

CAB432 Cloud Computing

Getting Started with Node, JS and Mashups

This prac and the following sheet provide an extensive introduction to node and to relevant JS and the consumption of web services. Much of the material will be directly usable as part of Assignment 1, and by the end of week 5 you should have a very good basis for working on that project.

There is a mix of material here. We will begin with some discussion of Node, which builds on some of the examples in the lectures. The main goal here is to provide a proper introduction to Node applications and to the package manager NPM for those who haven't used them before. We will also use node to provide a basic static webserver to serve some client side Javascript examples and in the end we will use this on a local machine to serve the Flickr example.

Note: Those students familiar with node and express can skip or skim over most of these introductory sections. However, everyone will benefit from the Flickr exercises and S3 hosting. It is crucial for Assignment 1 that people look at the Flickr Server Side exercises next week.

The sections of this prac sheet are as follows:

- ***Introducing Node and NPM – a basic introduction for those new to Node and NPM.***
- ***Flickr exercises Part I – the Client Side (see the next sheet for the server side)***
- ***Static Hosting on S3***

The follow-up sheet for Prac 4 will be devoted to the server side and you will create an Express app for this Flickr problem and explore some extensions.

Introducing Node and NPM

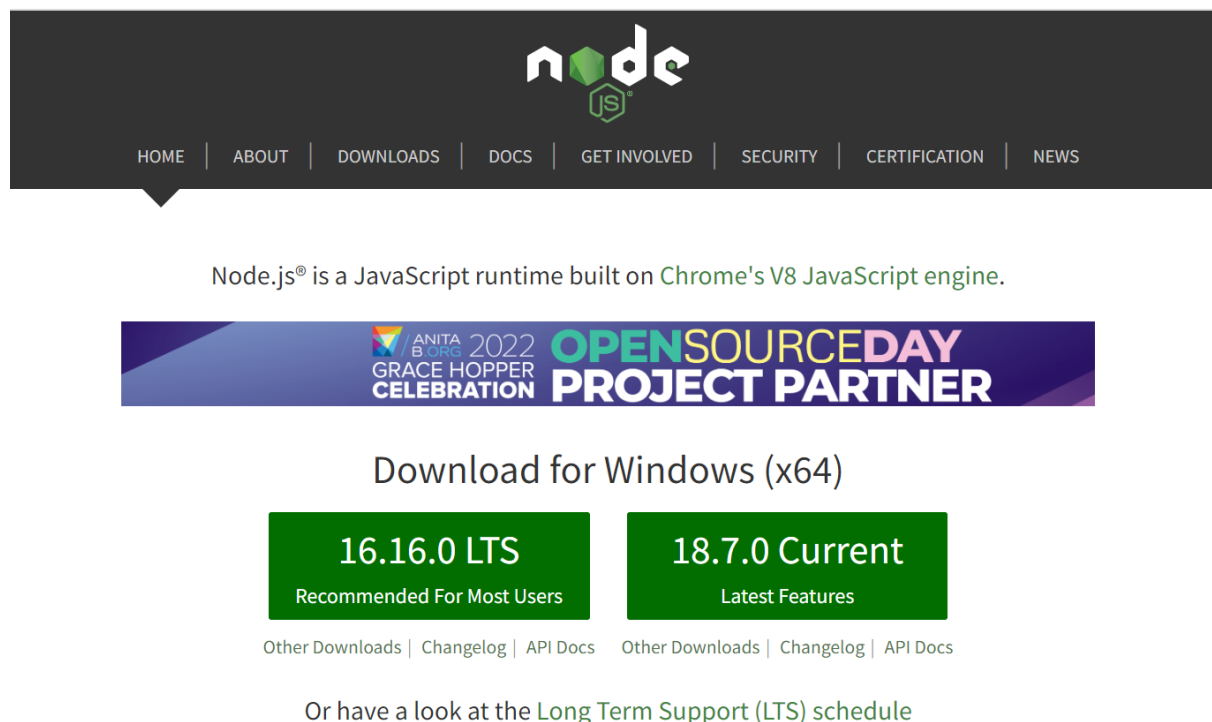
This section is intended for people with a limited background with Node. If you already have some node experience you can look at this as a quick revision or you can just skim over it to see if there is anything important that you need to look at more carefully. Many of these simpler exercises are run in a **node REPL** – either on your laptop or a lab machine or VM, or from one of the online sites like repl.it: <https://repl.it/languages/nodejs>.

The getting **started instructions below** are written mainly for Windows, and for the Ubuntu install, look at: <https://github.com/nodesource/distributions/blob/master/README.md#debinstall>. When using node.js under Linux please remember that in some installs the command may be `nodejs` due to a clash with an existing application.

We also provide instructions for the use of a **portable node** which you can run from a USB drive. You will find this in a separate file in this directory, and it works surprisingly well. In the next section, we are assuming that you are doing a fresh install on Windows. Under WSL you may follow the same instructions as for the relevant flavour of Linux, in my case Ubuntu 18.04, but here you can use later versions as well. We update the node versions regularly, but so do the people in the node project.

Getting started with Node:

Installation instructions for Node are readily available at the site: <https://nodejs.org/>.



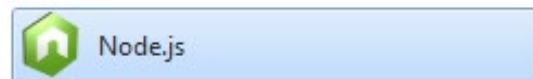
The screenshot shows the Node.js website interface. At the top is the Node.js logo and a navigation bar with links: HOME, ABOUT, DOWNLOADS, DOCS, GET INVOLVED, SECURITY, CERTIFICATION, and NEWS. Below the navigation bar is a statement: "Node.js® is a JavaScript runtime built on Chrome's V8 JavaScript engine." Underneath this is a banner for "ANITA B.ORG 2022 GRACE HOPPER CELEBRATION" and "OPENSOURCE DAY PROJECT PARTNER". The main section is titled "Download for Windows (x64)" and contains two green buttons: "16.16.0 LTS Recommended For Most Users" and "18.7.0 Current Latest Features". Below these buttons are links for "Other Downloads | Changelog | API Docs" for both versions. At the bottom of this section is a link: "Or have a look at the Long Term Support (LTS) schedule".

For Windows, use the download button and run the installer, accepting the defaults. There is a link here to "Other Downloads", and this is probably the best way of dealing with the Mac OS X install. For Linux, I think you are probably better to use your platform's package manager as discussed above. Please note that you will probably have to come to grips with a

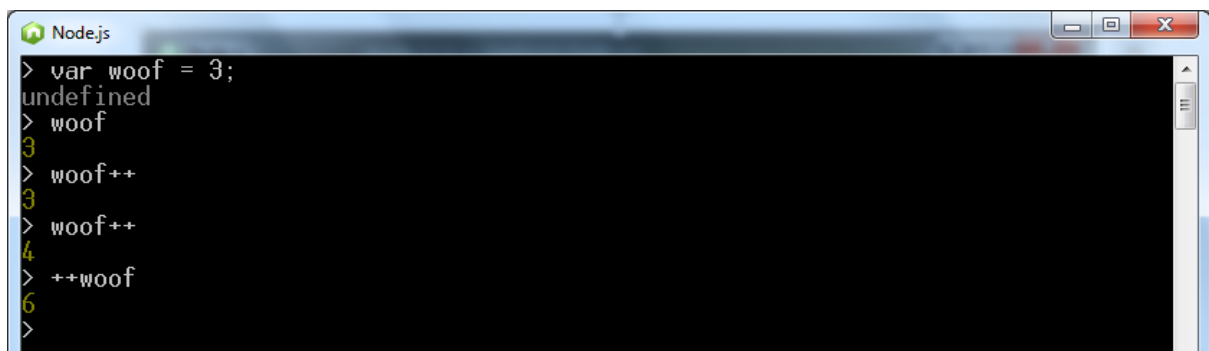
Node install under Ubuntu in order to construct the Docker container for assignment 1, but we won't make this part of this set of exercises.

The Node nomenclature is different from the usual – you will see the LTS and the Current releases, each with an explicit version in the form vX.x.x. The LTS stands for Long Term Support, and the promise is that the current LTS will be supported for some time period and that the Current, later release will then transition to be the next LTS. For consistency and stability we will use the LTS as the standard for this unit. However, you may use any version from around v10.x.x onwards without too many issues. The the later versions appear stable in any case.

Working with Node under Windows is very straightforward. For convenience, I normally work with the bash shell provided by Git (relying on the Windows install) or WSL2 (<https://docs.microsoft.com/en-us/windows/wsl/install-win10>; using a specific Ubuntu install). I can then use the Node commands at the command line as needed. If you like, you can also keep an interpreter window open to check out some of your basic JS. To do this, select the node.js option with the green icon as shown



The **resulting window** allows you to enter fragments of code and to see what might result. This shows (entirely correct) interactions with a variable called `woof`, that we initially define to have a value of 3. We will work with Node as a convenient and flexible server-side technology, but you may also see it as an interactive interpreter to allow you to deal with the basics.

A screenshot of a Node.js REPL window. The window has a title bar with "Node.js" and standard Windows window controls. The main area is a black console with white text showing the following interactions:

```
> var woof = 3;
undefined
> woof
3
> woof++
3
> woof++
4
> ++woof
6
>
```

A console for JS testing and development can also be supplied through developer toolsets available for the main browsers:

- The Firefox developer tools: <https://developer.mozilla.org/en-US/docs/Tools>
- The Chrome developer tools: <https://developers.google.com/web/tools/chrome-devtools/>

The **simplest editor to set up and use is VS code**, which automatically detects your node installation.

In what follows, we will explore a series of Node examples, which will rapidly increase in complexity. The **first of these will be just the simple server**. We will then move on to a much

more sophisticated static page server, which will allow us to explore the operation of the Node Package Manager or NPM, an essential tool for working with Node. The static page server will also allow us to explore some client-side JS examples. JavaScript language features and libraries are introduced in situ as needed, but chat with your tutor if you are unsure.

Working through some Node examples:

We assume at this point that you have successfully installed node and that you have chosen an editor and made it talk to the Node installation. You are now ready to proceed. Choose a Node working directory. From within this directory, download the basic server Node examples provided with this prac. The file basicservers.zip includes examples from the week 3 and week 4 lectures. The key issue is that none of these examples require the use of packages outside the core. The files are:

- simple.js – the basic HelloWorld example (port 2798)
- events.js – an example which handles events explicitly (port 2799)
- logheaders.js – HelloWorld server, but logging the headers (port 2800)
- asyncfile.js – the server which serves simple.js on request (port 2801)
- client.js/server.js – the tcp connection applications, linked on port 2802.

In each case, you may run the server using the command:

```
node server-name
```

and you may then connect at the address: `http://localhost:{port}`.

In the case of client.js, this will connect without the need for a browser but the launch is the same. If you are inexperienced or need a refresher, please work through at least some of these and confirm that you are able to make sense of what is happening in each case.

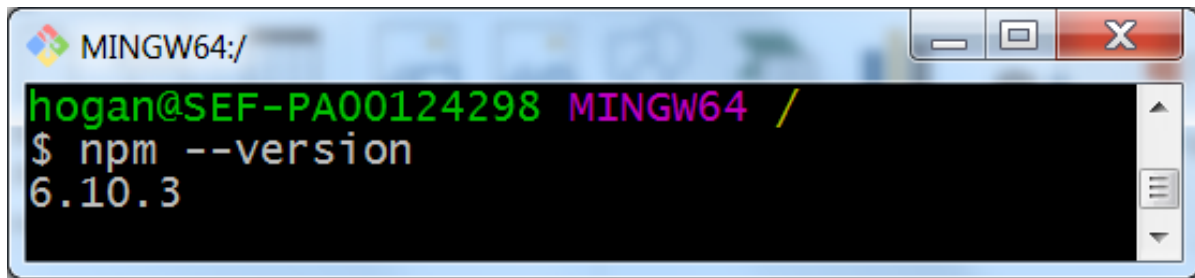
Introducing NPM and more sophisticated Node:

The Node Package Manager NPM allows us to manage the module dependencies for a Node application. This is an essential part of non-trivial Node development as otherwise we would do little else but manage package dependencies. In what follows we will step through some basic NPM and use it in the context of some simple Node applications.

The first step is to ensure that NPM – installed with Node – is in fact up to date. To do this, run:

```
npm install npm -g
```

This shows the NPM command `install`, here applied to the NPM package itself. The `-g` flag indicates that the package should be global and hence available to all your Node applications. The package manager itself is located as part of the Node installation and so is available on the application path. After the update, you should see a version display something like this, though the actual version number will be much higher:

A terminal window titled 'MINGW64:/' showing the command 'npm --version' being executed. The output is '6.10.3'. The prompt is 'hogan@SEF-PA00124298 MINGW64 /'.

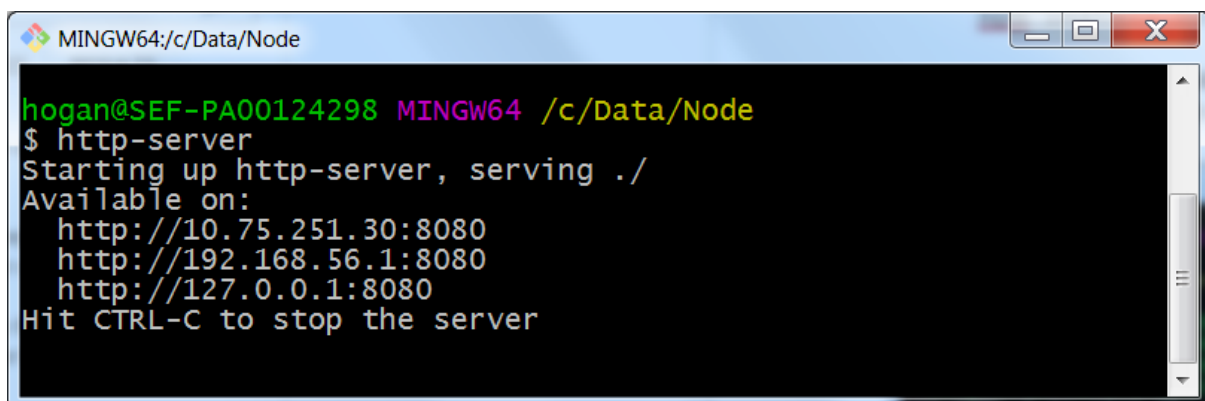
We will now consider a Node application, a basic http server for static web pages which builds on top of the simple servers we have played with over the first lecture and in the earlier exercises. We will install the Node http-server package: <https://www.npmjs.com/package/http-server>. The subsequent application may be handled in a number of ways, but we will operate within an application directory.

We will find it most useful to work with this Node server as an application, in much the same way as we might with Python's simple server. The advantage is that we can then serve pages from the working directory. While there is an argument for installing things globally to save space, generally we don't. Just install locally so that we are completely aware of any project dependencies:

```
npm install -g http-server
```

You will now see a package-lock.json file which shows the packages used in the application, and the node_modules directory. It is worth looking in the node_modules directory if you have never explored a node app before. There is a lot happening that is hidden from view.

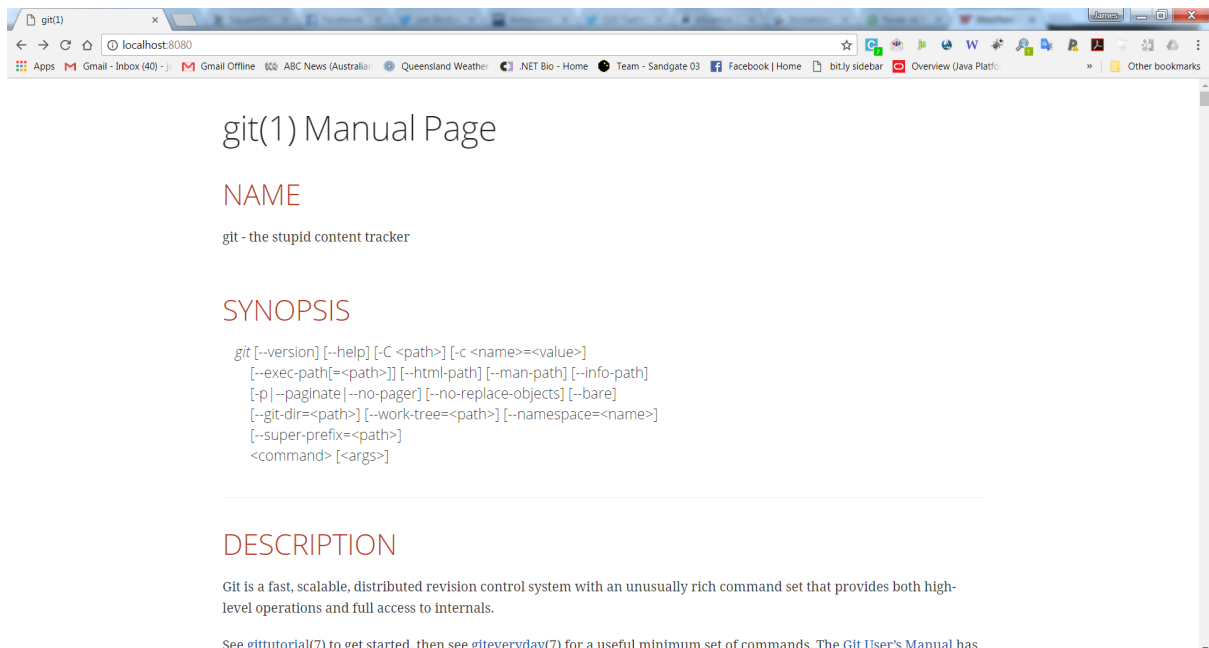
Once the package is installed, we can start the server straightforwardly as shown. As it is an application, we do not need to use Node to invoke it:

A terminal window titled 'MINGW64:/c/Data/Node' showing the command 'http-server' being executed. The output is: 'Starting up http-server, serving ./', 'Available on:', 'http://10.75.251.30:8080', 'http://192.168.56.1:8080', 'http://127.0.0.1:8080', and 'Hit CTRL-C to stop the server'. The prompt is 'hogan@SEF-PA00124298 MINGW64 /c/Data/Node'.

By default, the server will serve content from node_modules/http-server. You can override the directory from which content will be served by entering the path of the desired folder. For example, in the following snippet content is served from a directory named "c:/cab432/week03/basicservers/basicservers"

```
http-server c:/cab432/week03/basicservers/basicservers
```

In the absence of any sensible content, you will see a simple empty directory listing from the server. In my case, I have grabbed a simple `index.html` file from the git docs (included in the basic servers zip), and the resulting image is as shown on the next page.



If you experience any difficulties then work locally within the `node_modules/http-server` directory. This time you will need to use Node to invoke the server using the command below:

```
node bin/http-server
```

Working with Flickr (Client Side)

*This exercise is about using client side JavaScript to hit a **REST API** and to display the results on a page. This is a prelude to subsequent exercise when we do all this on the **server side**, which is what we require for the assignment. Please use your own judgement as to how much of this you need to work through, but most people will benefit from the exercise. Everyone, regardless of experience, should take a close look at the **server side Flickr exercises on the next prac sheet.***

The Flickr API Exercises. Part I (client side):

Having now set up a decent Node server, one that we can rely upon to deal with more complicated web pages, we test out some of these ideas by grabbing some photos from Flickr.

Our exercise here is to use the Flickr API in the manner considered in the lecture. To organise a Flickr key, you need to go to:

https://www.flickr.com/services/api/misc.api_keys.html

You will then be prompted to create a Flickr account or sign in. Once that is concluded, you will progress to

<https://www.flickr.com/services/apps/create/apply>

to make a choice between commercial and non-commercial API usage, and you will select non-commercial, and explain that it is for a course. You will then get a key pair, which you need to record somewhere. In the next worksheet you will adapt this client-side exercise to place all of this code in the Node application itself, and serve the resulting page(s) directly.

We will mainly use the standard API key listed to work with the REST API for image search. We begin by following the lecture example, replacing the symbol 'xxx' in the following link to obtain results the default XML format:

https://api.flickr.com/services/rest/?&method=flickr.photos.search&api_key=XXX&tags=golden-retriever

The JSON alternative is similar, requiring only an additional parameter. We will tend to prefer JSON as it is easy to read, parsing it is simple using the available libraries, and JSON is really the standard that people now work with:

https://api.flickr.com/services/rest/?&method=flickr.photos.search&api_key=XXX&tags=golden-retriever&per-page=50&format=json

Typical results from these queries are shown on the next page. Use these queries to test your new key. Make sure you understand what you are about to see by taking a look at the guide here:

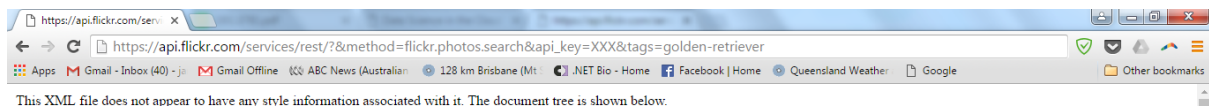
<http://www.flickr.com/services/api/response.json.html>.

Now, by default the JSON responses come wrapped in a JSON callback function, for historical reasons that we can very happily avoid discussing. We can get rid of this by the use of an additional nojsoncallback=1 option:

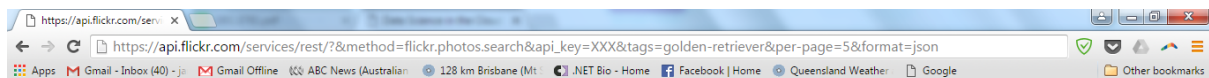
https://api.flickr.com/services/rest/?&method=flickr.photos.search&api_key=XXX&tags=golden-retriever&per-page=50&format=json&nojsoncallback=1

We can also eliminate videos from our results by the use of the media=photos option:

https://api.flickr.com/services/rest/?&method=flickr.photos.search&api_key=XXX&tags=golden-retriever&per-page=50&format=json&media=photos&nojsoncallback=1



```
<rsp stat="ok">
  <photos page="1" pages="1537" perpage="100" total="153686">
    <photo id="28675401270" owner="33159513@005" secret="0b8c8630c2" server="8562" farm="9" title="First swimlesson" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28344343033" owner="97030819@006" secret="f466d5a55a" server="8254" farm="9" title="Golden's Day (1)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28344342473" owner="97030819@006" secret="e2f74bfeaf" server="8795" farm="9" title="Golden's Day (4)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28675053630" owner="97030819@006" secret="2170a7ecc3" server="8735" farm="9" title="Golden's Day (5)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28855912372" owner="97030819@006" secret="34e89fe435" server="8016" farm="9" title="Golden's Day (6)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28344342183" owner="97030819@006" secret="a7113566bc" server="8400" farm="9" title="Golden's Day (8)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28855911732" owner="97030819@006" secret="4ce98596ac" server="7714" farm="8" title="Golden's Day (10)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884383621" owner="97030819@006" secret="610af87763" server="8279" farm="9" title="Golden's Day (11)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28855911352" owner="97030819@006" secret="ed325363ab" server="8732" farm="9" title="Golden's Day (13)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884383161" owner="97030819@006" secret="7289e69596" server="8817" farm="9" title="Golden's Day (12)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884381691" owner="97030819@006" secret="8ac7edd1b" server="7651" farm="8" title="Golden's Day (15)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28928614796" owner="97030819@006" secret="c464b0c790" server="8656" farm="9" title="Golden's Day (16)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884380711" owner="97030819@006" secret="82087ca3dd" server="8518" farm="9" title="Golden's Day (17)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884380211" owner="97030819@006" secret="57c9a7cb56" server="8671" farm="9" title="Golden's Day (18)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28675049780" owner="97030819@006" secret="d6002c2e20" server="8566" farm="9" title="Golden's Day (19)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884378751" owner="97030819@006" secret="cdbe84f283" server="8740" farm="9" title="Golden's Day (22)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28855908482" owner="97030819@006" secret="af10f879d4" server="8665" farm="9" title="Golden's Day (24)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28884378191" owner="97030819@006" secret="a9b443477f" server="8865" farm="9" title="Golden's Day (23)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28341817034" owner="97030819@006" secret="a88f684a54" server="8747" farm="9" title="Golden's Day (25)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28341816994" owner="97030819@006" secret="cfc8c55042" server="7560" farm="8" title="Golden's Day (31)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28341815944" owner="97030819@006" secret="21e04d32d0" server="8535" farm="9" title="Golden's Day (33)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28341816314" owner="97030819@006" secret="2788c23fab" server="8572" farm="9" title="Golden's Day (32)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28344337983" owner="97030819@006" secret="cfa48a4c86" server="8656" farm="9" title="Golden's Day (34)" ispublic="1" isfriend="0" isfamily="0"/>
    <photo id="28675048220" owner="97030819@006" secret="04ef2ec81d" server="8702" farm="9" title="Golden's Day (35)" ispublic="1" isfriend="0" isfamily="0"/>
  </photos>
</rsp>
```

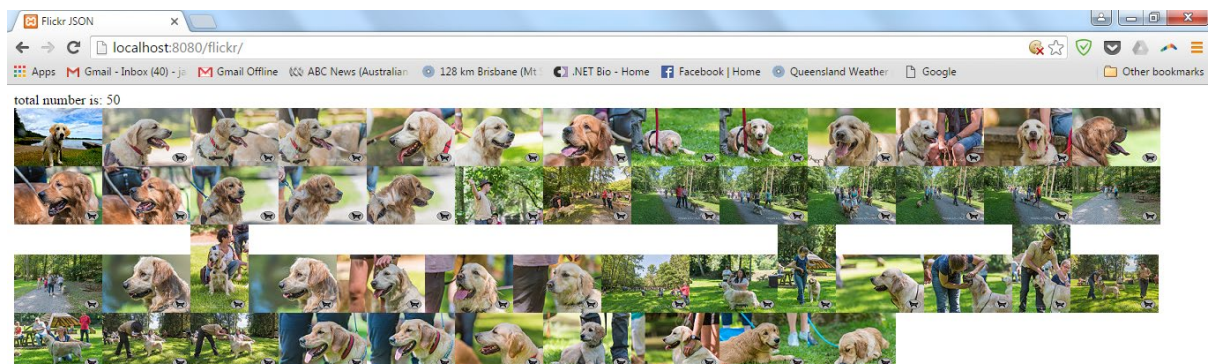


```
jsonFlickrApi({
  "photos": {
    "page": "1",
    "pages": "1537",
    "perpage": "100",
    "total": "153686",
    "photo": [
      {
        "id": "28675401270",
        "owner": "33159513@005",
        "secret": "0b8c8630c2",
        "server": "8562",
        "farm": "9",
        "title": "First swimlesson",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/33159513@005/28675401270/"
      },
      {
        "id": "28344343033",
        "owner": "97030819@006",
        "secret": "f466d5a55a",
        "server": "8254",
        "farm": "9",
        "title": "Golden's Day (1)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344343033/"
      },
      {
        "id": "28344342473",
        "owner": "97030819@006",
        "secret": "e2f74bfeaf",
        "server": "8795",
        "farm": "9",
        "title": "Golden's Day (4)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344342473/"
      },
      {
        "id": "28675053630",
        "owner": "97030819@006",
        "secret": "2170a7ecc3",
        "server": "8735",
        "farm": "9",
        "title": "Golden's Day (5)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28675053630/"
      },
      {
        "id": "28855912372",
        "owner": "97030819@006",
        "secret": "34e89fe435",
        "server": "8016",
        "farm": "9",
        "title": "Golden's Day (6)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28855912372/"
      },
      {
        "id": "28344342183",
        "owner": "97030819@006",
        "secret": "a7113566bc",
        "server": "8400",
        "farm": "9",
        "title": "Golden's Day (8)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344342183/"
      },
      {
        "id": "28855911732",
        "owner": "97030819@006",
        "secret": "4ce98596ac",
        "server": "7714",
        "farm": "8",
        "title": "Golden's Day (10)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28855911732/"
      },
      {
        "id": "28884383621",
        "owner": "97030819@006",
        "secret": "610af87763",
        "server": "8279",
        "farm": "9",
        "title": "Golden's Day (11)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884383621/"
      },
      {
        "id": "28855911352",
        "owner": "97030819@006",
        "secret": "ed325363ab",
        "server": "8732",
        "farm": "9",
        "title": "Golden's Day (13)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28855911352/"
      },
      {
        "id": "28884383161",
        "owner": "97030819@006",
        "secret": "7289e69596",
        "server": "8817",
        "farm": "9",
        "title": "Golden's Day (12)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884383161/"
      },
      {
        "id": "28884381691",
        "owner": "97030819@006",
        "secret": "8ac7edd1b",
        "server": "7651",
        "farm": "8",
        "title": "Golden's Day (15)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884381691/"
      },
      {
        "id": "28928614796",
        "owner": "97030819@006",
        "secret": "c464b0c790",
        "server": "8656",
        "farm": "9",
        "title": "Golden's Day (16)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28928614796/"
      },
      {
        "id": "28884380711",
        "owner": "97030819@006",
        "secret": "82087ca3dd",
        "server": "8518",
        "farm": "9",
        "title": "Golden's Day (17)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884380711/"
      },
      {
        "id": "28884380211",
        "owner": "97030819@006",
        "secret": "57c9a7cb56",
        "server": "8671",
        "farm": "9",
        "title": "Golden's Day (18)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884380211/"
      },
      {
        "id": "28675049780",
        "owner": "97030819@006",
        "secret": "d6002c2e20",
        "server": "8566",
        "farm": "9",
        "title": "Golden's Day (19)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28675049780/"
      },
      {
        "id": "28884378751",
        "owner": "97030819@006",
        "secret": "cdbe84f283",
        "server": "8740",
        "farm": "9",
        "title": "Golden's Day (22)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884378751/"
      },
      {
        "id": "28855908482",
        "owner": "97030819@006",
        "secret": "af10f879d4",
        "server": "8665",
        "farm": "9",
        "title": "Golden's Day (24)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28855908482/"
      },
      {
        "id": "28884378191",
        "owner": "97030819@006",
        "secret": "a9b443477f",
        "server": "8865",
        "farm": "9",
        "title": "Golden's Day (23)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28884378191/"
      },
      {
        "id": "28341817034",
        "owner": "97030819@006",
        "secret": "a88f684a54",
        "server": "8747",
        "farm": "9",
        "title": "Golden's Day (25)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28341817034/"
      },
      {
        "id": "28341816994",
        "owner": "97030819@006",
        "secret": "cfc8c55042",
        "server": "7560",
        "farm": "8",
        "title": "Golden's Day (31)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28341816994/"
      },
      {
        "id": "28341815944",
        "owner": "97030819@006",
        "secret": "21e04d32d0",
        "server": "8535",
        "farm": "9",
        "title": "Golden's Day (33)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28341815944/"
      },
      {
        "id": "28341816314",
        "owner": "97030819@006",
        "secret": "2788c23fab",
        "server": "8572",
        "farm": "9",
        "title": "Golden's Day (32)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28341816314/"
      },
      {
        "id": "28344337983",
        "owner": "97030819@006",
        "secret": "cfa48a4c86",
        "server": "8656",
        "farm": "9",
        "title": "Golden's Day (34)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344337983/"
      },
      {
        "id": "28675048220",
        "owner": "97030819@006",
        "secret": "04ef2ec81d",
        "server": "8702",
        "farm": "9",
        "title": "Golden's Day (35)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28675048220/"
      },
      {
        "id": "28344337633",
        "owner": "97030819@006",
        "secret": "1970a04e4a",
        "server": "8120",
        "farm": "9",
        "title": "Golden's Day (36)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344337633/"
      },
      {
        "id": "28344337523",
        "owner": "97030819@006",
        "secret": "355e0686d3",
        "server": "8597",
        "farm": "9",
        "title": "Golden's Day (37)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344337523/"
      },
      {
        "id": "28344337423",
        "owner": "97030819@006",
        "secret": "a4308c6547",
        "server": "8745",
        "farm": "9",
        "title": "Golden's Day (38)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28344337423/"
      },
      {
        "id": "28855906572",
        "owner": "97030819@006",
        "secret": "6448669772",
        "server": "8769",
        "farm": "9",
        "title": "Golden's Day (39)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28855906572/"
      },
      {
        "id": "28855906322",
        "owner": "97030819@006",
        "secret": "598da37628",
        "server": "8687",
        "farm": "9",
        "title": "Golden's Day (40)",
        "ispublic": "1",
        "isfriend": "0",
        "isfamily": "0",
        "url": "https://www.flickr.com/photos/97030819@006/28855906322/"
      }
    ]
  }
})
```

So far, we have used the API in a **very unsophisticated way**. Here we will do two things:

- We will **parse the JSON** returned by the API
- We will use it to format the returned information in a way that quickly displays the images and allows us to click through to the original photo and Flickr user account.

Below you will see an example of what we are trying to achieve. We take the output from the Flickr API and we then extract what we need, and place it inside some **rather elementary HTML**. We will supply basic styling later, and you should feel free to edit the CSS file.



It is important to understand the structure of the **JSON response** that is returned. At a global level, we have:

```
{
  photos: {},
  stat: "ok"
}
```

This **really isn't easy to see** – have a look at the extreme right of the bottom line of the page and you will finally reach the end of the photos element, and find that indeed there is a `stat: "ok"` report.

Next we look at the overarching photos record, which contains some parameters and then also an **array of individual photos**. An earlier response from this call produced the following:

```
photos: {
  page: 1,
  pages: 1567,
  perpage: 100,
  total: "156681",
  photo: [
    {
      id: "9403006062",
      owner: "97823106@N05",
      secret: "96a96106cb",
      server: "3682",
      farm: 4,
      title: "Instagram Photo -- Follow us @redbarninc",
      ispublic: 1,
      isfriend: 0,
      isfamily: 0
    },
    {
      id: "9402178630",
      owner: "87199074@N05",
      secret: "99e1d32064",
      server: "3802",
      farm: 4,
      title: "Whisper's First Front Cover Appearance! :)",
      ispublic: 1,
      isfriend: 0,
      isfamily: 0
    }
  ],
}
```

Now the information we need is far from obvious, but here goes. Flickr allows direct access to the photos by the use of the direct **farm id**, and access to the photo in its hosted page using the URL **we worked with in the lecture**. HTTPS is the standard, but the old HTTP form is still supported.

The `farm` URL has the form (updated from an earlier pattern):

```
https://farm{farm-id}.staticflickr.com/{server-id}/{id}_{secret}_{mstzb}.jpg
```

Our first photo – a rather dodgy golden retriever ☺ - has the form:

```
{
  id: "9403006062",
  owner: "97823106@N05",
  secret: "96a96106cb",
  server: "3682",
  farm: 4,
  title: "Instagram Photo -- Follow us @redbarninc",
  ispublic: 1,
  isfriend: 0,
  isfamily: 0
},
```

So we are able to construct a URL:

```
https://farm4.staticflickr.com/3682/9403006062_96a96106cb_t.jpg
```

Note that the [mstzb] selections determine size. We want the thumbnail, so use t.

We can similarly construct the ordinary URL as follows:

```
https://www.flickr.com/photos/{user-id}/{photo-id}
https://www.flickr.com/photos/97823106@N05/9403006062
```

Your practical task is as follows. In the flickr-starter-files zip you will find a simple HTML file that looks something like this:

```
<!DOCTYPE html>
<html>
<head>
  <title>The Flickr Experiment</title>
  <link rel="stylesheet" href="./src/styles.css" />
  <meta charset="UTF-8" />
</head>
<body>
  <h1>The Golden Retrievers</h1>
  <div id="app"></div>
  <script src="src/index.js">
  </script>
</body>
</html>
```

The CSS is added from another exercise and some of it won't be very relevant at this stage. Our focus is on the call and the parsing of the resulting JSON and its presentation as above. Here, all of that work will take place in the file index.js. We will work with the div shown above and update its innerHTML property, as is traditional in AJAX applications. We will provide you with a skeleton for the JavaScript file as shown on the next page. Your main task is to work through the JSON structure and construct the links for the images and display them.

Grabbing the JSON from Flickr is accomplished using the JavaScript fetch API. Many of you have seen before, but for those who haven't, an excerpt from the lecture notes is provided

with this week's files along with a video introducing it. Here we give you the code. You just have to fill in your API key:

```
const base = "https://api.flickr.com/services/rest/?"
const query = "&method=flickr.photos.search&api_key=XXXX&tags=golden-retriever&per-page=50&format=json&media=photos&nojsoncallback=1"
const url = base + query
fetch(url)
  .then( (response) => {
    if (response.ok) {
      return response.json()
    }
    throw new Error("Network response was not ok.")
  })
  .then( (rsp) => parseAndCreatePage(rsp))
  .catch(function(error) {
    console.log("There has been a problem with fetch: ",error.message)
  })
  );
```

Note the following points:

- Fetch uses the **GET method** by default and I have split the url for clarity.
- The return values are chained from one clause to the next, so **rsp** will contain **response.json** and so on.

The work of parsing the JSON and constructing the page is handled in a function called **parseAndCreatePage**. This is outlined below:

- *Initialise a string to hold the HTML*
- *Add a message showing the total number*
- *Loop over all the available photos*
 - *Each time, create URLs for the thumbnail and the original*
 - *Update the string using an image tag and hyperlink*
- *Finally, use the string to replace the innerHTML property of the app div.*

You need to access the photos. You do this by exploiting the record structure of the JSON. So to access the photo array, we use the following:

```
rsp.photos.photo
```

This is an array, like any other JS array. Individual photos are then available by indexing:

```
rsp.photos.photo[i]
```

And we can loop up to the length of this array using the, ahem, length:

```
rsp.photos.photo.length
```

For those who need a refresher on the hyperlink element of HTML, if you look at the page, you see that we have a set of images, and we will make these live, by encasing the entire image within the hyperlink. Our example will look something like this:

```
<a href="https://www.flickr.com/photos/97823106@N05/9403006062">  
    
</a>
```

Note of course that this is the final HTML, not the JS that creates it. To help you, here is a skeleton of the final JS function:

```
function parseAndCreatePage(rsp) {  
  let s = "";  
  s = "total number is: " + rsp.photos.photo.length + "<br/>";  
  // http://farm{farm-id}.static.flickr.com/{server-id}/{id}_{secret}_{mstb}.jpg  
  // http://www.flickr.com/photos/{user-id}/{photo-id}  
  for (let i=0; i < XXXX; i++) {  
    photo = XXXX;  
    t_url = XXXX;  
    p_url = XXXX;  
    s += '<a href="' + XXXX + '"' + '>' +  
      '' + '</a>';  
  }  
  
  const appDiv = document.getElementById("app");  
  appDiv.innerHTML = s;  
}
```

If you want to go further, try:

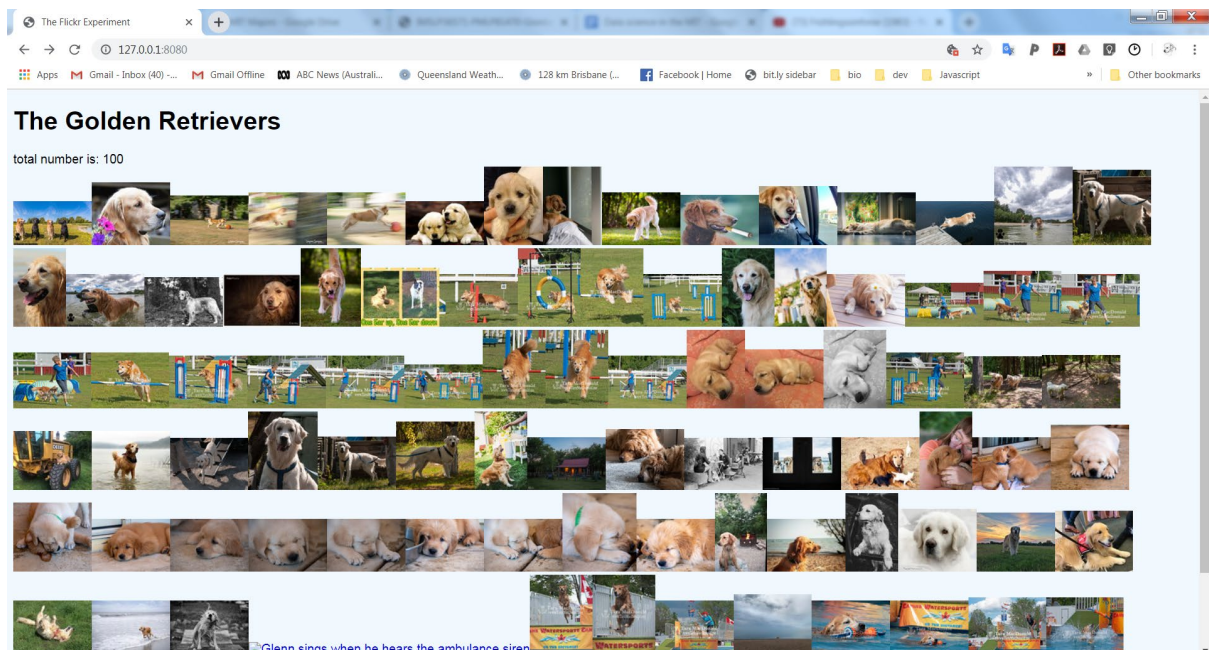
<http://www.flickr.com/services/api/flickr.places.find.html>

The final step is to deploy on localhost using the node server we installed above. In my case, I created a local directory called `public`, and placed the Flickr files there as shown below, before launching the web server to serve files from the `./public` directory:

```
MINGW64:/c/temp
hogan@SEF-PA00124298 MINGW64 /c/temp
$ ls ./public
index.html  src/

hogan@SEF-PA00124298 MINGW64 /c/temp
$ http-server ./public
Starting up http-server, serving ./public
Available on:
  http://192.168.0.6:8080
  http://192.168.56.1:8080
  http://127.0.0.1:8080
Hit CTRL-C to stop the server
```

As you can see, we can use these URLs (all of which are local) to access the page:



On the next prac sheet we will work through this application using Express, with the API access all taking place on the server side. The reason we use services of this nature is that they provide useful inputs for our mashups, and there are many advantages to working on the server rather than the client. This will be direct preparation for assignment 1.

A Quick Note on Apache Servers: While we have used `http-server`, we could equally have used Apache. For Windows, see XAMPP: <http://www.apachefriends.org/en/xampp.html> for a full Apache webserver and MySQL backend. New projects should be unpacked in the `htdocs` directory, using an app specific directory name. So, something fresh like "MyAppJS" would have a directory `MyAppJS` and be accessible at <http://localhost/MyAppJS>.

Static Website Hosting on S3

In week 3 we introduced Amazon S3 – a storage service for files and folders. S3 can also be used as a service for hosting static websites. These are websites that do not require a server/back-end runtime – just like the golden retriever web page that we have created. S3 is increasingly being used to host static websites because it scales well, has high availability, and offers attractive pricing.

In this final exercise, we will host our golden retriever web page on S3 and generate a publicly available URL. The steps are as follows:

1. Create a new bucket
2. Upload the website files to the bucket
3. Configure the bucket for static website hosting

1. Create a new bucket

Log in to the AWS management console. Click on “Services” on the top navigation bar. In the drop-down, select “S3” from the Storage section. You should now see the S3 dashboard.

Click on “Create bucket”. Enter a unique bucket name – this name needs to be globally unique.

General configuration

Bucket name

Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

AWS Region

Asia Pacific (Sydney) ap-southeast-2

Copy settings from existing bucket - *optional*

Only the bucket settings in the following configuration are copied.

Choose bucket

Block Public Access settings for this bucket

Public access is granted to buckets and objects through access control lists (ACLs), bucket policies, access point policies, or all. In order to ensure that public access to this bucket and its objects is blocked, turn on Block all public access. These settings apply only to this bucket and its access points. AWS recommends that you turn on Block all public access, but before applying any of these settings, ensure that your applications will work correctly without public access. If you require some level of public access to this bucket or objects within, you can customize the individual settings below to suit your specific storage use cases. [Learn more](#)

☐ **Block all public access**

Turning this setting on is the same as turning on all four settings below. Each of the following settings are independent of one another.

☐ **Block public access to buckets and objects granted through new access control lists (ACLs)**

S3 will block public access permissions applied to newly added buckets or objects, and prevent the creation of new public access ACLs for existing buckets and objects. This setting doesn't change any existing permissions that allow public access to S3 resources using ACLs.

☐ **Block public access to buckets and objects granted through any access control lists (ACLs)**


S3 will ignore all ACLs that grant public access to buckets and objects.

☐ **Block public access to buckets and objects granted through new public bucket or access point policies**

S3 will block new bucket and access point policies that grant public access to buckets and objects. This setting doesn't change any existing policies that allow public access to S3 resources.

☐ **Block public and cross-account access to buckets and objects through any public bucket or access point policies**

S3 will ignore public and cross-account access for buckets or access points with policies that grant public access to buckets and objects.

 **Turning off block all public access might result in this bucket and the objects within becoming public**

AWS recommends that you turn on block all public access, unless public access is required for specific and verified use cases such as static website hosting.

☒ I acknowledge that the current settings might result in this bucket and the objects within becoming public.

Scroll down to the “Block Public Access settings for this bucket” section. Uncheck “Block all public access” and accept the acknowledgement. We need to ensure that the bucket is publicly accessible so that others can access our website.

Scroll down to the bottom of the page and click “Create bucket”.

2. Upload the website files to the bucket

From the S3 dashboard, click on the name of the bucket you just created. Click the “Upload” button. Click “Add files” and select the “index.html” file. Click “Add folder” and select the “src” folder.

Please double check that the index.html file is in the root of the bucket (i.e. the listed folder should be “-”). This is very important!

Upload [Info](#)

Add the files and folders you want to upload to S3. To upload a file larger than 160GB, use the AWS CLI, AWS SDK or Amazon S3 REST API. [Learn more](#)

Drag and drop files and folders you want to upload here, or choose **Add files**, or **Add folders**.

Files and folders (3 Total, 1.7 KB)

Remove

Add files

Add folder

All files and folders in this table will be uploaded.

< 1 >

<input type="checkbox"/>	Name ▲	Folder ▼	Type ▼	Size ▼
<input type="checkbox"/>	index.html	-	text/html	273.0 B
<input type="checkbox"/>	index.js	src/	text/javascript	1.4 KB
<input type="checkbox"/>	styles.css	src/	text/css	70.0 B

Once you have confirmed the above, click “Upload”.

3. Configure the bucket for static website hosting

From the S3 dashboard, click on the name of your bucket. Click on the “Properties” tab. Scroll down to the “Static website hosting” section. Currently static website hosting is disabled. Click the “Edit” button and change it to “Enable”. Specify the index document as “index.html”.

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting

☐ Disable

☒ Enable

Hosting type

☒ Host a static website

Use the bucket endpoint as the web address. [Learn more](#)

☐ Redirect requests for an object

Redirect requests to another bucket or domain. [Learn more](#)

i For your customers to access content at the website endpoint, you must make all your content publicly readable. To do so, you can edit the S3 Block Public Access settings for the bucket. For more information, see [Using Amazon S3 Block Public Access](#)

Index document

Specify the home or default page of the website.

index.html

Save changes.

Scroll back down to the Static website hosting section. Your bucket website endpoint should now be available:

Static website hosting

Use this bucket to host a website or redirect requests. [Learn more](#)

Static website hosting

Enabled

Hosting type

Bucket hosting

Bucket website endpoint

When you configure your bucket as a static website, the website is available at the AWS Region-specific website endpoint of the bucket. [Learn more](#)

 <http://michael-test-worksheet.s3-website-ap-southeast-2.amazonaws.com>

If you try to open the URL endpoint, you should currently see a 403 Forbidden message:

403 Forbidden

- Code: AccessDenied
- Message: Access Denied
- RequestId: P40NVX8AYCMEFQ9D
- HostId: wdKhZbGGyRUGsrheVcPv7MaKxf1DP2leMvq8F8ETU+434YUrdlZamOuoXfXzG/sMrya7jgPJNZA=

We receive this message because although we edited the S3 Block Public Access settings in step one, we still need to implement a bucket policy that grants public s3: GetObject permissions.

From the S3 dashboard, select the name of your bucket. Click the “Permissions tab”. Scroll down to the “Bucket policy” section and click “Edit”.

Paste in the following policy – replacing Bucket-Name with the name of your bucket:

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "PublicReadGetObject",
      "Effect": "Allow",
      "Principal": "*",
      "Action": [
        "s3:GetObject"
      ],
      "Resource": [
        "arn:aws:s3:::Bucket-Name/*"
      ]
    }
  ]
}
```

Click “Save Changes”.

Open your bucket website endpoint. You should now see the golden retriever web page.