Open-Source Software Practice

6. Javascript Advanced

Instructor: Jaemin Jo (조재민, jmjo@skku.edu)
Interactive Data Computing Lab (*IDCLab*),
College of Computing and Informatics,
Sungkyunkwan University

Modules

- You don’t have to develop every feature you want from scratch.
- Most of them are already implemented and available as open-source packages!

- Where are they?
  - i.e., package index, package registry, package repository, …
  - Javascript packages: https://www.npmjs.com/
  - Python packages: https://pypi.org/
  - Ruby packages: https://rubygems.org/
  - …

Node Package Manager

- **Node Package Manager** or NPM is a package manager for Node.js.
- You have this. It comes with Node.

```
Microsoft Windows [Version 10.0.19043.2006]
(c) Microsoft Corporation. All rights reserved.

D:\skku-menu>npm -v
7.15.1
```

- Type `npm -v` on your terminal.
- Type `npm --help` to see all the commands.

- `npm init`
- `npm install`
- `npm uninstall`
- `npm test`

Why do We Need a Manager?

- Why do we use a package manager? Isn’t it enough to copy-and-paste JS code from GitHub?

- Dependency control
- Package update
- Compatibility
- Prerequisite management

Dependency Control

- Suppose you want to use package A@1.0.0.
  - 1.0.0 is a version number.

- A@1.0.0 depends on B@2.0.0 and C@1.5.3, so if you want to use A, you need to install all the dependencies.
  - B@2.0.0 depends on D@1.0.1...
  - C@1.5.3 depends on E@0.5.3...

- The package manager will install all the dependencies.

Package Update

- Suppose you have used A@1.0.0.
- A new version of A, e.g., A@2.0.0, is out. So, you want to update the package.

- The new version A@2.0.0 depends on B@3.0.0 and C@1.6.0. So, in order to update A, you must update B and C first.

- The package manager will update all the dependencies required to use A@2.0.0.

Compatibility

- A@2.0.0 depends on B@3.0.0 or newer (e.g., B@3.1.0).
- Z@1.0.0 depends on B@3.1.0 or newer.

- To use A and Z at the same time, you must install B@3.1.0 or newer.

- The package manager will resolve these compatibility issues.

Prerequisite Management

- Suppose you used 100 open-source packages to develop your project.
- Development is done, so you are about to deploy the project.

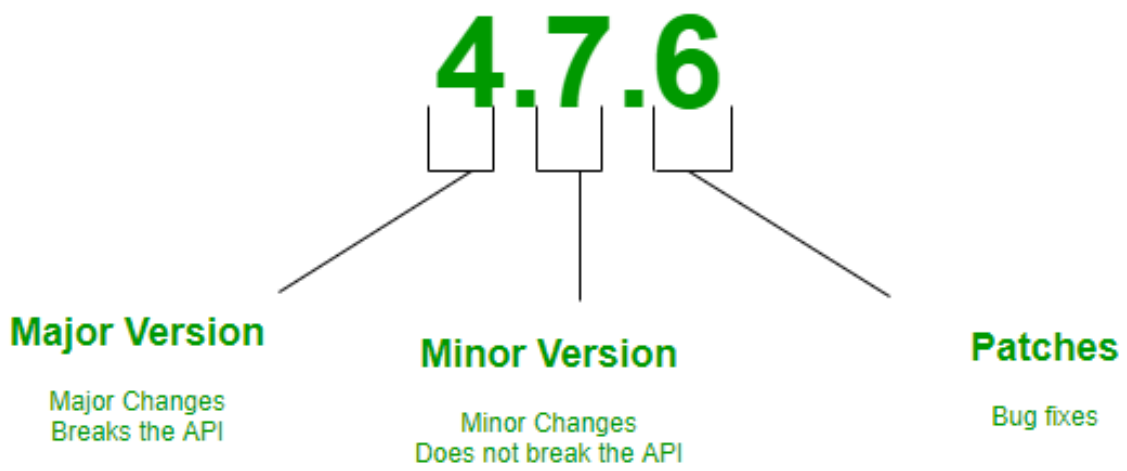
- Should all the source code of the 100 packages be included in your bundle?

- No, you don’t have to include the actual code, the names and versions of the 100 package are enough.

- The package manager keeps track of the prerequisites of your project and when you deploy your project, it will ship the information about the prerequisites together with your package.

Semantic Versioning

- Btw, what does version 1.2.3 mean?
- **Semantic versioning**



Semantic Versioning

- Suppose you were using A@1.0.0.

- It is safe to update it to 1.0.5. Maybe some bugs have been fixed.
  - Patch release
- It is safe to update it to 1.1.0. Maybe some new features have been added or the existing features have been updated without breaking changes (backward compatible).
  - Minor release
- It is **NOT** safe to update it to 2.0.0. There may be changes that break backward compatibility. To use this version, you must modify your code.

Semantic Versioning

- Suppose you deployed version 1.2.3 last time.

- If you just fixed some bugs, name the new version as 1.2.4.
- If you added some features that are backward compatible, name it as 1.3.0.
- If you completely changed APIs, name it as 2.0.0.

- Version numbers are very important!
- To deploy your package to NPM, you must increment the version number at least by 0.0.1 (no same version, no previous version).

Semantic Versioning

- Sometimes, we see a version number smaller than 1.0.0, such as 0.13.0.
- These versions are considered unofficial or unstable versions.
- More details about semantic versioning (with examples): https://semver.org/spec/v2.0.0.html

package.json

- Every Node project has a special JSON file in its root directory, ***package.json***.

- This file contains the metadata of a project.
  - Package name and version
  - Description
  - Keywords
  - Homepage
  - License
  - Dependencies
  - Scripts
  - …

package.json

```
{
  "name": "d3",
  "version": "6.6.0",
  "description": "Data-Driven Documents",
  "keywords": [
    "dom",
    "visualization",
    "svg",
    "animation",
    "canvas"
  ],
  "homepage": "https://d3js.org",
  "license": "BSD-3-Clause",
  "author": {
    "name": "Mike Bostock",
    "url": "https://bost.ocks.org/mike"
  },
  "main": "dist/d3.node.js",
  "unpkg": "dist/d3.min.js",
  "jsdelivr": "dist/d3.min.js",
  "module": "index.js",
  "repository": {
    "type": "git",
    "url": "https://github.com/d3/d3.git"
  },
  "files": [
    "dist/**/*.js",
    "index.js"
  ],
  "scripts": {
    "pretest": "rimraf dist && mkdir dist &&
```

```
  "devDependencies": {
    "json2module": "0.0",
    "rimraf": "3",
    "rollup": "2",
    "rollup-plugin-ascii": "0.0",
    "rollup-plugin-node-resolve": "5",
    "rollup-plugin-terser": "7",
    "tape": "4",
    "tape-await": "0.1"
  },
  "dependencies": {
    "d3-array": "2",
    "d3-axis": "2",
    "d3-brush": "2",
    "d3-chord": "2",
    "d3-color": "2",
    "d3-contour": "2",
    "d3-delaunay": "5",
    "d3-dispatch": "2",
    "d3-drag": "2",
    "d3-dsv": "2",
    "d3-ease": "2",
    "d3-fetch": "2",
```

Create a Node Project

- Let’s create our first Node project.
- In the root directory, type `npm init`.

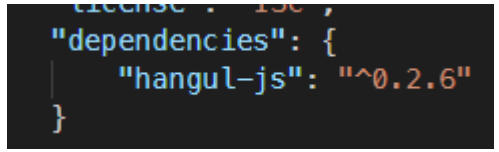
- It will ask a few questions about your project, but you can just press Enter to use the default.

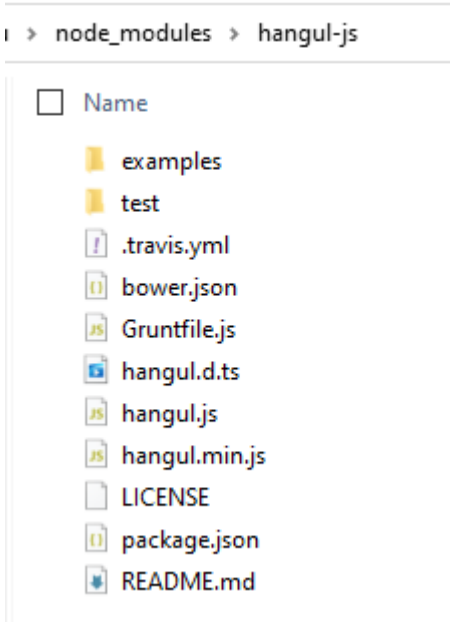
- It will create *package.json*.
- You can modify the metadata by editing *package.json* directly.

Create a Node Project

```
{
  "name": "skku-menu",
  "version": "1.0.0",
  "description": "",
  "main": "main.js",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "",
  "license": "ISC",
}
```

Install a Package





- Let's install a package.

- Go to the root directory of your project.
- `npm install hangul-js`

- This command will
  - add hangul-js to your *package.json* file as a dependency
  - create a directory named *node_modules* under the root directory
  - download the source code of *hangul-js* and save it to *node_modules/hangul-js*

Install a Package

- Dependencies installed via `npm install` are stored in *node_modules*.
- You must add this directory to *.gitignore*. Otherwise, the source code of all the dependencies will be under version control.

- Do not add or remove a directory manually. Use `npm install` and `npm uninstall` instead.

Three Installation Types

- Local installation (`npm install <name>`)
  - Use this command for dependencies that are used to run your package.
  - The package name <name> will be listed as dependencies in *package.json*.
  - Use this command if you are not sure.

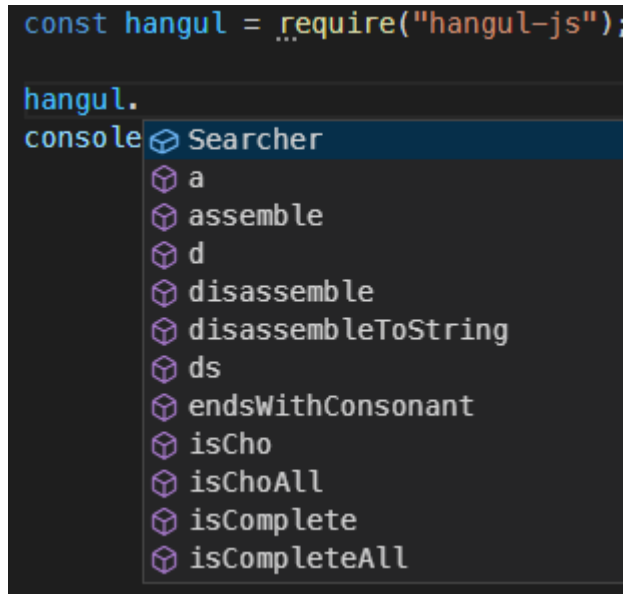
- Local dev installation (`npm install <name> --save-dev`)
  - Use this command for dependencies that are used to develop your package.
  - <name> will be listed as dev-dependencies in *package.json*.
  - Packages for testing, linting, packaging, ...

Three Installation Types

- Global installation (`npm install –g <name>`)
  - This command will not install <name> under *node_modules* but under a system-wide directory.
  - <name> will ***NOT*** be listed in *package.json*.
  - Use this command only when you install a command-line tool written in Javascript.
  - In most cases, you will install packages locally.

Use a Package

- After installing a package, you can load it into your code.
- Use `require`
- IntelliCode will help you see what's inside the package.




```
const hangul = require("hangul-js");

console.log(hangul.disassemble("가나다"));
```

```
D:\skku-menu>node main.js
['ㄱ', 'ㅏ', 'ㄴ', 'ㅏ', 'ㄷ', 'ㅏ']
```

Module

- You can create modules of your own.

circle.js

```
const PI = Math.PI;

exports.area = (r) => PI * r ** 2;

exports.circumference = (r) => 2 * PI * r;

/*
function area(r) {
    return PI * r ** 2;
}

exports.area = area;
*/
```

main.js

```
const hangul = require("hangul-js");

console.log(hangul.disassemble("가나다"));

const circle = require("./circle.js");

console.log(`The area of a circle of radius 4 is
 ${circle.area(4)}`);
```

Module

- To export variables or functions, add them to the *exports* object.
  - *exports.myVariable = 5;* (in module.js)

- To import variables or functions, load the module using the *require* function.
  - *const myModule = require('./module.js');*
  - *console.log(myModule.myVariable);*

- Modules you created: relative path starting with "./" (*require('./abc/def.js')*)
- Modules installed via NPM: just module name (*require('hangul-js')*)

Case Study

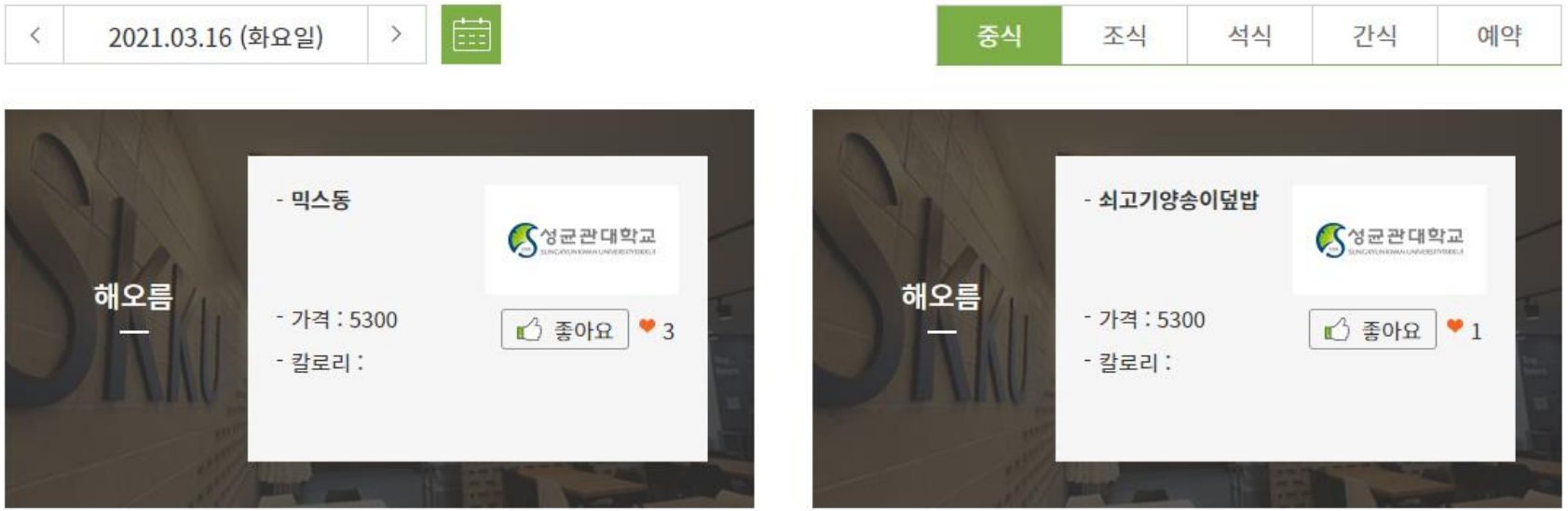
- Let’s practice what we learned through a case study.
- It is perfectly fine for you to not understand everything in this example at this moment.
  - Don’t worry. We will play with easier examples later.

- **Scenario:** I like the cafeteria at Bldg. 26, so I want to check out today’s menu on the terminal.

```
D:\skku-menu>node main.js
믹스동
쇠고기양송이덮밥
```

How?

- Before automate this, think about how you do the same thing manually.
- SKKU menu page: https://www.skku.edu/skku/campus/support/welfare_11_1.do?mode=info&srDt=2022-09-23&srCategory=L&conspaceCd=20201251&srResId=12&srShowTime=D.



How?

- The url I got: https://www.skku.edu/skku/campus/support/welfare_11_1.do?mode=info&**srDt=2022-09-23**&srCategory=L&conspaceCd=20201251&srResId=12&srShowTime=D

- It only shows the menu for the date specified by *srDt*. Should we replace it?

- Today's menu: https://www.skku.edu/skku/campus/support/welfare_11_1.do?mode=info&srCategory=L&conspaceCd=20201251&srResId=12&srShowTime=D
  - "hunch"

HyperText Markup Language

- How do computers see the web page you just saw?
- As text!
- There is a language that is used to describe a web page, HyperText Markup Language.

- Let’s see the HTML code of the menu page.

- Prepend “view-source:” to the url (Chrome).

HTML Code of the Menu Page


```
<!doctype html>
<html lang="ko">
<head>
<title>성균관대학교 | 대학생활 | 편의  /  복지 | 식당  /  메뉴 | 자연과학캠퍼스</title>
<link rel="shortcut icon" href="/_res/skku/img/common/favicon.png">
<meta name="viewport" content="width=device-width,initial-scale=1.0,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<!-- Global site tag (gtag.js) - Google Analytics -->
<script async src="https://www.googletagmanager.com/gtag/js?id=UA-53596226-1"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());

  gtag('config', 'UA-53596226-1');
</script><link rel="canonical" href="http://www.skku.edu/skku/campus/support/welfare_11_1.do" />

<link rel="stylesheet" type="text/css" href="/_common/cms.css" />
<link rel="stylesheet" type="text/css" href="/_res/skku/_css/layout.css" />
<!--[if IE 8]>
    <link rel="stylesheet" type="text/css" href="/_common/ie8.css" />
<![endif]-->
<link rel="stylesheet" type="text/css" href="/_common/css/toastr.css" />
<link rel="stylesheet" type="text/css" href="/_custom/skkuedu/resource/css/board.css" />
<link rel="stylesheet" type="text/css" href="/_common/css/ui.datepicker.css" />
<link rel="stylesheet" type="text/css" href="/_custom/skkuedu/resource/css/app.restaurant.css" />

<link rel="stylesheet" type="text/css" href="/_res/skku/_css/user.css" />

<script type="text/javascript" src="/_common/js/jquery/jquery-1.9.1.js"></script>
<script type="text/javascript" src="/_common/js/jquery/jquery-ui-1.11.4.min.js"></script>
```

HTML Code of the Menu Page

- If you scroll down a little bit, you will find what we are looking for.

```
3187        <h5>
3188            <span>해오름</span>
3189
3190        </h5>
3191        <div class="corner_info">
3192            <ul>
3193                <li>
3194
3195
3196
3197                        <img src="/_res/skku/img/sub/welfare_133.png" alt="믹스동 기본 썸네일 이미지" />
3198
3199
3200
3201
3202                    <div class="menu_title">
3203                        <pre>믹스동</pre>
3204                    </div>
3205                </li>
3206                <li>
3207
3208                    <span>
3209                        가격 :
3210                        5300
3211                    </span>
3212                </li>
3213                <li>
3214
3215                    <span>
3216                        칼로리 :
3217
3218                    </span>
3219                </li>
3220
3221
3222
3223            </ul>
3224
3225            <p class="btnList">
3226                <a class="likeBtn good" href="#none" data-key="17928">좋아요 </a>
3227                <span>3</span>
3228            </p>
3229        </div>
3230    </div>
```

Workflow

1. Get the menu page as HTML text.
2. Extract menu names.
   - There are two menus as you may know!
3. *console.log* the menu names.

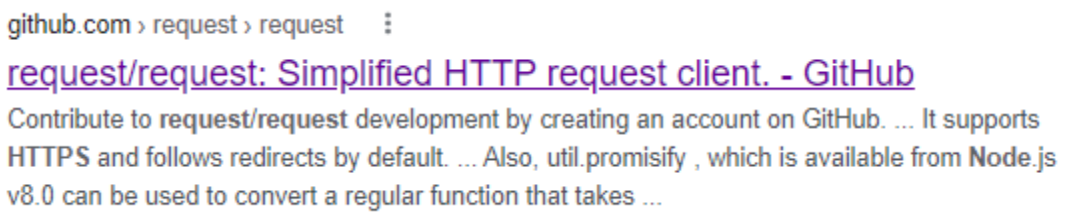
- In this example, we will “scrap” web pages from the SKKU server.
- **In many cases, it is not legal! You need a permission.**
- ***Use this example only for educational purposes***, and do not run the code too much. It will overload the server.

Download HTML from a URL

- So, we know the address or url of the menu page, and we want to download its content.

- So, I googled “node https request”.
  - You need some “hunches” to make up search terms.

- I got a library on GitHub.



Download HTML from a URL

- Unfortunately, it seemed that the lib was no longer maintained.

README.md

Deprecated!

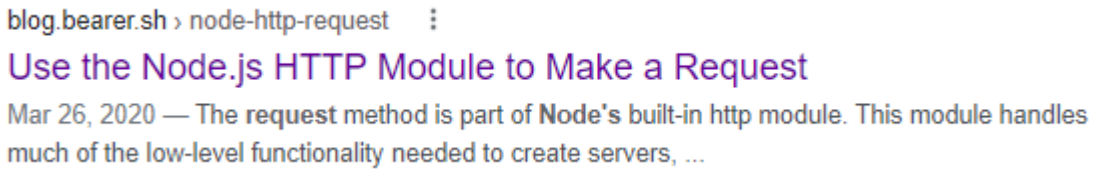
As of Feb 11th 2020, request is fully deprecated. No new changes are expected to land. In fact, none have landed for some time.

For more information about why request is deprecated and possible alternatives refer to this issue.

Request - Simplified HTTP client

Download HTML from a URL

- I got another link from Google.



- This link seemed to only use Node.js standard modules.

- It is always a good practice to use “standard modules” which come with Node.js itself.
  - Avoid making unnecessary dependencies!

Download HTML from a URL

- I found an example code.
  - https://blog.bearer.sh/node-http-request/

```
const http = require("http")

http.get("https://postman-echo.com/status/200", res => {
  let data = ""

  res.on("data", d => {
    data += d
  })
  res.on("end", () => {
    console.log(data)
  })
})
```

Download HTML from a URL

- I found an example code.
  - https://blog.bearer.sh/node-http-request/

```
const http = require("http")

http.get("https://postman-echo.com/status/200", res => {
  let data = ""

  res.on("data", d => {
    data += d
  })
  res.on("end", () => {
    console.log(data)
  })
})
```

```
const http = require("http");

let url =
    "<url of the menu page>";

http.get(url, (res) => {
    let data = "";

    res.on("data", (d) => {
        data += d;
    });
    res.on("end", () => {
        console.log(data);
    });
});
```

Download HTML from a URL

- `node main.js`
- Unfortunately, I got an error.
- It was because our url starts with “https” but we used the “http” module.

```
TypeError [ERR_INVALID_PROTOCOL]: Protocol "https:" not supported. Expected "http:"
    at new ClientRequest (_http_client.js:152:11)
    at request (http.js:46:10)
    at Object.get (http.js:50:15)
    at Object.<anonymous> (D:\skku-menu\main.js:6:6)
    at Module._compile (internal/modules/cjs/loader.js:1137:30)
    at Object.Module._extensions..js (internal/modules/cjs/loader.js:1157:10)
    at Module.load (internal/modules/cjs/loader.js:985:32)
    at Function.Module._load (internal/modules/cjs/loader.js:878:14)
    at Function.executeUserEntryPoint [as runMain] (internal/modules/run_main.js:71:12)
    at internal/main/run_main_module.js:17:47 {
  code: 'ERR_INVALID_PROTOCOL'
}
```

Download HTML from a URL

- So, I changed “http” to “https”.
- `node main.js`

```
const https = require("https");

let url =
    "https://www.skku.edu/skku/campus/support/welfare_11_1.do?mode=info&srCategory=L&conspaceCd=20201251&srResId=12&srShowTime=D";

https.get(url, (res) => {
    let data = "";

    res.on("data", (d) => {
        data += d;
    });
    res.on("end", () => {
        console.log(data);
    });
});
```

Download HTML from a URL

- I got an HTML document, but the content was wrong.
- It seemed that the SKKU server considered my request unauthorized.
- Why?

```
D:\skku-menu>node main.js
<meta http-equiv="Content-Type" content="text/html; charset=utf-8">
<br>
<br>
<center>
<img src="https://www.skku.edu/new_home/images/Emblem_1.jpg" width="300" height="300">
<h2>정보보호를 위해 부적절한 접근이 차단 되었습니다.</h2>
<h4>(정상적인 접근이 차단된 경우, 031-290-5217/5228 혹은 security@skku.edu로 연락 주시기 바라며,<br>
아래 'Detect time' / 'Client IP'를 알려주시면, 보다 정확한 처리가 가능합니다.)<h4>
<h2>Sorry, You have been blocked to protect our service.</h2>
<h4>(In order to resolve this, please contact us at 031-290-5217/5228 or security@skku.edu, <br>
if you let us know 'Detect time' / 'Client IP' below, more accurate processing is possible.)
<h4>
```

Download HTML from a URL

- When your browser connects to a server, it sends extra information to the server in addition to the url you requested.
  - HTTP headers
  - **Referrer**: the address of the previous web page from which a link to the currently requested page was followed
  - **User-Agent**: a characteristic string about your OS and web browser
  - https://developer.mozilla.org/en-US/docs/Web/HTTP/Headers

- However, if you connect to the page using the HTTPS module, these headers are not sent. So, from the server's point of view, your request seems unnatural.

Download HTML from a URL

- Let’s add User-Agent.
- In practice, you need to do some trial and error to trick the server.

```
https.get(
    url,
    {
        headers: {
            "User-Agent":
"Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36",
        },
    },
    (res) => {
        let data = "";

        res.on("data", (d) => {
            data += d;
        });
        res.on("end", () => {
            console.log(data);
        });
    }
);
```

Download HTML from a URL

- Finally, I got the source code that I saw in Chrome!
- Let's extract menu names.

```
<!doctype html>
<html lang="ko">
<head>
<title>성균관대학교 | 대학생활 | 편의  /  복지 | 식당  /  메뉴 | 자연과학캠퍼스</title>
<link rel="shortcut icon" href="/_res/skku/img/common/favicon.png">
<meta name="viewport" content="width=device-width,initial-scale=1.0,minimum-scale=1.0,maximum-scale=1.0,user-scalable=no">
<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />
<meta http-equiv="Content-Script-Type" content="text/javascript" />
<meta http-equiv="Content-Style-Type" content="text/css" />
<meta http-equiv="X-UA-Compatible" content="IE=edge" />
<!-- Global site tag (gtag.js) - Google Analytics -->
<script async src="https://www.googletagmanager.com/gtag/js?id=UA-53596226-1"></script>
<script>
  window.dataLayer = window.dataLayer || [];
  function gtag(){dataLayer.push(arguments);}
  gtag('js', new Date());

  gtag('config', 'UA-53596226-1');
</script><link rel="canonical" href="http://www.skku.edu/skku/campus/support/welfare_11_1.do" />

<link rel="stylesheet" type="text/css" href="/_common/cms.css" />
<link rel="stylesheet" type="text/css" href="/_res/skku/_css/layout.css" />
```

Parse HTML

- An HTML document consists of tags.
- **Tags**: strings contained in angle brackets
  - Opening tag (e.g., <html>) vs Closing tag (e.g., </html>)

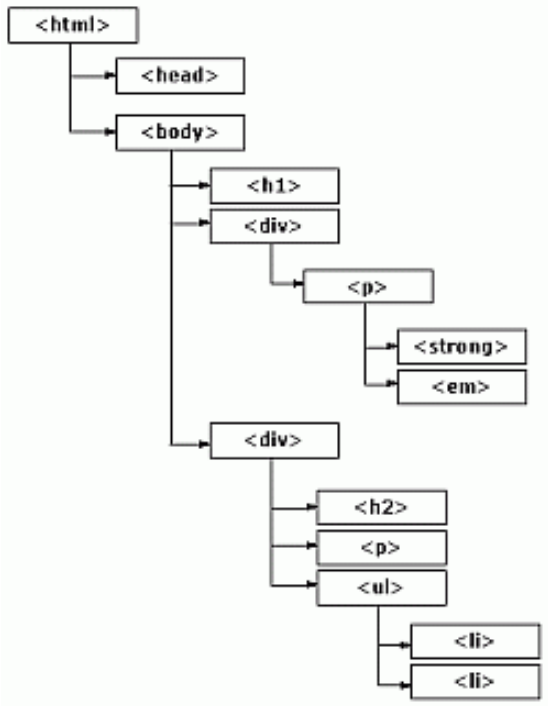
```
<html>
<head>...</head>
<body>
  <div>...</div>
</body>
</html>
```

Parse HTML

- Tags constitute a hierarchy (i.e., a tree).
- This hierarchy is called **a DOM tree.**
  - DOM: Document Object Model

- Finding menu names = finding tags that contain menu names in the DOM tree.

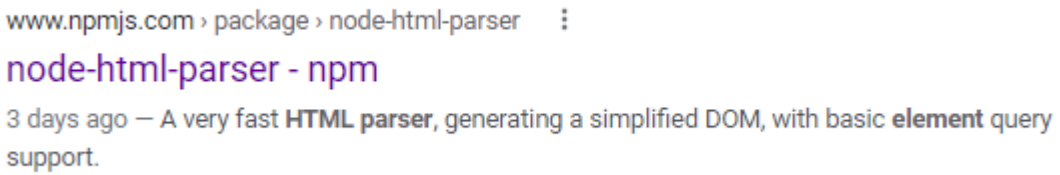
- However, the downloaded HTML is just text.
- Let’s reconstruct the DOM tree from HTML.



Parse HTML

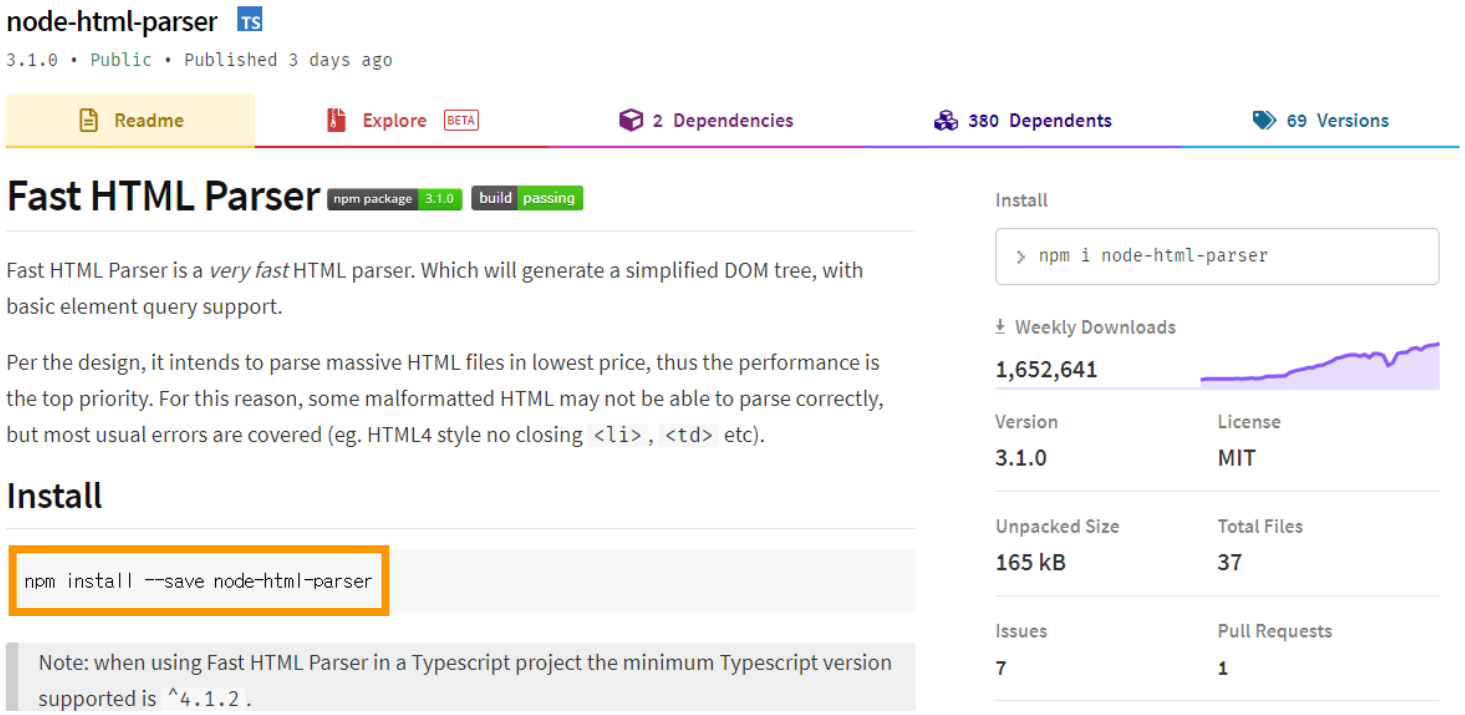
- **An HTML parser** constructs a DOM tree from an HTML document (text).

- I googled “node html parser”



Parse HTML

- Seemed well maintained (published 3 days ago).
- Seemed there are many users.
- Seemed to support the feature that I needed.



Parse HTML

- `npm install node-html-parser`
- *const parser = require("node-html-parser");*
- *let root = parser.parse(data);*

```
const https = require("https");
const parser = require("node-html-parser");

let url =
    "Url";

https.get(
    url,
    {
        headers: {
            "User-Agent": "long string ",
        },
    },
    (res) => {
        let data = "";

        res.on("data", (d) => {
            data += d;
        });
        res.on("end", () => {
            let root = parser.parse(data);
        });
    }
);
```

Parse HTML

- Take a look at the HTML code.
- The menu names are in a tag that has a class attribute “menu_title”.
- So, we will find tags by class names.

```
<h5>
    <span>해오름</span>

</h5>
<div class="corner_info">
    <ul>
        <li>


                    <img src="/_res/skku/img/sub/welfare_133.png" alt="믹스동 기본 썸네일 이미지" />


            <div class="menu_title">
                <pre>믹스동</pre>
            </div>
        </li>
        <li>

            <span>
                가격 :
                5300
            </span>
        </li>
        <li>

            <span>
                칼로리 :

            </span>
        </li>


    </ul>

    <p class="btnList">
        <a class="likeBtn good" href="#none" data-key="17928">좋아요 </a>
        <span>3</span>
    </p>
</div>
</div>
```

Parse HTML

- *root.querySelectorAll(".menu_title")* finds all tags who have a class name "menu_title" and returns an array of tags.
  - A dot before "menu_title" indicates it is a class name.
  - We will learn this later. So, don't panic.

```
res.on("end", () => {
    let root = parser.parse(data);
    root.querySelectorAll(".menu_title").forEach((menu) => {
        console.log(menu.innerText.trim());
    });
});
```

Parse HTML

- We will iterate the returned array of tags using the *forEach* method and print out the text (i.e., menu names) inside the tag.

- .trim() is called to remove leading and trailing whitespaces.
  - “ abc ”.trim() => “abc”

```
res.on("end", () => {
    let root = parser.parse(data);
    root.querySelectorAll(".menu_title").forEach((menu) => {
        console.log(menu.innerText.trim());
    });
});
```

Final Code

```
const https = require("https");
const parser = require("node-html-parser");

let url =
    "https://www.skku.edu/skku/campus/support/welfare_11_1.do?mode=info&srCategory=L&conspaceCd=20201251&srResId=12&srShowTime=D";

https.get(
    url,
    {
        headers: {
            "User-Agent":
                "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/88.0.4324.190 Safari/537.36",
        },
    },
    (res) => {
        let data = "";

        res.on("data", (d) => {
            data += d;
        });
        res.on("end", () => {
            let root = parser.parse(data);
            root.querySelectorAll(".menu_title").forEach((menu) => {
                console.log(menu.innerText.trim());
            });
        });
    }
);
```

Results

- `node main.js`

```
D:\skku-menu>node main.js
해물순두부찌개
데리야끼삼겹숙주덮밥
```

- Wait...! Do I have to run this script inside the project directory (D:\skku-menu)?
- For now, yes.

- But, you can install our package globally.

Develop a Command-line Interface

1. Add “#!/usr/bin/env node” on the first line of *main.js*.

```
#!/usr/bin/env node

const https = require("https");
const parser = require("node-html-parser");
```

2. Modify *package.json* as follows:

```
    "test": "echo \"Error
},
"bin": {
    "menu": "./main.js"
},
"author": "",
"license": "ISC",
```

- This “bin” property means that when this package is installed, you will expose a terminal command “menu” which runs “main.js”.

Develop a Command-line Interface

3. `npm pack`
   - This will pack your code to a self-contained package.

```
npm notice === Tarball Details ===
npm notice name:          skku-menu
npm notice version:       1.0.0
npm notice filename:      skku-menu-1.0.0.tgz
npm notice package size:  15.8 kB
npm notice unpacked size: 93.1 kB
npm notice shasum:        2535014d5e416954e3db2e47da03728f4a7121c3
npm notice integrity:     sha512-ITOzP5V9wMx74[...]OEjIS2g9zjmyg==
npm notice total files:   7
npm notice
skku-menu-1.0.0.tgz
```

4. `npm install skku-menu-1.0.0.tgz -g`
   - This will install your package globally.
5. `menu`

```
d:\>menu
해물순두부찌개
데리야끼삼겹숙주덮밥
```

Summary: Javascript Advanced

- You can reuse Javascript packages to speed up your development.
  - Don’t reinvent the wheel.
- Using external packages introduces a lot of versioning and dependency issues, but a package manager will resolve them.
  - Node Package Manager or NPM
- *package.json* contains the metadata of your project. You must include this file in your project.
- We first developed a small but useful command line tool using Javascript.