# 1 Project 1

**Due**: Feb 15 by 11:59p

**Important Reminder**: As per the course *Academic Honesty Statement*, cheating of any kind will minimally result in receiving an F letter grade for the entire course.

This document first provides the aims of this project. It then lists the requirements as explicitly as possible. This is followed by a log which should help you understand the requirements. Finally, it provides some hints as to how those requirements can be met.

## 1.1 Aims

The aims of this project are as follows:

- To ensure that you have set up your VM as specified in the *Course VM* and *Gitlab* set up documents.
- To get you to write a simple but non-trivial JavaScript program.
- To allow you to familiarize yourself with the programming environment you will be using in this course.
- To introduce you to text processing within JavaScript.

## 1.2 Background

A url-shortener transforms a `longUrl` into a `shortUrl`. The general idea is well described in this *Wikipedia article*, but this project has more specific requirements.

As far as this project is concerned, a url consists of the following 3 fields:

**scheme** a non-empty sequence of characters terminated by the first occurrence of the string `://`. Examples include `http` or `https`. A scheme is case-insensitive.

**domain** the longest non-empty sequence of characters following the `://` which do not include a `/` character. A domain is case-insensitive.

**rest** any string after the domain (it could be empty). The rest field is case-sensitive.

For example, in the URL

```
http://zdu.binghamton.edu/cs544/projects/prj1/prj1.html
```

the scheme is `http`, domain is `zdu.binghamton.edu` and rest is `/cs544/¬ projects/prj1/prj1.html`.

A url-shortener always has an associated `SHORTENER_DOMAIN` specifying the domain for returned short-url's.

The algorithm for shortening a URL `longUrl` is described below, illustrated with an example for a URL shortener with `SHORTENER_DOMAIN` set to `cs544.io` and input `longUrl https://ZDU.bingHAMTON.eDU/cs544/index.html`:

1. If the `domain` and `rest` components of `longUrl` have not been seen earlier (for example, `zdu.binghamton.edu/cs544/index.html` has not been seen earlier):
   (a) Compute a random non-negative integer less than 2**32. For example, For example, `444544`
   (b) Compute its lower-case base-36 (the characters `0` through `9`, followed by `a` through `z`) representation. For example, the base-36 representation of the integer `444544` is `9j0g`.
   (c) Associate the above base-36 string with the `domain` and `rest` of the input long-url. In the example, the `domain` and `rest` of the input URL, namely `zdu.binghamton.edu/cs544/index.html` will be associated with `9j0g`.
2. If the `domain` and `rest` components of `longUrl` have been seen earlier, retrieve the base-36 representation associated with that `domain` and `rest`. (in the example, if `zdu.binghamton.edu/cs544/index.html` has been seen earlier, its associated base-36 retrieved may be `9j0g`
3. Produce a short URL `SCHEME://SHORTENER_DOMAIN/BASE36`, where `SCHEME` is the `scheme` of the input URL, `SHORTENER_DOMAIN` is the domain for the URL-shortener and `BASE36` is the base-36 representation computed or retrieved above. For the example, we would get `https://cs544.io/9j0g`.

## 1.3 Requirements

You must check in a `submit/prj1-sol` directory in your gitlab project such that typing `npm install` within that directory is sufficient to run the project using `./index.js`.

You are being provided with `index.js` which provides the required command-line behavior. What you specifically need to do is add code to the provided url-shortener.js source file as per the requirements in that file.

Additionally, your `submit/prj1-sol` **must** contain a `vm.png` image file to verify that you have set up your VM correctly. Specifically, the image must show a x2go client window running on your VM. The captured x2go window must show a terminal window containing the output of the following commands:

```
$ hostname; hostname -I
$ ls ~/git-repos
$ ls ~/cs544
$ ls ~/git-repos/i?44
$ crontab -l | tail -3
```

Other miscellaneous requirements:

- All URLs returned by your implementation must have case normalized to lower-case for the scheme and the domain.
- Your project **must** implement the API specified for the `UrlShortener class`.
- A non-functional requirement is that looking up a url should usually take time independent of the number of urls tracked by your module.
- Your module will need to break up a URL into its components. Library routines to implement this functionality may be available. The use such routines is **strictly forbidden**; you should implement the functionality yourself using either string or regex library routines.

## 1.4 Example Log

The behavior of the program is illustrated by the following annotated log:

```
#usage message
$ ./index.js
usage: index.js SHORTENER_DOMAIN
$ ./index.js cs544.io         #SHORTENER_DOMAIN === cs544.io
>> h                          #help message
? SHORT_URL: query SHORT_URL
+ LONG_URL:  add LONG_URL
- URL:       remove URL
# URL:       count URL
h:           help
{}                           #output of help command
>> + http:/zdu.io/a.html      #missing ://
{ code: 'URL_SYNTAX',
  message: 'URL_SYNTAX: bad url http:/zdu.io/a.html' }
>> + http:///a.htmp           #missing domain
{ code: 'URL_SYNTAX',
  message: 'URL_SYNTAX: bad url http:///a.htmp' }
>> + http://zdu.io/a.html     #successfully adding
{ value: 'http://cs544.io/3or718' }
>> + http://zdu.io/a.html     #idempotent
{ value: 'http://cs544.io/3or718' }
>> ? http://zdu.io/a.html     #not SHORTENER_DOMAIN url
```

```
{ code: 'DOMAIN',
  message:
   'DOMAIN: domain of url http://zdu.io/a.html not equal to cs544.io' }
>> ? http://cs544.io/3or718   #successful lookup
{ value: 'http://zdu.io/a.html' }
>> ? htTPS://cs544.IO/3or718  #case ignored in scheme and domain
{ value: 'https://zdu.io/a.html' }
>> # http://zdu.io/a.html     #number of successfule lookups
{ value: 2 }
>> ? htTPS://cs544.IO/3or718x #never returned this url
{ code: 'NOT_FOUND',
  message: 'NOT_FOUND: htTPS://cs544.IO/3or718x not found' }
>> - http://zdu.io/a.html     #deactivate
{}
>> ? htTPS://cs544.IO/3or718  #cannot be looked up
{ code: 'NOT_FOUND',
  message: 'NOT_FOUND: htTPS://cs544.IO/3or718 not found' }
>> + http://zdu.io/a.html     #add it back in, same short url
{ value: 'http://cs544.io/3or718' }
>> ? htTPS://cs544.IO/3or718  #look it up again successfully
{ value: 'https://zdu.io/a.html' }
>> # http://zdu.io/a.html     #count updated
{ value: 3 }
>>                            #typed ^D for EOF
$

#use undocumented random-seed nodejs option to get reproducible results
$ nodejs --random-seed=444544 index.js cs544.io
>> + http://zdu.io/a.html
{ value: 'http://cs544.io/182qe07' }
>> ? http://cs544.io/182qe07
{ value: 'http://zdu.io/a.html' }
>> ? HTTPs://cs544.IO/182qe07
{ value: 'https://zdu.io/a.html' }
>> # HTTPs://cs544.IO/182qe07
{ value: 2 }
>> # HTTPs://cs544.IO/182QE07
{ code: 'NOT_FOUND',
  message: 'NOT_FOUND: HTTPs://cs544.IO/182QE07 not found' }
>>
$
```

## 1.5 Provided Files

The prj1-sol directory contains a start for your project. It contains the following files:

**url-shortener.js** This skeleton file constitutes the guts of your project. You will need to flesh out the skeleton, adding code as per the documentation. You should feel free to add any auxiliary function or method definitions as required.

**index.js** This file provides the complete command-line behavior which is required by your program. It requires url-shortener.js. You **must not** modify this file; this ensures that your `UrlShortener` meets its specifications and facilitates automated testing by testing only the `UrlShortener` API.

**README** A README file which must be submitted along with your project. It contains an initial header which you must complete (replace the dummy entries with your name, B-number and email address at which you would like to receive project-related email). After the header you may include any content which you would like read during the grading of your project.

## 1.6 Hints

The following standard library functions may come in useful: Number.prototype.toString(), Math.random() and Math.floor().

The following steps are not prescriptive in that you may choose to ignore them as long as you meet all project requirements.

1. Set up your course VM as per the instructions specified in the *Course VM* and *Gitlab* documents.
2. Read the project requirements thoroughly. Look at the sample log to make sure you understand the necessary behavior.
3. Look into debugging methods for your project. Possibilities include:
   - Logging debugging information onto the terminal using console.log() or console.error().
   - Use the chrome debugger as outlined in this article.
4. Consider what kind of indexing structure you will need to track the urls. For each url, you will need to track a one-to-one mapping which maps between a `longUrl` ( . - `domain` and `rest` only) and its associated `shortUrl` (`domain` and `rest` only). You will also need to track whether a mapping is currently active and the number of times the mapping has been successfully queried.

When designing your indexing structure, you should first think in terms of the abstract data structure **Map** and then consider possibilities for implementing such `Map` data structures in JavaScript. Possibilities:

- Use standard JavaScript objects as maps. Advantages include the ability to using simple `[]`-based access and trivial JSON conversion for subsequent projects. Disadvantages include the lack of any defined order for property keys and the overhead of all the machinery associated with objects like inheritance.
- Use the relatively new Map addition to the standard library. An advantage of this approach is that it preserves insertion order. It may also be lighter than the `Object` alternative suggested above. Disadvantages include a clumsy API (`get()` and `set()`) and non-trivial work required for JSON conversion which may be useful for subsequent projects.

5. Start your project by creating a `work/prj1-sol` directory. Change into that directory and initialize your project by running `npm init -y`. This will create a `package.json` file; this file should be committed to your repository.

6. Within a terminal window in your `x2go` client window, type in the following commands:

```
$ hostname; hostname -I
$ ls ~/git-repos
$ ls ~/cs544
$ ls ~/git-repos/i?44
$ crontab -l | tail -3
```

   Use an image capture program on your workstation to capture an image of your `x2go` client window into the file `vm.png` in your `work/prj1-sol` directory. The captured image should clearly shows the terminal window containing the output of the above commands.

7. Copy the provided files into your project directory:

```
$ cp -p $HOME/cs544/projects/prj1/prj1-sol/* .
```

   This should copy in the `README` template, the `index.js` and the `url-shortener.js` skeleton file into your project directory.

8. You should be able to run the project, but all commands will fail with `TODO` errors since your `url-shortener.js` file needs code added.

```
$ ./index.js #show usage message
$ ./index.js cs544.io
>> h
? SHORT_URL: query SHORT_URL
+ LONG_URL:  add LONG_URL
```

```
- URL:        remove URL
# URL:        count URL
h:            help
{}
>> + sad
{ code: 'TODO', message: 'TODO: add()' }
>>
```

9. Replace the `XXX` entries in the `README` template. Commit your project to gitlab:

```
$ git add .
$ git commit -a -m 'started prj1'
```

10. Open the copy of the `url-shortener.js` file in your project directory. It contains

    (a) A `strict` declaration.

    (b) A skeleton `UrlShortener class` definition with `@TODO` comments. Note that the `class` syntax is a recent addition to JavaScript and is syntactic sugar around JavaScript's more flexible object model. It was used in this project because JavaScript's object model has not yet been covered in the course. Note that even though the use of this `class` syntax may make students with a background in class-based OOPL's feel more comfortable, there are some differences worth pointing out:

    - No data members can be defined within the `class` body. All "instance variables" must be referenced through the `this` pseudo-variable in both `constructor` and methods. For example, if we want to initialize an instance variable `urls` in the `constructor()` we may have a statement like:

      ```
      this.urls = new Map();
      ```

    - There is no implicit `this`. If an instance method needs to call another instance method of the same object, the method **must** be called through `this`.

    - There is no easy way to get private methods or data. Instead a convention which is often used is to prefix private names with something like a underscore and trust `class` clients to not misuse those names.

    (c) An assignment of the `UrlShortener class` to `module.exports`. This makes `UrlShortener` the only declarations available outside the `url-shortener.js` file; all other declarations are local to the file.

11. Add initialization code to your `constructor()`.

12. Add any utility functions or methods which may be useful. Some possibilities:

    **splitUrl()** Splits a string URL into its components.

    **error()** A simple wrapper which returns an error result.

    It may be possible to factor out behavior common to multiple public methods into a common private method.

13. Implement the `add()` method as per its specifications.

14. Implement the `query()` method as per its specifications. Test using the command-line `index.js`.

15. Implement the `count()` method as per its specifications. Test using the command-line `index.js`.

16. Implement the `remove()` method as per its specifications. Test using the command-line `index.js`.

17. Iterate until you meet all requirements.

It is a good idea to commit and push your project periodically whenever you have made significant changes. When it is complete please follow the procedure given in the *gitlab setup directions* to submit your project using the `submit` directory.