**1. Spring Web Starter**

* What is Spring boot?
  + 기본적이고 적절한 옵션 및 써드파티 설정을 제공해주며 쉽게 프로그램을 시작할 수 있도록 하는 기능을 제공한다.
  + 모든 스프링 개발에 있어서 빠르고 폭넓은 사용성을 제공한다. 물론 얼마든지 기본적으로 설정된 값들을 바꾸기도 쉽게 되어있다.
  + Non-functional 한 기능들도 제공하기 때문에 비즈니스 로직에 집중할 수 있도록 도와준다.
  + 스프링 부트는 자바 8부터 사용할 수 있다.
* 어떻게 스프링 부트는 수많은 설정과 의존성을 지원하는가? 아래의 내용에 포함되어있음.(principle)
  + EnableAutoConfiguration
  + Spring boot web starter
* SpringBootApplication 어노테이션을 기준으로 Component 스캔이 일어난다.

**2. Spring boot principle**

**의존성 관리의 이해 및 응용**

* SpringBoot parent가 기본적으로 들어오는데 이것은 스프링부트에서 제공하는 다양한 의존성을 묶어놓은 것을 상속받고 있기 때문에 모든 의존성이 한번에 버전과 함께 들어오는 것.
* 왜 좋은가 ? 기존에 우리가 직접 관리하는 경우 의존성 충돌이 굉장히 자주 일어났는데 Spring boot 가 직접관리해줌으로써 이러한 충돌이 굉장히 줄어들었음.
* 물론 이 경우에도 자신의 의존성을 만들고 싶다면 Dependency Mangement 를 참고해서 변경할 수 있다. 이 방법보단 스프링이 기본 제공하는 의존성의 부모형태로 사용하는 것이 좋다.
* 버전을 명시적으로 의존성을 추가하는 것이 바람직하며 스프링이 버전을 제공하는 경우는 작성하지 않아도 된다.
* Springboot 가 제공하는 버전을 변경하고 싶은 경우 의존성에 작성하면 Overriding을 통해 가능하다.
* <properties>
* <spring.version>5.0.6.RELEASE</spring.version>

</propeties>

**AutoConfiguration**

* EnableAutoConfiguration (SpringbootApplication이 상속받고 있는 Annotation)
  + Component Scan 을 통해서 1차적으로 빈을 찾아서 등록이 이루어지고 EnableAutoConfiguration을 통해 2차적으로 빈을 등록 및 설정한다.
  + (META-INF) spring.factories 파일을 읽어들여 관련된 설정을 추가로 제공한다. 즉 Springboot에서 제공하는 다양한 설정(의존성 말고)에 대한 관리를 담당하는 곳으로 다양한 파일을 스스로 읽어와 설정을 하도록 되어있다.
  + 참고로 Component Scan이 먼저이고 AutoConfiguration이 이후에 일어난다. 즉 같은 빈을 등록하려고 하면 후자로 덮어쓴다.
* ConfigurationProperties
  + 서로 다른 모듈에서 빈을 참조하여 가져올 수 있다. 가져오는 쪽에서 [Application.properties](http://application.properties/) 를 통해서 가져올 수 있도록 지원해주는데 일반적으로 프로젝트 규모가 크거나 멀티 모듈 프로젝트에서 주로 사용될 것 으로 보인다.
  + 추후에 참고하자. 일반적인 Bean 을 가져와서 사용하기에는 귀찮고 설정할 것이 많기 때문에 설정파일에 주로 사용될 것으로 보인다.
  + 사용 ! 기존 프로젝트의 설정파일과 유사한데 특정 값만 다른 경우 기존것을 가져와서 사용하고 값만 바꿔주는 형태

**내장 웹 서버**

* 개념
  + Spring boot 는 서버가 아니다. Tomcat을 쉽게 사용할 수 있도록 내장해서 제공하는 것 뿐
  + 이 또한 자동설정에서 제공해서 주는 것(AutoConfiguration → spring.factories)
  + 자동설정에서 내장 톰캣이나 제티 등을 생성하고 내장 서블릿을 만들어서 매핑하는구나 라고 알고있으면 된다. 물론 직접 만들 수도 있음.
  + 자신이 내장 서블릿과 톰캣을 생성하여 띄울 수 있는데 이러한 설정을 자동설정파일을 통해서 지원하고 있구나 (서버생성과 서블릿 생성을 나누어서)
* 응용
  + 기본제공되고 있는 Tomcat 을 Jetty혹은 Undertow 로 변경할 수 있음 의존성 제거해주고 추가하면
  + Https & SSL
    - Http2 설정 (SSL 설정이 되어있어야 가능함)
    - SSL 설정

**독립적으로 실행가능한 JAR 파일**

* 프로그램을 Package하면 Jar 파일 하나가 생긴다. 그 안에 모든 클래스 관련 정보가 담기기 때문에 독립적으로 실행이 가능한데 그렇다면 의존성은 어떻게 주입받는가?
* Spring boot가 제공해주는 내부 프로세스에 의해서 Jar file을 읽어주는 클래스가 포함되어 하나의 Jar파일로 합쳐지기 때문에 모든 기능을 하나의 Jar 파일을 통해 프로그램을 사용할 수 있는 것이다.
* 참고로 자바에서는 기본 내장 Jar 를 로딩하는 표준적인 방법이 존재하지 않는다. Springboot 가 이를 가능하도록 설정해둔 것이기 때문에 Springboot에서 만든 하나의 Jar파일은 독립적으로 배포가 가능하다.

**3. Spring boot practical use**

**SpringApplication**

* 배너를 자신이 원하는 형태로 다양하게 바꿀 수도 있다.
* ApplicationEvent
  + ApplicationListener 를 통해 이벤트를 자동으로 발생시킨다.
  + 다만 ApplicationContext가 로드되기 이전에 발생한 이벤트는 빈으로 등록되더라도 발생시키지 못한다.
  + 이 경우는 직접 Main에서 등록해주는 과정을 통해 해결 가능.
* 프로그램 실행 이후 뭔가를 하고 싶을 때
  + ApplicationRunner → 추상화된 API 를 사용할 수 있다는 장점이 있음.
  + 여러가지 Runner 들이 있다면 Order(1) 과 같은 옵션으로 실행순서를 정해줄 수 있다. 낮을수록 빨리 실행된다.

**외부설정**

* 설정과 관련된 파일은 일반적으로 .properties 라는 파일에 작성하는 것이 일반적인데 이러한 파일에는 우선순위가 존재한다. 뿐만 아니라 type-safe하게 관리하기 위해서 빈으로 등록하여 자바코드로도 조작이 가능한데 이를 알아보자.
* 우선순위
  + Build 를 실행하면 Java Source 전체를 빌드한 이후 테스트 코드를 빌드한다 즉 테스트코드에 있는 Properties가 우선순위를 갖게된다.
  + Java & Test 에서 각기 다른 Application.properties를 통해 사용하는 경우 문제가 되기 쉽다. 왜? 어플리케이션 빌드 과정에서 Test에 존재하는 properties가 오버라이딩 하는데 만약 자바에 있는 설정정보가 테스트엔 없다면? 빌드 실패.
  + 테스트에 필요한 properties를 재정의 하고 싶은경우? 혹은 부분만 오버라이딩 하고싶은 경우
    - @TestPropertySource(locations = "classpath:/test.properties") 를 통해 사용가능.
  + 추가적으로 Home 아래에 [application.properties](http://application.properties/) 를 추가로 생성하면 우선순위가 높다(Java source에 국한해서)

**프로파일**

* Profile("somthing1") , Profile("somthing2)
  + 특정한 상황에 맞게 다른 빈을 사용하고 싶거나 Profile로 무엇인가를 나눠서 관리하고 싶을 때 사용
  + [application.properties](http://application.properties/) 에서 [spring.profiles.active](http://spring.profiles.active/) = somthing1 으로 설정하는 경우 somthing1 과 관련된 형태로 빈이 생성 및 주입되고
  + 이 또한 위의 외부설정파일 처럼 우선순위가 적용된다. 따라서 배포 및 커멘드라인에서 설정해주는 방법도 가능하다. (java -jar target/...jar —spring.profiles.active =something2 와 같이 설정하는 경우 somthing2 의 프로파일로 프로그램이 동작한다.

**로깅**

* 로깅 퍼사드(Commons Logging, ***SLF4J***) VS 로거
  + 로깅 퍼사드는 다양한 로거를 추상화해놓은 형태이다. 따라서 이러한 로깅 퍼사드를 사용하는 경우 쉽게 다른 로거로 변환이 가능하다.
  + 즉 로깅 퍼사드가 구현체는 아니라는 말이기 때문에 우리가 실제로 사용하는 구현체는 로깅 퍼사드가 아니다.
  + 그럼 Commons Logging 과 SLF4J 를 쓸 때 기본적으로 사용하는 로거는 무엇인가? 둘다 LogBack을 사용하도록 되어있다. 우리가 둘중 무엇을 쓰든 상관없이 구현체는 LogBack으로 기본 설정 되어있음.

**테스트**

* 통합 테스트
  + SpringBootTest
  + 사용 방법
    - Mockmvc
    - WebTestClient with webflux ( 비동기 )
  + 통합 테스트가 실행되면 모든 빈을 등록한 이후 자신이 Mock으로 교체한 부분만 교체하여 테스트를 돌린다
  + 즉 비용이 크다. 꼭 필요한 경우가 아니면 통합테스트 보단 아래의 슬라이싱 테스트를 이용하자.
* 슬라이싱 테스트
  + 레이어별로 잘라서 적용이 된다. 필요한 부분만 테스트할 수 있도록 지원하는 것임.
  + JsonTest → JacksonTester ..
  + ***WebMvcTest(SampleController.class) → @MockBean SampleService sampleService → 웹과 관련된 빈만 설정되기 때문에 (Controller, Component...) Service와 같은 것은 Mock을 통해 채워줘야한다. → 꼭 MockMvc 로 테스트해야한다.***
  + @DataJpaTest 와 같은 경우 Repository만 등록되는 것처럼 필요한 부분만 테스트를 하기 위해서 슬라이싱 테스트가 주로 사용된다.
* Test Util by Junit
  + OutputCapture
    - Junit에서 제공하는 기능으로 @Rule public OutputCapture capture = new OutputCapture()을 통해서 사용할 수 있다.
    - 해당 부분이 실행될 때 찍히는 로그와 Sout 내용을 확인할 수 있다. (capture.contains()...)

**Spring Web MVC**

**소개**

* SpringBoot 는 기본적으로 Web MVC와 관련된 AutoConfiguration을 제공합니다. 이렇게 제공하고 있는 자동 설정이 존재하기 때문에 특별한 설정없이 바로 WebMVC를 사용할 수 있는데요. WebMvc가 제공하는 대표적인 혹은 스프링 Boot가 설정하여 작동시킨 MVC 의 기능과 내부동작을 일부 알아보도록 합시다.
* Spring boot의 자동설정에 의해서 우리는 스프링 MVC 관련 설정을 하지 않아도 이미 사용할 수 있는 형태이다. 다만 MVC와 관련해서 추가적인 작업이나 설정이 필요하다면 아래와 같은 설정을 통해서 이를 가능하게 할 수 있다. 참고로 @EnableWeb 관련 어노테이션을 붙이면 기본 자동설정이 모두 사라지기 때문에 지양해야한다.

@Configuration  
public class WebConfiguration implements WebMvcConfigurer

**MessageConverter**

* HTTP 본문의 요청 및 응답 본문을 변경할 때 사용한다.
  + RequestBody
  + ResponseBody
* 내부적으로 메세지를 Converting하는 다양한 구현체들이 존재하는데 요청 및 응답의 방식에 따라 알맞은 구현체가 선택된다. 예를 들어 Json 타입의 요청이 들어오는 경우 JsonConverter 가 선택되는 형태
* 그럼 Response에서는 어떤 구현체를 사용할까? 기본적으로는 Json 형태(객체타입)를 리턴하고 String, Integer과 같은 경우는 String으로 리턴하도록 되어있다. 참고로 RestController의 경우 ResponseBody를 사용하지 않아도 Json형태로 기본설정되어있다.
* 참고로 Json타입의 응답 및 요청의 경우 카멜케이스면 카멜케이스, 스네이크 케이스면 스네이크 케이스를 명시적으로 사용해줘야 서버에러가 발생하지 않는다.

**ViewResolve**

* 구현 : ContentNegotiatingViewResolver
* 마찬가지로 SpringWebMvc 에서는 View에 해당하는 부분을 매핑해주는 다양한 구현체들이 존재한다. 어떻게 매핑하여 View를 제공하는가?
  + 기본적으로는 Request Header 를 통해서 현재 반환할 수 있는 모든 타입 중 Header와 일치하는 타입을 리턴
  + 가끔 Request Header 가 존재하지 않는 경우도 많음. 그런 경우 추가적으로 format이라는 것을 내부적으로 확인해서 리턴.
* 예를 들어 XML과 같은 경우 ViewResolver로 등록되어있지 않다. 그럼 어떻게 하면 되는가? → ViewResolver로서 XML을 처리할 수 있도록 의존성을 추가해주면 XML 헤더의 요청이 들어오는 경우 적절한 ViewResolver(우리가 추가해준) 구현체가 일을 해서 정상 동작하게 된다.

**정적 리소스 자원**

* 기본 리소스의 위치(자동 설정되어있는 위치)
  + classpath:/static
  + classpath:/public
  + classpath:/resources/
  + classpath:/META-INF/recourses
  + 참고 : Template Engine을 사용하는 경우 /resource/templates/ 밑에 사용하면 됨.
* 추가적인 리소스의 위치를 설정하고 싶다?
  + WebConfiguration 클래스를 통해 WebMvcConfigurer 를 구현하고 addResourceHandlers 를 통해 추가 할 수 있다.

**웹 Jar & 파비콘 & Index page**

* Web Jar
  + Web 을 위한 Jar 파일 예를 들어 Jquery , React , Vue 등등은 모두 Maven, Gradle에 의존성을 추가하여 사용할 수 있음 → Jar File이 추가됨.
* Index page
  + Index.html & Index.template 을 정적 리소스 기본위치에 두면 기본 호출시에 자동으로 매핑해서 제공함. 만약 둘다 없는 경우 에러처리\
* 파비콘
  + Title 부분에 있는 그림을 바꾸는 것으로 정적 리소스 기본위치에 favicon.ico 라는 파일의 형태를 주면 작동한다.
  + 가끔 브라우저의 캐싱기능 때문에 바뀌지 않는 경우가 있는데 이 경우 favicon.ico 를 URL로 직접 요청하고 이후에 다시 브라우저를 키면(껐다가) 변경된다.

**Thymeleaf**

* Spring Web Mvc 는 기본적으로 템플릿 엔진을 일부 지원함.
  + Mustache
  + Groovy
  + Thymeleaf
* 이러한 템플릿 엔진을 사용하면 ViewResolver에게 Model을 통해 값을 전달할 수 있고 이러한 값을 전달 받아 View에서 사용할 수 있도록 지원함.
* 참고로 JSP는 서블릿을 통해 뷰를 렌더링 하기 때문에 슬라이싱 테스트에 불리할 뿐 아니라 Springboot 자동설정에도 제외되어있음. 따라서 요즘 잘 쓰지 않는 추세
* 참고
  + SSL & CSL → Template Engine & SPA

**HtmlUnit**

* HTML을 단위테스트 하기 위한 툴

**ExceptionHandler**

* 기본적으로 BasicErrorHandler 가 동작하여 에러페이지를 보여준다.
  + 에러코드에 따라 다른 웹페이지를 보여주고 싶은 경우
    - /resources/template/
    - /resources/static/
    - 아래에 404, 4xx 등의 Html을 통해서 지원 가능
  + Customizing한 에러를 보여주고 싶은 경우
    - 예외를 던지고 해당 컨트롤러에서 @ExceptionHandler(Class.class) 를통해 처리할 수 있음.
  + 부가적으로 에러에 대한 작업을 하고싶은 경우 ErrorViewResolver를 구현하여 오버라이딩을 통해 사용가능.

**HATEOAS**

* Restful api를 구현할 때 관련된 링크정보까지 제공하는 기능을 의미함
* Hypermedia As The Engine Of Application State
* 리소스와 연관되어 있는 링크 정보를 클라이언트에게 제공한다.
* 의존성을 추가하여 사용할 수 있으며 연관 정보와, Hypertext Reference를 제공한다.
* 참고 ObjectMapper(객체 → Json , Json → 객체) 는 Springboot 에서 기본적으로 제공하고 있으며 spring.jackson.\* 를 [application.properties](http://application.properties/) 부분에서 수정함으로써 커스터마이징 할 수 있다.

**CORS**

* 기본적으로 Single Origin Policy가 적용되어 있는데 이를 우회하는 방법으로 Cross Origin Resource Sharing 방법이 사용된다.
* Spring web mvc가 제공하는 기능이고 Spring boot는 CORS를 사용하기 위한 web mvc 설정을 자동설정으로 지원한다.
* 서로다른 포트에서 데이터를 받아올 수 있다. (CORS를 쓰면 AJAX로도 Same Origin Policy의 제약을 넘어 다른 도메인의 자원을 사용할 수 있다.)
* 참고로 서버와 서버간의 관계가 아닌 클라이언트와 서버간의 관계에서 적용되는 것이다.

**Spring Data(RDS(JDBC, DATA JPA) , NOSQL)**

**인메모리 데이터베이스**

* Spring boot 에서 제공하는 AutoConfiguration으로 인해 JDBC 와 DataSource관련된 설정이 기본적으로 지원되기 때문에 Jdbc & H2 의존성만 추가하여 사용할 수 있다. (아무 설정을 추가로 하지 않는 경우 H2 데이터베이스를 선택하여 제공)
* 간단하게 의존성만 추가하면 DataSource 와 JdbcTemplate 를 빈으로 등록하여 제공해주며 추가적으로 관련 설정들을 지원하는 형태.

**Mysql**

* DBCP (Database connection pool) 중 HikariCP 를 지원하며 이러한 디비 커넥션관련 설정이 어플리케이션의 성능에 상당히 많은 영향을 미친다. 따라서 다양한 DBCP 중 신중한 학습을 통해 선택하는 것이 바람직하다.
* 설정된 DBCP를 커스터마이징 하기 위해서 [application.properties](http://application.properties/) 아래와 같은 명령어를 통해 제어가능
  + **spring.datasource.hikari.\***
  + spring.datasource.tomcat.\*
  + spring.datasource.dbcp2.\*

**Spring Data Jpa**

* 소개
  + SDJ → JPA → Hibernate (구현체) → Datasource (JDBC)
  + ORM JPA → 객체와 Relation을 매핑할 때 개념적 불일치를 해결하는 프레임워크
  + JPA → ORM을 위한 자바(EE) 표준
  + Spring Data Jpa
    - Repository 빈 자동생성
    - 쿼리 메소드 자동 구현
    - @EnableJpaRepositories ← Springboot 가 자동설정
* 데이터베이스 초기화 및 Sql문
  + spring.jpa.generate-ddl=true
  + spring.jpa.show-sql=true
  + spring.jpa.hibernate.ddl-auto=
    - update : 기존데이터를 유지하면서 개발
    - create-drop
    - create
    - validate : 운영상황에 적절함. ddl → false 로 두고
  + 일반적인 개발환경에서는 true, update 옵션으로 개발을 하고 어느정도 안정화된 시점부터는 보통 validate, false 옵션을 가져감.
  + 다만 Test도 이러한 설정에 영향을 받기 때문에 테스트용 스키마 파일을 만들어서 데이터베이스를 초기화하는 작업도 함께 이루어짐(H2) 아래의 파일에 생성할 스키마를 정의하여 사용하는 방식
    - schema.sql
    - data.sql
* 데이터베이스 마이그레이션
  + 데이터베이스에서 테이블이나 데이터의 변경도 버전 관리를 할 수 있도록 지원해주는 툴
    - flyway
    - liquibase
  + 필요할 때 보고 참고하길

**Nosql**

* Redis
* Mongodb
* Neo4j

**Spring Security**

* Spring UserDetails
* Spring boot → WebSecurityConfigurer
* PasswordEncoder

**4. Spring boot management**

* Spring boot가 기본적으로 제공하는 admin page
  + 서버용 프로젝트와 실제 프로젝트가 필요하다.
  + 서버용
  + <dependency>
  + <groupId>de.codecentric</groupId>
  + <artifactId>spring-boot-admin-starter-server</artifactId>
  + <version>2.0.1</version>

</dependency>

* + 클라이언트용
    - * spring.boot.admin.client.url = "localhost:8080"

de.codecentric spring-boot-admin-starter-client 2.0.1