Flickr Mapshup

CAB432 – Cloud Computing

Assignment 1 – API Mashup Project Report

Student: Jacinta Roberts (n9954619)

Due Date: Monday, Week 9 - 16/09/2019

Table of Contents

1.0 Introduction

1.1 Motivation:

Have you ever scrolled through photos online and wondered where that jaw-droppingly beautiful waterfall or hidden beach-side retreat was that caught your eye? Do you wish you could seek out new, local experiences or share your favourite photos from around the world with friends? Well, now you can do all of that and more with FlickrMapshup. A picture speaks a thousand words, and we've grown into a society that shares our experiences, feelings and thoughts by sharing images with one another – having a way to readily share our favourite collections alongside exactly where they were taken has more uses than you would initially think.

1.2 Overview:

FlickrMapshup aims to provide a fun, easy way to find images from around the world. This mashup combines the Google Maps JS API with the Flickr API, to allow users to search for and view photos from Flickr and display their geotagged locations onto an interactive map.

Users can search for images by keyword tag and/or username. There is also a capability to search for nearby images (within a 32km radius of their device) using the geolocation service from Google Maps and even find directions to the selected image. From the results returned from Flickr based on the user's search parameters, the Google Map will populate the area with markers to represent each image. The map can be zoomed into a desired level to show a variety of levels ranging from city, country or global views. If the user hovers over a marker, an information window popup will appear displaying the image and if the user clicks a marker, it will appear in the left sidebar and display more information. This information includes the title, date taken, owner, a link to the image on Flickr, a directions button, description and tags. If the user clicks the 'Get Directions' button it clears the map and shows the user how to get from their current location to the selected image. For a full user guide, please see Appendix A.

1.3 Flickr API:

URL: https://www.flickr.com/services/api/

Flickr provides a set of methods and API endpoints to retrieve information from the website including general image searches and user profile data, or even for logging in and modifying content.

FlickrMapshup utilises *flickr.images.search* to return a list of photos (containing information such as geotags (longitude and latitude), title, id, owner, server) from the Flickr website based on a user-provided query based on tags, username or location.

1.4 Google Maps Javascript API:

URL: https://developers.google.com/maps/ documentation/javascript/

The Maps Javascript API is used to display the photo locations and permits the customisation of maps with stylised content and imagery for display on web pages and mobile devices. It features four basic map types which can be modified using layers, controls, along with various services and additional libraries.

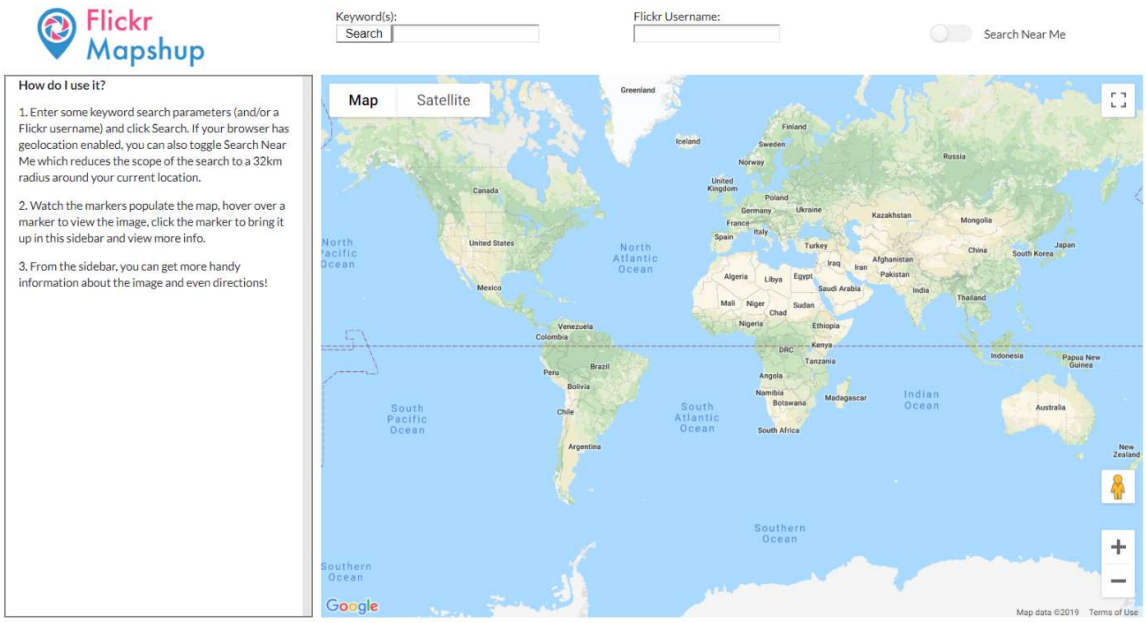
The API uses the *geocoding service* to place the location (Latitude and Longitude) of the queried photos and plot them as markers on the map. The API can also get the browser's current location using the *geolocation service*, allowing maps to display the closest images to the user (wild card keyword tags).

2.0 Use Cases

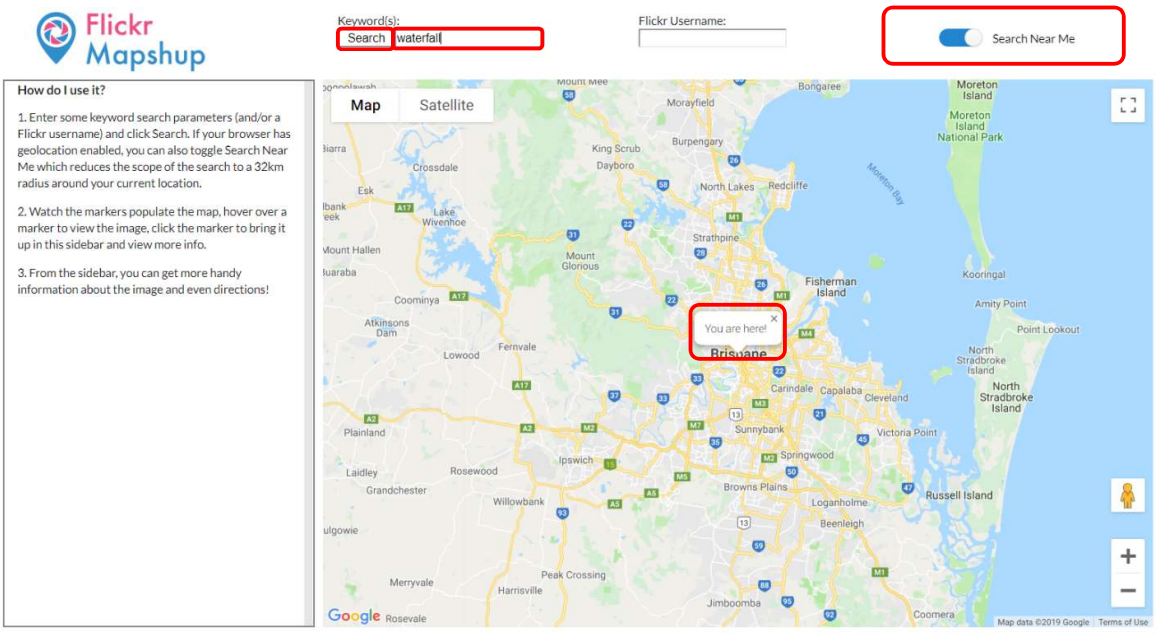
2.1 Case A:

As an avid member of the community/person who wants to try new things, I want the system to show me images taken near me so I can pursue new experiences and so that I can navigate to the location to see it.

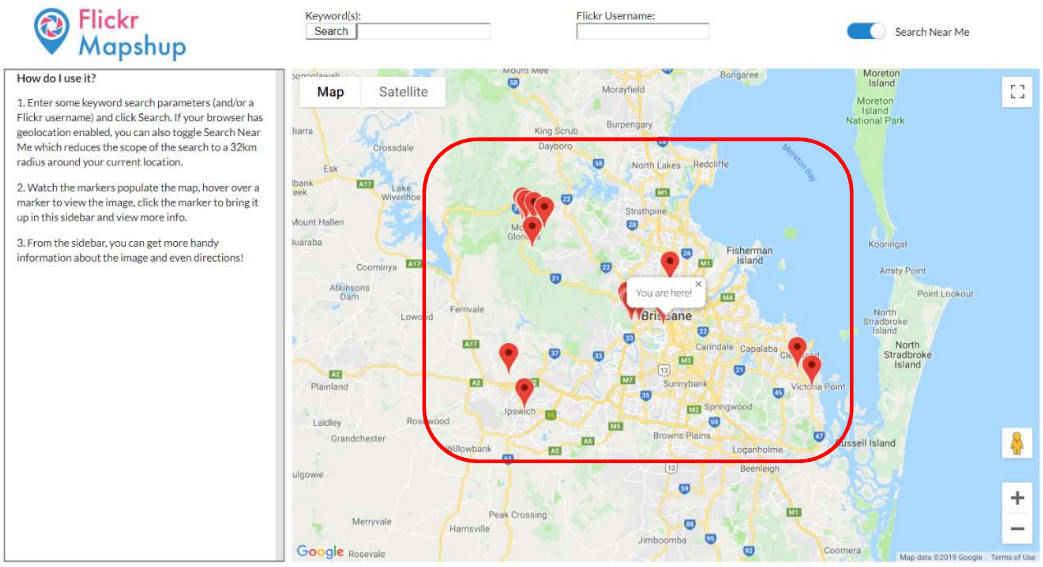
This user navigates to the index page of FlickrMapshup, with the intent to search for a specific tourist attraction, local hotspot or landmark in their area they want to find and scope out.



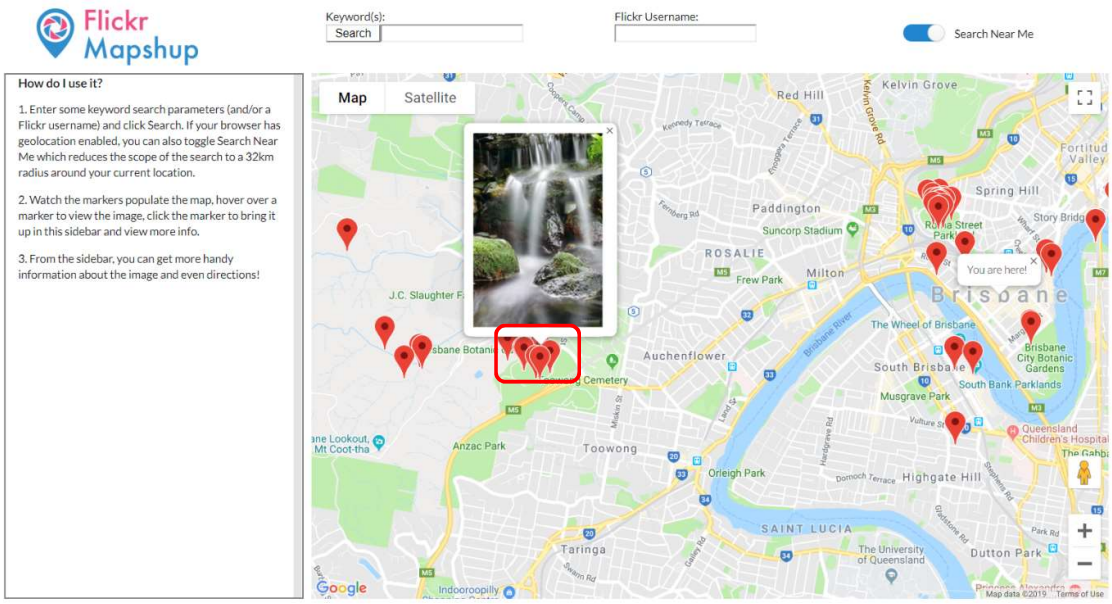
In this scenario, they decide to search for 'waterfall' near their current location (Brisbane). The map will zoom into the browser's location upon toggling the slider.



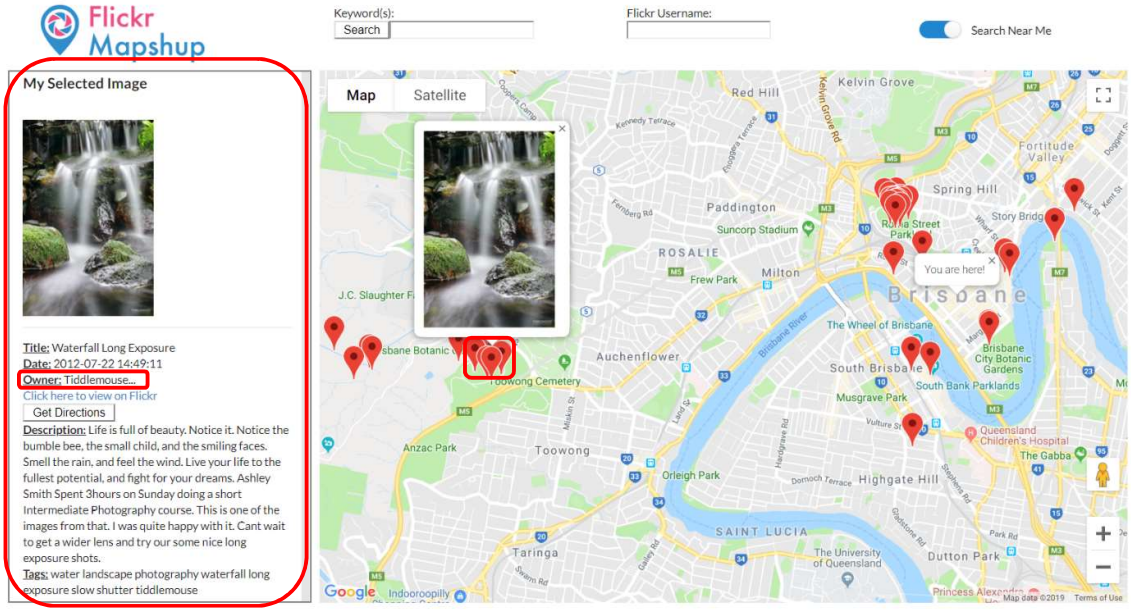
Upon clicking, 'Search' they are shown the markers for numerous geotagged images on Flickr which had the keyword 'waterfall'.



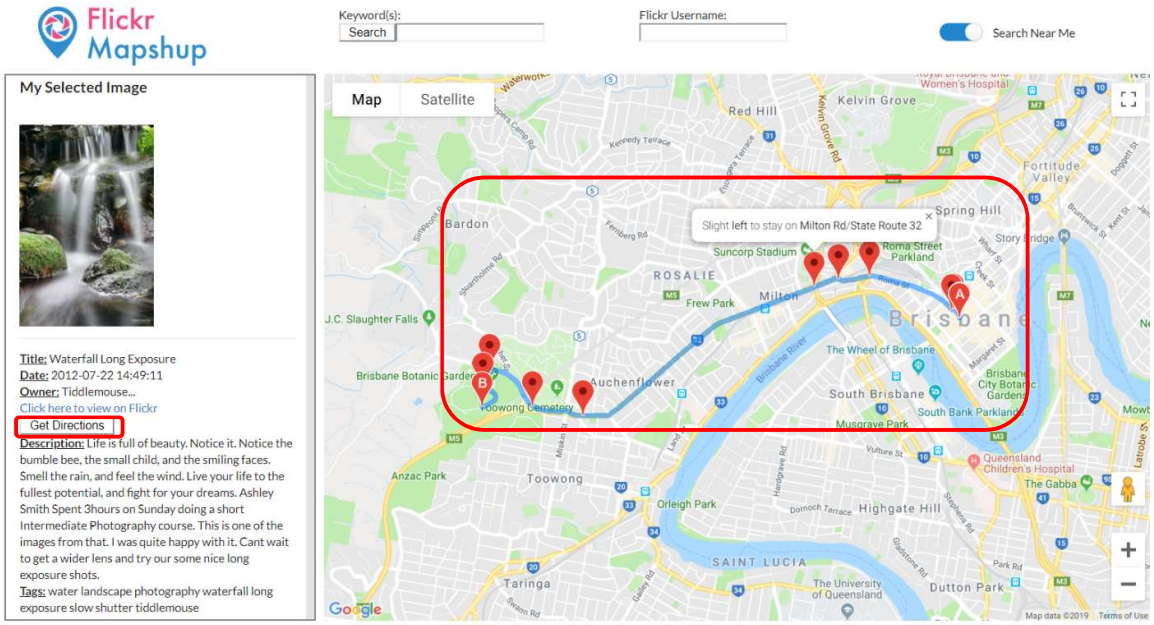
After exploring the map, the user finds a waterfall that has sparked their interest by hovering over the markers.



After clicking on this interesting marker, the user is presented with additional information in the sidebar. This user wants to find how to get there from their current location, so they click on the 'Get Directions' button in sidebar.



When this button is clicked, the other photo markers are removed and the directions (and handy instructions), are displayed on the map for the user to follow and navigate to this photo location.



The API Endpoint Calls for Case A are:

- **Flickr - Photos Search:**

https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=e03dbfe935c27fb30cc83c3b370e47bc&has_geo=1&extras=geo,description,date_taken,owner_name,tags**&tags=waterfall&lat=-27.469770699999998&lon=153.0251235&radius=32**&format=json&media=photos&nojsoncallback=1

Further information on the Flickr Photos Search API Call can be found here:
https://www.flickr.com/services/api/flickr.photos.search.html

- **Geolocation Browser API**

navigator.geolocation.getCurrentPosition(function(position) {
…
});

Further information on the Geolocation Browser API can be found here:
https://developer.mozilla.org/en-US/docs/Web/API/Geolocation_API

- **Google Maps - Directions Service:**

DirectionsRequest object literal
{
origin: infoWindow.getPosition(),
(NOTE: this value is a LatLng which marks the brower's current location and is retrieved via navigator.geolocation.getCurrentPosition)
destination: marker.position,
(NOTE: this value is a LatLng which marks the selected photo's latitude and longitude)
travelMode: 'WALKING',
(Selected by default but could be selected by the user for future improvements)
}

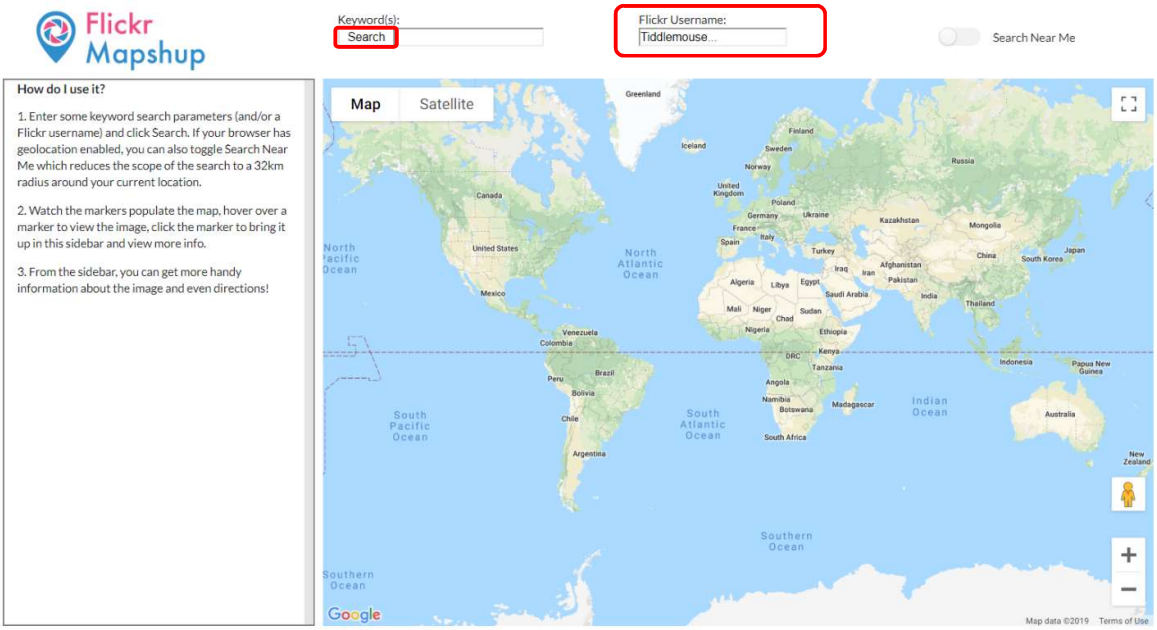
Note: If the browser does not have geolocation capability or permissions set, the 'get directions' and 'search near me' functionality will not be able to work.

Further information on the Google Maps Directions Service API Call can be found here:
https://developers.google.com/maps/documentation/javascript/directions

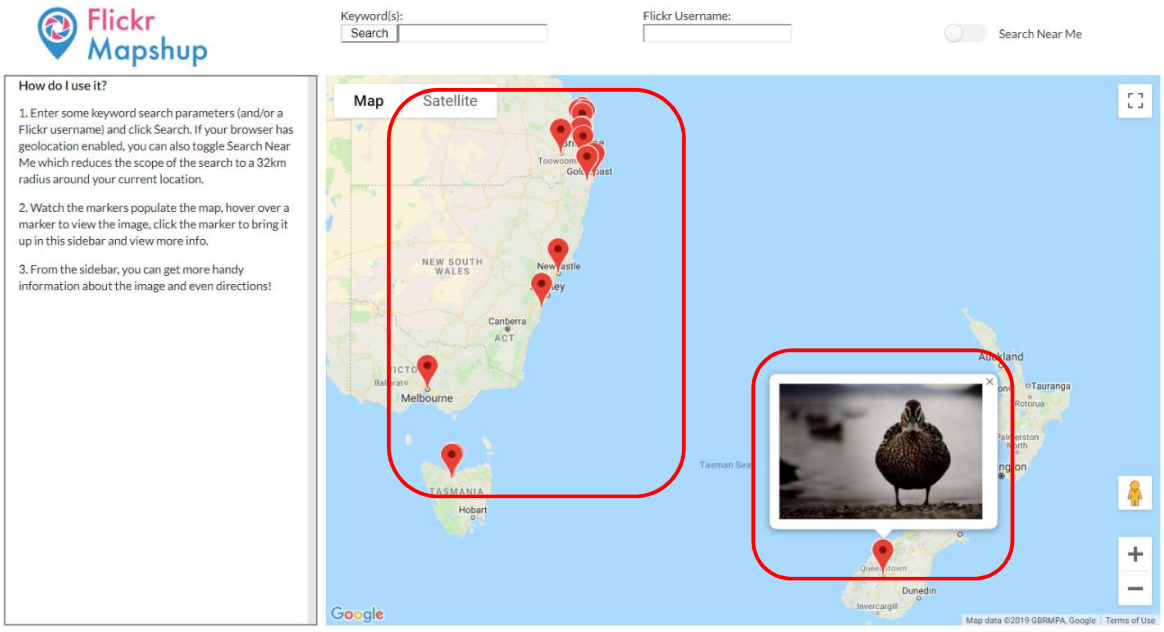
2.2 Case B:

As a frequent Flickr Member and travel blogger, I would like to view all of my images I've taken on my journeys and geotagged, on an interactive map so I can view them and share them with my audience.

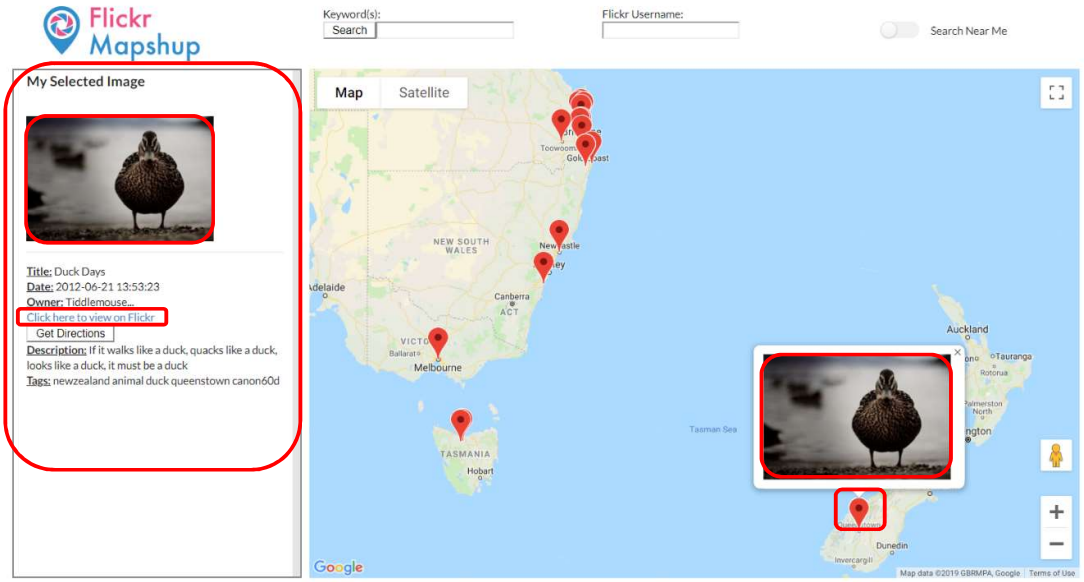
In the previous case, the owner of the selected image was 'Tiddlemouse…', suppose that user wishes to display a map of all their geotagged images on Flickr. First, they would navigate to the home page and enter 'Tiddlemouse…' into the Flickr Username box (no keywords tags are required).



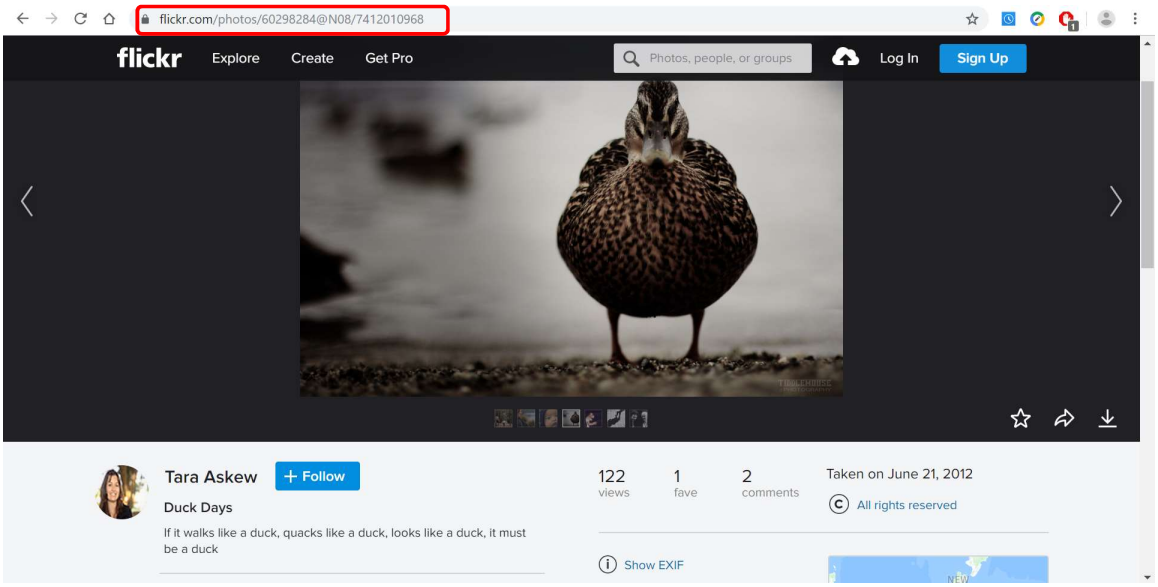
Then, they would click search and the map displays their geotagged images. They can then hover over these images to view the photo, exploring their images taken from all over the world.



Alternatively, they can click on a marker to view more information about the image such as the title, date, owner, Flickr link (which can be accessed either by clicking the photo or the visible hyperlink), directions, description and tags.



By clicking on the link to view on Flickr, the user can also see the image on the Flickr website.



The API Endpoint Calls for Case B are:

- **Flickr - Photos Search:**

https://api.flickr.com/services/rest/?method=flickr.photos.search&api_key=e03dbfe935c27fb30cc83c3b370e47bc&has_geo=1&extras=geo,description,date_taken,owner_name,tags**&username=Tiddlemouse...**&format=json&media=photos&nojsoncallback=1

Further information on the Flickr Photos Search API Call can be found here:
https://www.flickr.com/services/api/flickr.photos.search.html

3.0 Technical Breakdown

From early on in the development, it was ensured that the client-side processing was minimised and that the response information from the server was refined to reduce the payload size. Apart from rendering and displaying features of the Google Map, the server was to handle the client's requests and perform the Mashup. This meant that the data had to be retrieved from the APIs, and appropriately parsed on the server before being sent to the client. Express generator was utilised to create the boilerplate and structure for the directories of FlickrMapshup (ExpressJS, 2019). Some of these directories included, 'public' (client directory including stylesheets, client-side javascript and images) and 'routes' (single application page was held at the '/' index router). This endpoint managed the client input through forms, performed the API calls, and returned the processed results back to the client. A basic representation of the flow is presented in Figure 1 below.

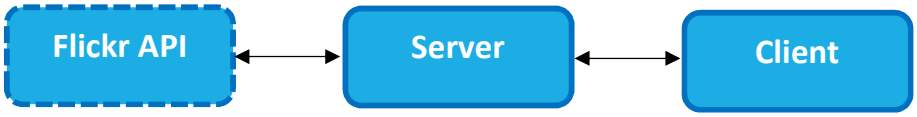


Figure 1: Overview of communication between server, client and APIs

3.1 Client-side:

The client will initiate a request to the server, in which they will be served the Index page. At this point, a request will be formed by the user (i.e search for images) that will be sent to the server via HTML forms (GET method). The form's elements (ranging from tags, username, latitude/longitude – which are sent via a hidden form if the 'Search Near Me' slider is on) will help inform the server what request to make to the API with the Axios GET (Figure 2).

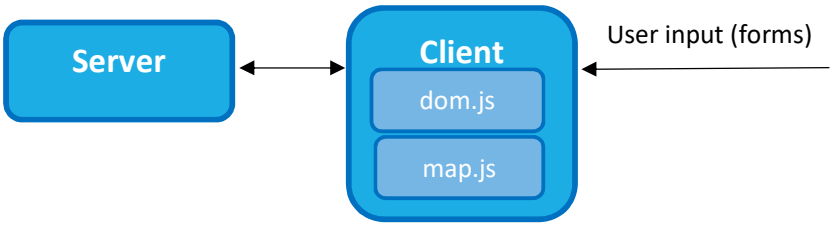


Figure 2: Client-side interactions breakdown

The user input form (tags, username, nearby) was checked in dom.js to provide feedback to the user if the input was invalid (i.e. send an alert if none of the search parameters were entered or if the search was too refined and returned no photos).

HTML5 and CSS3 with Embedded JavaScript templates were used to form the front-end. HTML5 & CSS3 are markup and stylesheet languages used structure and style (colours, layouts and fonts) to build web pages. The client-side of the JavaScript enables the interactive features of the website, which is used to get the browser's location, as well as render and place the image locations of the maps or populate the sidebar. JQuery was utilised to perform the event listening and event handling.

The response data from the server was used to create the map markers, and event listeners were used to create an interactive experience. This was able to show/hide the associated photos when the mouse was hovered over a specific marker and the sidebar was populated with specific data about the image when the marker was clicked. The Google Map was initialised with each search and

various methods were designed to alter the map accordingly. The photos were placed by using the 'addMarkers' method which created markers from the photos array received from the server and added their respective event listeners.

The styling of the basic design was provided via Bootstrap. This framework was chosen as it is the most popular HTML, CSS and JavaScript framework used to develop simple and consistent UI for web pages (Bootstrap, 2019). It contains the HTML and CSS based design template for typography, forms, buttons, navigation and other interface components. The only element that was made using another framework was the slider for 'Search Near Me' – which was made using Semantic UI as the specific style of slider desired could not be found in Bootstrap.

3.2 Server-side:

Given a request from the client, the server deals with the requests to the Flickr API and responds to the client with the relevant information. The Node server deals exclusively with the routing of incoming client requests and outgoing responses, and additional JS libraries were created (flickr.js) to assist with the handling of requests to external APIs and refining/parsing with the response data.

The server (index.js) routes to the index page (/), retrieving the search elements from the browser and then calls the 'createFlickrOptions' function defined in flickr.js to determine which url to use with Axios GET (Figure 3). Axios is a promised based HTTP client for the browser and Node JS, which is able to make HTTP requests from the Node server (Axios, 2019). Given the server's response from the Axios GET, the relevant information is processed and displayed accordingly on the client-side (i.e. if that is a search response, the map is populated with markers).

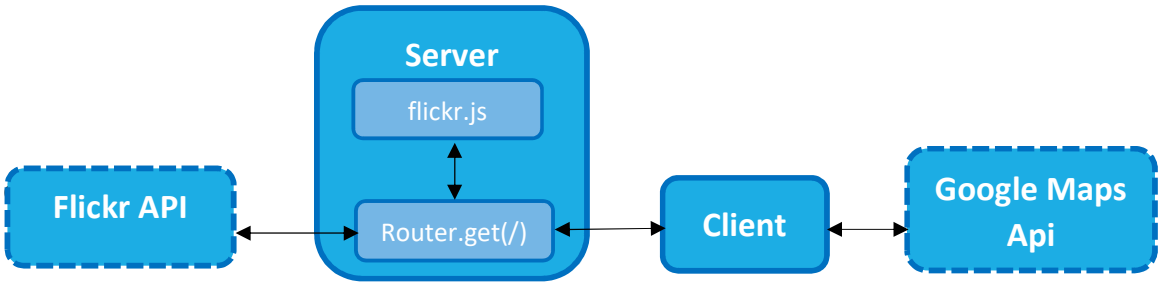


Figure 3: Server-side interactions breakdown

The server runs on the Express JS web framework (sets the routes and views easily allowing client and server-side integration – where API calls are made), which sits on top of Node JS (asynchronous Javascript runtime framework for building event-driven network apps), which is on top of on a Docker container. This Docker container runs on an EC2 instance (Ubuntu 18.04 box) and allows containerized services to run the same regardless of the environment and operating system it is running on.

Pug was used as a HTML templating engine, as the Google Maps API is a client-side library and trying to implement it on the server side proved to be difficult. Pug fully integrates with Express, a popular Node web framework and was incorporated into the project with ease (Pug, 2019). With Pug, it was not necessary to create the HTML by concatenating strings into variables, and instead Pug templates were compiled and used to generate these HTML strings based on the response data. This approached proved far more efficient rather than constructing a HTML representation on the client-side which reduced the processing load and permitted flexibility in the formatting.

3.3 Docker:

A Docker container is a lightweight, executable package of software that bundles up all of code, necessary dependencies/JS libraries, Node server and Pug templates for an application so that it is able to run quickly and reliably (Docker, 2019). Docker removes the worries of environmental inconsistency in deployment as it allows containerized services to run the same regardless of the environment and operating system it is on. The Dockerfile to build the FlickrMapshup image is depicted below in Figure 4.

```
# Use node base image
FROM node:10
# Copy app source
COPY . /src
# Set working directory to /src
WORKDIR /src
# Install app dependencies
RUN npm install
# Expose port to outside world
EXPOSE 3000
# Start command as per package.json
CMD ["npm", "start"]
```

Figure 4 – Dockerfile to construct the FlickrMapshup image

The build uses the official Docker Node image with the latest active Node JS LTS version (version 10.16.3) implemented (NodeJS, 2019). The next line copies the FlickrMapshup source code into the base directory of the image, sets the working directory, and exposes port 3000. Following this, the RUN command is used to install and include the required Node packages in the Docker image to avoid the need download them each time the image is run. Finally, the Dockerfile uses CMD *npm start* to begin the execution of the application (which is found in the package.json file as '*node ./bin/www*').

This mashup is deployed using Docker and is hosted on an Amazon EC2 running Ubuntu 18.04 – and employs a similar procedure to the sample given in the Containerising Express App Practical. It was ensured that the EC2 ran daemonised and mapped port 80 to port 3000 used by the server (with www also normalising port 3000). The application could then be accessed by navigating to the EC2's Public IP address with port 3000.

4.0 Difficulties and Limitations

4.1 Time/Lack of Experience:

The biggest limitation for me was time as I kept running into errors which would easily consume hours of my time because I was not experienced in web applications. An issue I ran into at the start when I tried implementing the Google Maps JS API without the Pug templating engine (which quickly proved to be difficult). I even had issues with selecting which sort of method to use (GET vs POST) and whether to use router vs app when handling the server/client requests. I struggled for a long time trying to get my search form with a submit button on the client-side to communicate to the server because of this. I also forgot that I could use Axios to make an API request inside of the router – which I later remembered implementing in the GitHub API practical. One more complex issue I ran into was when I was tried to dockerize my application as I kept getting the error: "*Cannot connect to the Docker daemon at unix://var/run/docker.sock. Is the docker daemon running?"* I ended up resolving this issue by dockerizing the application on my clean EC2 instance (with just Docker, NodeJS and npm installed) rather than on a WSL (Windows Subsystem for Linux).

4.2 Geolocation API Call:

If the Geolocation API call was made from an insecure origin (i.e. not over https), the functionality would fall apart – unfortunately, the Amazon EC2 does not have a secure origin and so it requires an SSL certificate. This can be amended by either generating a self-signed certificate on the VM for https, or it was also permissible to just show the geolocation functionality on another device. I will aim to work out how to generate this self-signed certificate as the Google Maps Directions Service currently also relies upon the browser location (origin) to generate the directions. This Geolocation API Call is also sometimes quite inaccurate, so it could be frustrating for users if their detected location differs from their actual location – or if they wish get directions from an alternative location.

5.0 Extensions

FlickrMapshup could be further extended by implementing additional functionality (i.e. by extending current functionality or introducing new APIs to the mashup).

5.1 Current Functionality:

- Search by Groups: Allow an additional field where users can search based upon a certain group on Flickr rather than a tag or username.
- Photos near me: Include a 'radius' input field to allows users to reduce the search radius for photos near them (it is currently fixed at the maximum of 32km to reduce the likelihood of getting no results returned).
- Directions service: The Directions Service is currently defaulted to just show the directions from an origin to destination with the fixed transport mode of walking. Clearly, improvements could be made here by allowing the user to customise this (i.e. select from 3 types of transport mode: walking, public transport or driving) or to receive live traffic updates. A significant improvement to the app could be made by allowing using to search a specific area based upon where the map is currently zoomed, or for them to specify an initial starting point, rather than defaulting to the browser's current location.

5.2 Additional API Functionality:

- Flickr Login: Allow the user to login to their Flickr account directly to easily comment, favourite, save and share photos they find.
- Exportable photo map: It would greatly increase the usability of FlickrMapshup if the user was able to select certain images to export a customised map for sharing. This feature could be used by freelance photographers or bloggers to share their latest travels and experiences with their audiences more easily. Alternatively, it could even be used by travel agents to showcase the highlighted sights of their advertised tours or for promoting social/community events nearby.
- Instagram: This app could also incorporate the Instagram API to increase the number of photos in the collection and results returned to the user. However, the Instagram API's access is restricted, and a request would need to be submitted to Facebook to apply for usage.

6.0 Testing

6.1 Error Handling:

Testing was conducted to ensure that the app performed as expected and errors were appropriately handled. Alterations to the code were made incrementally, with a focus on effectiveness over elegance initially. Later, the code was refactored to improve the modularity and reduce the risk of errors. Various errors were encountered throughout the development including logic errors, syntax errors, server errors and runtime errors.

Logic errors were difficult to fix, as the code was functioning but not in the way I expected it to. To diagnose logic errors, a test plan was formed. From the test cases on the following page, it was observed that all functionality implemented worked as intended. Syntactic errors were easily prevented by the use of a code editor such as VS Code. Runtime errors were identified by unexpected behaviours, and an error was logged in the console. Server errors were similar with instead an error status code being logged. These were amended by identifying the error from the log and redesigning the code if necessary. Checked exceptions (i.e. error could be expected in certain cases) were handled with catch statements.

6.2 Test Cases:

ID	Purpose	Expected Outcome	Result
Search Form			
1	Search for images with no keywords, no username entered and 'Search Near Me' toggled off.	Error – Alert user to input some search parameters.	**Passed**
2	Search for images with no keywords, no username entered and 'Search Near Me' toggled on.	Show nearby images within a 32km radius (wildcard tags).	**Passed**
3	Search with valid username and 'Search Near Me' toggled off.	Show images owned by the specified username (global view – shrinks to fit all markers).	**Passed**
4	Search with valid username and 'Search Near Me' toggled on.	Show nearby images within a 32km radius owned by the specified username.	**Passed**
5	Search with invalid username	Error – Alert user that no photos were found for their search parameters.	**Passed**
6	Search with tags and 'Search Near Me' toggled off.	Show images with the specified tag (global view – shrinks to fit all markers).	**Passed**
7	Search with tags and 'Search Near Me' toggled off.	Show images within a 32km radius with the specified tag.	**Passed**
8	Search with tags, username and 'Search Near Me' toggled off	Show images that are owned by the specified username and have the specified tags.	**Passed**
9	Search with tags, username and 'Search Near Me' toggled on	Show images within a 32km radius if they match all 3 criteria.	**Passed**
10	Search that is too refined and does not return any photos matching the criteria	Error – Alert the user that no photos were found that matched their search.	**Passed**
Map			
11	Hovering over a photo marker on the map	When the cursor is over the marker, the corresponding image is shown. When the cursor moves away the image is hidden.	**Passed**

12	Clicking a photo marker on the map	When the marker is clicked, additional information about the image appears in the side bar (title, date, owner, link to Flickr, Get Directions button, description and tags).	**Passed**
13	Moving the map around to explore	Moving the map around does not cause the markers to disappear.	**Passed**
14	Changing the state of the 'Search Near Me' slider.	Changing the state of the slider causes all currently existing markers to be removed from the map (prepares for new search).	**Passed**
Directions			
15	Pressing the 'Get Directions' button in the side bar, with 'Search Near Me' on	All other photo markers are cleared and the directions (from current browser location) to the selected image's location are displayed.	**Passed**
16	Pressing the 'Get Directions' button in the side bar, with 'Search Near Me' off (when a photo from the same continent was clicked on)	As above.	**Passed**
17	Pressing the 'Get Directions' button in the side bar, with 'Search Near Me' off (when a photo from a different continent was clicked on)	Error – Invalid route. Alert the user with the following message, "Directions request failed due to ZERO_RESULTS."	**Passed**

7.0 References

Axios. (2019). *Axios.* Retrieved from github.com: https://github.com/axios/axios

Bootstrap. (2019). *Introduction.* Retrieved from getbootstrap.com: https://getbootstrap.com/docs/4.3/getting-started/introduction/

Docker. (2019). *What is a Container?* Retrieved from Docker.com: https://www.docker.com/resources/what-container

ExpressJS. (2019). *Express Application Generator.* Retrieved from expresssj.com: https://expressjs.com/en/starter/generator.html

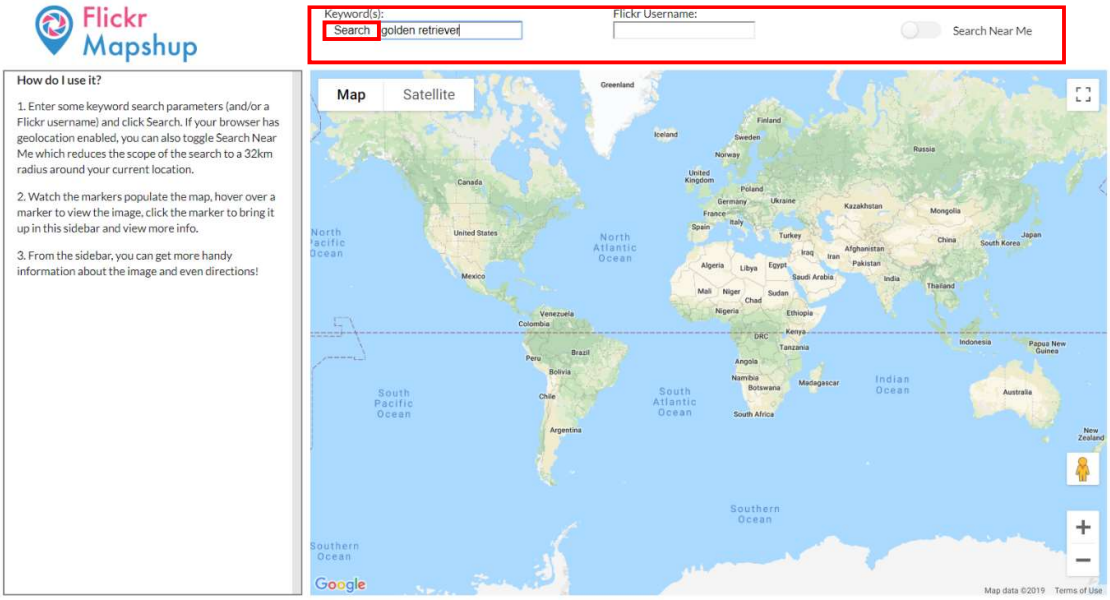
NodeJS. (2019). *Official Docker Image for Node.js.* Retrieved from Github.com: https://github.com/nodejs/docker-node

Pug. (2019). *Getting Started.* Retrieved from pugjs.org: https://pugjs.org/api/getting-started.html

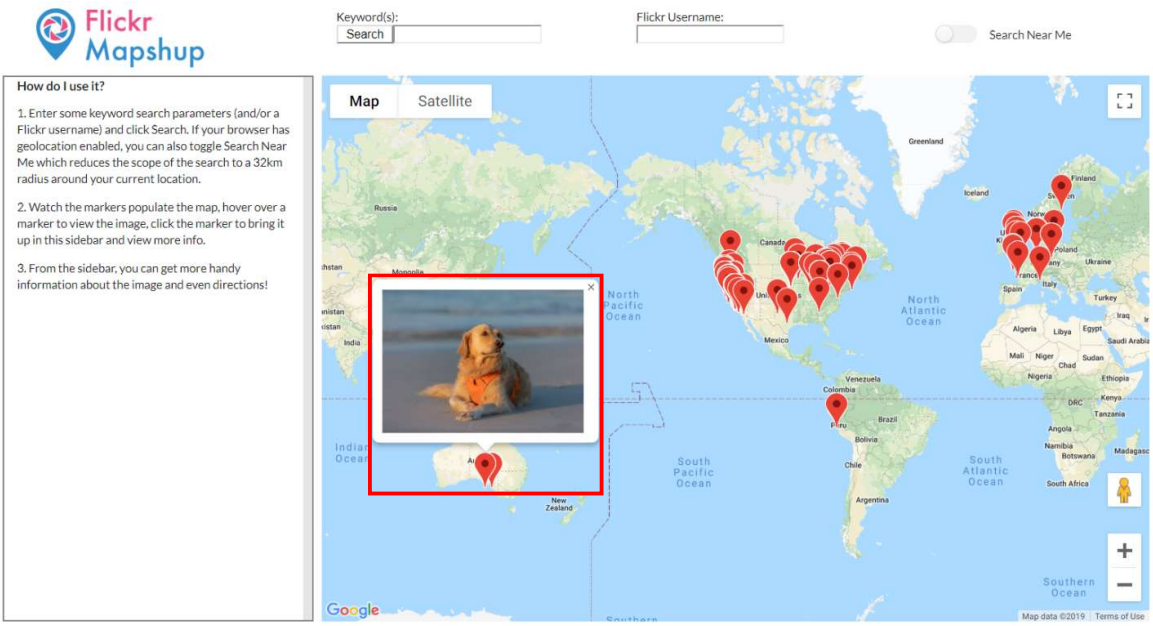
8.0 Appendix

7.1 Appendix A – User Guide

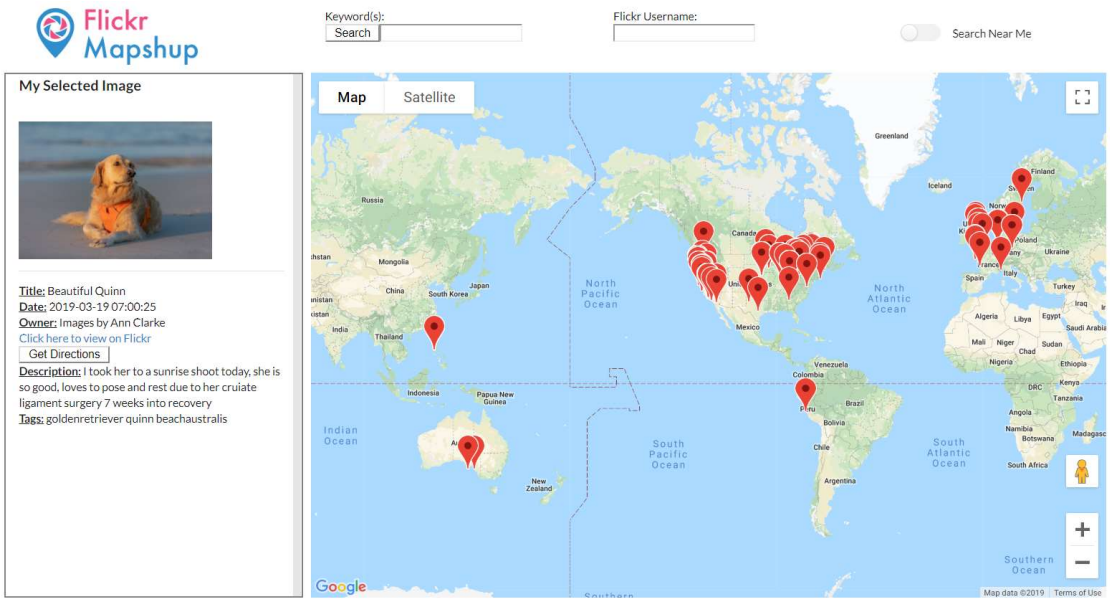
1. Enter some tags into the keyword(s) form and/or a Flickr username. If your browser has geolocation enabled, you can also toggle the Search Near Me button which reduces the scope of the search to a 32km radius around your current location. Press 'Search'.



2. The markers populate the map corresponding to the search results of the query. Hover over a marker to view the image.



3. Click the marker to bring it up into the sidebar and to view more information about the photo (including title, date, owner, link to Flickr, directions, description and tags).



4. Click or hover over other markers available in this search, make further searches or get the directions to the image by pressing the 'Get Directions' button in the sidebar. Below, I have clicked the button to see the directions to this beautiful golden retriever – he's far away but well worth it.