Node Web Crawler

Module 1 Tutorial: Node Web Crawler Tutorial

In this tutorial lab, you will build a script to download webpages. This lab will use concepts and skills learned in the module 1, concepts such as core modules, CLI arguments, launching Node scripts, using an npm module, making a GET request and processing the body, creating a folder and a file.

Node Web Crawler

The implementation of the lab consists of the following steps:

1. Create the project folder and make a file in it. The name doesn't matter.
2. Create a `package.json` file using `npm init -y`
3. Install a module `uuid` for generating random timestamped values which we will use for folder names
4. Create a script file which downloads a page at a given URL and writes page's HTML to a file. URL must be provided from the command-line argument.

Let's start!

Open your Terminal / Command Prompt (for Windows, use bash). Create a new folder with `mkdir`:

```
mkdir download-page
cd download-page
```

Create a new file in this newly created folder:

```
touch download.js
```

Now, create a `package.json` file by using `npm init -y`. Once the file is created, install `uuid` module using the `npm install` (or `npm i`) command:

```
npm i uuid@3.1 -E
```

That's all the set up. Now, you are ready to start developing the program itself.

Let's see the program at a higher level without the details. The program will have this structure:

```
const http = require('http')
...

const downloadPage = (url='http://nodeprogram.com') => {
  ...
}

downloadPage(process.argv[2])
```

Now you can start the actual implementation.

Open `download.js`, and start writing code to import modules: `http`, `fs`, `path` and `uuid` for creating of an HTTP agent, a folder and a file, working with a folder/file path and generating random values.

```
const http = require('http')
const fs = require('fs')
const path = require('path')
const uuidv1 = require('uuid/v1')
```

Next, you will create a function which takes the URL, downloads the HTML from that URL and saves the HTML into a newly created file. The name of this function is `downloadPage`. It just shows how creative I am (sarcasm).

At a high level, the function looks like this:

```
const downloadPage = (url='http://nodeprogram.com') => {
  const fetchPage = (urlF, callback) => {
    ...
  }
  const folderName = uuidv1()

  fs.mkdirSync(folderName)
  fetchPage(url, (error, data)=>{
    ...
    fs.writeFileSync(path.join(__dirname, folderName, 'file.html'), data)
    console.log('downloading is done in folder ', folderName)
  })
}
```

Now, the details! First, the default value of the `url` is set to a website nodeprogram.com, in case the CLI argument URL value is not provided by the CLI argument (`process.argv[2]`).

Next, there's a function named `fetchPage` which takes the URL and a callback function and makes a GET request. The html of the page is sent as the *second* argument of the callback function once the response has been completed. This is the definition of the `fetchPage` function:

```
const downloadPage = (url='http://nodeprogram.com') => {
  console.log('downloading ', url)
  const fetchPage = (urlF, callback) => {
    http.get(urlF, (response) => {
      let buff = ''
      response.on('data', (chunk) => {
        buff += chunk
      })
      response.on('end', () => {
        callback(null, buff)
      })
    }).on('error', (error) => {
      console.error(`Got error: ${error.message}`)
      callback(error)
    })
  }
```

The `downloadPage` function is not over yet. You need to create a unique folder name using the npm module `uuid`. Then use that folder name value to create a folder with `mkdirSync()`.

Finally, invoke `fetchPage` which was defined earlier and takes a callback function. The logic to create the files `url.txt` and `file.html` goes inside of the callback because the GET method is *asynchronous*. The `mkdirSync` and `writeFileSync` methods are *synchronous*.

The last line will execute the entire `downloadPage` method with the command line argument indicating the page URL:

```
  const folderName = uuidv1()
  fs.mkdirSync(folderName)
  fetchPage(url, (error, data)=>{
    if (error) return console.log(error)
    fs.writeFileSync(path.join(__dirname, folderName, 'url.txt'), url)
    fs.writeFileSync(path.join(__dirname, folderName, 'file.html'), data)
    console.log('downloading is done in folder ', folderName)
  })
}

downloadPage(process.argv[2])
```

Save the file. You are done with the program so let's go ahead and test it. From the Terminal app, launch the script with some URL:

```
node download.js http://www.google.com
```

You would see an output similar to this:

```
downloading  http://www.google.com
downloading is done in folder  b9697730-9764-11e7-8546-bfe35e924b2c
```

Open the newly created subfolder in your code editor or even in a browser to view the HTML (the browser view won't be a perfect copy, because there might be missing some CSS dependencies or AJAX/XHR content which is loaded on the fly by the target website but inaccessible by the GET request).

Experiment some more to see that each time you run the node script, the URL is saved into a file named `url.txt` within the folder and that the html data is saved within `file.html`:

```
node download.js http://azat.co
node download.js http://webapplog.com
node download.js http://node.university
node download.js
```

Congratulations! You are done with the lab.