ATOM Configuration Guide

softcontext@gmail.com
2017-01-11

1. ATOM 설치 및 JavaScript 기동 테스트

Atom is a text editor that's modern, approachable, yet hackable to the core—a tool you can customize to do anything but also use productively without ever touching a config file.
Atom is a desktop application built with HTML, JavaScript, CSS, and Node.js integration.

- Cross-platform editing
- Built-in package manager
- Smart autocompletion
- File system browser
- Multiple panes
- Find and replace

1.1. Installing Node.js

https://nodejs.org/ko/download/

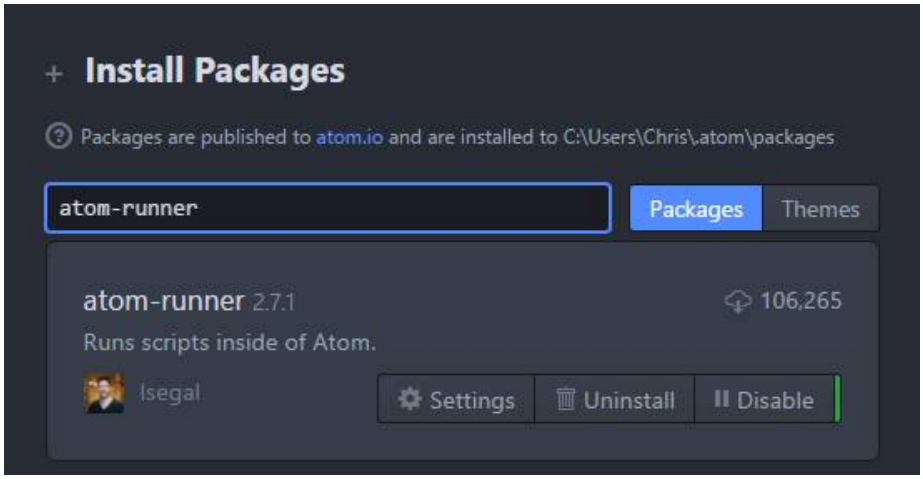
1.2. Installing Atom Editer

https://atom.io/

1.3. JavaScript Settings : atom-runner

File > Settings > Install >
type 'atom-runner' in search box and enter > Install

다음 페이지 그림을 참고하세요.



atom 에서 **alt+R** 을 눌러 javascript, ruby 파일 등을 바로 실행할 수 있습니다.

1.4. Test : JS Running

File > Open Folder >
make folder 'hello-world' or select target folder >

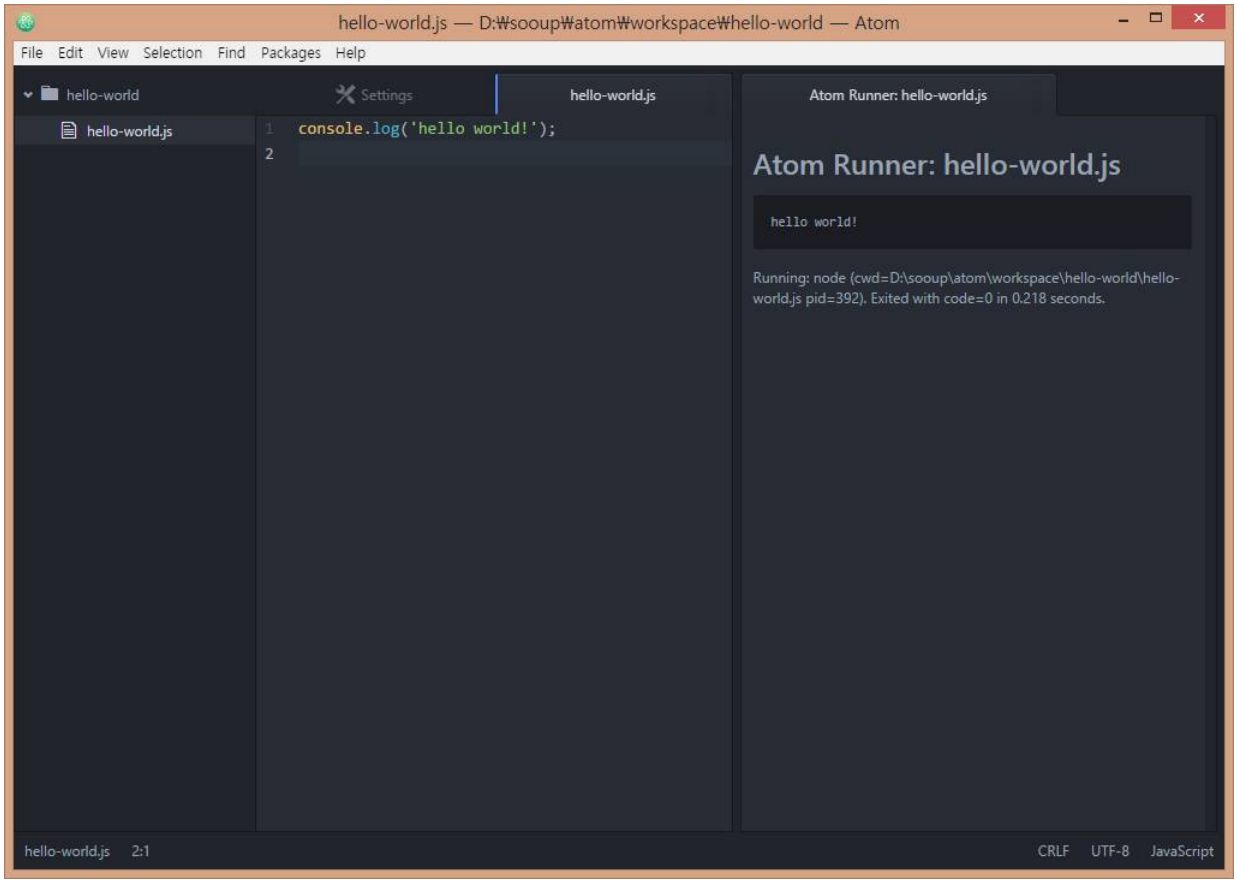
New File >
type hello-world.js

hello-world.js

```
console.log('hello world!');
```

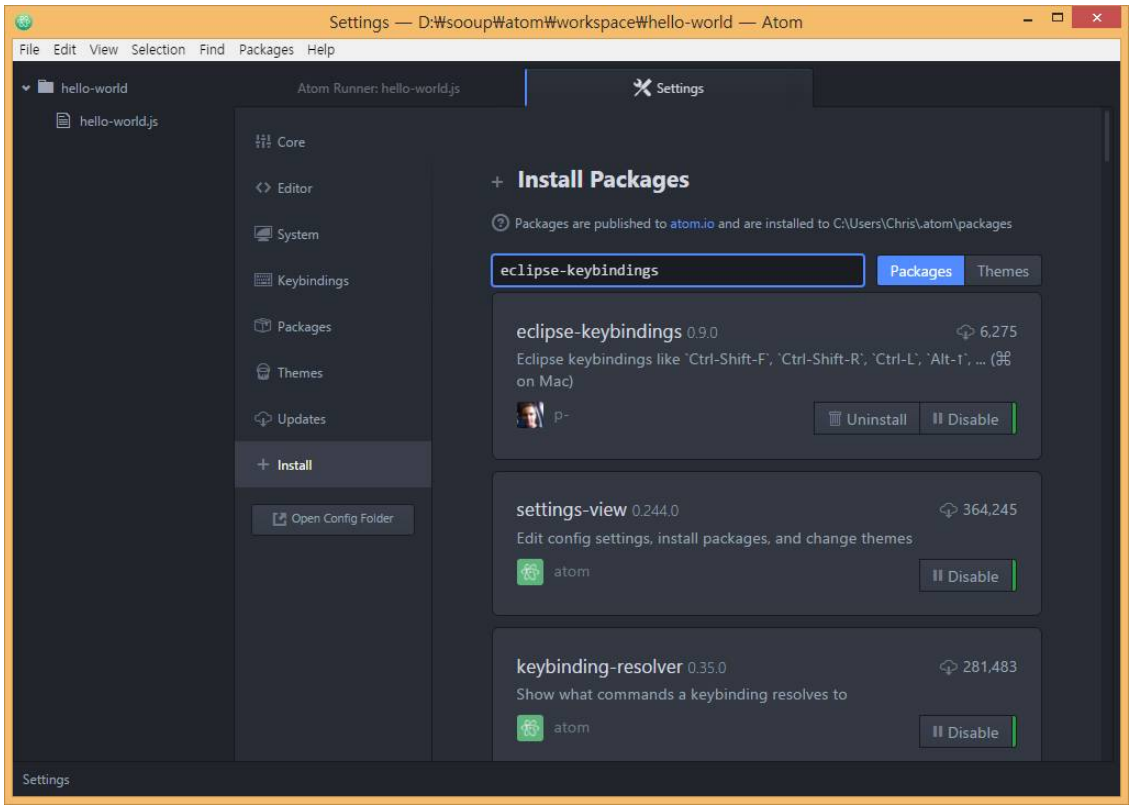
save file > **alt + r**

다음 페이지 결과화면을 참고하세요.


hello-world.js — D:₩sooup₩atom₩workspace₩hello-world — Atom
File Edit View Selection Find Packages Help
hello-world
Settings
hello-world.js
Atom Runner: hello-world.js
hello-world.js
1 console.log('hello world!');
2
Atom Runner: hello-world.js
hello world!
Running: node (cwd=D:\sooup\atom\workspace\hello-world\hello-world.js pid=392). Exited with code=0 in 0.218 seconds.
hello-world.js 2:1
CRLF UTF-8 JavaScript

2. Use Eclipse Shortcut : eclipse-keybindings

File > Settings > Install >
type 'eclipse-keybindings' and enter > Install



인스톨 하는 것만으로 이클립스에서 사용하던 단축키를 사용할 수 있다.

keybinding-cheatsheet

아톰이 제안하는 단축키를 그대로 사용하고자 할 때, 단축키를 쉽게 조회할 수 있는 추천 패키지

Quickly view and filter atom keybindings.

3. Install Useful Packages

ATOM 에서 패키지를 추가해서 기능을 확장할 수 있다. 패키지는 애드온, 플러그인, 익스텐젼이라 부르는 개념과 비슷하다.

highlight-selected

단어를 더블클릭했을때 해당 같은 단어들을 표시한다.

linter

Linter is a base linter provider for the Hackable Atom Editor. It provides a top-level API to its consumer that allows them to visualize errors and other kind-of messages, easily.
해당 파일에 문법 오류가 있을때 알려준다. 이 패키지는 기능을 제공하는 패키지이고 linter 데이터는 따로 설치한다. 아래 linter-jshint 를 먼저 설치하면 같이 설치된다.

linter-jshint

jshint 데이터를 linter 에 추가한다. javascript 파일(.js)의 문법 오류를 알려준다.

linter-csslint

csslint 데이터를 linter 에 추가한다. css 파일(.css)의 문법 오류를 알려준다.

file-icons

Pretty and good for visual grepping, these filetype icons appear next to files in your filetree, fuzzy finder, and tabs. You have the choice of colored or monochrome icons.

autocomplete-modules

Autocompletes require/import statements for you by following the path you type and showing you the available files at that location.

ask-stack

Quickly get answers and code samples from Stack Overflow in Atom

atom-beautify

Beautify HTML (including **Handlebars**), CSS (including Sass and Less), JavaScript, and much more in Atom. 적용하는 방법 : Packages > Atom Beautify > Beautify 클릭!

atom-jade

Jade language support in Atom

테마

https://atom.io/themes 사이트를 참고해서 마음에 드는 테마를 추가하세요.

autocomplete-snippets

Simply type 'html' and press Enter for basic structure.

autoclose-html

Will automatically add closing tags when you complete the opening tag.

open-in-browser

Simple Package to open file in default Browser

docblocker

A helper package for writing documentation

emmet

the essential tool for web developers

atom-ternjs

JavaScript code intelligence for atom with Tern. Adds support for ES5, ES6 (JavaScript 2015), Node.js, jQuery & Angular. Extendable via plugins. Uses suggestion provider by autocomplete-plus.

auto-detect-indentation

Automatically detect indentation of opened files.

pigments

A package to display colors in project and files.

editorconfig

Helps developers maintain consistent coding styles between different editors

resize-indent

Easily change the indent size of a file

4. TypeScript Settings : atom-typescript

4.1. 패키지 설치

File > Settings > Install >

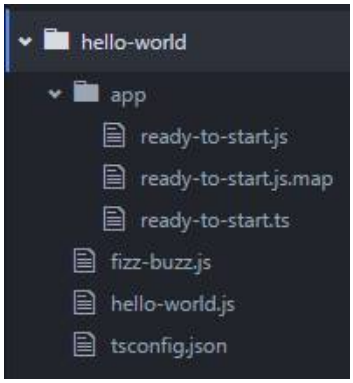
type 'atom-typescript' in search box and enter > Install

4.2. 기능 설명

https://atom.io/packages/atom-typescript

4.3. Test

테스트 프로젝트 구조



tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "module": "commonjs",
    "emitDecoratorMetadata": true,
    "experimentalDecorators": true,
```

```
        "sourceMap": true,
        "suppressImplicitAnyIndexErrors":true
    },
    "compileOnSave": true,
    "buildOnSave": false,
    "exclude": [
        "node_modules"
    ],
    "filesGlob": [
        "app/**/*.ts",
        "typings/index.d.ts"
    ],
    "atom": {
        "rewriteTsconfig": false
    }
}
```

tsconfig.json 파일은 TypeScript 를 위한 프로젝트 단위 환경 파일이다. Exclude 하고 싶은 폴더나 파일, Compile 옵션, File 옵션등을 설정 할 수 있다. filesGlob 항목의 설정에 따라 app 폴더 밑에 .ts 확장자로 끝나는 파일을 대상으로 한다.

app/ready-to-start.ts

```
console.log('ready to start TypeScript!');
```

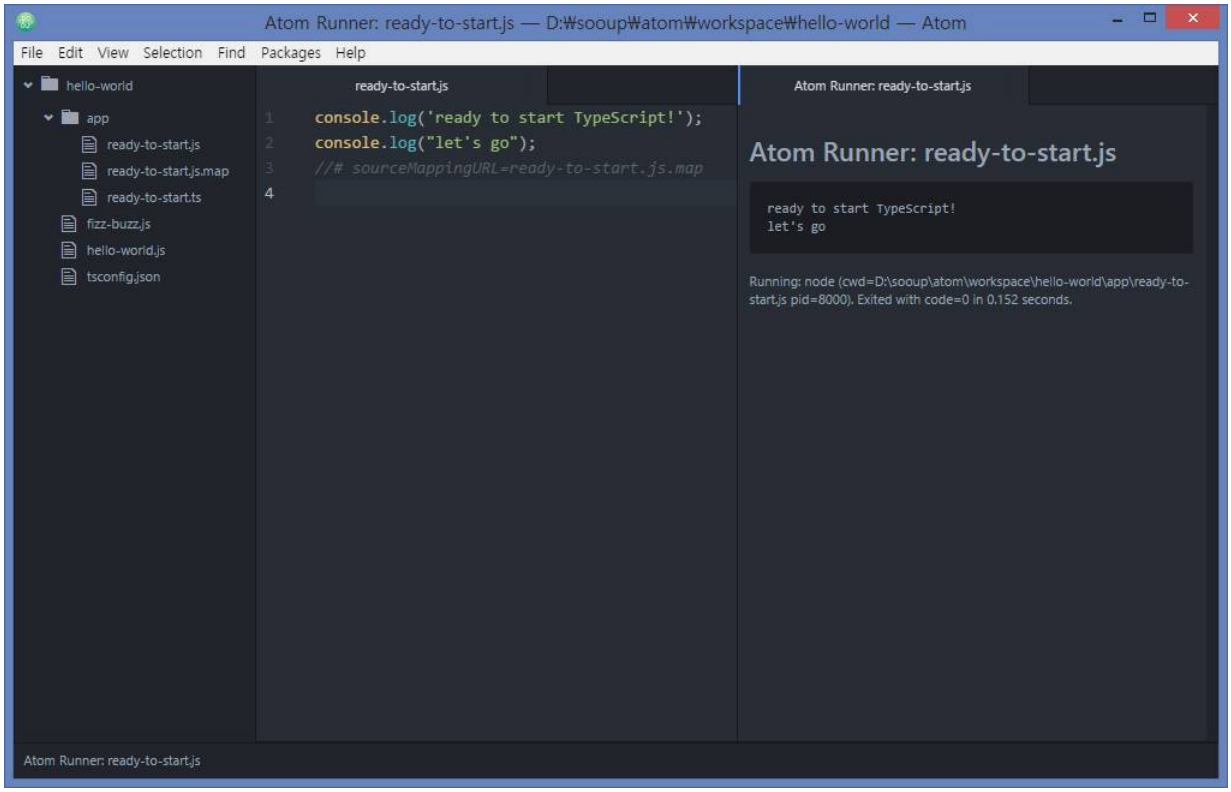
트랜스파일링

shortcut for compiling : **F6**

수행결과로 ready-to-start.js 파일과 ready-to-start.js.map 파일이 만들어 진다.

ready-to-start.js 파일 실행

shortcut for running : **alt + r**



4.4. Test : One More Time

test-interface.ts

```
interface IGreeter {
    greet(): string;
}

class Greeter implements IGreeter {
    greeting: string;

    constructor(message: string) {
        this.greeting = message;
    }

    greet(): string {
        return 'Hello, ' + this.greeting;
```

```
    }
}

var greeter = new Greeter('Everybody');
var result = greeter.greet();
console.log(result);
```

tsconfig.json

타입스크립트 설정파일에서 "compileOnSave": true 설정이 되어 있다면 ATOM 이 자동으로 트랜스파일링 작업을 수행한다.

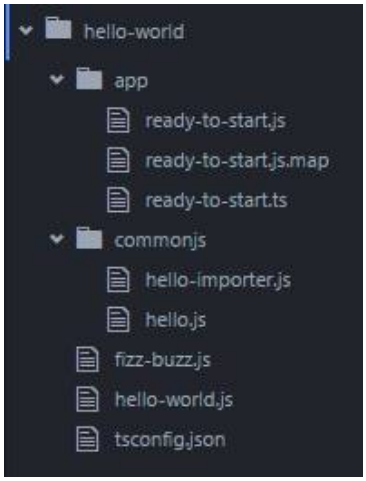
test-interface.js 파일 실행

shortcut for running : **alt + r**

5. Module Test : CommonJS

노드가 선택한 모듈기술이다. 노드를 컴파일러로 사용하기 때문에 특별한 부가 설정없이 그대로 사용할 수 있다.

테스트 프로젝트 구조



commonjs/hello.js

```
let hello = function(who){
   console.log('hello '+who);
};

exports.hi = hello;
```

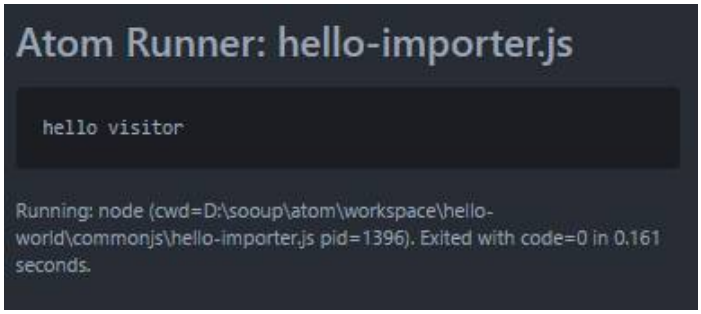
commonjs/hello-importer.js

```
let hi = require('./hello').hi;

hi('visitor');
```

Test

shortcut for running : **alt + r**



다음 챕터에서 노드가 사용하는 CommonJS 방식의 모듈시스템 대신 ES6 가 제안하는 모듈시스템을 사용해 볼 것이다. 이를 통해 ES6 문법을 ES5 문법으로 변경하여 사용하는 방법을 살펴본다.

6. Use ES6 in ATOM : Babel(6to5)

앞 챕터 6 에서는 노드가 지원하는 CommonJS 방식의 모듈시스템을 사용했다.
이번에는 ES6 가 제안하는 모듈시스템을 사용해 보자.

6.1. Babel 의 필요성 확인

hello.js

```
var asia = function () {
  console.log('Hello Asia');
};

export let korea = asia;

export let world = function () {
  console.log('Hello World');
};
```

hello-importer.js

```
import {world, korea} from './hello';

world();
korea();
```

테스트 실행

위 파일에서 실행(alt+r)하면 에러가 발생한다.

```
D:₩sooup₩atom₩workspace₩hello-es6₩es-import₩hello.js:7
export let korea = asia;
^^^^^^
SyntaxError: Unexpected token export
... 생략
```

노드가 지원하지 않는 export/import 문법을 사용할 수 없음을 파악했다.
바벨을 사용해서 6to5 작업을 통해 실행할 수 있는 소스코드(노드가 지원하는 문법의 코드)를 얻도록 설정해 보자.

참고로 언급하자면 노드 버전 4.0 부터 ES6 문법인 class 키워드를 지원한다. 따라서 ATOM 에서 .js 확장자로 파일을 만들고 그 안에서 class 문법을 사용하고 바로 실행하는 것이 가능하다.

6.2. language-babel 패키지

1. 설치

File > Settings > Install >
type 'language-babel' in search box and enter > Install

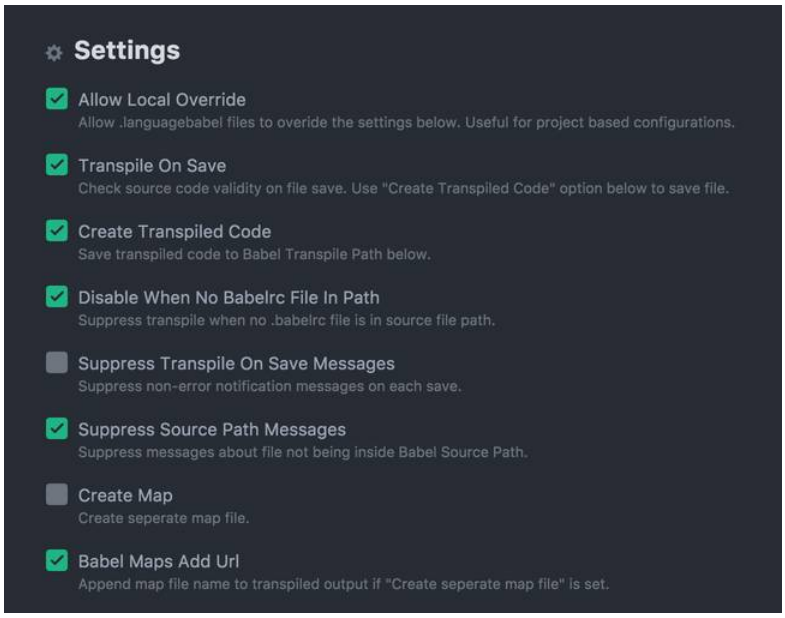
2. 패키지 기능 설명

https://atom.io/packages/language-babel

3. 패키지 설정

File > Settings > Packages >
type 'language-babel' in Installed Packages Search Box >
Community Packages > click 'Settings' button

다음화면을 참고하여 설정을 변경한다.



6.3 프로젝트 환경 설정

1. package.json 파일생성

```
npm init --yes
```

2. 바벨 디펜던시 설치

```
npm install --save-dev babel-core babel-preset-es2015 babel-preset-es2017
npm install --save-dev babel-preset-react babel-plugin-syntax-flow
npm install --save-dev babel-plugin-transform-es2015-arrow-functions
npm install --save-dev babel-plugin-transform-class-properties
```

3. 바벨 설정파일 작성

.babelrc

```
{
   "only": [ "*.babel", "*.jsx", "*.es6" ],
   "presets": [ "es2015", "es2017", "react" ],
   "plugins": [ "transform-es2015-arrow-functions", "transform-class-properties" ]
}
```

.languagebabel

```
{
"babelMapsPath": "",
"babelMapsAddUrl": false,
"babelSourcePath": "",
"babelTranspilePath": "",
"createMap": false,
"createTargetDirectories": false,
"createTranspiledCode": true,
"disableWhenNoBabelrcFileInPath": true,
```

```
"suppressSourcePathMessages": false,
"suppressTranspileOnSaveMessages": false,
"transpileOnSave": true
}
```

For most projects, it is better to configure language-babel via project-based **.languagebabel** file properties which will override the package settings.

Transpile On Save

On any file save of a language-babel grammar enabled file the Babel transpiler is called. No actual transpiled file is saved but any Babel errors and/or successful indicators are notified by an ATOM pop-up. Not all files are candidates for transpilation as other settings can affect this. For example see Disable When No Babelrc File In Path and Babel Source Path below.

```
{"transpileOnSave": true} or {"transpileOnSave": false}
```

.languagebabel 파일의 기타 설정은 사이트를 참고한다.

https://atom.io/packages/language-babel

4. 트랜스파일링 작업대상으로 설정하기

hello.js/ hello-importer.js 파일의 확장자를 .es6 로 모두 바꾼다.

파일을 저장하는 단축키(ctrl-s)를 누르면 바벨이 트랜스파일링을 진행하여 파일명은 같고 확장자가 .js 인 파일을 같은 폴더위치에 생성한다.

5. 테스트

확장자가 .js 인 파일을 실행(alt+r)한다.

7. Angular Project Generator : angular-cli

7.1. 설치

```
npm install -g angular-cli
```

7.2. 프로젝트 만들기

프로젝트에 필요한 디펜던시가 자동으로 다운로드되어 추가된다.

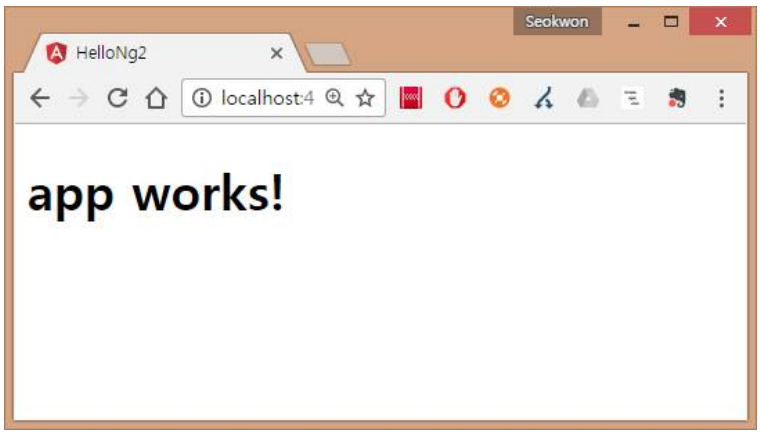
```
ng new hello-ng2
```

7.3. 개발서버 실행

```
cd hello-ng2
ng serve
```

7.4. 개발서버 접속

http://localhost:4200



8. Express Project Generator : express-generator

8.1. 설치

```
npm install -g express-generator
```

8.2. 프로젝트 만들기

```
express --view=jade hello-express
```

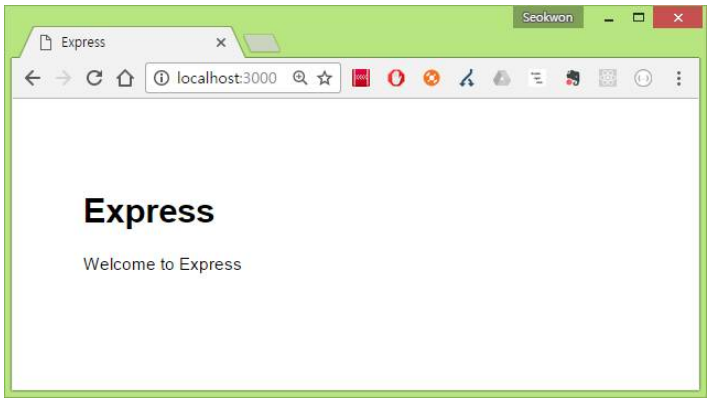
8.3. 디펜던시 설치

```
cd hello-express
npm install
```

8.4. 서버 실행

```
npm start
```

8.5. 서버 접속



9. React Project Generator : create-react-app

9.1. ATOM 패키지 react 설치

```
File > Settings > Install >
type 'react' in search box and enter > Install
```

2017-01-18 language-bable 패키지와 충돌이 발생한다. 해결될 때까지 사용하지 않도록 한다.

9.2. react 패키지 기능 설명

https://orktes.github.io/atom-react/

9.3. React Project Generator 설치

```
npm install -g create-react-app
```

9.4. create-react-app 기능 설명

https://facebook.github.io/react/blog/2016/07/22/create-apps-with-no-configuration.html

9.5. Test

1. 테스트 프로젝트 만들기

```
create-react-app hello-react
```

2. 디펜던시 설치

```
cd hello-react
npm start
```

3. 개발서버 접속

수고하셨습니다.