스프링 부트 + 리액트 개발 셋업 2018

게시일: [2018-09-14](https://start.goodtime.co.kr/2018/09/%ec%8a%a4%ed%94%84%eb%a7%81-%eb%b6%80%ed%8a%b8-%eb%a6%ac%ec%95%a1%ed%8a%b8-%ea%b0%9c%eb%b0%9c-%ec%85%8b%ec%97%85-2018/)

오늘 오랜만에 올리는 글은 [스프링 부트](https://spring.io/projects/spring-boot) + [리액트](https://reactjs.org/) 개발을 2018년 현재 시점의 도구들로 셋업하는 방법에 대해 써보고자 한다. 아직까지 상용 프로젝트에서는 두 가지 조합을 본 적이 없지만 현재까지 추세나 장점을 볼 때 앞으로 대세가 될 것이라 확신한다. 우리나라 스프링 기반의 상용 웹 프로젝트를 보면 UI는 JSP를 기본으로 jQuery로 DOM 조작을 직접하거나 X-Platform이나 Nexacro 같은 상용 프레임웍을 사용하는 경우가 많은데 React는 학습이 쉽고 UI 컴포넌트화, 재사용성이 좋고 개발자 층이 두터운 등 다양한 이유로 UI 프레임웍의 대표 주자가 되고 있다. React를 추천하기도 하고 이미 대세가 된 또 한 가지 큰 이유는 웹 아키텍처의 변화다. 웹에서 클라이언트가 갈수록 복잡한 기능을 담당하기 때문에 시스템 구조도 그에 맞게 서버측은 API 역할만 담당하고 사용자 업무 로직 상당 부분을 클라이언트에서 처리하는 경우가 많아졌다는 것이다. Eclipse에서 JSP를 코딩하던 방식으로는 이런 복잡한 UI나 업무 처리가 쉽지 않다. Angular를 포함한 여러 UI 프레임웍은 이런 환경에서 개발자에게 인기가 많아지고 있다. 이 글은 스프링 부트, 리액트 각각에 대한 소개글이 아니므로 각각을 공부하려는 경우는 별도 자료를 찾아보도록 한다.

**준비**

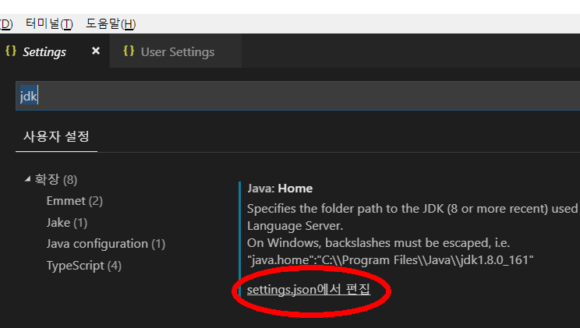
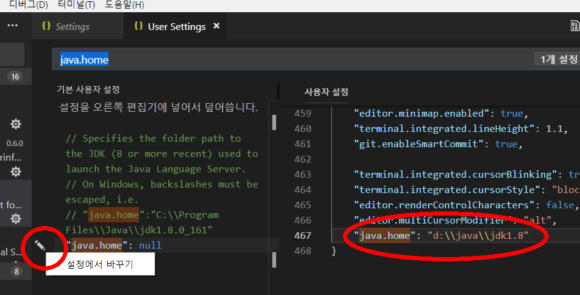
내가 권장하는 스프링 부트 + 리액트 개발 환경의 도구들은 다음과 같다. 아직 설치 안했다면 링크를 눌러 하나씩 설치해보도록 한다.

* [Visual Studio Code](https://code.visualstudio.com/download) – “뭐라구요! 아직까지 Visual Studio Code를 플레이해본 적이 없다구요? 농담이지요?”  이미 많은 사람이 텍스트 에디터로서도 Visual Studio Code를 사용하고 있을 것이다. 워낙 기능이 커지고 있어서 이제는 Eclipse를 대체할 IDE로 사용할 수 있을 정도다. 특히 JavaScript 개발에 있어서의 기능은 다른 어떤 도구들도 따라오지 못할 정도가 되고 있어서 VSC 사용을 강력 권장한다.
* [Node.js](https://nodejs.org/en/download/) – 이제 JavaScript 개발에 있어서 필수도 아니고 기본 도구라고 할 수 있을 정도로 일반화가 된 것 같다. 웹 앱을 개발하더라도 JavaScript를 사용한다면 Node.js를 설치하는 것이 점점 더 편한 시대가 되었다.
* [JDK](https://www.oracle.com/technetwork/java/javase/downloads/index.html) – 자바 개발 환경을 셋업하는 데에 당연히 필요한 항목이다. 그런데 버전은 이제 8 이상이어야 한다. 이미 Oracle에서는 10을 대표 버전으로 내세우고 있다.

**Visual Studio Code 확장**

비주얼 스투디오 코드의 기능을 확장하는 프로그램이 상당히 많다. 여기서는 다음 두 가지를 설치하는 것이 좋다. 비주얼 스투디오 코드에서 ctrl-shift-x를 누르거나 사이드바에서 확장 아이콘을 누른 후 검색란에서 검색하여 설치한다.

* Java Extension Pack (Microsoft) – Java 언어 지원 기능(Eclipse 도구 활용), 디버거, 테스트 실행, Maven 프로젝트 관리 등의 확장을 패키징한 것이다.
* Spring Boot Extension Pack (Pivotal) – Spring 프레임웍에 적용할 수 있는 몇 가지 유용한 기능이 들어있다.

위 확장을 설치했으면 JDK 위치를 알려줘야 한다. ctrl-,를 누르거나 메뉴에서 기본 설정 > 설정으로 들어간 후 jdk를 검색하고 편집 기능을 통해 사용자 설정에 java.home 값을 넣도록 한다.[](http://start.goodtime.co.kr/wp-content/uploads/2018/09/vcs-jdk-setting.png)[](http://start.goodtime.co.kr/wp-content/uploads/2018/09/vcs-jdk-setting2.png)마찬가지 방법으로 spring-boot.ls.java.home 설정에 대해서도 JDK 경로를 설정해주도록 한다.

**Spring Boot 웹 프로젝트 생성**

도구가 준비됐으면 프로젝트부터 생성해보자. Visual Studio Code에서 ctrl-shift-p를 누르거나 보기 > 명령 팰릿을 선택하여 spring을 검색하면 나오는 *Spring Initalizr: Generate Maven Project Spring*을 선택한다. 연달아 나오는 질문에서 언어로 Java를 선택하고 Group Id는 여기서는 myweb, Artifact Id는 test로 하겠다. 또한 Spring Boot 버전은 최신인 2.0.5를 선택하고 의존성은 필요에 따라 선택하면 된다. 기본적으로 다음 두 가지는 포함하도록 한다.

* DevTools – 클래스 소스 변경시 재시작 등의 기능이 있다.
* Web – Tomcat을 기반으로 서블릿 API를 사용할 수 있는 프로젝트가 된다.

이제 폴더 선택 대화창이 나타나면 임의로 적당한 위치에 폴더를 만들고 선택한다. 그 다음 폴더를 열겠냐는 팝업에서 열기를 선택하면 프로젝트가 열리면서 준비 과정에 약간 시간이 지나간다. 만들어진 폴더 구조는 Maven에 의해 만들어진 것이므로 Eclipse에서와 거의 동일하다. /src/main/webapp 폴더만 직접 만들어주도록 한다. 개인적으로 VSC Java 도구에서 한 가지 불만은 Ecipse처럼 자바 클래스 트리를 플랫하게 보여주지 않는다는 것이다. 나는 트리 구조보다 플랫한 목록형 나열이 훨씬 보기 좋다고 생각한다. 프로젝트에서 JSP를 사용하려면 라이브러리 하나를 추가해야 한다. pom.xml 파일을 열고 <dependencies> 섹션에 다음을 추가한다.

**<dependency>**

**<groupId>**org.apache.tomcat.embed**</groupId>**

**<artifactId>**tomcat-embed-jasper**</artifactId>**

**<scope>**provided**</scope>**

**</dependency>**

참고로 JSTL을 사용하려면 다음 의존성도 필요하다.

**<dependency>**

**<groupId>**javax.servlet**</groupId>**

**<artifactId>**jstl**</artifactId>**

**<scope>**provided**</scope>**

**</dependency>**

스프링 부트에서 프로젝트 설정은 어노테이션이나 src/main/resources/application.properties로 처리되는데 개인적으로 properties 파일은 한글을 쓰기도 어렵고 설정 값을 구별해 보기도 불편해서 yaml 파일로 바꿀 것을 권장한다. application.properties 파일을 삭제하고 아래와 같이 application.yml 파일을 추가한다.

spring:

http:

encoding:

charset**:**  UTF-8

mvc:

view:

prefix**:**  /jsp/

suffix**:**  .jsp

server:

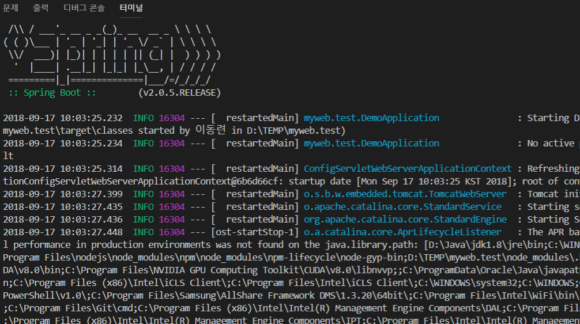
tomcat:

uri-encoding**:** UTF-8

현 시점에서 웹 프로젝트를 실행하면 정상 구동돼야 한다. ctrl+shift+` 또는 메뉴에서 터미널 > 새 터미널을 선택한 후 .\mvnw spring-boot:run 또는 mvnw test를 실행하면 오류 없이 실행돼야 한다. 아래는 구동 과정의 콘솔 출력 일부다.

시스템환경변수 추가 반드시 할 것 – 위명령어 실행전에 꼭

JAVA\_HOME c:\jdk1.8

[](http://start.goodtime.co.kr/wp-content/uploads/2018/09/vcs-spring-boot-1.png)(윈도 명령 프롬프트에서는 기본적으로 색이 나오지 않는데 위 설정에 spring.output.ansi.enabled 값을 always로 설정하면 나온다.)

**React 개발 환경 셋업**

이번에는 React 환경을 셋업해보자. 터미널을 하나 더 연 후 npm init을 실행하여 앞서 만든 프로젝트를 Node.js 프로젝트로 만든다(결과적으로 package.json 파일이 생기는 것 뿐이다). 모든 질문에 엔터만 눌러도 무방하다. 다음으로 아래와 같이 입력하여 React 개발을 위한 의존 라이브러리를 설치한다.

d:\myweb.test>npm i react react-dom

d:\myweb.test>npm i @babel/core @babel/preset-env @babel/preset-react babel-loader css-loader style-loader webpack webpack-cli -D

설치 결과 package.json 파일에 아래 내용과 같이 의존성이 추가됐을 것이다.[](http://start.goodtime.co.kr/wp-content/uploads/2018/09/vcs-package-json.png)여기서 한 가지, React 앱을 개발할 때 사실 [create-react-app](https://github.com/facebook/create-react-app)이라는 훌륭한 도구를 사용할 수 있는데 이 절 이하에서 복잡하게 수동으로 설정하는 이유는 상용으로 개발하는 Java 웹 프로젝트 대부분은 다수의 웹페이지로 구성되는데 반해 create-react-app은 단일 페이지 앱(SPA)만 기본적으로 염두에 두었다는 점 때문이다. 상용 프로젝트에서도 조건만 맞으면 SPA 방식으로 개발하는 것도 문제가 없겠으나 좀더 복잡한 시나리오나 조건을 고려하여 이렇게 다수 웹 페이지인 경우로 설정해보고자 한다.

**webpack 설정**

다음으로는 webpack 설정을 추가한다. webpack을 통해 React 개발시 JavaScript의 최근 언어 기능을 사용할 수 있고 JSP에 포함할 .js 파일들을 만들 수 있다. 이 과정은 자동화가 안되므로 그냥 아래 내용을 프로젝트 루트 경로에 webpack.config.js라는 파일로 만들도록 한다.

**var** path = require('path');

module.exports = {

context: path.resolve(\_\_dirname, 'src/main/jsx'),

entry: {

main: './MainPage.jsx',

page1: './Page1Page.jsx'

},

devtool: 'sourcemaps',

cache: **true**,

output: {

path: \_\_dirname,

filename: './src/main/webapp/js/react/[name].bundle.js'

},

mode: 'none',

module: {

rules: [ {

test: */\.jsx?$/*,

exclude: */(node\_modules)/*,

use: {

loader: 'babel-loader',

options: {

presets: [ '@babel/preset-env', '@babel/preset-react' ]

}

}

}, {

test: */\.css$/*,

use: [ 'style-loader', 'css-loader' ]

} ]

}

};

내용은 다음과 같다.

* React 소스 경로는 src/main/jsx로 한다
* (현재로서는) MainPage.jsx, Page1Page.jsx 두 스크립트를 각각 빌드한다
* 빌드한 결과 js 파일들은 src/main/webapp/js/react 아래에 [웹페이지 이름].bundle.js 파일로 놓는다
* .jsx 파일은 ES6 등의 JavaScript 구문을 사용하며 React 스크립트다
* .css 파일은 웹페이지내에 포함한다

**서버 코드 개발**

이제 코딩에 들어갈 수 있게 됐다. VSC에서 임의 패키지 경로에 MyController라는 클래스 파일을 만들어보자. 내 계획은 일단 Java단에서 처리할 로직은 없으므로 다수의 URL을 한 매핑으로 처리하되 React 스크립트만 구별해서 JSP가 로딩하게 하는 것이다. 코드를 입력할 때 Eclipse에서와 마찬가지로 내용 도움(content assist)이 잘 작동하므로 Java 코딩 작성도 손쉽게 할 수 있다. 혹시 import가 안된 클래스나 annotation이 있으면 alt-shift-o를 눌러 자동으로 찾게 한다.

**package** myweb.test;

**import** org.springframework.stereotype.Controller;

**import** org.springframework.ui.Model;

**import** org.springframework.web.bind.annotation.GetMapping;

**import** org.springframework.web.bind.annotation.PathVariable;

@Controller

**public** **class** MyController {

@GetMapping("/{name}.html")

**public** String page(@PathVariable String name, Model model) {

model.addAttribute("pageName", name);

**return** "page";

}

}

그리고 이번엔 src/main/webapp/jsp 경로 아래에 page.jsp라는 파일을 만든다. webpack에서 만들어낼 [페이지 이름].bundle.js에 해당하는 스크립트를 불러들이는 부분이 들어가 있다. 또한 React의 최상위 요소가 되는 div도 들어가 있다.

<%@ page language="java" contentType="text/html; charset=utf-8"

%>

<!doctype html>

<**html**>

<**head**>

<**title**>${pageName}</**title**>

</**head**>

<**body**>

<**div** id="root"></**div**>

<**script** src="/js/react/${pageName}.bundle.js"></**script**>

</**body**>

</**html**>

이번엔 src/main/webapp/css/custom.css 파일을 만든다. webpack에서 CSS도 로딩되게 설정했으므로 나중에 이 파일이 웹페이지에 잘 포함됐는지 확인해볼 수 있다.

.main { **font-size**: 24px; **border-bottom**: solid 1px black; }

.page1 { **font-size**: 14px; **background-color**: yellow; }

**클라이언트 코드 개발**

이제 JSX 파일을 작성할 차례다. src/main/jsx 폴더 아래에 아래와 같이 별 의미는 없지만 내용을 구별할 수 있게 자바스크립트 클래스를 작성한다. css 파일을 import하는 게 보이는데 개인적으로 CSS 파일은 JavaScript 위에서 import하는 걸 선호한다. MainPage.jsx:

import '../webapp/css/custom.css';

import React from 'react';

import ReactDOM from 'react-dom';

class MainPage extends React.Component {

render() {

**return** <div className="main">메인 페이지</div>;

}

}

ReactDOM.render(<MainPage/>, document.getElementById('root'));

Page1Page.jsx:

import '../webapp/css/custom.css';

import React from 'react';

import ReactDOM from 'react-dom';

class Page1Page extends React.Component {

render() {

**return** <div className="page1">Page1 페이지</div>;

}

}

ReactDOM.render(<Page1Page/>, document.getElementById('root'));

**클라이언트 스크립트 빌드**

webpack을 실행하면 바로 클라이언트 스크립트가 빌드되지만 JSX 파일 수정시에 자동으로 지속적으로 빌드되는 것이 필요하다. 이를 위해 webpack에는 watch 명령이 준비돼 있다. VSC 터미널에서 아래 명령을 실행해준다. 개발시에는 -d 옵션을 붙이고 운영 환경에서는 -p 옵션으로 바꾸도록 한다.

node\_modules\.bin\webpack --watch -d

webpack.config.js에 설정한 대로 정상 빌드되어 src/main/webapp/js/react아래에 main.bundle.js, page1.bundle.js가 생겼다면 성공이다. 개발할 때마다 명령어를 복잡하게 입력하기는 불편하므로 package.json의 scripts 항목에 아래와 같이 자주 사용하는 명령을 넣어둔다. 터미널에서 npm start, npm run watch로 간단하게 실행할 수 있다.

"scripts": {

"start": "set JAVA\_HOME=d:**\\**java**\\**jdk1.8&&mvnw spring-boot:run",

"watch": "node\_modules**\\**.bin**\\**webpack --watch -d",

**테스트 및 맺음말**

자 이제 웹 브라우저에서 웹페이지를 열어보자. 모두 이상 없이 구성됐고 mvnw spring-boot:run 명령으로 Spring Boot가 정상 구동되었다면 http://localhost:8080/main.html, http://localhost:8080/page1.html을 열었을 때 각각 맞는 페이지 내용이 스타일을 설정한 대로 표시돼야 한다. 결과는 아주 간단한 웹페이지지만 Eclipse를 대체하면서도 Java와 JavaScript 개발을 모두 편리하게 할 수 있는 개발 환경을 구성해봤다. 앞으로 React를 사용할 때 이 작은 글이 조금이나마 도움이 되길 바라본다.