환경설정

Open JDK 17 ( unzip ) base

IntelliJ IDEA CE ( Install )

MariaDB ( Install )

PostMan ( Install )

*1. open JDK17 :* openjdk-17\_windows-x64\_bin.zip *파일을 unzip 하시면 됩니다.*

*2. Intellij IDEA CE :* ideaIC-2022.3.2.exe *인스톨 하시면 됩니다.*

***3. Postman :*** Postman-win64-Setup.exe*인스톨 하시면 됩니다.*

*4. MariaDB10.3 : mariadb-10.3.11-winx64.msi 인스톨 하시면 됩니다.*

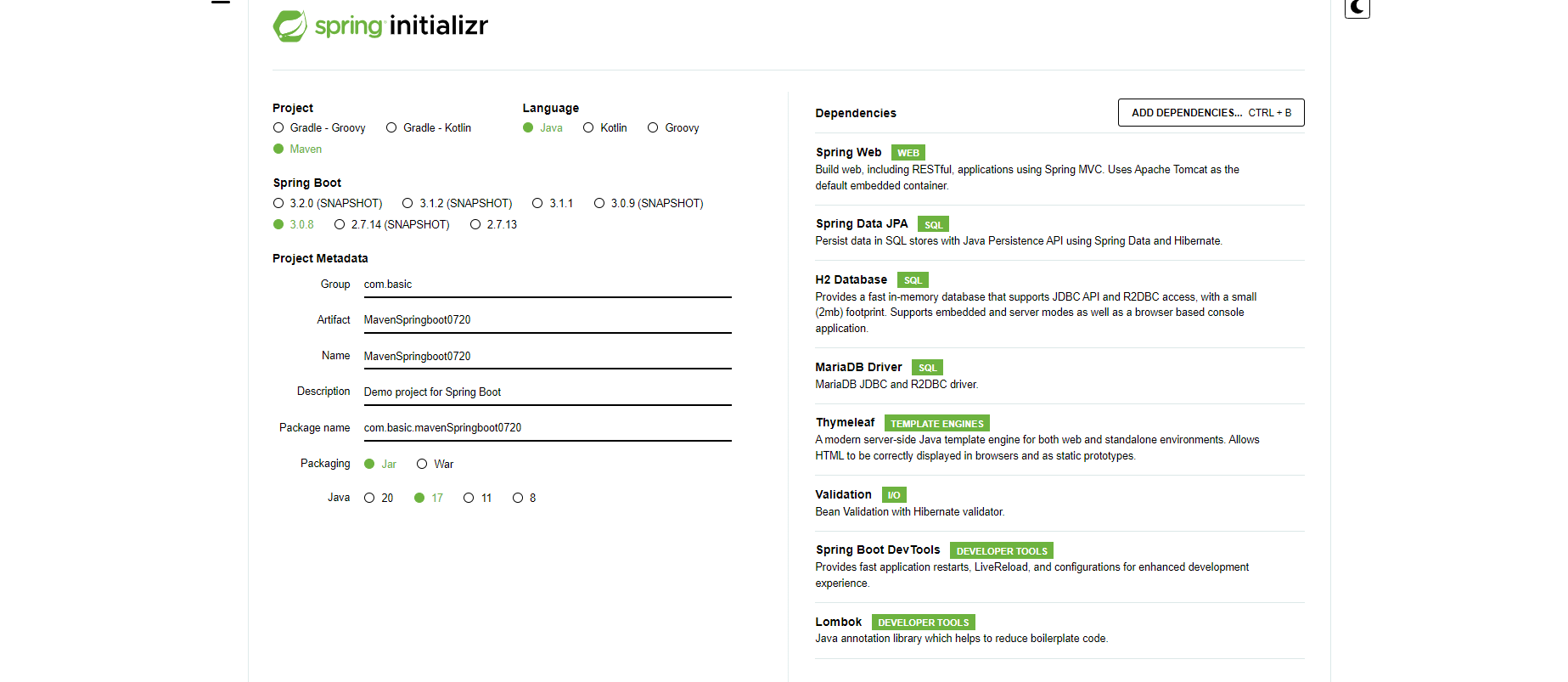
MariaDB 설치시 유의사항

***: Root 패스워드 maria / maria***

***: Use UTF-8 as default server's character set 반드시 체크 해주어야 한글 깨지지 않습니다.***

Spring Initializer에서 SpringBoot 프로젝트를 생성할 때 추가 할 의존성

webmvc, data jpa, h2, mariadb, thymeleaf, validation, devtools, lombok (only for Dev)

  
(maven project 예시)

C:\SpringBoot3\_App 폴더 아래에 Source Code 작성

SpringBoot 튜토리얼

<https://docs.spring.io/spring-boot/docs/current/reference/html/>

SpringBoot API Doc

<https://docs.spring.io/spring-boot/docs/current/api/>

SpringBoot Admin(모니터링UI 제공)  
<https://github.com/codecentric/spring-boot-admin>

JavaSE 17 API Doc

<https://docs.oracle.com/en/java/javase/17/docs/api/>

J2EE API Docs

<https://jakarta.ee/specifications/platform/8/apidocs/>

Jakarta EE 9 API Docs

<https://jakarta.ee/specifications/platform/9/apidocs/>

SpringFramework API Docs

<https://docs.spring.io/spring-framework/docs/current/javadoc-api/>

Transaction Propagation

https://helloino.tistory.com/127

Spring Boot가 선택한 **Third-Party 라이브러리**

Connection Pooling ( DBCP ) => HikariCP ( <https://github.com/brettwooldridge/HikariCP> )

JSON Parser => Jackson ( <https://github.com/FasterXML/jackson> )

Logging => Facade ( SLF4j https://github.com/qos-ch/slf4j) 인터페이스

구현체 ( Logback https://github.com/qos-ch/logback)

Spring Boot 프로젝트 => unzip => IDE 설정

jar 형태로 작성해서 배포함

java -jar xxx.jar

Graalvm framework : https://www.graalvm.org/  
https://www.oracle.com/kr/java/graalvm/what-is-graalvm/

Spring3 버전의 특징

**GraalVM** 을 사용하여 AOT(ahead of time)

JIT 컴파일러 - 바이트 코드를 Runtime에 어셈블리 코드로 변환

AOT 컴파일러 - 빌드 타임에 어셈블리 코드로 변환

os가 윈도우라면 Native Image(.exe) 로 생성해서 배포 가능함

docker image로 빌드 가능함

J2EE8 버전 기준으로 이클립스 재단에 기부함

J2EE - javax.xxx

* Jakarta EE - jakarta.xxx <https://jakarta.ee/>

톰캣 내장 : java -xx.jar 로 단독실행이 가능함.(stand alone)

**특정 third-party libraries** 자동바인딩 – 설정 필요 X  
no-code generation, no-xml  
spring boot **JPA**(jdbc포함되어있음) maven dependency 설정 :   
<https://mvnrepository.com/artifact/org.springframework.boot/spring-boot-starter-data-jpa/3.1.0>

기본생성 main 클래스의

@SpringBootApplication =

: **@SpringBootConfiguration + @EnableAutoConfiguration + @ComponentScan**

@Configuration ( JavaConfig는 XML 대신 클래스에 설정(Configuration)이 가능하도록 )

: XML 대신 Spring Bean에 대한 정보를 클래스에 선언한다.

: 설정 역할을 하는 Java 클래스임을 나타내는 어노테이션

: Java Config

@ComponentScan

: @Component, @Service, @Repository, @Controller – Sprign Bean

@Autowired, @Resource – Spring Bean간의 의존성주입

**@ComponentScan 과 같은 역할을 하는 xml tag**

**<context:component-scan base-packages=”” />**

@Bean 과 @Component 의 차이점은 ??

: 메서드 내부에서 개발자가 직접 생성한 객체를 Spring Bean으로 생성 해주는 역할

: @Component => **클래스에 선언**

: @Bean => **메서드에 선언**

: 외부 라이브러리에서 제공하는 클래스를 Spring Bean으로 생성할 때 주로 사용함

@Configuration **public** **class** **SpringFoxConfig** {

@Bean   
**public** Docket **api**() {

**return** **new** **Docket**(DocumentationType.SWAGGER\_2)

.select()

.apis(RequestHandlerSelectors.any())

.paths(PathSelectors.any())

.build(); }

}

(Swagger-springfox 는 REST API docs 를 생성 해주는 역할 )

**@EnableAutoConfiguration**

: Spring Boot가 제공하는 여러개의 Auto Configuration 들을 활성화 시킨다.

**%% 스프링 컨테이너가 관리하는 Spring Bean은 Singleton Object 이다.**

Spring Boot와 Spring Framework의 차이점

=> Spring Framework : XML, Java Config 에 Bean Configuration 정보를 설정한다.

=> Spring Boot : Java Config 에 Bean Configuration 정보를 설정한다.

terminal을 열고 (Only Cmd 에서만 가능합니다)

환경변수 설정 => 현재 cmd창에서만 적용!! 커맨드창 닫으면 초기화됨)

set JAVA\_HOME=C:\java\jdk-17

set PATH=%JAVA\_HOME%\bin;%PATH%

bootApp 실행

**mvnw spring-boot:run**

jar 생성 command

**mvnw package**

=

java -jar .\target\myspringboot3-app-0.0.1-SNAPSHOT.jar  
jar 실행시 MyRunner.java 클래스에 args 전달하기 :

java -jar -Dbar target\MySpringBoot-App-0.0.1-SNAPSHOT.jar --foo

* 터미널에서 한글 깨질경우 아래 명령어 입력:  
  명령어 : chcp 65001

//4번째 우선순위

java -jar target\MySpringBoot-App-0.0.1-SNAPSHOT.jar --myboot.name=Spring

java -jar .\target\MySpringBoot-App-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod

java -jar -Dspring.profiles.active=prod .\target\MySpringBoot-App-0.0.1-SNAPSHOT.jar

**Gradle Project 에서의 jar 생성 및 실행**

**gradlew build**

java -jar .\build\libs\myspringboot-gradle-app-0.0.1-SNAPSHOT.jar --spring.profiles.active=prod

.properties(yml) 설정된 환경변수 읽어 오는 방법

1. @Value 어노테이션 사용하기

: @Value(“${server.port}”)

String portNumber;

1. Environment의 getProperty() 메서드 사용하기

@Autowired

Environment env;

String portNumber = env.getProperty(“server.port”);

1. 사용자 정의 Properties 클래스 작성하기

LogBack의 Logger 구현 클래스

Logger Class Name = ch.qos.logback.classic.Logger

Log4j2의 Logger 구현 클래스

Logger Class Name = org.apache.logging.slf4j.Log4jLogger

@Component - Spring Bean 임을 선언할 때 사용되는 어노테이션

@Repository, @Service, @Controller, @RestController

@Autowired, @Resouce - Spring Bean 을 주입(injection) 받을 때 사용되는 어노테이션

**Dependency Injection(DI)** 유형

1. Setter Injection

: setter method, 변수 위에 @Autowired, @Resource 선언

@Autowired 는 Type 기반으로 의존하는 객체를 찾아서 주입 / @Qualifier

@Resource 는 Bean의 name(id)으로 의존하는 객체를 찾아서 주입

2. Constructor Injection

: argument가 있는 생성자를 선언한다.

2-1. Constructor Injection 변형

: 주입 받으려는 변수를 final 로 선언한다.

Lombok의 @RequiredArgsConstructor 어노테이션을 사용하여 생성자가 자동으로

생성이 되도록 한다.

-----------------------------------------------

Java8 : 함수형 프로그래밍 지원

함수형 인터페이스

=> 추상메서드를 1개만 가지고 있는 인터페이스

함수형 인터페이스가 가진 추상메서드를 재정의 하는 구문을 람다식으로 표현 할 수 있다.

java.util.function 팩키지의 함수형 인터페이스

Predicate

**boolean test(**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Predicate.html) **t);**

Consumer

**void accept(**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Consumer.html) **t);**

**Supplier**

**T get();**

**Function**

[**R**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html) **apply(**[**T**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html) **t)**

**UnaryOperator**

[**Function**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)**<T,T>**

**BinaryOperator**

[**Function**](https://docs.oracle.com/javase/8/docs/api/java/util/function/Function.html)**<T,T,T>**

EJB = Session Bean + Entity Bean

-----------------------------------------------

ORM(Object Relational Mapping)

Mapping Rule

Entity Class ⇔ Table

Entity Object ⇔ Row(Record)

Entity Variables ⇔ Columns

**JPA(Java Persistence API)**

: ORM 기능을 위한 Jakarta EE 표준 스펙

**Hibernate**

: JPA 표준스펙을 구현한 ORM 구현체 https://github.com/hibernate/hibernate-orm

**Spring Data JPA**

: Spring Boot 개발자가 Hibernate를 좀 더 쉽게 사용할 수 있도록

Hibernate 구현체를 더 추상화 한 라이브러리

<https://spring.io/projects/spring-data-jpa>

h2 Console 접근 URL

http://localhost:8080/h2-console

Jakarta EE 9 API Docs

<https://jakarta.ee/specifications/platform/9/apidocs/>

Spring Data Core Api doc

<https://docs.spring.io/spring-data/commons/docs/current/api/index.html>

Spring Data JPA Api doc

<https://docs.spring.io/spring-data/jpa/docs/current/api/index.html>

# **Hibernate JavaDoc (6.2.3.Final)**

<https://docs.jboss.org/hibernate/stable/orm/javadocs/>

CrudRepository의 메소드에 대한 설명

T 는 Entity 클래스의 Type

등록: <S **extends** T> S save(S entity);

리스트 조회: **Iterable**<T> findAll(); (CrudRepository)

**List**<T> findAll(); (ListCrudRepository)

PK로 조회: **Optional**<T> findById(ID id);

삭제: **void** delete(T entity);

**void** deleteById(ID id);

PK존재여부: **boolean** existsById(ID id);

select \* from accounts where id = 3L=> findById(ID)와 매핑

select \* from accounts where username = ‘spring’  
-> ex) Optional<T> findByUsername(username) 등 직접 finder method 선언

Iterable

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Optional.html>

Optional

<https://docs.oracle.com/en/java/javase/17/docs/api/java.base/java/util/Optional.html>

JPA Repository query(finder) method 메소드 명 규칙

<https://docs.spring.io/spring-data/jpa/docs/current/reference/html/#jpa.query-methods>

Dialect(방언, 사투리)

spring.jpa.properties.hibernate.dialect=org.hibernate.dialect.MariaDB103Dialect

Hibernate Dialect Api doc

<https://docs.jboss.org/hibernate/orm/5.2/javadocs/org/hibernate/dialect/package-summary.html>

Transactional Propagation(트랜잭션 전파속성)

Required

None ===> T1

T1 ===> T1 (기존 트랜잭션이 유지됨)

Requires\_new

None ===> T2

T1 ===> T2 (항상 새로운 트랜잭션이 시작됨)

Mandatory

None ===> Exception이 발생함

T1 ===> T1 (기존 트랜잭션이 유지됨)

//Primary Key의 값 자동 증가 전략 4가지

public enum GenerationType {

/\*\*

\* Indicates that the persistence provider must assign

\* primary keys for the entity using an underlying

\* database table to ensure uniqueness.

\*/

*TABLE*,

/\*\*

\* Indicates that the persistence provider must assign

\* primary keys for the entity using a database sequence.

\* Sequence 를 제공하는 Oracle

\*/

*SEQUENCE*,

/\*\*

\* Indicates that the persistence provider must assign

\* primary keys for the entity using a database identity column.

\* Column Auto Increment를 제공하는 MySql, MariaDB

\*/

*IDENTITY*,

/\*\*

\* Indicates that the persistence provider should pick an

\* appropriate strategy for the particular database. The

\* <code>AUTO</code> generation strategy may expect a database

\* resource to exist, or it may attempt to create one. A vendor

\* may provide documentation on how to create such resources

\* in the event that it does not support schema generation

\* or cannot create the schema resource at runtime.

\*/

*AUTO (default)*

}

**GenerationType.Auto**

MariaDB [boot\_db]> desc account;

+----------+--------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+----------+--------------+------+-----+---------+-------+

| id | bigint(20) | NO | PRI | NULL | |

| password | varchar(255) | YES | | NULL | |

| username | varchar(255) | YES | UNI | NULL | |

+----------+--------------+------+-----+---------+-------+

account\_seq 테이블이 추가로 생성된다.

MariaDB [boot\_db]> desc account\_seq;

+-----------------------+---------------------+------+-----+---------+-------+

| Field | Type | Null | Key | Default | Extra |

+-----------------------+---------------------+------+-----+---------+-------+

| next\_not\_cached\_value | bigint(21) | NO | | NULL | |

| minimum\_value | bigint(21) | NO | | NULL | |

| maximum\_value | bigint(21) | NO | | NULL | |

| start\_value | bigint(21) | NO | | NULL | |

| increment | bigint(21) | NO | | NULL | |

| cache\_size | bigint(21) unsigned | NO | | NULL | |

| cycle\_option | tinyint(1) unsigned | NO | | NULL | |

| cycle\_count | bigint(21) | NO | | NULL | |

+-----------------------+---------------------+------+-----+---------+-------+

**GenerationType.Identity** ( auto increment column )

+----------+--------------+------+-----+---------+----------------+

| Field | Type | Null | Key | Default | Extra |

+----------+--------------+------+-----+---------+----------------+

| id | bigint(20) | NO | PRI | NULL | auto\_increment |

| password | varchar(255) | YES | | NULL | |

| username | varchar(255) | YES | UNI | NULL | |

+----------+--------------+------+-----+---------+----------------+

@Table(“user”)

*@RestController = @Controller + @ResponseBody*

Response (응답) : *@ResponseBody*

: Java Object -> JSON 변환 처리는 Jackson이 담당 함

: 변환된 데이터를 응답(response) body에 담아 주는 역할

Request (요청) : *@RequestBody*

: JSON -> Java Object 변환 처리는 Jackson이 담당함

: 변환된 데이터를 요청(request)에 담아서 컨트롤러의 메서드의 아규먼트로 매핑 해주는 역할

실질적인 변환 처리가 누가 할까요? Jackson

ResponseEntity

: Body + Http StatusCode + Header 한번에 담아서 응답을 주는 객체

<https://www.baeldung.com/spring-response-entity>

Put과 Patch 차이점

: put - 모든 항목 전체수정

: patch - 부분 항목 수정

등록

POST

[http://localhost:8080/users](http://localhost:8087/users)

header

content-type:application/json

body

{

"name":"스프링",

"email":"spring@a.com"

}

Id로 조회 GET

[http://localhost:8080/users](http://localhost:8087/users)/{id}

email로 조회 GET

[http://localhost:8080/users](http://localhost:8087/users)/email/spring@a.com/

목록조회 GET

[http://localhost:8080/users](http://localhost:8087/users)

수정 PATCH

header

content-type:application/json

body

[http://localhost:8080/users](http://localhost:8087/users)/{email}/

{

"name":"Spring"

}

수정 PUT

header

content-type:application/json

body

[http://localhost:8080/users](http://localhost:8087/users)/1

{

"name":"Spring",

"email":"spring@a.com"

}

삭제 DELETE

[http://localhost:8080/users](http://localhost:8087/users)/1

ResponseEntity 클래스 <https://docs.spring.io/spring-framework/docs/current/javadoc-api/org/springframework/http/ResponseEntity.html>

=> Body + Status Code + Header

[http://localhost:8080/users](http://localhost:8087/users)/xml

<dependency>

<groupId>com.fasterxml.jackson.dataformat</groupId>

<artifactId>jackson-dataformat-xml</artifactId>

<version>${jackson-bom.version}</version>

</dependency>

Thymeleaf Docs

<https://www.thymeleaf.org/doc/tutorials/3.1/usingthymeleaf.html>

Thymeleaf-Spring Docs

<https://www.thymeleaf.org/doc/tutorials/3.1/thymeleafspring.html>

Validation(입력항목 검증) 표준 스펙

Java Bean Validation

jakarta.validation.\*

Jakarta API Docs

<https://jakarta.ee/specifications/platform/8/apidocs/>

Java Bean Validation의 구현체

Hibernate Validator

<https://github.com/hibernate/hibernate-validator>

<https://hibernate.org/validator/>

Hibernate Validator API

https://docs.jboss.org/hibernate/validator/6.2/api/

Spring Framework5 API Docs

<https://docs.spring.io/spring-framework/docs/5.3.24/javadoc-api/>

jakarta validation의 어노테이션

@NotEmpty vs @NotBlank

@NotEmpty : white space(“ “) 는 허용

@NotBlank : 문자열의 공백을 제거(trim)하고 체크 하므로

white space(“ “) 허용하지 않음 trim() + notEmpty()

—------------------------

CORS(Cross Origin Resource Sharing)

=> SOP를 우회하기 위한 방법

Ajax(Asynchronous Javascript and Xml)

: Xml(JSON) - 서버에서 보내준 데이터

: Asynchronous - 비동기 XmlHttpRequest(XHR)

: Javascript - html을 동적으로 생성

: SOP(Single Origin Policy)

Client (ReactJS) -> SpringBoot(REST API)

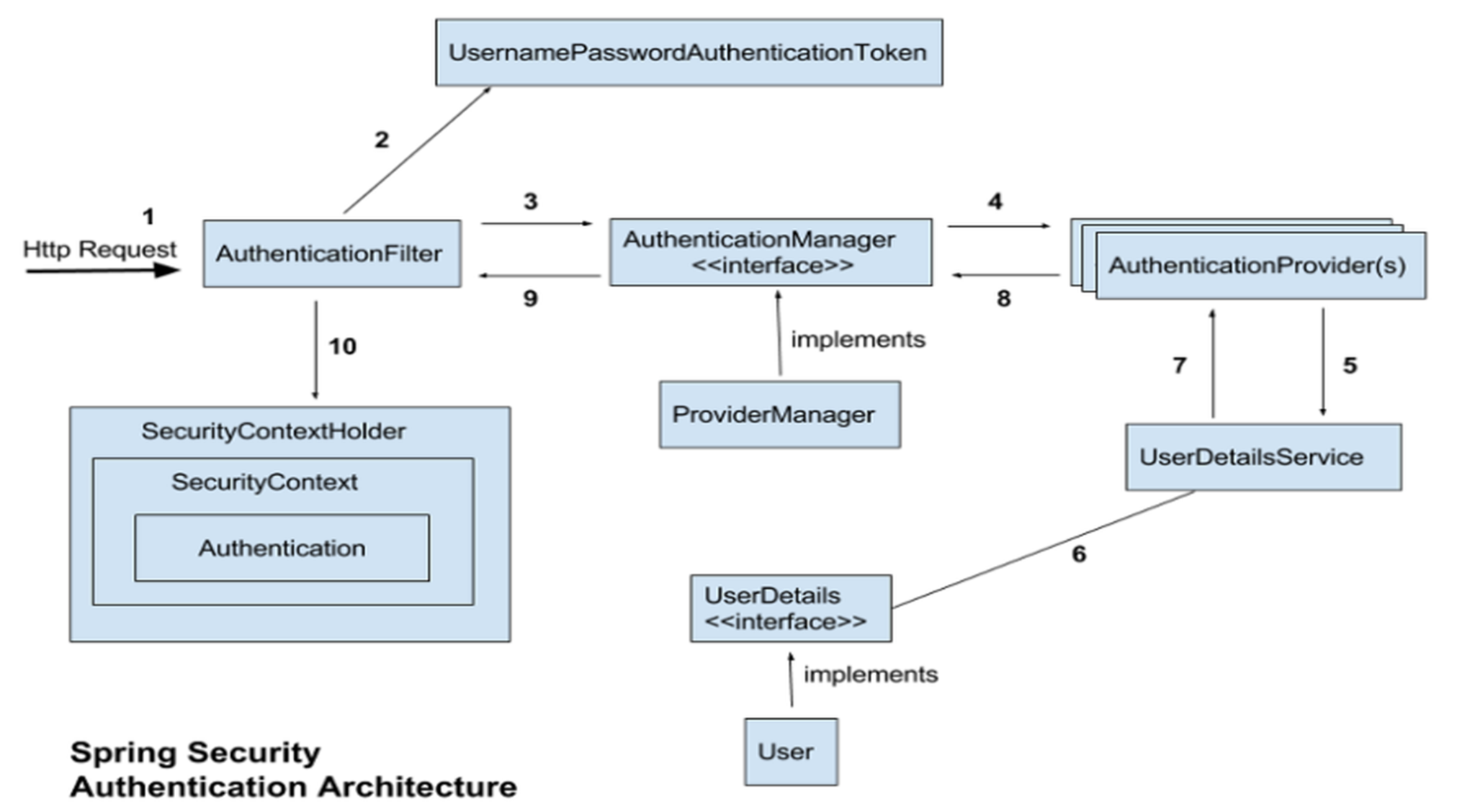
<http://localhost:3000> -> [http://localhost:8080](http://localhost:8087)

—------------------------

Actuator endpoints 요청하기

http://localhost:8080/actuator

Spring Security Architecture



Spring Security 6.1 Api doc

<https://docs.spring.io/spring-security/site/docs/current/api/>

**IDEA 단축키**

**ctrl + shift + f10 : Java run**

**ctrl + shift + t : 테스트 케이스 추가**

**ctrl + alt + o(오우) : auto import**

**ctrl + alt + v : return type 자동 생성 ( eclipse 는 alt + shift + l(엘) )**

**ctrl + alt + shift + l(대문자엘) : code format**

**alt + insert : generate constructor, getter, setter**

**alt + enter : import , create new class(interface, enum)**

**alt + shift + insert : column selection mode**

**ctrl + alt + m : extract method**

**ctrl + alt + n : inline method**

**ctrl + o (오우) : override method**

**ctrl + i (아이) : implements method**

**IDEA 설정**

1. JDK17 위치 알려주기

File -> Project Structure

: project -> project sdk - Add sdk

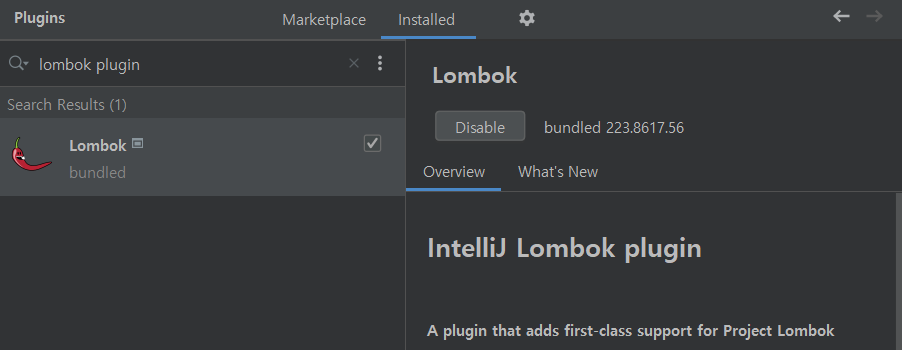
1. File -> Settings

: Font 설정

: **Lombok** 설정

**Plugins > Lombok Plugin install** 확인

Plugins > installed

Lombok 설치된 것을 확인 한다.  


​**Build, Execution, Deployment > Compiler > Annotation Processors >**

**Enable annotation processing** 체크

: Auto Import 설정

**File -> Settings -> Editor -> General -> Auto Import**

**1) Insert imports on paste 옵션은 코드를 붙여넣기 한 경우의 import문 삽입 옵션으로 Always,**

**Ask, Never을 사용할 수 있다.**

**Always : 모두 자동 import**

**Ask : 확인 후 선택**

**Never : 자동 import 하지 않음**

**2) Add unambiguous imports on the fly 옵션 체크**

: Devtools 관련 설정

​**Build, Execution, Deployment > Compiler**

**- Build Project Automatically 체크 합니다.**

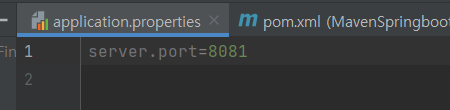
**Advanced Settings**

**- Allow auto-make to start even-if developed application is currently running 체크**

: properties file에서 한글이 깨지지 않게 하려면

**Editor -> File Encodings**

**- project encoding, default encoding for properties 을 utf-8 로 선택**

**서버 포트번호는 기본이 8080인데 변경 가능함:**

postman 요청 전송 방법:

