Node js

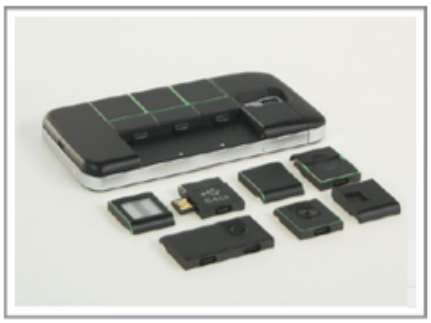
Node.js는 Chrome V8 JavaScript 엔진으로 빌드된 JavaScript 런타임.

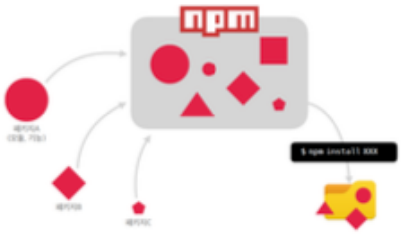
Node 설치하기

https://nodejs.org/ko/

npm 사용하기

NPM(Node Package Manager)은 전 세계의 개발자들이 만든 다양한 기능(패키지, 모듈)들을 관리.





Install 하기

```
npm init -y
```

```
npm install 패키지이름
```

개발용 의존성 패키지 (-D, —save-dev)

개발 당시만 필요하고 웹 브라우저 동작시에는 필요 없는 패키지

```
npm install parcel-bundler -D
```

일반 의존성 패키지

실제 웹브라우저에서 동작할때 필요한 패키지를

```
$ npm install jquery
```

번들러(Bundler) : parcel

번들러란

웹사이트를 만들기 위해선 HTML, CSS, JavaScript 이렇게 단 3가지의 재료만 있어도 가능하지만 이렇게 순수하게 3가지 재료만 가지고 웹사이트를 만드는 경우는 거의 없다.

HTML은 React나 Vue.js 등의 프레임워크로 대체되고, CSS는 Sass 같은 전처리기로 대체되곤 합니다. 그리고 JavaScript 대신 TypeScript를 사용하기도 하며 여기에 Firebase, Lodash 등 다양한 써드 파티 모듈(3rd party modules)까지 추가 사용된다.

대부분의 디펜던시(dependency)들이 CommonJS 방식을 사용하는데 ES6 모듈 문법과 맞지 않아 에러가 발생 되기도 하고 거대한 자바스크립트 파일 때문에 구동이 늦어지고, 구형 브라우저(legacy browser)에서도 코드가 돌아가도록 하기 위해 폴리필(polypill)도 추가하기도 한다.

모듈 번들러가 이러한 번거로운 과정을 도와 줄 수 있다. 모듈 번들러가 근본적으로 하는 일은 다수의 자바스크립트 파일과 그 밖의 써드 파티 모듈들을 가지고 브라우저에서 로드되는 하나의 커다란 자바스크립트 파일(즉 자바스크립트 번들)로 만드는 것입니다.

폴리필 : 브라우저에서 지원하지 않는 코드를 사용가능한 코드 조각이나 플러그인(추가기능)을 의미한다.

Parcel Bundler

설치하기

1. 폴더 생성하기
2. 초기화하여 package.json 파일 생성하기

```
npm init -y
```

3. parcel-bundler 설치하기

```
npm i -D parcel-bundler
```

i: install 약자

-D: dev Dependencis 의 약자 개발 의존성 패키지로 개발 당시만 필요하고

웹브라우저 동작시에는 필요없는 패키지

dependencies : 일반 의존성 패키지 실제 웹브라우저에서 동작 할때 필요한 패키지

package.json 설정하기

```
"scripts": {
   "dev": "parcel index.html",
   "build": "parcel build index.html"
 },
```

필요한 플러그인 셋업하기

외부파일 연결하기

1. install 하기

   ```
   npm i parcel-plugin-static-files-copy
   ```

2. 이미지 파일 준비하기 :
3. html 문서에 추가하기
4. static 폴더만들기
5. static 폴더에 favicon 추가하기
6. config 구성하기

Autoprefixer 설정하기

1. install 하기

```
`npm i -D postcss autoprefixer
```

2. root 폴더에 css 폴더 추가하기
3. 버전관리하기

```
` "browserslist": [
    "> 1%",
    "last 2 versions"
  ]
```

browserslist : NPM 프로젝트에서 지원할 브라우저의 범위를 명시하는 용도이다. 그 명시를 Autoprefixer패키지가 활용하게 됩니다.

>1% : 전 세계 브라우저 중 1% 이상인 모든 브라우저

last 2 version : 마지막의 2개 버전

4. rc 파일 만들기

   root에 .postcssrc.js 파일 만들기

```
module.exports = {
  plugins: [
    require('autoprefixer')
  ]
}
```

rc(Runtime Configuration)의 약자로 구성 파일입니다.

babel 연결하기

1. install 하기

   npm i -D @babel/core @babel/preset-env @babel/plugin-transform-runtime

2. root 폴더에 .babelrc.js 생성하기

```
module.exports = {
  presets: ['@babel/preset-env'],
  plugins: [
    ['@babel/plugin-transform-runtime']
  ]
}
```

Import , export

Import

import 문은 다른 모듈에서 내보낸 바인딩을 가져올 때 사용합니다. 경로 지정시 js 파일은 확장자를 생략해도 된다.

```
Import 이름 from ' 경로' ;
Import {모듈에서 하나의 멤버만 가져올 때 } from ' 경로' ;
```

```
import myModule from "my-module";
import myDefault, {foo, bar} from "my-module";
```

Export

export 문은 JavaScript 모듈에서 함수, 객체, 원시 값을 내보낼 때 사용합니다. 내보낸 값은 다른 프로그램에서 import 문으로 가져가 사용할 수 있습니다.

내보내는 모듈은 "use strict"의 존재 유무와 상관없이 무조건 엄격 모드입니다. export 문은 HTML 안에 작성한 스크립트에서 는 사용할 수 없습니다

내보내기에는 두 종류, 유명(named)과 기본(default) 내보내기가 있습니다. 모듈 하나에서, 유명 내보내기는 여러 개 존재할 수 있지만 기본 내보내기는 하나만 가능합니다. 각 종류는 위의 구문 중 하나와 대응합니다

Named 방법

```
// 먼저 선언한 식별자 내보내기 export { myFunction, myVariable };
// 각각의 식별자 내보내기
// (변수, 상수, 함수, 클래스)
export let myVariable = Math.sqrt(2);
export function myFunction() { ... };
```

Default 방법

```
// 먼저 선언한 식별자 내보내기
export { myFunction as default };
// 각각의 식별자 내보내기
export default function () { ... };
export default class { ... }
```