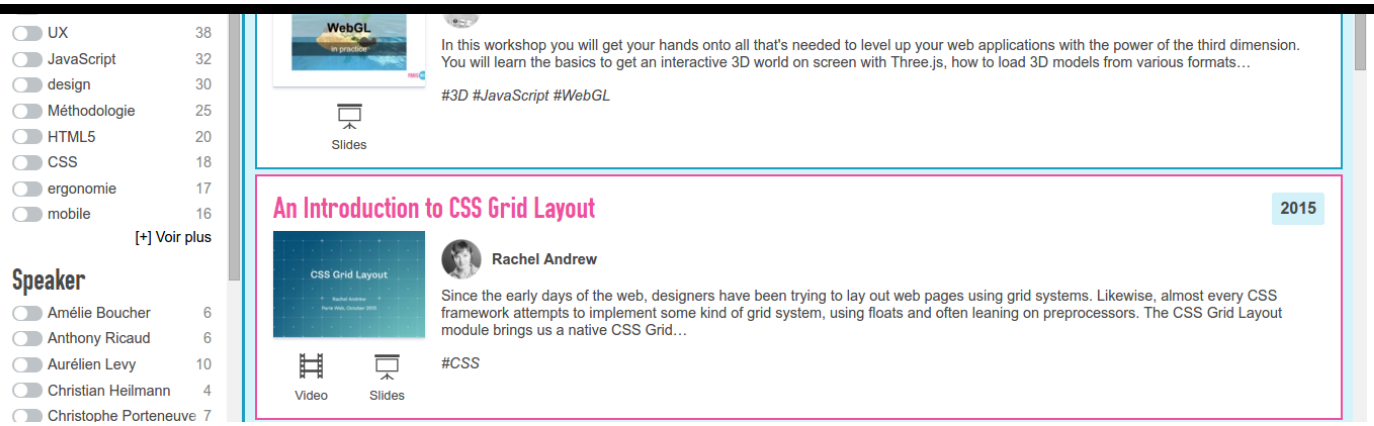
I wrapped this all into one script that will build a big array of all the talks, adding links to thumbnails to each. The final JSON is available here if you want to build something else out of it.

Configuring the search

I wrote another script to push the JSON file to the Algolia API and configure the index. I wanted it to search into the talk title, the authors name, tags and talk description, in that order. I also added filters (what we call facets) on the authors, tags and year as well as defining synonyms (like js/javascript or ux/user experience).

Building the front-end

The layout of the final page is a classical layout of search pages. A header on top with the search input, a sidebar on the left for filtering and the main area on the right. I re-used part of the code I already did for the Marvel Super-Search.



I used our instantsearch.js library. I started by creating the HTML markup, using empty `div` s as placeholder for where I wanted the various elements of the search: input, filters, results, etc.

Then, I instanciated `instantsearch` with my Algolia credentials and configured the widgets (defining if I wanted an `AND` or a `OR` on the filters, how many to display, etc). I also created a Mustache template for results and styled all that with the ParisWeb colors.

I used Cloudinary as a CDN for the images, which let me resize, compress and cache them on the fly. I also forced all author pictures into grayscale for a better visual harmony.

Getting feedback

As I was building it, I often pushed the site online (hosting it on GitHub pages), to get feedback from some beta users. It gave me a lot of interesting ideas and allowed me to improve it as I was building it.

I'm still eager to know what you think about it, and hope it will be helpful to you. If you find any issue or would like to suggest an improvement, feel free to add it to the issue tracker.

#search #parisweb

Public Speaking

07 Jun 2016

I wanted to write a recap of this session for a long time. I'm in a flight trip above the Atlantic ocean, so I have a bit of time in front of me. Let's get started.

Introduction

One of the most valuable learnings from my previous company was the public speaking course I attended (thanks again Marc). With two days of training I went from "Speaking in public? Are you

is basically a rewriting of my notes at that time.

The class was about what people see when they see you speaking publicly. We did not go into what is happening inside, because it is too intimate, we focused on what was visible on the outside.

We were asked to tell how we thought our public speaking skill was. What scared us, what we thought we do well, etc. I said that if I know the subject I'm talking about, I think I can handle the situation well. If I have a complex set of ideas to share I might get confused and forgot important details. I also know that I speak fast, so I need to be careful. Also, I've been game-mastering fantasy roleplay for more that 15 years, so I know a bit about improvisation. I'm also scared of how to start a talk, when everybody is talking among themselves.

Basic principles

We were taught the 4 most important rules:

1. We need to stay natural. This is not a theater play, and we are not playing characters, we are ourselves. We do not have to show everything, only what we want to show.
2. We must remember that this is nothing personal. You do not go on stage to be loved. You go on stage because you have something to share. People care about what you say, not who you are.
3. We should not listen to that little voice in a corner of our head that repeats "you'll never be able to do it", "what you say is not interesting", "they already know all I can say".
4. The speaker is the only accountable person. The listeners are always right. If they did not understand something it's not because they are foolish, it's because you did not explain it well enough.

Public speaking is not only about speaking. Oral speaking is just a small part of it. Your whole body is expressing your ideas when you are on stage. We are trying to convince people of what we are saying, and we do that by more than words. We do that by being convinced ourselves and our whole body will show it, in the way we breath, the way we walk, the way we move, etc.

Conversation, Information or Communication

We also made a clear distinction between a *conversation*, an *information* and a *communication*.

A *conversation* is mundane and informal. It's small-talk and there is no definite point. It can happen anywhere, anytime, there is no specific pattern.

An *information* is formal. It goes one way, from the one who knows to the one who does not know. The boundaries are clear. Think of the flight speech about exits in planes.

Make your message clear

You want people to have a *before* and an *after* your talk. Because if they don't, what was the point? You must have a message they will remember.

- You should be able to pitch that message in one or two short sentences.
- Do not hesitate to repeat the message throughout the talk, with different words, but with the same meaning.
- Give context to what you say. Give examples, and facts.

People might understand the theory of what you're talking about, but if you give them real examples, they will remember it better once you've finished talking. If it helps the listener understand and remember, you should do it. Remember that the audience is always right. If it's too complex for them, they will stop following.

People have a limited attention span. Maybe you're the first talk of the day and they are not yet fully awake, maybe you're the last talk before lunch and they are hungry, maybe the speaker before you was boring and they are falling asleep, maybe you're the last speaker of the day and they are tired. There are countless ways that the audience will not be in the perfect mood to listen to you. This will not matter if you've adapted your speech so it is really easy to understand.

Split your talk in chunks. Do small breaks. Don't let the audience doze off because of your monotonous voice. Throw an image here and there to wake them up. Make it feel like there is something new to discover at each new slide. Try to hook them. If they are not hooked, they will just go to their smartphones and check Twitter. Talk about you and how you relate to the subject you're talking about.

Now that we've seen generic ideas, we'll go a bit further into the content itself.

Introduction

Keep about 1mn for an introduction. Start by creating a desire, explaining why, sparking curiosity. Giving a real anecdote is nice way to start.

At this point, we were suggested to use sentences like "Have you ever noticed...", "Are you like me and..." or "From the dawn of ages, humankind have..." to quickly rally the audience to what we are going to say. I personally don't want those introduction because they look too much like one-man shows. Also, the talk being unidirectional, the audience cannot answer and if they do not agree they can't do anything but listen, which will frustrate them. So my personal advice is to avoid those over-generic sentences.

then dive into the subject wholeheartedly.

Clearly show where you are

Whenever you transition from something to another, from one slide to another or from one idea to another, verbalise it. Say that we're moving to something else, that way people can more easily clear their mental buffer.

If you ever enumerate things, count on your fingers, or clearly show that every item in the list is different. Do not put too much content in your slides. People will read your slides, and while they do, they don't listen to you. So try to keep the information in the slides to a minimum. It should be just enough to let someone see what your current point is and where you are in your explanation.

Add a clear mention of the current chapter you're currently explaining, so people know where you are. Clearly show how many slides you have, so they know when to expect the end of your talk. If they think it is a never ending talk, they will get bored before the actual end.

You are not transparent

Do not confess your internal state to the audience. Do not tell them how stressed you are, do not say "Oh I'm sorry, I don't remember what I wanted to say". Remember that everything that happens inside of you is not visible on the outside. If you tell the audience that you are stressed, then they'll know. If you say nothing, they won't have a clue.

If you don't remember what you want to say next, just leave a blank. When people add silence to their talks, it makes the audience think about what was said before. It is way better to just put a silence than a "hmm, so... Hmm... Yeah". The former makes your previous sentence feel really important while the latter makes you sound foolish.

Own the stage

Try to be sure of what you say, people will feel your presence on stage and will feel like you belong there. This will give you legitimacy.

Be dynamic, don't be static. This show that you have a strong belief in what you say. And people don't want to hear you talking about something for which you don't really care. They already have plenty of things they don't care about, they don't need yours.

Be synthetic. Everybody can talk about any subject, but only the real professionals can be synthetic about it. This is related to the Dunning-Kruger effect, but the more synthetic you are on a subject, the more people will see that you've mastered it. On the other hand if you start explaining every little details, it show that you do not see the big picture.

Ask if people have any questions, and don't forget to smile, we don't like to ask questions to someone with a closed face.

Do not, ever, finish your talk with "I hope you liked it", it undermine everything you said. Instead, say "With everything I talked about, you are now ready to do {whatever you talked about}".

Bonus

We also got a list of mantras that you can repeat before going on stage:

- I do not talk because I want to be loved. I talk because I want to be understood.
- I know it's normal to be scared when speaking publicly, and I know this is not a weakness.
- My internal stress is not visible on the outside. People cannot read inside of me.
- If I feel the stress coming up, I slow down, I breath, I take my time.

Bonus number two

We also had a nice discussion about face-to-face talks. For example when you're meeting with customers and you want to explain why something they did is wrong and why your way is better. You want to convince them of your way. Here are a few things to keep in mind:

- Never say to the others that they are wrong. Their reasons are perfectly valid, knowing what they know.
- Being convinced of being right is not enough, you need to also be convincing.
- No-one was ever convinced by an authoritative argument. You have to be reasonable, and nuanced.
- Give facts, examples, evidences.
- Avoid antagonism. You and them are working toward the same goal: using the best possible solution. Understand their issues, put yourself in their shoes.
- Say things clearly, do not make them think they may have not understood things clearly or that you are hiding something.

#public-speaking

Marvel Super-Search

01 Jun 2016

Last December I discovered that Marvel had an API. Yes, that's right, Marvel, the publisher of all the comics I used to read. I'm a developer, but I'm also an avid consumer of APIs and love to build things with all the available data.

Code available on GitHub

dataset.

You can see the full demo here

So I started registering to the Marvel developer program to get an API key and started pulling some data. Or actually that was my original plan, but the developer website was buggy at that time and I could not register at all.

At first I was disappointed because I really wanted to index all these characters and superpowers, but I wasn't going to let that bring me down. I started rolling up my sleeves and went on the hunt for another interesting source of data.

Wikipedia

I obviously quickly ended up on the Wikipedia, where I could find a serie of pages listing all the heroes and villains of the Marvel universe (or actually, universes). I wrote a small node script using x-ray to get the list of names and matching urls and saved it on my disk.

```
import xray from 'x-ray';
let x = xray();

const targetUrl = 'https://en.wikipedia.org/wiki/Category:Marvel_Comics_superheroe
s';
const selector = '#mw-pages .mw-category-group li a@href';

x(targetUrl, selector)((urlList) => {
```

Code available on GitHub

Then my journey of extracting data from the Wikipedia began.

As I said, I love APIs. And the Wikipedia being the huge project that it is, I was sure that they had an API to get clean data from each page. They do have an API for sure, but it only returns the raw dump of the page, including a mix of HTML and custom Wikipedia markup. This is not formatted at all, so was of no use to me.

```
*Enhanced mental faculties *Master martial artist and hand-to-h
s a [[American patriotism|patriotic]] [[supersoldier]] who ofter
ego]] of '''Steve Rogers''', a frail young man enhanced to the
)|movie serial]], ''[[Captain America (serial)|Captain America]]
le=IGN's Top 100 Comic Book Heroes|date= 2011|publisher= [[IGN]]
uperheroes|title=Top 25 Best Marvel Superheroes|first1= Joshua|l
No, it didn't work. There were too many "Supers" around. "Captai
as soon as possible. Needing to fill a full comic with primarily
e eager to join in on the new ''Captain America'' book, but Jack
ction as only he could. Then he tightened up the penciled drawir
wanted to have our say too."<ref name="Wright 36">{{Cite book |
st2=Sanderson|first2=Peter |authorlink2=Peter Sanderson |title =
s Publishing]]|date = June 2005}}</ref> While most readers respo
= 135–136|isbn = 978-0-452-29532-2}}</ref> Though preceded as a
opular hero with nearly a million copies of his comic sold per m
. Stanley Lieber, now better known as [[Stan Lee]], contributed
e art in a Simon and Kirby comics story illustrates the followir
|Harry N. Abrams]]|year = 1991|page = 37|isbn = 9780810938212}}<
the [[All-Winners Squad]], in its two published adventures, in '
cs]] attempted to revive its superhero titles when it reintroduc
```

Example taken from here

DBPedia

I kept searching and found the DBPedia, which is an unofficial project of creating an API of structured data on top of the original API. The people in this project did an Herculean job of converting the raw markup of my previous example into nice looking JSON responses.


```
{
  "abstract": [
    {
      "lang": "en",
      "value": "Hercules is a fictional superhero appearing in […]"
    }
  ],
  "aliases": [
    {
      "lang": "en",
      "value": "The Prince of Power, The Lion of Olympus, Harry Cleese, Victor Tegle
r"
    }
  ]
}
```


Code available on GitHub

where no taken into account. And worse than that, some really popular characters did not even had any data attached to them.

Infoboxes

That's about that time that I realized that the only information that I actually needed was the one displayed in the infobox. The infobox is this little white box on the right side of any Wikipedia page that displays an overview of the most important facts of the page. In my case it was the name of the character, its potential aliases, powers, known teams and authors.

The Hulk	
Publication information	
Publisher	Marvel Comics
First appearance	*The Incredible Hulk* #1 (May 1962)
Created by	Stan Lee Jack Kirby
In-story information	
Alter ego	Robert Bruce Banner[1]
Team affiliations	Avengers Defenders Horsemen of Apocalypse Fantastic Four[2] Pantheon Warbound
Notable aliases	Joe Fixit, War, Green Scar, World-Breaker, Sakaarson, Doc Green
Abilities	**Bruce Banner:** • Genius-level intellect **Hulk:** • Immense superhuman strength, durability, and longevity

I did not really care about the rest of the page. What I had in mind for my demo would be a simple way to search through all characters and filter them based on exactly those criteria (power, teams, authors) and being able to find them with any of their aliases. So I needed a way to extract content from the infobox.

Fortunately, I started this project in node. And node comes with npm, where there is a module for anything. I quickly found wiki-infobox that let me extract a *mostly* structured representation of the data in the infobox, by just feeding it the name of the page.

```
import infobox from 'wiki-infobox';
```

Code available on GitHub

```
  //   },
  //   "aliases": [
  //     {
  //       "type": "text",
  //         "value": "<br>Green Scar<br>World-Breaker<br>Jade Giant"
  //     },
  //     […]
  //   ],
  //   […]
  // }
});
```

I say *mostly* because the module tries its best to manually parse the raw dump I showed earlier. And it did that using regexp and trying to handle all possible edge cases. Overall it worked quite well, but I still had to clean the output to have something that I could work with. My respect for the team behind DBPedia grew even more at that time, because extracting formatted data from the initial dump is clearly not an easy task.

Using both DBPedia and the data I got from my infoboxes, I started to have an interesting dataset. One thing that was missing were popularity metrics. I wanted my heroes to be displayed by order of popularity. If I start typing *iron*, I want Iron Man to be displayed first, not the unknown Iron Monger character.

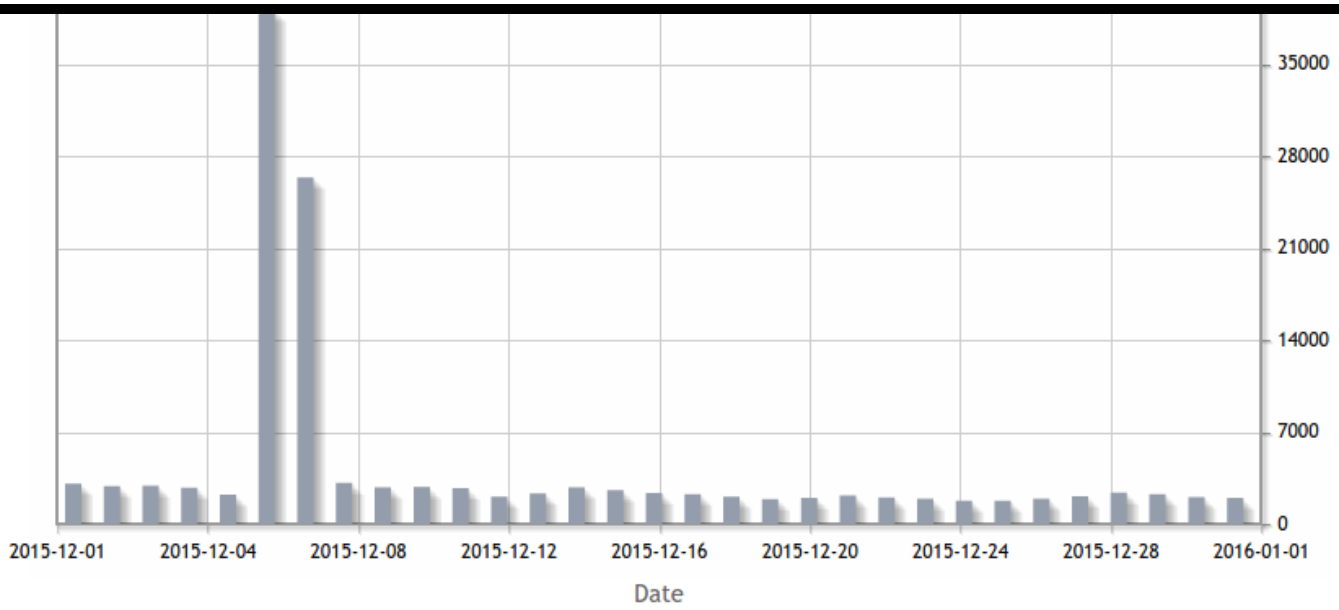
Wikidata

In order to get this information, I tried the Wikidata API. This API gave me a lot of metadata information about each Wikipedia page. But that's it, only metadata. Data about the data. Like the name of each page in each language or the other names that redirect to the same page. This wasn't exactly what I was looking for, but let me grab a nice list of aliases for my characters. Thanks to that data, I could now find Spider-Man with Peter Parker, Spidey or Webhead.

Of course, there's an npm module to do that easily as well: wikidata-sdk.

Page views

The more I searched for Wikipedia-related APIs, the more I found weird projects. The last one I used is stats.grok.se, which is the personal project of a core contributor that exposes as an API, the pageview count of each Wikipedia article on the past 90 days. This could give me the popularity ranking I was looking for. The page for Iron Man was visited way more often than the one for Iron Monger, so I could use that data to rank them.



Unfortunately, the stats are only valid up to December 2015. After that, the endpoints were just returning empty results. But what I also discovered is that those results were Netflix-biased. I mean that at the time I did the crawling, Netflix just released its Jessica Jones show, so Jessica Jones and all the other characters from the show (Purple Man or Luke Cage) had the more important number of pageviewsr. While the show is great, Jessica Jones is in no way a more popular character than, say, Spider-Man, Iron Man or Captain America in the comics.

My dataset was starting to look promising but there was one information that I still did not manage to get. Believe me or not, but from all the APIs I tried, absolutely none of them were able to give me the url of the image used to display the character. I absolutely needed this image to build a demo that looked nice, so I coded another small `x-ray` script to go scrap every Wikipedia HTML page and extract the image url. Sometimes the best solution is the more obvious one.

Marvel API

It took me a few days to do everything I mentioned above. Just to be sure, I had a new look at the Marvel developer portal and I discovered that they fixed it. So I registered for an API key and started exploring their API to see what I could extract from it.

First thing to know is that the Marvel API is tedious to use. I had countless timeouts, infinite redirect loops, empty results, errors and other things that made the whole experience unpleasant. But in the end it was all worth it because of the quality of the data you can extract from the API.

Code available on GitHub

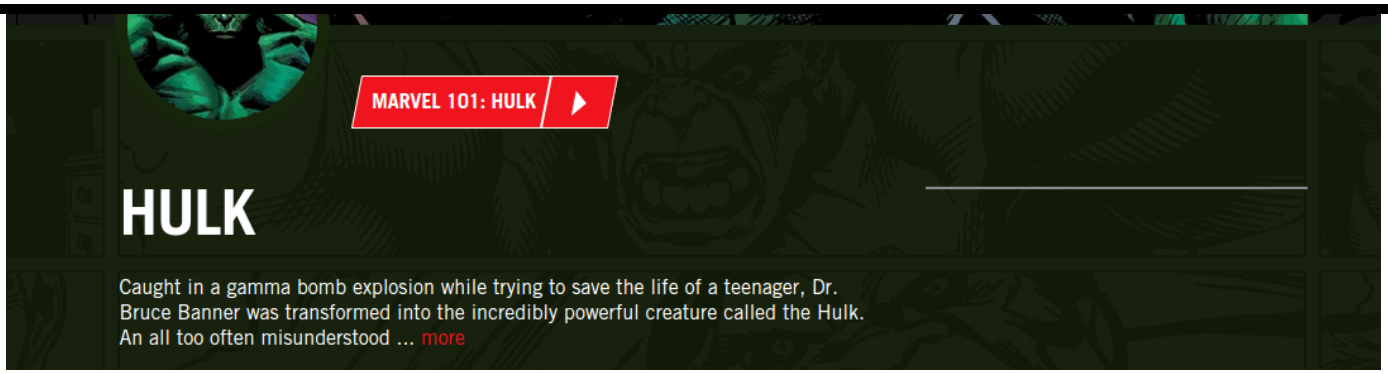
First of all, they do provide url to an avatar image of each hero. And not just any avatar image, one that is correctly cropped and with all background noise removed. It also gives you an *in-universe* description of each character. So now I could display that Daredevil was the secret identity of Matt Murdock, whose father was a boxer. As opposed to simply saying that "Daredevil is a fictional character from the Marvel universe", which is not very relevant. And finally the API gave me the popularity ranking I was looking for. For each character I have the count of comics, stories and events they were involved in.


```
{
  "name": "Daredevil",
  "description": "Abandoned by his mother, Matt Murdock was raised […]",
  "thumbnail": "http://i.annihil.us/u/prod/marvel/i/mg/6/90/537ba6d49472b/standard_
xlarge.jpg"
  "comicCount": 827,
  "eventCount": 11,
  "serieCount": 163,
  "storyCount": 1326
}
```


All data I could get from the Marvel API was of much better quality than anything I could have had from the Wikipedia. Still, each character only had a few bits of information, so I merged results with my previous Wikipedia scraping, using fallbacks to always use the best possible value.

Marvel website

One last drawback of the Marvel API is that their website does not even use it. You can find on the official Marvel website pages for each character that display more information about them that what you could find in the API (things like their weight or height). The designers at Marvel did an incredible job at defining a main page color for each character. It means that the Hulk page will have a green tint, while it will be purple for Hawkeye and red for Daredevil.



They also defined custom background images for major characters. All character pages have several panels of comics in the background, but for major characters, all panels are actually taken from real adventures of that character.

Through tedious scraping and parsing of CSS and JavaScript, I managed to extract this information for each character, and could use it to improve the UI.

Tips & tricks

I will give more details about how I build the UI in another post, and don't worry it will be way easier than getting the data. Before closing this post I'd like to share a few tips I learned from this whole process of getting data out of APIs and websites.

First of all, you should create isolated and repeatable scripts. In my case my data was coming from various sources, so I created a script per source. That way I could just run the DBPedia script again and update only the data coming from the DBPedia.

Crawling always comes in two steps. First you have to get the data from the website or API, then you have to extract and format it the way you want.

Code available on GitHub

but chances are that you'll have to rewrite your script dozens of times until you got all the edge cases right.

My advice is to always create two different scripts. The first one will blindly download all the urls and save the output on disk, without doing any formatting on it. The second one will read the local version and extract the data. Doing so, you only pay the price of the download once, and can then quickly iterate on the extraction part.

I would not commit the temporary files into a git repository, but only the output formatted files. Whatever the format in which you store the output file, I would make sure that the way it is saved is consistent across extractions, so you can easily do a diff between two versions of the file. For JSON, this means ordering your keys alphabetically.

Finally, when dealing with external sources, especially the Wikipedia, I'd be extremely careful on the inputs. You're basically handling data that has been written by somebody else. Chances are that they forgot to close a tag, or that the data will not be correctly formed one way or another. Add scripts to fix the data for you, and add tests to those scripts so you're sure that when fixing one issue you're not creating a new one. I have more than 300 tests for this example. It's a lot, but it's needed.

Conclusion

This was a really cool project to do. You can see the demo online, or browse through the code. Everything is in it, from the import scripts to the final UI, even including the JSON data files.

#algolia #search #marvel

Older Newer