

# ANGULAR 시작하기 with Spring

By Taehyo(dukeom@gmail.com)

# Javascript 인식의 변화

---

- 과거의 Javascript
  - 개발자들이 하찮게 여겨 배우지 않던 기술
  - Coder
    - Html, Javascript를 사용하여 웹 페이지를 만들던 개발자
- 현재의 Javascript
  - 개발자들이 어려워 배우기 힘든 기술
  - Frontend 개발자
    - 다양한 Frontend 기술들을 사용하는 개발자



# 왜 Javascript는 주목받게 되었을까?

---

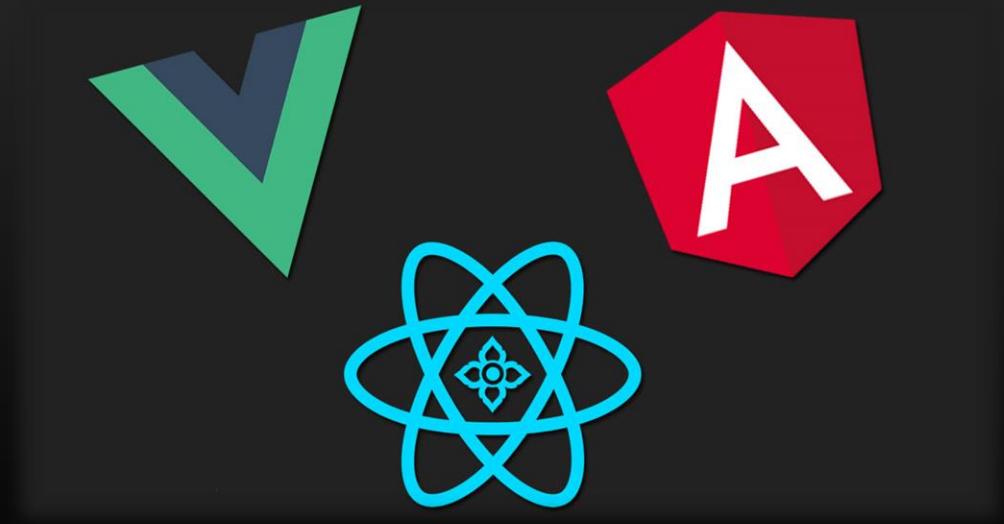
- 일등공신 Node.js
  - Javascript는 브라우저에서만 작동하는 언어
  - Node.js의 등장
    - 서버에서 고성능 I/O 처리를 위해 만들어짐
    - 브라우저 없이 단독으로 실행이 가능해짐
    - NPM(Node Package Manager)은 Node.js를 사용하는 패키지 관리도구
    - Package = HTML, CSS, Javascript등을 포함한 리소스 묶음
    - NPM을 사용하여 의존성 관리
    - NPM을 이용해 소스를 패키지 형태로 공유 가능



# 왜 Javascript는 주목받게 되었을까?

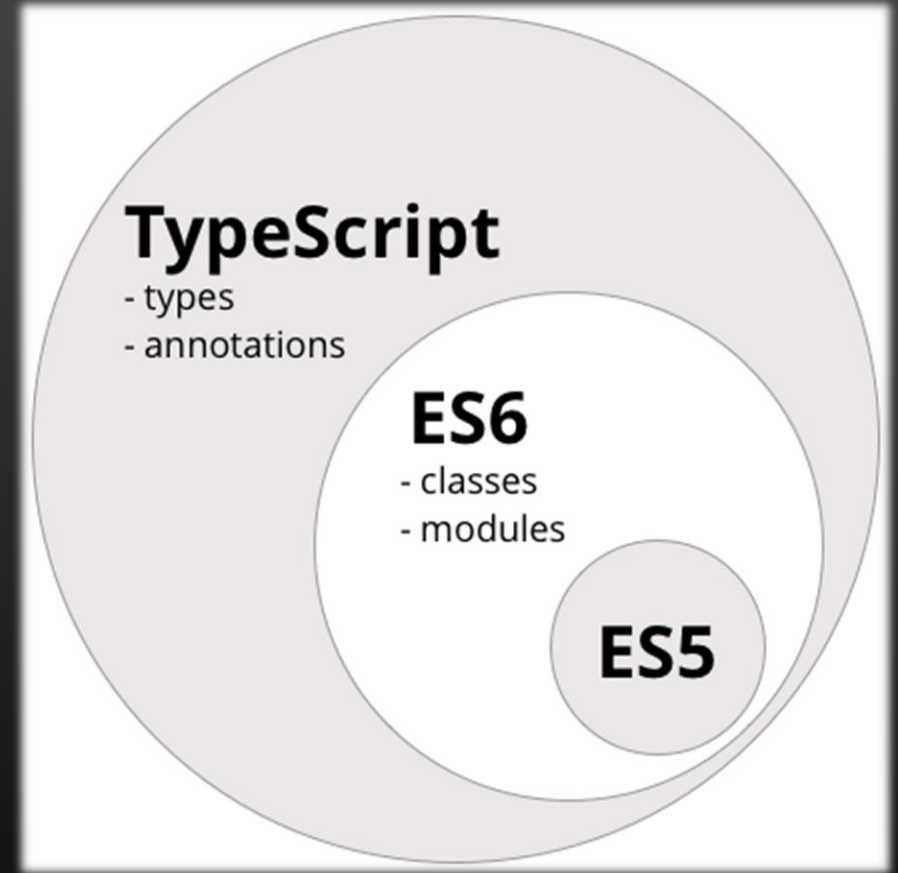
---

- jQuery의 등장
- 모던웹 개발을 위한 Javascript 프레임워크 및 라이브러리 등장
  - Angular, React.js, Vue.js 등
- 브라우저의 변화
  - 빠른 엔진을 탑재한 브라우저의 등장
    - 크롬 V8 엔진



# 왜 Javascript는 주목받게 되었을까?

- Typescript의 등장
  - Typescript is Superset
  - Typescript는 compile 하면 Javascript가 된다.
  - 대규모 웹 어플리케이션 및 하이브리드 앱 개발에 Javascript가 사용되기 시작하면서 안정적인 개발이 필요해짐
  - Microsoft에서 개발된 Javascript type 적용 언어
  - Compile을 통해 문법 체크가 가능해짐



# Frontend 개발 및 성능 향상을 위한 도구들

---

- Frontend 개발을 편리하게 해주는 도구
  - 패키지 관리자
    - NPM, Yarn, Yoman
  - Typescript 문법 체크
    - Tslint
- Frontend 성능 향상을 위한 도구
  - Webpack
    - 난독화, 압축, 번들링



# Angular 특징

---

- Javascript Framework
- Google에서 개발자들에 의해 만들어짐.
- Typescript를 사용
  - `var name: string;`
  - `class user {  
 var name: string;  
 constructor(name) {  
 this.name = name;  
 }  
}`

# AngularJS vs Angular

---

- AngularJS는 version 1.x
- Angular는 version 2.x~
- 현재 version 4.x
- AngularJS는 MVC 디자인 패턴
- Angular는 컴퍼넌트 기반 프레임워크
- Angular는 Typescript를 사용

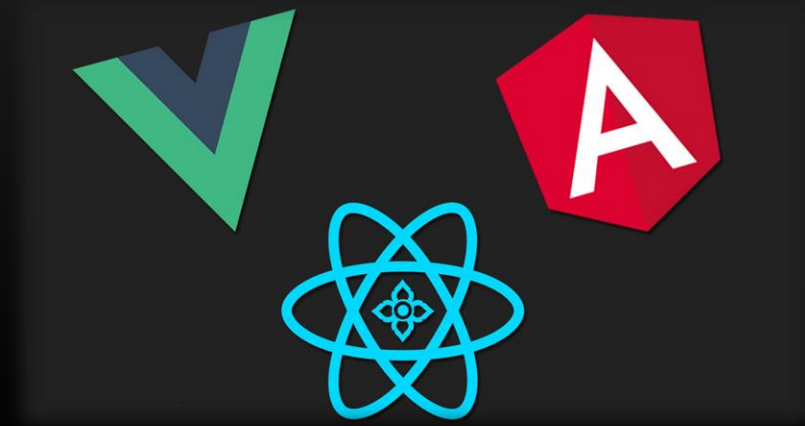




# Angular vs React.js vs Vue.js

---

- 현재 가장 주목받는 Javascript 개발 언어
- React
  - Facebook에서 만들어 사용 중
  - MVC에서 View영역만 다루기 때문에 다른 라이브러리와 사용 가능
  - 자체적인 Virtual DOM 객체를 생성해 개발자가 브라우저 DOM에 접근하는 것을 최소화 하는 방식으로 성능을 향상 시킴
  - Javascript 문법에 XML과 비슷한 문법을 도입한 JSX형식을 사용하는 것을 권장한다.
- Learning curve
  - Angular > React.js > Vue.js



# 개발 환경 설정



# 사전 설치 항목

---

- 아래 항목들의 최신 버전을 설치
  - Node.js
  - Visual Studio Code
  - Java SDK
  - Eclipse (Spring Tool)

# Node..js 설치

---

- Node.js
  - <https://nodejs.org/en/download/>
  - Cmd에서 다음을 확인
    - `$ node -v`  
v6.11.2
    - `$ npm -v`  
3.10.10
  - 브라우저 없이 실행
    - Node.js Command Prompt 실행
      - `$ node`
      - `$ console.log("Hello, Node.js")`

# NPM 기본 명령어

---

- npm help
- npm init
  - 실행된 위치에 package.json 파일을 생성한다.
  - 패키지 관리를 위한 초기화 작업을 수행한다.
- npm install
  - package.json에 있는 의존 패키지를 설치한다.
- npm install 패키지명
  - 패키지명을 package.json에 추가하고 해당 패키지를 설치한다.
- npm list
  - 현재 설치된 패키지 목록을 트리 형태로 보여준다.
- npm run
  - package.json의 script에 선언된 명령을 수행한다.
- example
  - npm install jquery
  - npm install jquery@1.12.4
  - npm install -g http-server
    - 전역으로 설치

# Angular-cli 설치

---

- Angular-cli는 angular용 scaffolding tool
- Angular-cli를 전역(global)으로 설치한다.
  - `$ npm install -g @angular/cli`
- 설치 확인
  - `$ ng --version`

# Angular-cli로 프로젝트 생성 및 실행

---

- 프로젝트 생성
  - `$ ng new my-app`
  - My-app 폴더에 프로젝트가 생성됨
  - Npm을 이용해 dependency package를 자동으로 download 함.
- 프로젝트 실행
  - `$ cd my-app`
  - `$ ng serve --open`

# Yarn 패키지 매니저 설치

---

- Yarn은 페이스북, Exponent, 구글과 Tilde의 엔지니어 그룹들이 함께 협력하여 npm의 핵심 이슈를 해결하기 위해 새로운 패키지 매니저
- 설치
  - `$ npm i -g yarn`
- 실행
  - `$ yarn`
  - node\_module을 다운받는다.



# Visual Studio Code 설치

---

- 다운로드 URL
  - <https://code.visualstudio.com/download>
- 아이콘 플러그인 설치
  - Ctrl+P
  - ext install vscode-icons

실습

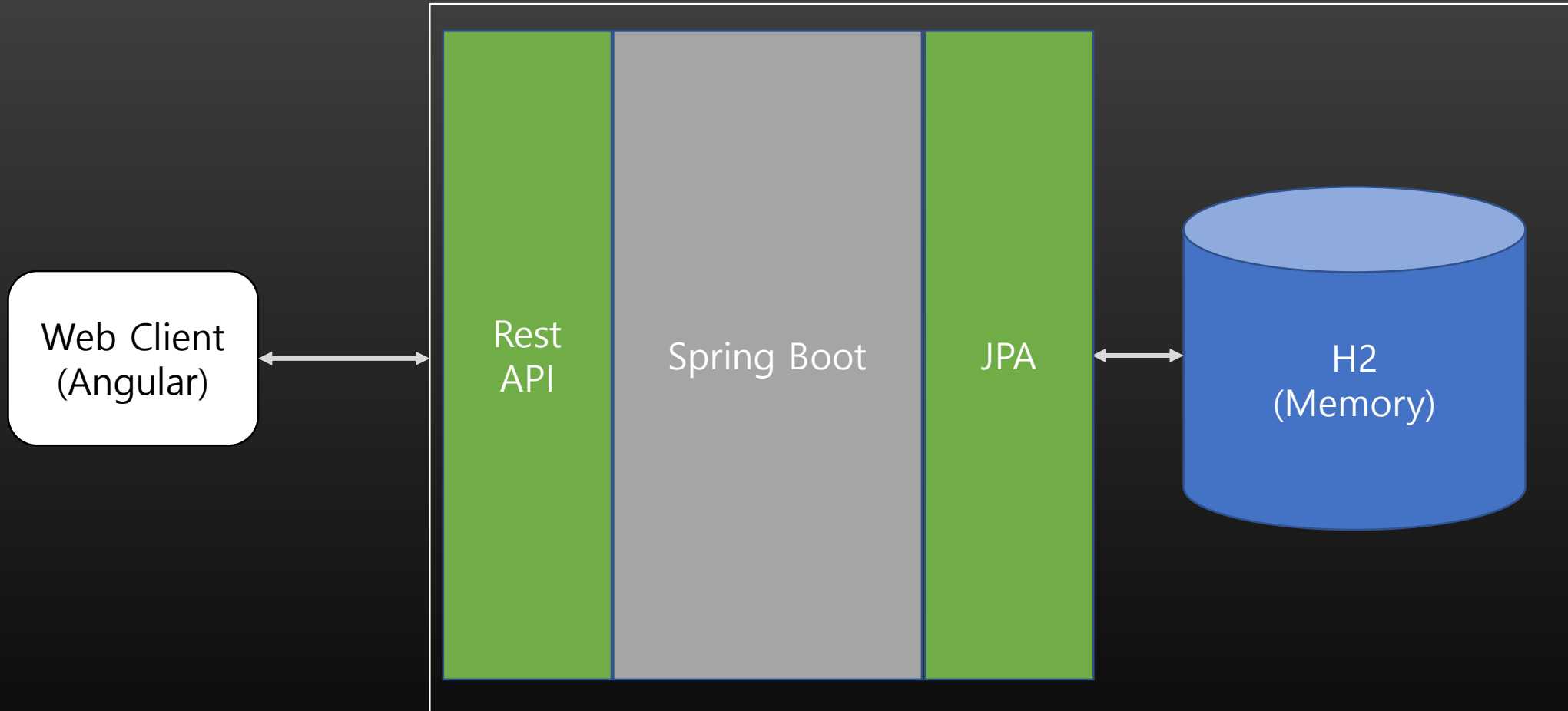
```
<body style="margin: 0; background-color: #f0f0f0;">  
<div id="main_content">  
  <h1 style="font-family: cursive; font-size: 1.5em;">Hello, World!  
  <p style="font-size: 1.2em;">This is a simple web page.</div>
```

# Java & Eclipse 설치

---

- Java 다운로드 (1.8)
  - <http://www.oracle.com/technetwork/java/javase/downloads/index.html>
- Eclipse 다운로드 (
  - <http://www.eclipse.org/downloads/>

# 개발 아키텍처 구성도



# 최종 실행 화면

Users

추가

userid	password	name	
admin	admin	Administrator	삭제
user	user	User	삭제

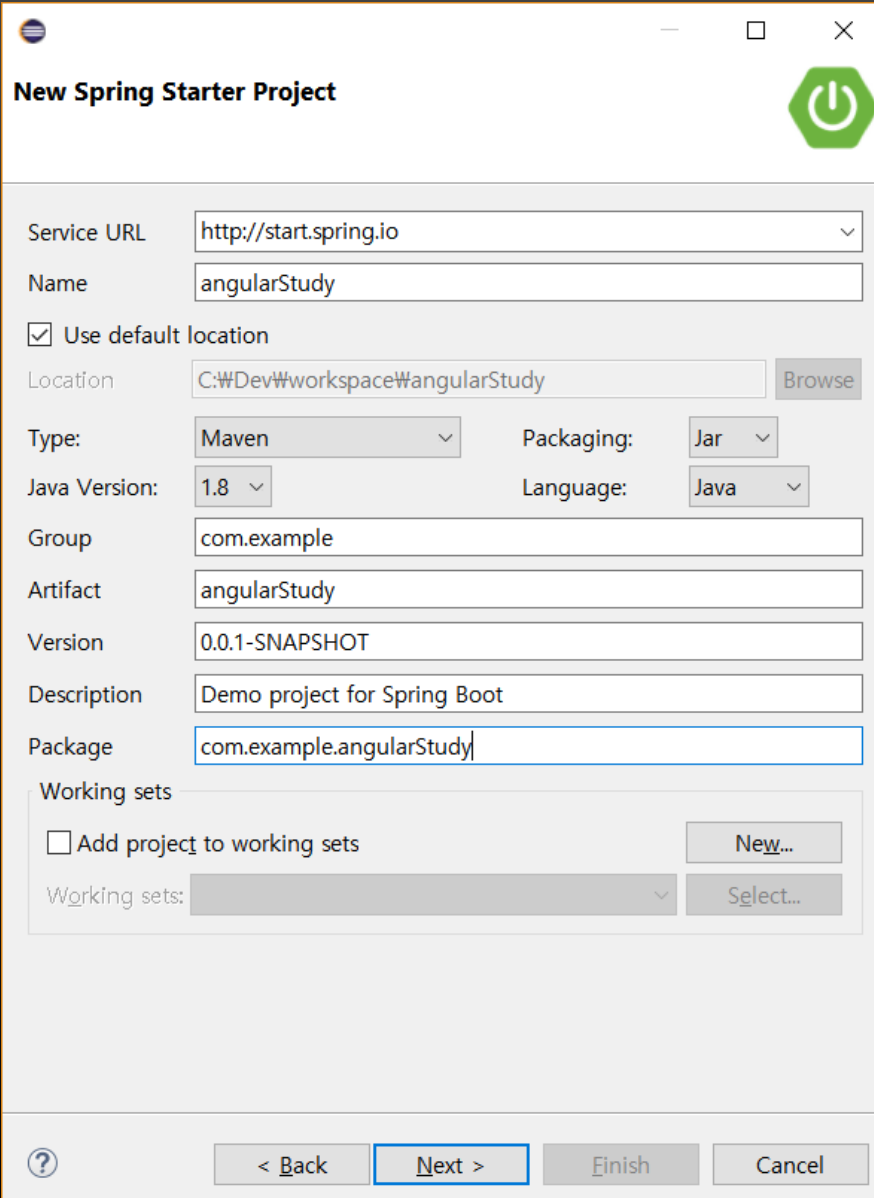
Angular is running in the development mode. Call enableProdMode() to enable the production mode. core.es5.js:2925

user app.component.ts:50

user app.component.ts:50

# Springboot Project 생성

- File -> new -> Project...
- Spring boot -> Spring starter project
- Project 설정
  - Type : Maven
  - Java Version : 1.8
  - Packaging : jar
  - Name : angularStudy
  - Group : com.example
  - Artifact : angularStudy
  - Package : com.example.angularStudy



**New Spring Starter Project**

Service URL:

Name:

☒ Use default location

Location:

Type:  Packaging:

Java Version:  Language:

Group:

Artifact:

Version:

Description:

Package:

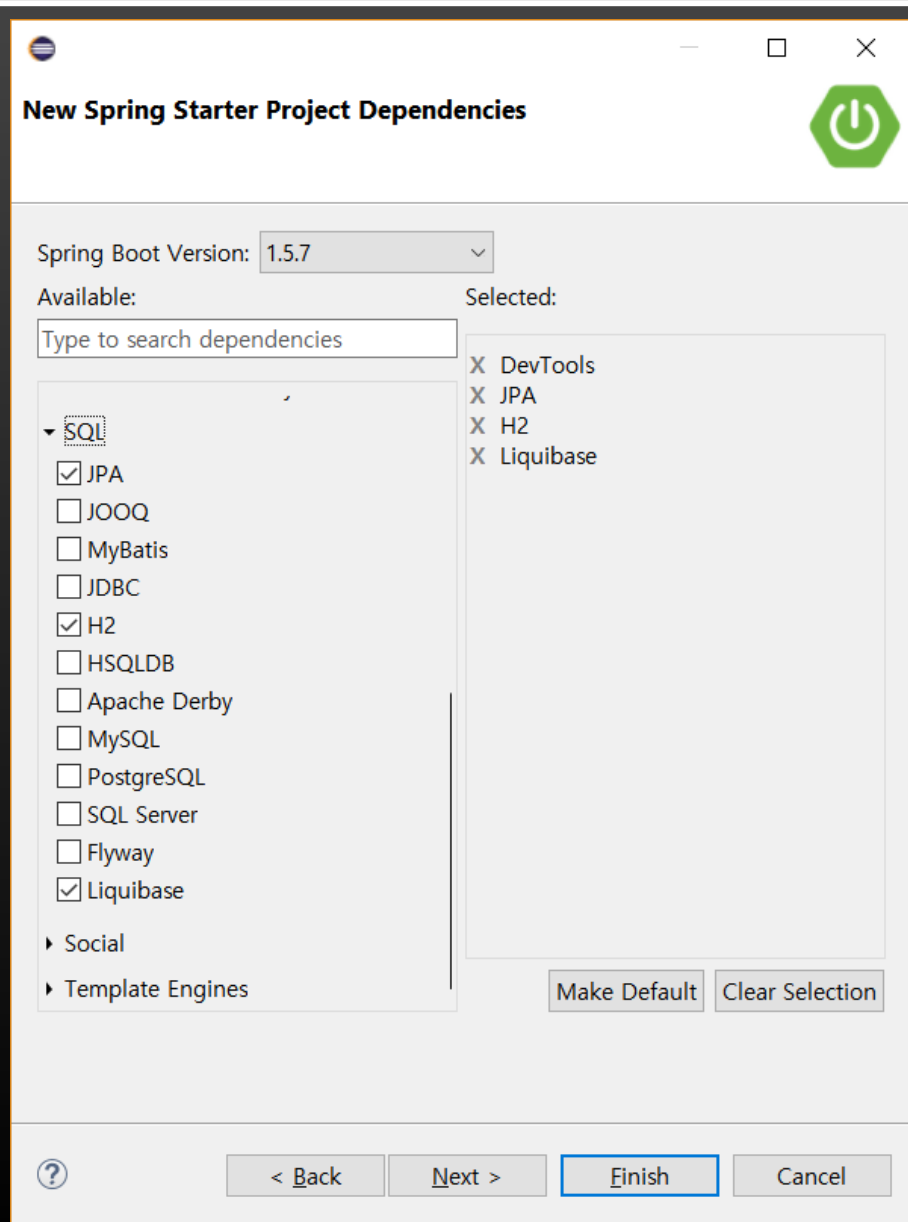
Working sets

☐ Add project to working sets

Working sets:

# Dependency 설정

- 아래 항목을 Check
  - Core
    - DevTools
  - SQL
    - JPA
    - MyBatis
    - H2
    - Liquibase
  - Web
    - Rest Repositories



The image shows a 'New Spring Starter Project Dependencies' dialog box. At the top, it says 'Spring Boot Version: 1.5.7'. Below this, there are two columns: 'Available:' and 'Selected:'. The 'Available:' column has a search bar 'Type to search dependencies' and a list of dependencies with checkboxes. The 'Selected:' column shows the dependencies that have been chosen. At the bottom right, there are buttons for 'Make Default' and 'Clear Selection'. At the very bottom, there are navigation buttons: '< Back', 'Next >', 'Finish' (highlighted with a blue border), and 'Cancel'.

Spring Boot Version: 1.5.7

Available: Selected:

Type to search dependencies

SQL

- ☒ JPA
- ☐ JOOQ
- ☐ MyBatis
- ☐ JDBC
- ☒ H2
- ☐ HSQLDB
- ☐ Apache Derby
- ☐ MySQL
- ☐ PostgreSQL
- ☐ SQL Server
- ☐ Flyway
- ☒ Liquibase

Social

Template Engines

X DevTools

X JPA

X H2

X Liquibase

Make Default Clear Selection

< Back Next > Finish Cancel

# 서버 설정

---

- src/main/resources/application.properties

```
server.contextPath=/  
server.port=8080
```

```
logging.level.org.springframework.web=DEBUG  
logging.level.org.hibernate=ERROR
```



# Datasource 설정

---

- src/main/resources/application.properties

```
# ***** H2 In Memory Database Connection Info *****
spring.datasource.url=jdbc:h2:mem:example;DB_CLOSE_DELAY=-1;DB_CLOSE_ON_EXIT=FALSE
spring.datasource.username=sa
spring.datasource.password=
spring.datasource.driver-class-name=org.h2.Driver
spring.datasource.platform=h2
spring.datasource.initialize=true
spring.datasource.continue-on-error=false
spring.jpa.hibernate.ddl-auto=update
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true
spring.h2.console.enabled=true
```

# Schema & Data 생성 - Liquibase

---

- 폴더 및 파일 생성
  - src/main/resource/db/user.csv

```
user_id;password;name  
admin;admin;Administrator  
user;user;User
```

# Schema & Data 생성 - Liquibase

---

- 폴더 및 파일 생성
  - src/main/resource/db/changelog/db.changelog-master.yaml

```
databaseChangeLog:
  - changeSet:
      id: 1
      author: sa
      changes:
        - createTable:
            tableName: user
            columns:
              - column:
                  name: user_id
                  type: varchar(20)
                  constraints:
                    primaryKey: true
                    nullable: false
              - column:
                  name: password
                  type: varchar(20)
                  constraints:
                    nullable: false
```

```
      - column:
          name: name
          type: varchar(50)
          constraints:
            nullable: false
      - loadData:
          encoding: UTF-8
          file: db/user.csv
          separator: ;
          tableName: user
```

# Server Start

---

- Boot Dashboard
  - angularStudy click -> start
- <http://localhost:8080>
  - Whitelabel Error Page

# Domain Class

---

- Com/example/angularStudy/domain/User.java

```
@Entity
public class User {
    @Id
    private String userId;
    private String password;
    private String name;
}
```

# Repository Interface

---

- Com/example/angularStudy/repository/UserRepository.java

```
public interface UserRepository extends JpaRepository<User, String>{  
  
}
```

# Rest Controller Class

---

- Com/example/angularStudy/rest/UserResource.java

```
@CrossOrigin
@RestController
@RequestMapping("/api")
public class UserResource {
    @Autowired
    private UserRepository userRepository;

    @GetMapping("/users")
    public ResponseEntity<List<User>> getAllUsers(){
        List<User> users = userRepository.findAll();
        return new ResponseEntity<>(users, HttpStatus.OK);
    }
}
```

# Rest Test

---

- <http://localhost:8080/api/users>

```
1 // 20170915111037
2 // http://localhost:8080/api/users
3
4 [
5   {
6     "userId": "admin",
7     "password": "admin",
8     "name": "Administrator"
9   },
10  {
11    "userId": "user",
12    "password": "user",
13    "name": "User"
14  }
15 ]
```



# Rest Controller Class

---

- Com/example/angularStudy/rest/UserResource.java

```
@CrossOrigin
@RestController
@RequestMapping("/api")
public class UserResource {
    ...
    @PostMapping("/users")
    public ResponseEntity<Void> createUser(@RequestBody User user){
        userRepository.save(user);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @DeleteMapping("/users/{userId}")
    public ResponseEntity<Void> deleteUserere(@PathVariable String userId ){
        userRepository.delete(userId);
        return new ResponseEntity<>(HttpStatus.OK);
    }
}
```

# Angular 프로젝트 생성

---

- 프로젝트 상위 디렉토리에서 실행
- `$ ng new angularStudy --sd src/main/webapp -d`
  - `angularStudy` : 프로젝트명
  - `--sd` : `--source-dir`의 약자(소스 위치 지정)
  - `-d` : `-dry-run`의 약자

# Angular 프로젝트 생성

```
PS C:\Dev\workspace\angularStudy> ng new angularStudy --sd src/main/webapp
installing ng
  create .editorconfig
  create README.md
  create src\main\webapp\app\app.component.css
  create src\main\webapp\app\app.component.html
  create src\main\webapp\app\app.component.spec.ts
  create src\main\webapp\app\app.component.ts
  create src\main\webapp\app\app.module.ts
  create src\main\webapp\assets\.gitkeep
  create src\main\webapp\environments\environment.prod.ts
  create src\main\webapp\environments\environment.ts
  create src\main\webapp\favicon.ico
  create src\main\webapp\index.html
  create src\main\webapp\main.ts
  create src\main\webapp\polyfills.ts
  create src\main\webapp\styles.css
  create src\main\webapp\test.ts
  create src\main\webapp\tstconfig.app.json
  create src\main\webapp\tstconfig.spec.json
  create src\main\webapp\typings.d.ts
  create .angular-cli.json
  create e2e\app.e2e-spec.ts
  create e2e\app.po.ts
  create e2e\tstconfig.e2e.json
  create .gitignore
  create karma.conf.js
  create package.json
  create protractor.conf.js
  create tsconfig.json
  create tslint.json
You can `ng set --global packageManager=yarn`.
Installing packages for tooling via npm.
Installed packages for tooling via npm.
Successfully initialized git.
Project 'angularStudy' successfully created.
```

# Angular 프로젝트 실행

---

- \$ yarn start
- 브라우저에서 다음 입력localhost:4200

Welcome to app!



Here are some links to help you start:

- [Tour of Heroes](#)
- [CLI Documentation](#)
- [Angular blog](#)



# bootstrap package 추가하기

---

- \$ yarn add bootstrap
- angular-cli.json 수정
  - Apps -> styles 항목을 아래와 같이 수정한다.

```
"styles": [  
  "styles.css", "../node_modules/bootstrap/dist/css/bootstrap.min.css"  
]
```

# Html 수정

---

- webapp/app/app.component.html
  - bootstrap table 소스 참조
    - [https://www.w3schools.com/bootstrap/bootstrap\\_tables.asp](https://www.w3schools.com/bootstrap/bootstrap_tables.asp)

## Bordered Table

The `.table-bordered` class adds borders to a table:

Firstname	Lastname	Email
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

# 변경된 html 화면

- 아래 화면과 같이 html을 수정한다.  
webapp/app/app.component.html

## 사용자 관리

사용자를 조회, 수정, 생성, 삭제할 수 있습니다.

사용자ID	비밀번호	이름
John	Doe	john@example.com
Mary	Moe	mary@example.com
July	Dooley	july@example.com

# User model class 생성

---

- app/model/user.model.ts

```
export class User {  
  userId: string;  
  passwd: string;  
  name: string;  
  
  constructor() {}  
}
```



# UserService

---

- webapp/app/service/user.service.ts

```
import { Injectable } from '@angular/core';
import { Http, Response } from '@angular/http';

@Injectable()
export class UserService {
  api_users = "http://localhost:8080/api/users";

  constructor(
    public http: Http
  ) {}

  getUsers() {
    return this.http.get(this.api_users);
  }
}
```

- Service는 Angular-cli를 사용하여 자동 생성할 수 있다.
- \$ ng g s service/user

# module에 service 추가

- webapp/app/app.module.ts

```
import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';
import { HttpClientModule } from '@angular/http'; ←
import { UserService } from '../service/user.service'; ←
import { AppComponent } from './app.component';

@NgModule({
  declarations: [
    AppComponent
  ],
  imports: [
    BrowserModule,
    HttpClientModule ←
  ],
  providers: [UserService], ←
  bootstrap: [AppComponent]
})
export class AppModule { }
```

- Rest Api 호출을 위해 HttpClientModule을 import한다.
- UserService를 사용하기 위해 import한다.
- Service의 경우 providers에 선언해야 사용할 수 있다.

# app.component.ts

---

- webapp/app/app.component.ts

```
import { User } from './model/user.model';
import { UserService } from './service/user.service';

export class AppComponent implements OnInit {
  title = 'app';
  users: User[];
  constructor(private userService: UserService) {}
  ngOnInit() { this.loadAllUsers(); }
  loadAllUsers() {
    this.userService.getUsers()
      .subscribe (res=> {
        this.users = res.json();
      })
  }
}
```

- AppComponent에 User class와 UserService를 import 한다.
- Constructor에서 userService를 생성하도록 한다.
- Service 호출 결과는 json 형식으로 받는다.

# app.component.html

```
<tbody *ngIf="users">
  <tr *ngFor="let user of users">
    <td>{{user.userId}}</td>
    <td>{{user.password}}</td>
    <td>{{user.name}}</td>
  </tr>
</tbody>
```

- Html에서 결과 값을 출력한다.
- Looping은 \*ngFor를 사용한다.
- \*ngIf를 이용하여 users의 값이 있을 경우에만 결과를 보여준다.

## 결과 화면

### 사용자 관리

사용자를 조회, 수정, 생성, 삭제할 수 있습니다.

사용자ID	비밀번호	이름
admin	admin	Administrator
user	user	User

# 사용자 입력 화면 구성

- 사용자 입력화면은 추가 버튼 클릭시 나타나게 구성한다.

## 사용자 관리

추가

사용자를 조회, 수정, 생성, 삭제할 수 있습니다.

사용자ID

패스워드

이름

취소

저장

# 사용자 입력

---

- 사용자 객체 생성 (app.component.ts)

```
user: User = new User();
```

- 화면 처리 (app.component.html)

- 입력창을 보이기 위한 입력 버튼

```
<button type="button" class="btn btn-primary" (click)="addForm()">추가</button>
```

- Input text 필드

```
<input type="text" class="form-control" name="userid" style="width:100px" [(ngModel)]="user.userId">  
<input type="text" class="form-control" name="password" style="width:100px" [(ngModel)]="user.password">  
<input type="text" class="form-control" name="name" style="width:100px" [(ngModel)]="user.name">
```

- 저장/취소 버튼

```
<p><button type="button" class="btn btn-primary" (click)="cancel()">취소</button></p>  
<p><button type="button" class="btn btn-primary" (click)="save()">저장</button></p>
```

# 사용자 입력

---

- ngModel 이 정의된 FormModule import

```
import { FormsModule } from '@angular/forms';
```

- Save button click시 console에 log 출력해 보기

```
console.log(this.user);
```

# 사용자 입력

---

- 입력 화면 토글 기능

- app.component.ts

```
showForm: boolean = false;

addForm() {
  this.showForm = true;
}

cancel() {
  this.showForm = false;
}
```

- showForm 변수를 선언하여 값을 바꾸어주는 방식으로 toggle 기능을 구현한다.
- 입력 버튼 클릭시 addForm()을 호출
- cancel() 이벤트로 폼을 숨긴다...

- app.component.html

```
<div *ngIf="showForm">
  /* 입력폼 위치 */
</div>
```



# 사용자 저장

---

- 저장 버튼 이벤트 정의(app.component.ts)

```
save() {  
    console.log(this.user);  
    this.userService.saveUsers(this.user).subscribe(  
        (res) => this.loadAllUsers(),  
        (res) => console.log(res)  
    );  
    this.isNew = false;  
}
```

- 서비스에 저장 RestApi 정의(user.service.ts)

```
saveUsers(user: User) {  
    return this.http.post(this.api_users, user);  
}
```

# Spring rest controller 저장 api

---

- UserResource.java

```
@PostMapping("/users")  
public ResponseEntity<Object> saveUsers(@RequestBody User user){  
    userRepository.save(user);  
    return new ResponseEntity<>(HttpStatus.OK);  
}
```

# 사용자 삭제 화면 구성

- 삭제 버튼 클릭시 confirm 창이 뜨고 확인 버튼 클릭시 삭제된다.

The screenshot shows a web browser window with multiple tabs. The active tab is titled 'localhost:4200 내용:' and displays a confirmation dialog box asking '삭제하시겠습니까?' (Are you sure you want to delete?). The dialog has two buttons: '확인' (Confirm) and '취소' (Cancel).

In the background, the '사용자 관리' (User Management) page is visible. It features a '추가' (Add) button and a description: '사용자를 조회, 수정, 생성, 삭제할 수 있습니다.' (You can search, modify, create, and delete users.). Below this is a table listing users.

사용자ID	비밀번호	이름	
admin	admin	Administrator	삭제
user	user	User	삭제
1	1	1	삭제

# 사용자 삭제

---

- app.component.ts

```
delete(userId: string) {  
    console.log(userId);  
    if (confirm ('삭제하시겠습니까?')) {  
        this.userService.deleteUsers(userId).subscribe(  
            (res) => this.loadAllUsers(),  
            (res) => console.log(res)  
        );  
    }  
}
```

- user.service.ts

```
deleteUsers(userId: string) {  
    return this.http.delete(`${this.api_users}/${userId}`);  
}
```

# 사용자 삭제

---

- 삭제 버튼 (app.component.html)

```
<button type="button" class="btn btn-danger" (click)="delete(user.userId)">삭제 </button>
```

# Spring rest controller 삭제 api

---

- UserResource.java

```
@DeleteMapping("/users/{userId}")  
public ResponseEntity<Void> deleteUsers(@PathVariable String userId){  
    userRepository.delete(userId);  
    return new ResponseEntity<>(HttpStatus.OK);  
}
```

# 최종 화면

## 사용자 관리

추가

사용자를 조회, 수정, 생성, 삭제할 수 있습니다.

사용자ID	비밀번호	이름	
admin	admin	Administrator	삭제
user	user	User	삭제
1	1	1	삭제

THANK YOU!

