

Flickr Uploadr

Antero Duarte

40211946@live.napier.ac.uk

Edinburgh Napier University - Advanced Web Technologies (SET09183)

Abstract

Flickr Uploadr is an application that allows users of the photography oriented platform Flickr to schedule photos to be uploaded at a later date. The users connect their flickr account to the app and can schedule photos to be uploaded and can change the title, description and tags of a photo.

Keywords – Advanced, Web, Technologies, Python, Flask, Flickr, Scheduler

1 Introduction

Flickr [1], like many social media platforms, prioritises content from users who post regularly. This means that the same user, with the same following and a comparable quality photo is more likely to get more exposure and better reception for a photo that is posted in the 10th day of a 10 day streak, than one posted on the first day.

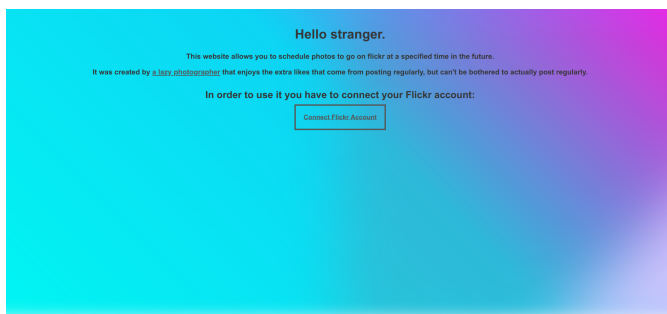


Figure 1: Flickr Uploadr - Home page

This application, by allowing users to schedule posts for later days, allows people who see photography as a hobby to enjoy the same level of exposure within the Flickr platform. For photography professionals, it allows them to batch edit photos and not have to worry about being consistent with their posts and focus more on the rest of their work.

Flickr Uploadr uses the Flickr REST API [2] to upload photos as the logged in user. It uses a microservice oriented architecture and decouples the frontend from the backend [3].

2 Design

This web app was designed in line with modern standards and best practices.

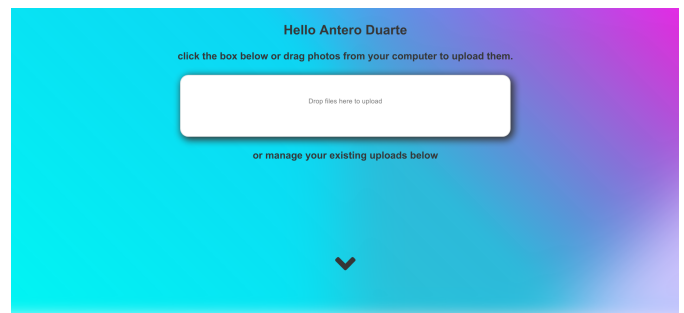


Figure 2: Flickr Uploadr - App home

2.1 Architectural Overview

The backend uses python Flask [4] for the web related activities, an embedded SQLite[5] database to store user information and uploading jobs ¹.

The frontend uses Vue.js [6], a javascript framework. It is a framework that is increasing in popularity and is considered one of the top 3 javascript frontend SPA frameworks, alongside React.js [7] and Angular.js [8]. In my opinion, and the reason why I like it is that it stands exactly in the middle between the other two. Where Angular is too enterprise focused with little room for customisation in terms of code organisation and design patterns in favor of corporate-oriented standardisation, and React.js makes no assumptions and offers little in the way of suggesting best practice, leaving that for the developer to deal with. Vue.js offers a standardised way of organising code, by providing single-file components [9], which include markup, logic and presentation in the same file, but at the same time, it can be used in more complex applications, with state [10] and route [11] management or simpler applications as a substitute for older Document Object Model manipulation based solutions like jQuery [12].

2.2 Database and data modelling

As mentioned before, the database is a SQLite disk persisted instance that is accessed from the code through

¹ An uploading job is a photo and all of the associated settings needed to upload said photo to Flickr

the Peewee Object Relational Mapper [13]. The database schema is as seen in figure 3.

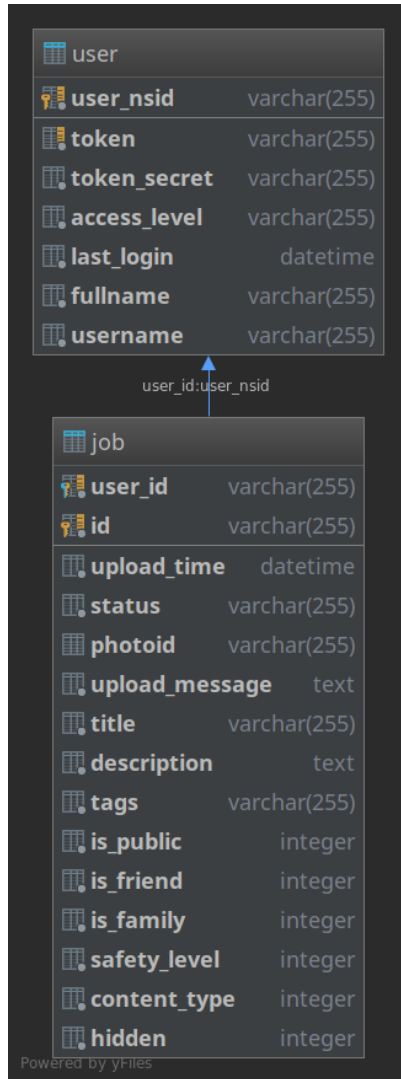


Figure 3: Database Schema

Figure 3 shows 2 main tables, user and job. Peewee recommends that auto-incrementing IDs are used as primary keys for the database tables, but since there is no canonical answer to this question [14], I decided to go with what we learned in the 2nd year Database module, that when natural primary keys exist, they should be used in favor of surrogate keys.

Although this is an advantage in terms of data design, it introduced a workaround, since peewee supports the opposing view.

```

1 class BaseModel(Model):
2     # We override save because we use self managed primary keys, and peewee docs
3     # say that we must always call it with force_insert=True when that is the case
4     # So I'm expecting this to save countless hours of debugging
5     def save(self, *args, **kwargs):
6         if "force_insert" not in kwargs:
7             kwargs["force_insert"] = True
8         return super(BaseModel, self).save(*args, **kwargs)
9 
```

Listing 1: Peewee save function workaround

The code excerpt 1 shows the workaround for peewee's

default save function. As explained in the documentation [15], self managed primary keys need the argument `force_insert` to be set to `True` on every save.

2.3 Authentication/Authorisation

This application extends Flickr, which means users need a Flickr account to use the application. Flickr provides an OAuth [16] server as part of their API [17], which means that the amount of data this application needs to keep about a user is minimal. It also takes the pressure away from this app to manage users and passwords. The application links a user account to a Flickr `user_nsuid`, which is Flickr's unique identifier for users. It also stores an OAuth Token, so it can upload photos as the user when scheduled, even if the user is offline. Although it is generally discouraged to store access tokens, it is the only option when scheduling activities on behalf of the user.

2.4 Decoupled frontend

As mentioned before, this application uses Vue.js to provide a modern, feature-rich interface that connects to the server using a RESTful API [18]. Vue.js is a reactive library, which means that changes to the application state will trigger components that receive the state to re-render. In practice this means that a developer does not have to manually listen to events in the DOM², to trigger changes to the DOM². This enables faster development and less time debugging/dealing with DOM² quirks.

2.5 Application functionality

Upon logging in, a user can upload images to the server by using the file drop box, provided by `vue-dropzone` [19], which is a plugin that implements `Dropzone.js` as a Vue component. Uploaded images automatically generate a job ID (v4 UUID[20]) and the job fields are set to their defaults (`title='name_of_the_uploaded_file'`, `description=''`, `tags=''`, `upload_time=current_time + 1 day`). After creating, a job's status is set to `WAITING`, meaning it is on the queue to be processed shortly after `current_time > job.upload_time` (See figure 4). Shortly after, because there are no guarantees that a job will run exactly on the specified time, for example, if there's a queue of jobs all set to run at 5pm, they will run in the order they come from the database, presumably, row creation time, which essentially means first come, first served.

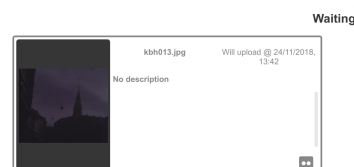


Figure 4: Jobs waiting to be processed

A job can be edited, so a user can update the photo's title on Flickr, the photo's description on Flickr, the

²Document Object Model

photo's tags on Flickr, and the uploading time (See figure 5).

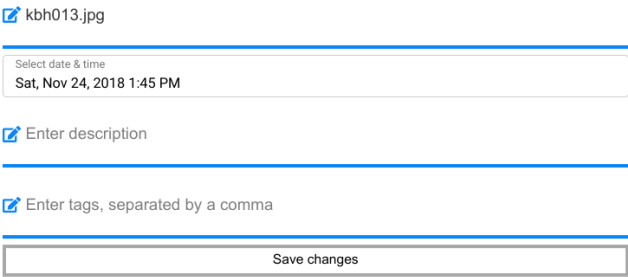


Figure 5: Jobs editing form

The uploading time is displayed in the form of a calendar, where past dates are blocked (See figure 6). As shown in the screenshot, taken on the 23rd, days 23 and before are blocked. The reasoning behind blocking the 23rd as well stems from a belief that people using a scheduler should not be putting pressure on the server for immediate/almost immediate uploads.

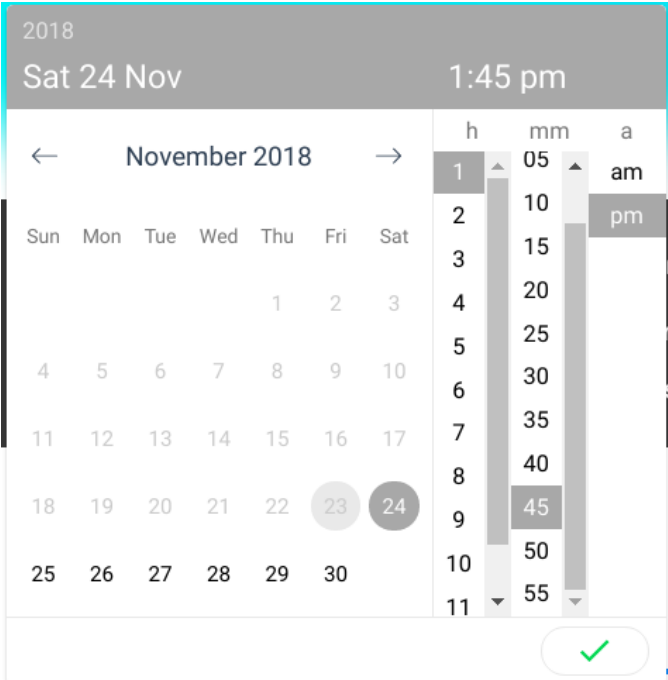


Figure 6: Calendar

This means that a user posting a job for now + 5 minutes is essentially placing the effort on the server, rather than waiting 5 minutes and posting the photo manually on Flickr. This is not a hard constraint, and is not checked on the server side, but by implementing it on the UI, it discourages users from doing it.

2.6 URL Structure

2.6.1 URL Map

Since this application is based on a REST API, the URL structure is quite simple and has a few conditional branches, as can be seen in Figure 7

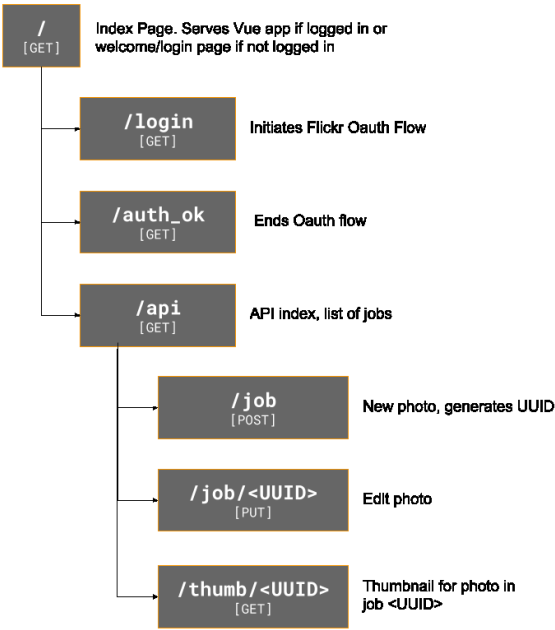


Figure 7: URL map

2.6.2 Navigation map

As for navigation, the application presents the map in figure 8.

Note the navigation around the Login flow, which as per the oauth spec [16], sends a user to Flickr to authenticate (figure 9³), then send the user to a consent page (figure 10) which authorises the application to perform activities on behalf of the user, and then back to the application, which receives the necessary parameters to request an access token via url query parameters, and finally redirects the user to the application home page, where the application already has access to user data, as see before on figure 2, where the user's Flickr full name is shown.

3 Enhancements

“If it ain’t broke, don’t fix it. Do improve it though...”

There are several things I would have done if I had more time to develop this project. Although the initial idea/functionality is all covered, a few more elements would greatly improve this application.

³ At the time of writing, flickr was in the process of transitioning from being owned by Yahoo and using Yahoo's login system, which is why the flickr login screen is a yahoo login screen.

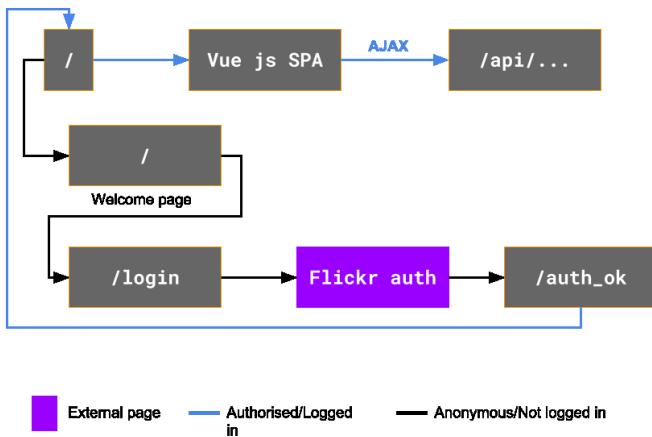


Figure 8: Navigation map

3.1 Tiered user system

As is, all users are the same to the application, but during development, I had the idea to create a tiered user system, where different usage tiers would have different privileges. For example, there is no limit to the amount of photos a user can upload, but to make this application feasible and free to use, there would have to be restrictions on the amount of images a user can post so the disk usage on the server doesn't grow endlessly. One of the user tiers, could have the user pay for unlimited or a higher cap on storage. This would mean the paid users would be paying for the server/hosting, while free users could still enjoy some of the benefits of the system. Introducing a cap on storage would mean that the application would have to keep track of how much disk space a user is currently using, which it doesn't at the time of writing.

3.2 Setlist flow

An idea I had since the start was to measure a setlist's flow.

By flow here, I mean how one song progresses into the next one. Key changes, beats per minute, loudness and other acoustic metrics that I have access to in the data could be used to calculate how easily one can move from one song to the other. Other data could also be gathered to try to make the flow calculations better.

For example, songs in different tunings sometimes mean that artists have to switch instruments or tune them differently, which breaks the flow in a live concert scenario. Songs with different instruments than the previous one might mean that a band member has to switch instruments or come on/off stage. Songs with a low dynamic complexity (low variation between calm/agitated periods) and high bpm, or high danceability and bpm, might make the band tired after playing a lot of them in a row, which means they would need to rest between songs.

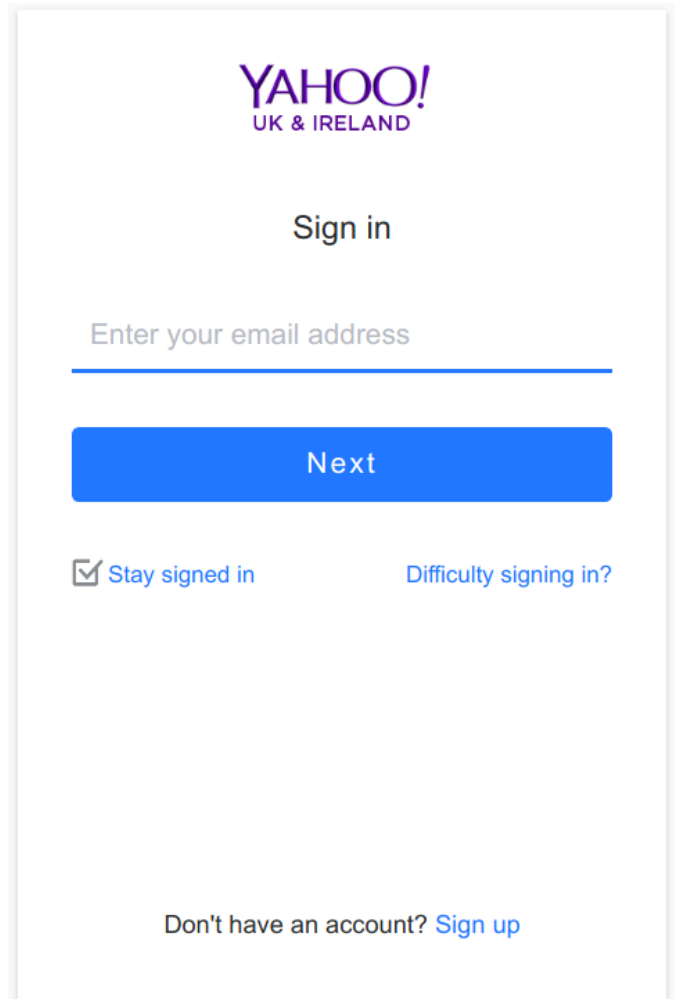


Figure 9: Oauth flow - Flickr/yahoo login³

These and other conditions could be made into a program that tries to leverage the elements of different songs to keep a crowd's engagement.

3.3 Data collection

Finally, a feature that could be extremely useful in growing an app like this one would be to collect data about how people are using the website. This could become a big advantage when developing new functionality, and could also enable machine learning based features, if said data was used to train models.

3.4 Dedicated setlist page

When creating this application, the initial idea involved a page dedicated to managing the songs in the current setlist. This idea was dropped when due to time constraints I realised that I would not be able to create the necessary mechanisms to have enough functionality that makes it worth having such a page.

The setlist page would include the previously mentioned track suggestion feature, a track reordering (using drag and drop) and overall setlist metrics, such as total running time, and possibly even a dashboard type

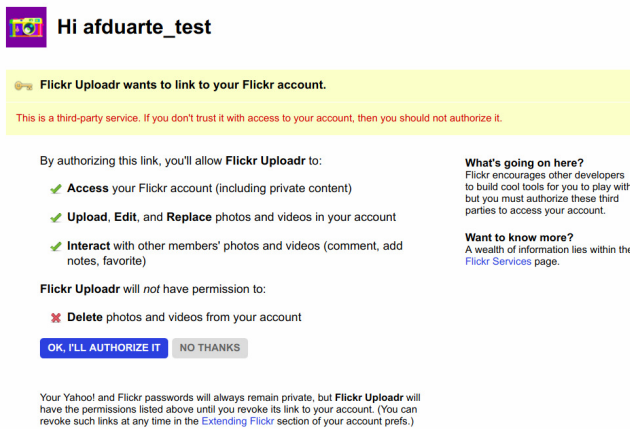


Figure 10: Oauth flow - Flickr consent page

interface that would show the characteristics of the current playlist.

3.5 Dedicated track page

Similarly to the setlist page, the amount of effort needed to create a track page, when compared to the functionality that would be in that page, just did not justify its creation.

3.6 Direct connection to data APIs

All of the data collection process could have been avoided if a direct connection was made to the APIs offered by both data sources. This would have opened up a lot of possibilities, considering that the artist catalogue is quite limited, but on the other hand, it would have made the range of different outcomes too unwieldy, which would have slowed development because the code would have to accommodate for gaps in the data that were filled during the data gathering stage.

4 Critical evaluation

While I'm happy with the current state of Set.lystr, several things about it make me want to spend more time on it, even if just for fun/portfolio building.

4.1 Architecture

One of the things I enjoy the most about developing projects such as this one is to look at the holistic system architecture. While being a relatively simple project that is composed of a single codebase, it's architecture is also rather simplistic. While there are things about this project that might not pass the scalability test, for example not splitting different routes into different files, I personally consider this a good architecture when taking into account the wider context. This application and its development have a scope, and one of the best skills when thinking about the architecture of an application is to take into account the balance of time vs features.

I could have made this application more scalable. I could have decoupled the frontend from the backend, which would mean less work on the server side. I could have deployed n instances of the application and load balance them. But calling that overkill would be an understatement.

That being said, I am particularly happy about features such as the streaming search results function, which is backed by a generator function that yields results as they become available. This improves the perceived loading times, and when matched with client-side functionality that sorts the results as they come, creates a good User Experience.

Another example that I am particularly happy with is the balance between AJAX functionality for improved user experience and the fall-backs that are in place for the case where a user is not running javascript. While it could be argued that users whose browsers don't support javascript would not benefit from the modern CSS in the page, having these fall-back capabilities means that even with a broken layout and a few ugly pages (I'm looking at you, default Flask redirect-after-form-submit page), a user is fully capable of using the system. This is also true for screen readers, which makes the web application more accessible.

4.2 Responsiveness

While responsive to a point, this web application still has some pages that are not optimised for mobile view. One of the worse elements being the setlist, which should be made to take the whole screen when on mobile views.

This not only makes the website less appealing, but also harder to interact with when on mobile devices. This becomes a major problem, when considering that upwards of 60% of the web's traffic according to statistics from April 2018, are from mobile devices.

4.2.1 Sub par search results

When searching for a set of keywords, a user can clearly see that the search results are just not up to the standard that a user has come to expect when using a website. And while this is partly due to users being "spoiled" by great user experiences, such as the ones that big companies/products provide, the truth is that the technology for achieving such functionality is more accessible than ever. Besides dedicated search engines that index all pages in a website, there are libraries that could be used to improve string comparison, which in itself would have made search results better.

While string comparison using the naive python builtin *SequenceMatcher* stems from a limitation imposed as part of the coursework, of not being able to install additional libraries, if I had put some more effort into the search functionality, better results would have been possible. For example, for the cases where an artist has a self-titled release (which is very common, e.g: Pearl Jam

by Pearl Jam), due to the text index being just a dictionary of *string* to *string*, whatever comes first is overridden by what comes after, which means searching for Pearl Jam would never return the artist, when more often than not, that is the desired outcome.

This could be fixed quite easily by just making the text index a dictionary of *string* to *list of string*, but I decided that this would come on a following iteration of the development, due to time constraints.

5 Personal Evaluation

Web development is an area of computing that has always fascinated me. Partly because of how easy it is nowadays to create something that can be accessible by the whole world, but also because it can be quite challenging to adhere to living standards and create a web application that follows all best practices in the industry. That being said, I work as a part-time web developer, so a lot of these best-practices are ingrained in the way I develop a project such as this one.

This can be a blessing and a curse. It is very good to know my way around a codebase and to be able to understand the basics of a new framework in a programming language that I have barely used before in a matter of minutes, as was the case with Flask, simply because I understand the underlying concepts that are common to most web applications. But this brings with it a sense of perfectionism and always trying to push myself past the next boundary that can work against me. I am guilty of having developed coursework applications that go beyond demonstrating that I am capable of achieving the level sought after by the marker, simply because that is the bar I have set for myself and my work outside of university.

In this coursework in particular, I would say that I was able to strike a balance between achieving the learning outcomes proposed in the coursework description and achieving a level of self-fulfilment that comes from doing the best I can. As mentioned in the enhancements section3, I realise that there are things I could have made prettier or more functional, or better in general. But I also realise that I introduced challenges for myself that were out of the scope of the coursework but that I decided to do in order to improve my abilities (for example, not using frontend libraries).

In the end, I am happy about where I got to because I can see a clear path to improve the things I am not completely happy about with this coursework. I was able to manage the whole project from start to end, balance that with the rest of my university workload and a particularly busy period at my job.

References

- [1] "Flickr." <https://www.flickr.com/about>.
- [2] "Flickr app garden - rest api." <https://www.flickr.com/services/api/>.
- [3] SitePen, "Microservices and spas." <https://www.sitepen.com/blog/2017/02/20/microservices-and-spas/>, Feb 2017.
- [4] "Flask (a python microframework)." <http://flask.pocoo.org/>.
- [5] "Sqlite." <https://www.sqlite.org/index.html>.
- [6] "Vue.js." <https://vuejs.org/>.
- [7] "React.js." <https://reactjs.org/index.html>.
- [8] "Angular.js." <https://angular.io/>.
- [9] Vue.js, "Single file components." <https://vuejs.org/v2/guide/single-file-components.html>.
- [10] Vue.js, "Vuex (centralized state management for vue.js)." <https://vuex.vuejs.org/>.
- [11] Vue.js, "Vue router (the official router for vue.js)." <https://router.vuejs.org/>.
- [12] SmashingMagazine, "Replacing jquery with vue.js." <https://www.smashingmagazine.com/2018/02/jquery-vue-javascript/>, Feb 2018.
- [13] "Peewee orm." <http://docs.peewee-orm.com/en/latest/>.
- [14] agiledata.org, "Choosing a primary key: Natural or surrogate?." <http://www.agiledata.org/essays/keys.html>.
- [15] "Models and fields — peewee 3.7.1 documentation." <http://docs.peewee-orm.com/en/latest/peewee/models.html#id4>.
- [16] oauth.net, "Oauth community site." <https://oauth.net/>.
- [17] "Flickr app garden - oauth." <https://www.flickr.com/services/api/auth.oauth.html>.
- [18] SearchMicroservices, "What is a restful api?." <https://searchmicroservices.techtarget.com/definition/RESTful-API>.
- [19] rowanwins, "vue-dropzone - a vue component for file uploads, powered by dropzone.js." <https://github.com/rowanwins/vue-dropzone>.
- [20] I. T. Union, "Universally unique identifiers (uuids)." <https://www.itu.int/en/ITU-T/asn1/Pages/UUID/uuids.aspx>.

6 Appendices

6.1 Data gathering code