

Spring Restful API Service 개발

김 순곤

soongon@hucloud.co.kr

목 차

1. 스프링과 스프링 부트 소개
2. REST 와 Resource 이해
3. RESTful API 디자인 가이드
4. CRUD 구현
5. RESTful 웹서비스 보안
6. 성능을 위한 설계

01 장

스프링과 스프링 부트
Spring & Spring Boot

01 Spring

■ Spring

- EJB를 주 프레임워크로 사용할 때 불편했던 점들을 해소.
- 2002년 로드존슨이 출판한 도서 **Expert One-on-One J2EE Design and Development**에 선보인 코드가 Spring의 근간이 됨.
- 이 도서를 읽은 개발자들이 “코드가 책으로만 존재하기에는 너무 아깝다”며 로드 존슨의 허가를 받은 뒤 프레임워크로 발전시킴.
- 2003년 6월 Apache 2.0 License로 공개됨.
- 버전 약력
 - 1.0 : 2004년 3월
 - 2.0 : 2006년 10월
 - 2.5 : 2007년 11월
 - 3.0 : 2009년 12월
 - 3.1 : 2011년 12월
 - 4.0 : 2013년 12월
 - 5.0 : 2017년



<http://spring.io>

Spring Framework 개요

✓ 스프링 프레임워크 란?

Java 엔터프라이즈 개발을 편하게 해주는 오픈소스 **경량급 애플리케이션 프레임워크**이다.

애플리케이션 프레임워크

- 특정 계층이나 기술, 업무 분야에 국한되지 않고 애플리케이션의 전 영역을 포괄하는 범용적인 프레임워크를 말한다.

경량급 프레임워크

- 단순한 웹컨테이너에서도 엔터프라이즈 개발의 고급기술을 대부분 사용할 수 있다.

Spring Framework 개요

☒ 스프링 프레임워크 란?

Java 엔터프라이즈 개발을 편하게 해주는 오픈소스 경량급 애플리케이션 프레임워크이다.

엔터프라이즈 개발 용이

- ◉ 개발자가 복잡하고 실수하기 쉬운 Low Level에 많이 신경 쓰지 않으면서 Business Logic 개발에 전념할 수 있도록 해준다.

오픈소스

- ◉ Spring은 OpenSource의 장점을 충분히 취하면서 동시에 OpenSource 제품의 단점과 한계를 잘 극복함

스프링 프레임워크 특징

1. 컨테이너 역할

- Spring 컨테이너는 Java 객체의 LifeCycle을 관리하며, Spring 컨테이너로 부터 필요한 객체를 가져와 사용할 수 있다.

2. DI (Dependency Injection) 지원

- Spring은 설정 파일이나 어노테이션을 통해서 객체 간의 의존관계를 설정할 수 있도록 하고 있다.

3. AOP(Aspect Oriented Programming) 지원

- Spring은 트랜잭션이나 로깅, 보안과 같이 공통적으로 필요로 하는 모듈들을 실제 핵심 모듈에서 분리해서 적용할 수 있다.

스프링 프레임워크 특징

4. POJO(Plain Old Java Object) 지원

- Spring 컨테이너에 저장되는 Java 객체는 특정한 인터페이스를 구현하거나, 특정 클래스를 상속받지 않아도 된다.

5. 트랜잭션 처리를 위한 일관된 방법을 지원

- JDBC, JTA 등 어떤 트랜잭션을 사용하던 설정을 통해 정보를 관리하므로 트랜잭션 구현에 상관없이 동일한 코드 사용 가능

6. 영속성 (Persistence)과 관련된 다양한 API 지원

- Spring은 MyBatis, Hibernate 등 데이터베이스 처리를 위한 ORM(Object Relational Mapping) 프레임워크들과의 연동 지원

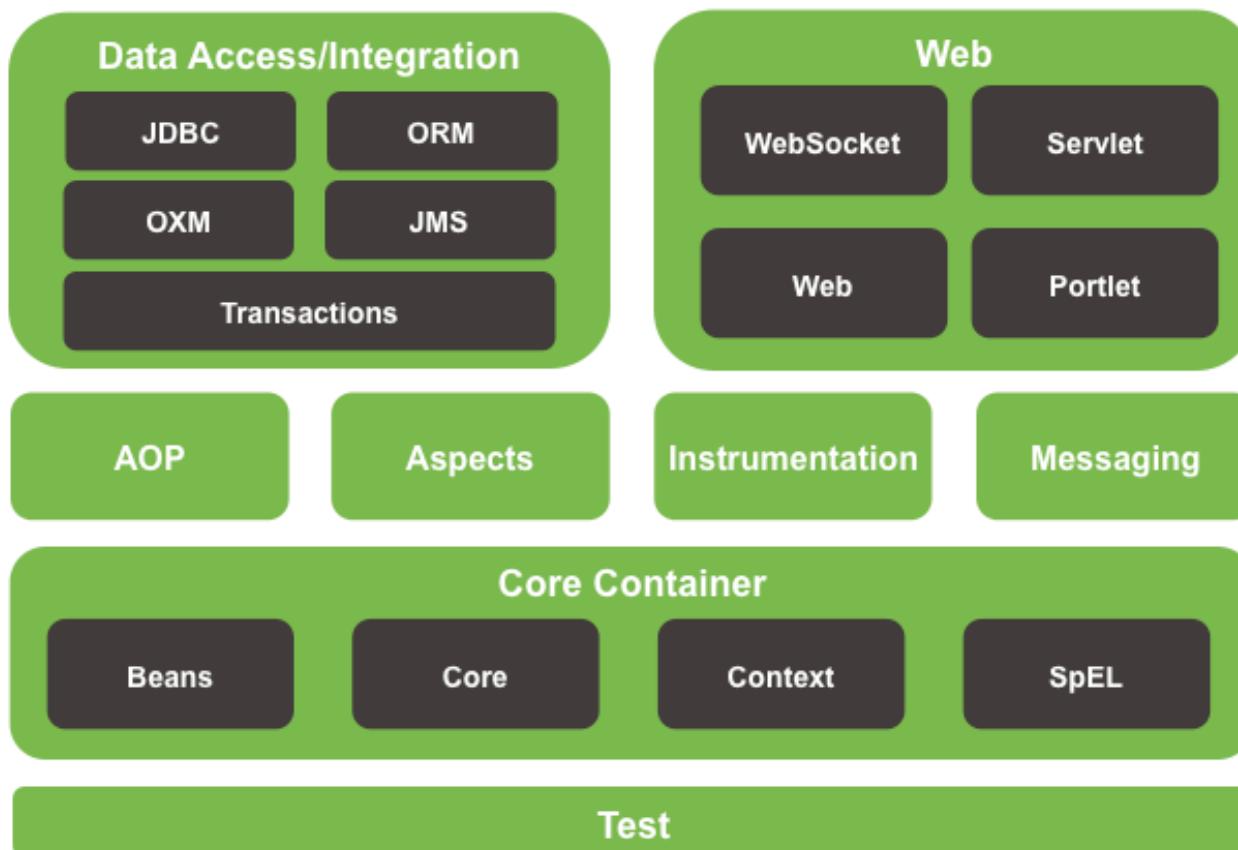
스프링 프레임워크 기능요소

- **Spring**

스프링의 주요 모듈 목록



Spring Framework Runtime



스프링과 메이븐

■ Spring 의 시작

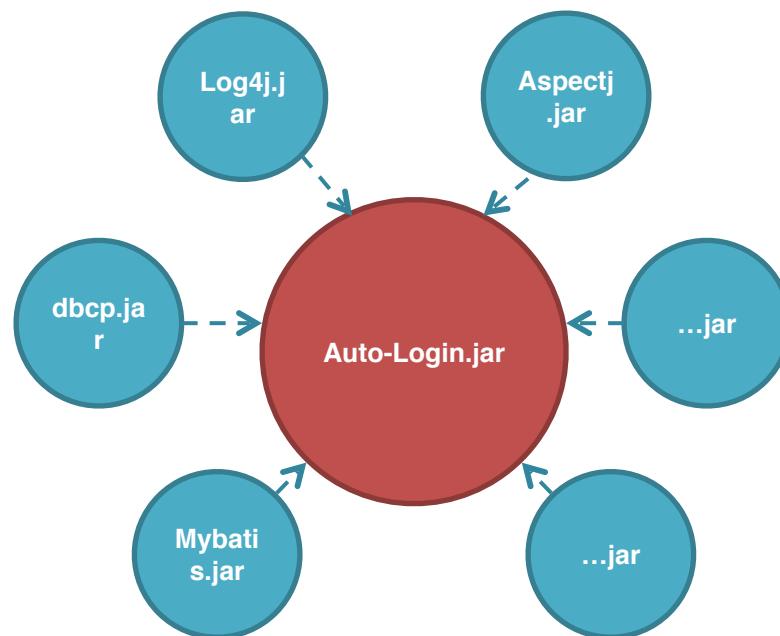
- Spring 을 사용하기 위해 Maven Build 가 필요함.

■ Maven

- Spring 의 의존 라이브러리를 간편하게 추가할 수 있도록 의존성 주입을 제공.
- 그 외 오픈 소스 라이브러리도 Maven Repository 에 업로드 함으로써 대부분의 라이브러리를 자동으로 추가 가능함.
- 프로젝트의 구조를 자동으로 구성함.
- Build 를 통해 배포를 위한 파일을 생성함.

■ Maven 과 유사한 Gradle

- 설정 파일을 XML 대신 Groovy 언어를 사용
- 안드로이드 기본 빌드 툴



Spring 개발 시작

Java Development Kit 1.8

JDK 설치

- 스프링 5.0(스프링 부트 2.0)을 위해서는 JDK8 이상 필요 (스프링 4.0은 JDK 6 이상)

The screenshot shows the Oracle Java SE Downloads page. At the top, there's a navigation bar with links for Sign In/Register, Help, Country, Communities, I am a..., I want to..., Search, Products, Solutions, Downloads, Store, Support, Training, Partners, About, and OTN. Below the navigation is a breadcrumb trail: Oracle Technology Network > Java > Java SE > Downloads. On the left, there's a sidebar with links for Java SE, Java EE, Java ME, Java SE Support, Java SE Advanced & Suite, Java Embedded, Java DB, Web Tier, Java Card, Java TV, New to Java, Community, and Java Magazine. The main content area has tabs for Overview, Downloads (which is selected), Documentation, Community, Technologies, and Training. A section titled "Java SE Development Kit 8 Downloads" explains the purpose of the JDK and lists useful tools. It also includes sections for "See also:" and "JDK 8u111 Checksum" and "JDK 8u112 Checksum". Below this is a section for "Java SE Development Kit 8u112" with a note about accepting the Oracle Binary Code License Agreement. There are two radio buttons: "Accept License Agreement" (selected) and "Decline License Agreement". A table provides download links for various platforms:

Product / File Description	File Size	Download
Linux x86	162.42 MB	jdk-8u112-linux-i586.rpm
Linux x86	177.12 MB	jdk-8u112-linux-i586.tar.gz
Linux x64	159.97 MB	jdk-8u112-linux-x64.rpm
Linux x64	174.73 MB	jdk-8u112-linux-x64.tar.gz
Mac OS X	223.15 MB	jdk-8u112-macosx-x64.dmg
Solaris SPARC 64-bit	139.78 MB	jdk-8u112-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	00.00 MB	jdk-8u112-solaris-sparcv9.tar.gz

On the right side, there are two sections: "Java SDKs and Tools" and "Java Resources", each listing several links.

스프링 개발 툴 설치

Spring Tool Suite (STS) - 전자정부 프레임워크 개발 툴과 유사



Spring Tool Suite™

Your tools should be as refined as the code you write. STS is our Eclipse-based IDE crafted to serve the needs of building applications with Spring. We're always working on new features and performance in our mission to make STS the most productive Eclipse distribution available.

[Learn more about STS >](#)

TOOLS

Spring Tool Suite™

The Spring Tool Suite is an Eclipse-based development environment that is customized for developing Spring applications. It provides a ready-to-use environment to implement, debug, run, and deploy your Spring applications, including integrations for Pivotal tc Server, Pivotal Cloud Foundry, Git, Maven, AspectJ, and comes on top of the latest Eclipse releases.

Included with the Spring Tool Suite is the developer edition of Pivotal tc Server, the drop-in replacement for Apache Tomcat that's optimized for Spring. With its Spring Insight console, tc Server Developer Edition provides a graphical real-time view of application performance metrics that lets developers identify and diagnose problems from their desktops.

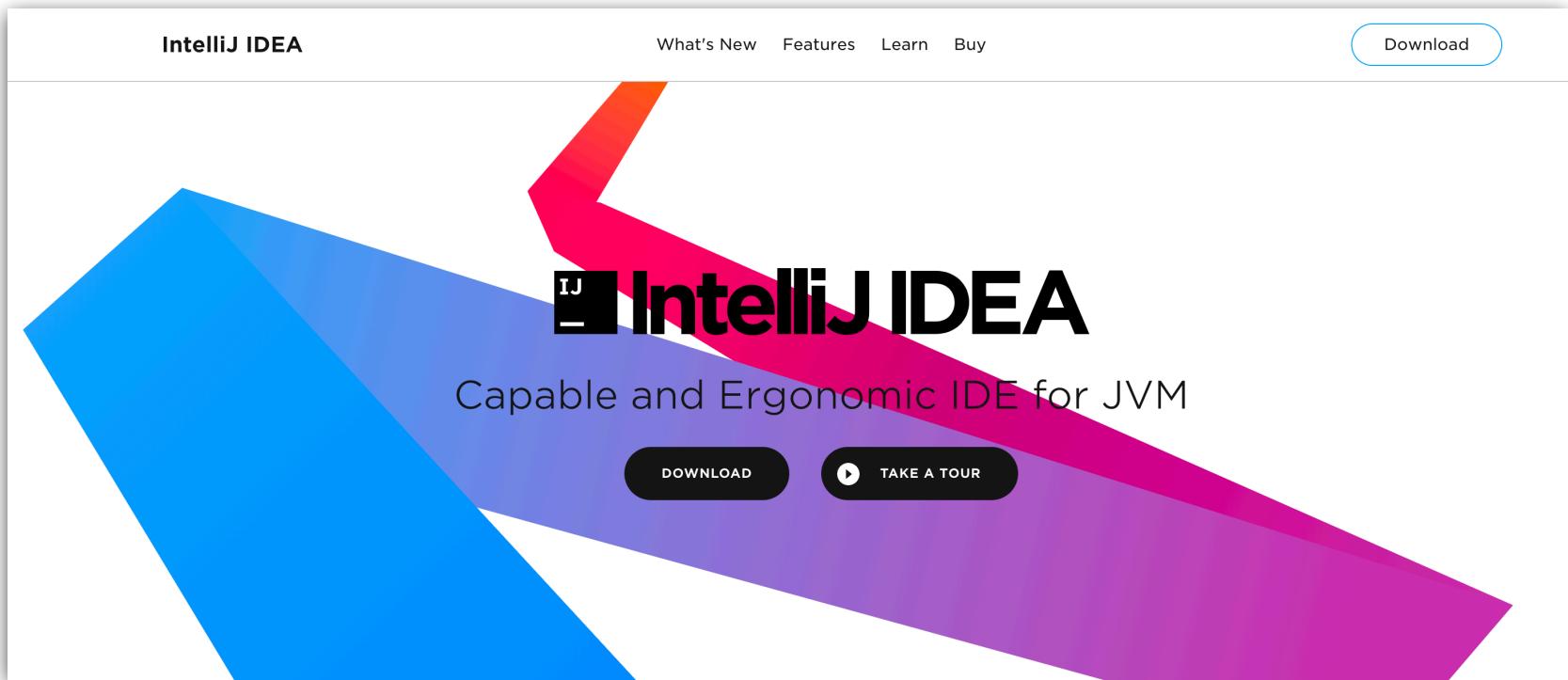
The Spring Tool suite supports application targeting to local, virtual and cloud-based servers. It is freely available for development and internal business operations use with no time limits, fully open-source and licensed under the terms of the Eclipse Public License.



DOWNLOAD STS
(3.8.4.RELEASE for Mac)

[See All Versions](#)

스프링 개발 툴 - IntelliJ IDEA



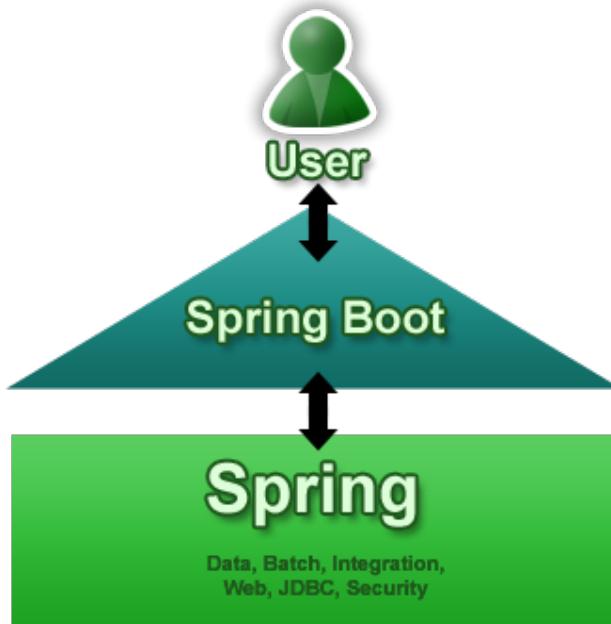
<https://www.jetbrains.com/idea/>

스프링 부트

스프링 부트 소개

스프링 부트 - 차세대 스프링 프레임워크

- 스프링 기반 production-ready 어플리케이션 개발
- Convention over Configuration 을 따름
- 톰캣이 내장되어 단독으로 실행 가능
- 설정을 최대한 줄이고 XML 설정파일 필요 없음



스프링 부트 소개

✓ 프로젝트 생성방법

1

<https://start.spring.io> - Spring Initializer 사이트 활용

The screenshot shows the Spring Initializer interface. At the top, it says "SPRING INITIALIZR bootstrap your application now". Below that, there's a header with "Generate a" followed by dropdown menus for "Maven Project", "Java" (set to 1.8), and "Spring Boot" (set to 1.5.4).

On the left, under "Project Metadata", there's a "Group" field containing "kr.co.acomp" and an "Artifact" field containing "hello".

On the right, under "Dependencies", there's a search bar with "Web, Security, JPA, Actuator, Devtools..." and a "Selected Dependencies" section with "Web" and "MyBatis" listed.

At the bottom center is a large green "Generate Project" button.

2

Spring Boot CLI

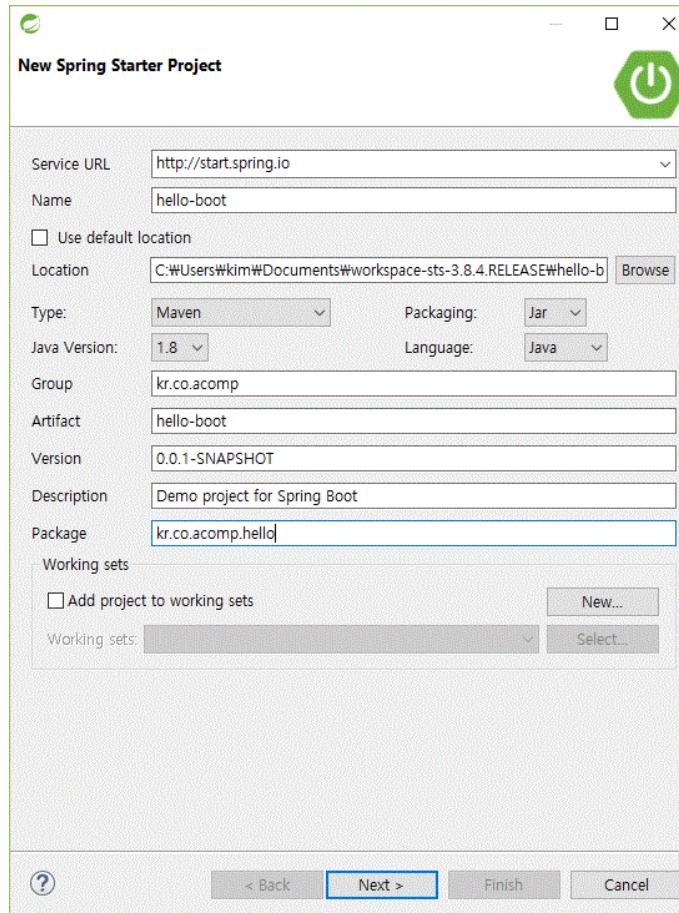
- 커맨드 라인 스프링 부트 툴

스프링 부트 소개

프로젝트 생성 방법

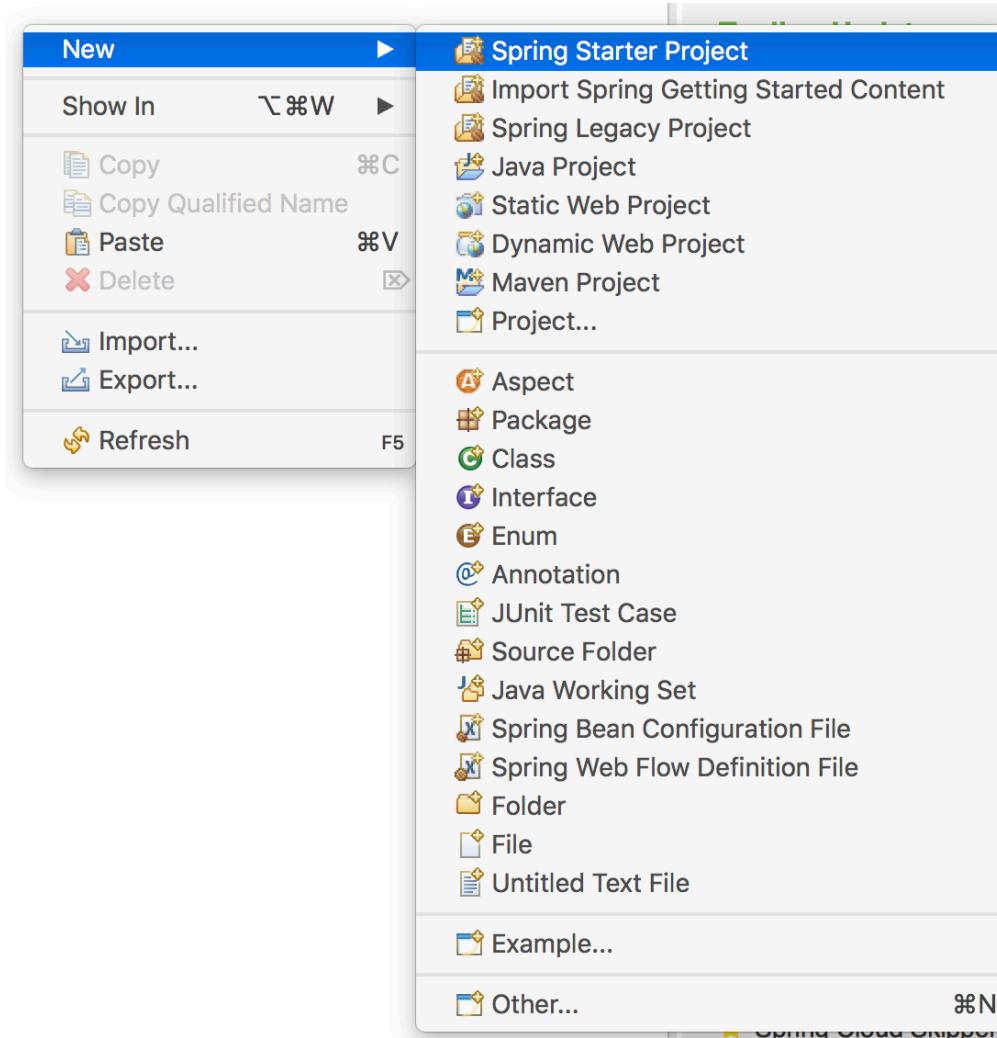
3

STS나 IntelliJ IDEA를 통한 방법



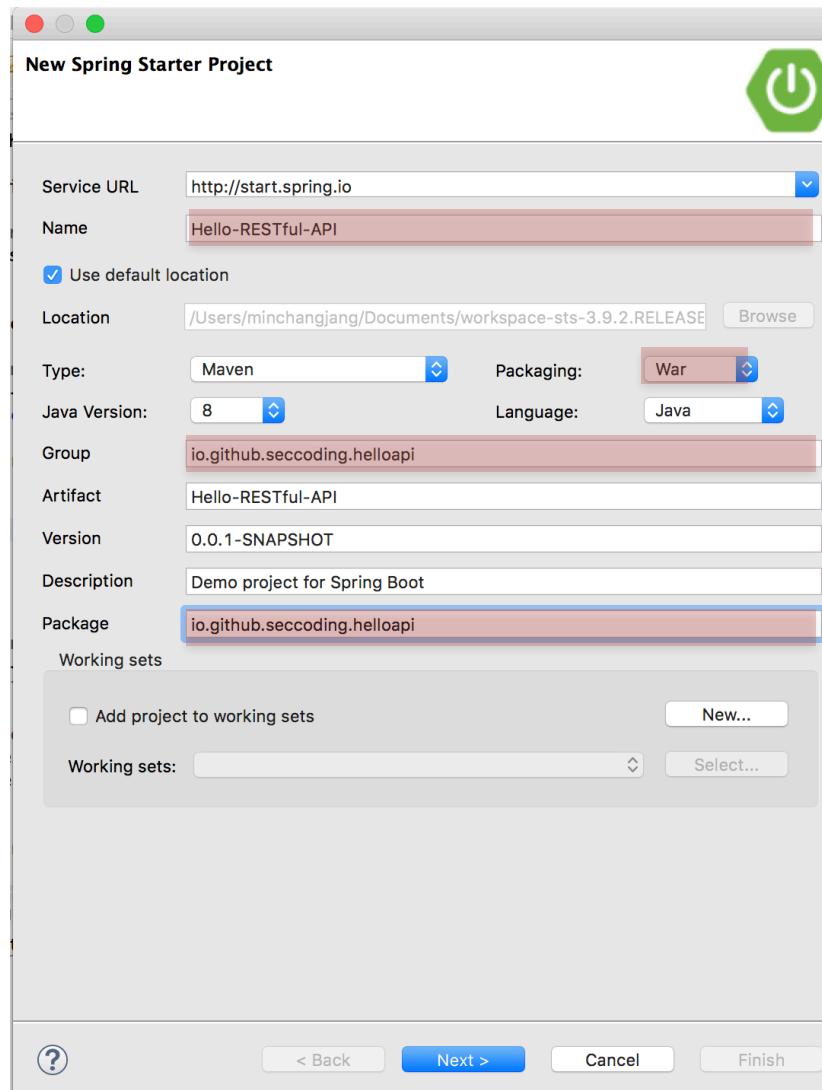
스프링 부트 소개

프로젝트 생성 방법 - Eclipse의 Spring Starter Project 이용



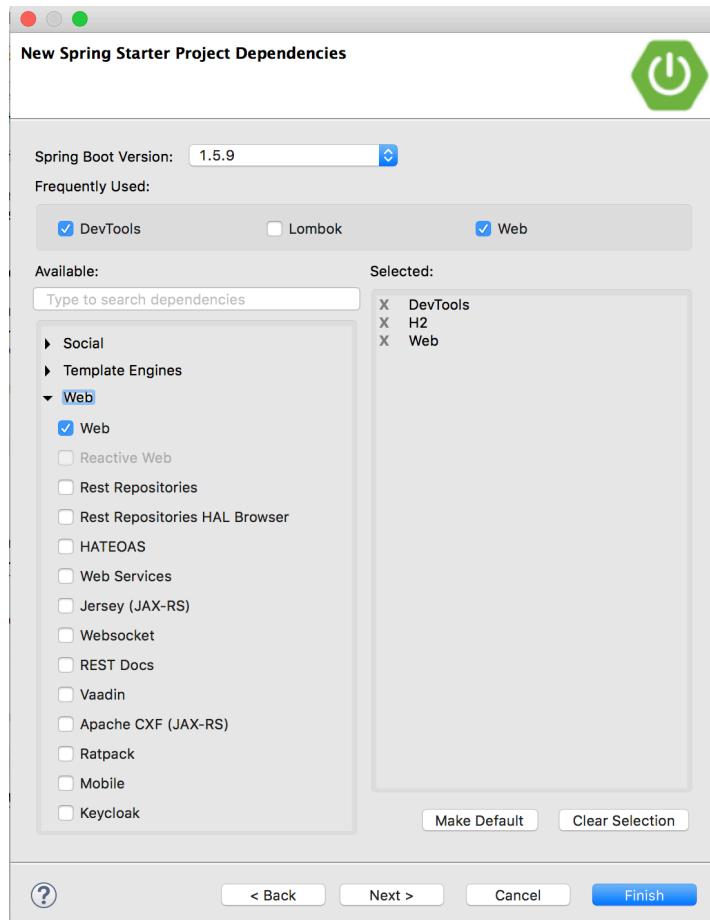
스프링 부트 소개

✓ 프로젝트 생성 방법



스프링 부트 소개

의존라이브러리 설정



**Web -> Web
Core -> DevTools
SQL -> H2**

선택

스프링 부트 프로젝트 기본 구조

pom.xml :

- 메이븐 빌드 명세

mvnw :

- maven wrapper

HelloBootApplication.java :

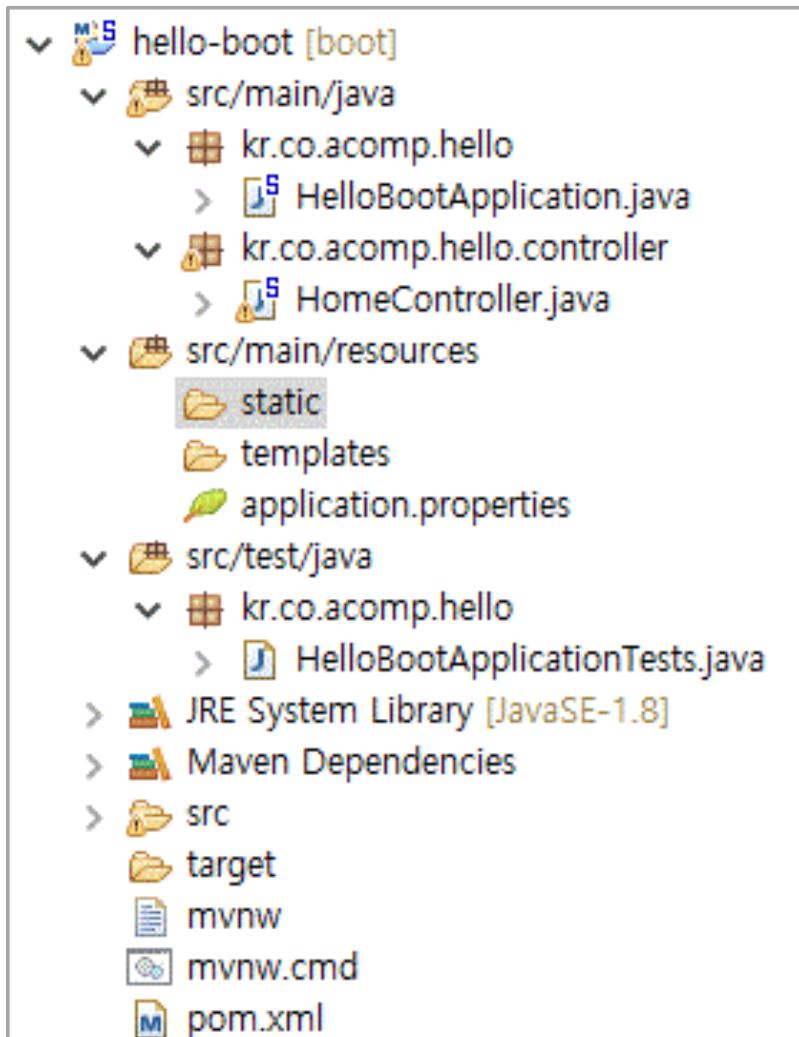
- main class

application.properties :

- 필요한 구성은 추가하는 프로퍼티

HelloBootApplicationTest.java :

- Junit Test class



Application.java

```
@SpringBootApplication // 컴포넌트 구성과 자동 구성
public class HelloBootApplication {

    public static void main(String[] args) {
        SpringApplication.run(HelloBootApplication.class, args);
    }
}
```

@SpringBootApplication

- **@Configuartion**
- **@ComponentScan**
- **@EnableAutoConfiguration**

스프링 부트 설정 파일

application.properties

- 선택적인 파일 (사용 안 해도 됨)
- 명시적으로 요청하는 부분이 없음
- spring boot 가 로드되면 자동으로 이 파일을 로드

기존의 XML 설정파일도 가져올 수 있음

- Application.java 파일에 아래 어노테이션 추가
- @ImportResource("classpath:spring/root-context.xml")

스프링 부트 개발자 툴

스프링 부트 개발모드 지원

- ⌚ thymeleaf 등 템플릿에 대한 개발 캐싱을 비활성화 처리
- ⌚ 소스의 변경이 일어나면 스프링 부트 서버가 새로 시작되면 바로 반영
- ⌚ html, css 부분을 변경하면 바로 브라우저에서 감지하여 반영
 - 크롬, 파이어폭스, 사파리에서 <http://livereload.com/extensions/> 플러그인 설치 필요

간단 설정

- ⌚ 디펜던시 추가 : spring-boot-devtools

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
</dependency>
```

스프링 부트 테스트

아래 의존성을 추가

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
    <exclusions>
        <exclusion>
            <groupId>org.junit.vintage</groupId>
            <artifactId>junit-vintage-engine</artifactId>
        </exclusion>
    </exclusions>
</dependency>
```

```
// @RunWith(SpringRunner.class) => jUnit5 에서는 필요없음
@SpringBootTest
public class HelloBootApplicationTests {

    @Test
    public void contextLoads() {
    }
}
```

실습 : 첫 번째 엔드 포인트

Hello, World!

Spring-WebMVC 에서의 RESTful 지원

✓ @Controller와

@RequestMapping(value="url", method=POST/GET/DELETE/PUT),

@ResponseBody 사용

```
@Controller  
public class IndexController {  
  
    @RequestMapping(value="/", method=RequestMethod.GET)  
    @ResponseBody  
    public String index() {  
        return "Hello, Boot";  
    }  
  
}
```

Spring Boot 에서의 RESTful 지원

- @RestController**와
@X-Mapping 사용

```
@RestController
public class IndexController {

    @GetMapping("/")
    public String index() {
        return "Hello, Boot";
    }

}
```

@RestController

@Controller

@ResponseBody

@GetMapping

**@RequestMapping(value="/",
method=RequestMethod.GET)**

REST 소개

What is Web Service?

- 컴퓨팅 디바이스 상호 간 제공되는 소프트웨어 서비스 또는 기능
- WWW 표준 프로토콜 사용

웹 서비스 표준이 없다면?

웹 서비스란?

- 두 엔드포인트 간 메세지가 전달되는 방식
 - ▶ API, Application Programming Interface

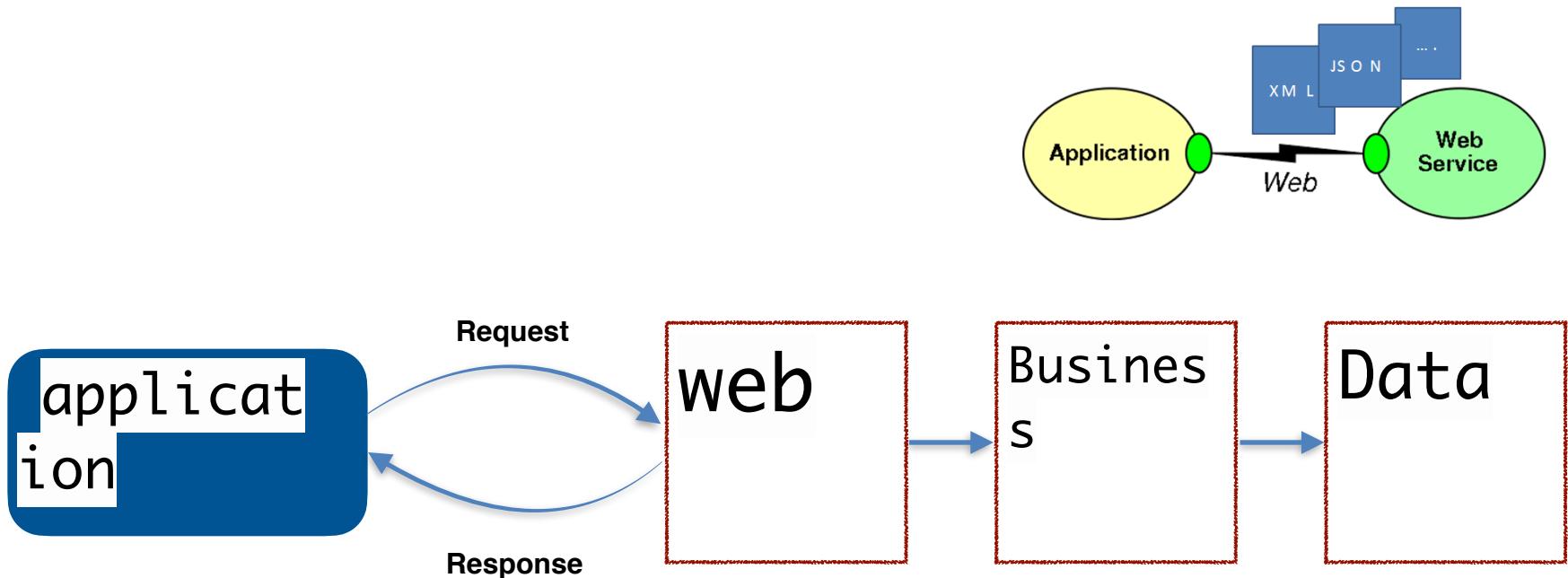
- IoT 의 증가로 API 필요성 증가
 - ▶ API as a Service (AaaS)
 - AWS, Google API, Naver API, Kakao API, 대한민국 공공 API …
 - REST API 를 통해 개발 가능, CLI 를 통해 접근 가능

- Use cases

- ▶ <http://booking.com> <https://www.expedia.com>
 - API 를 통해 실시간 가격정보 제공
 - 요청에 따른 과금 가능

SOA 개념에서 발전됨

웹 서비스란?



SOAP vs. REST

REST requirements 및 특징

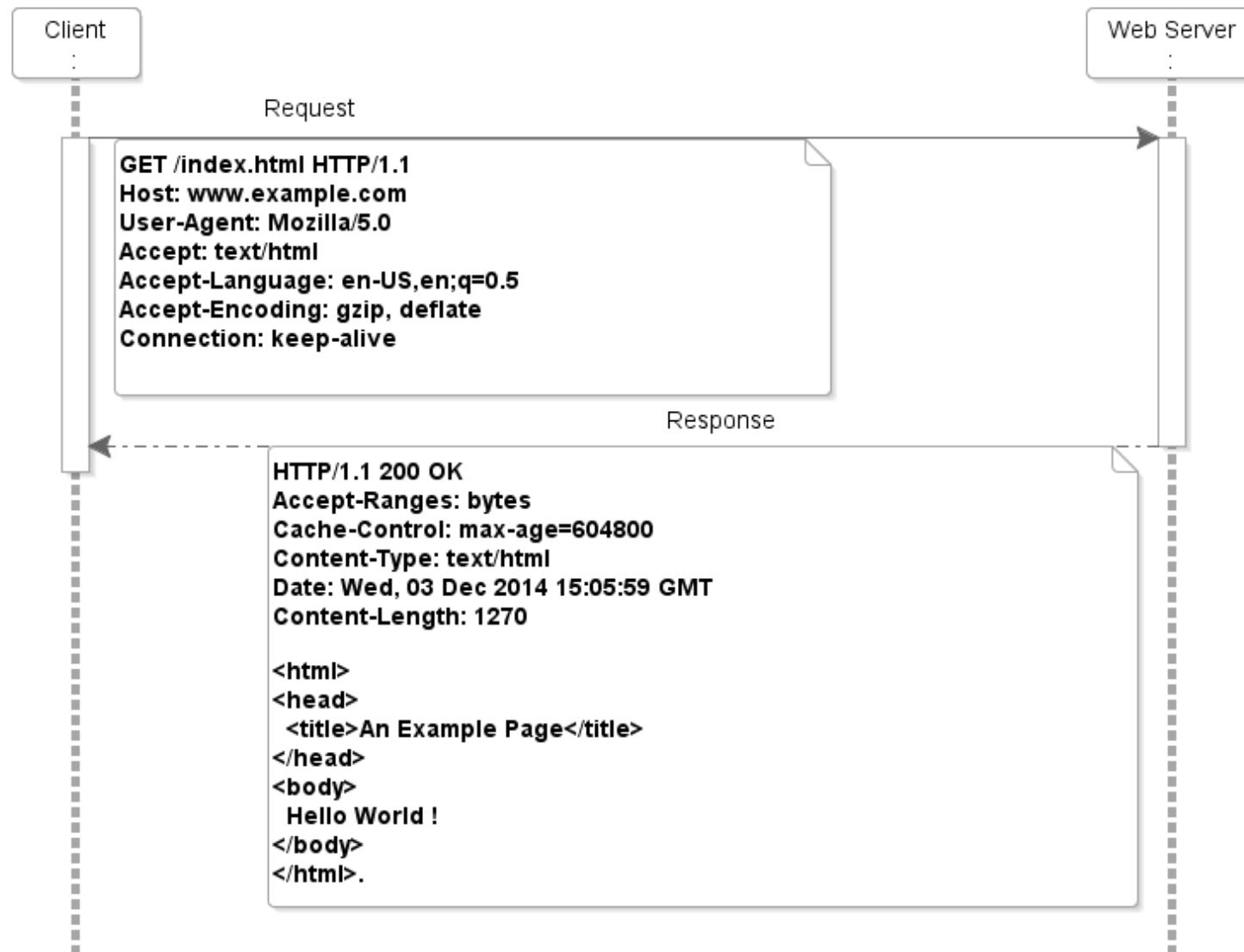
- Client-server 기반 아키텍처
- HTTP 전송규약 사용
- Stateless
- Cacheable
- Layered system
- Representation of resources
 - 📌 URI 와 리소스 (data) 가 매핑 됨
- 구현의 자유
 - 📌 다양한 방식으로 구현될 수 있는 아키텍처 스타일

HTTP - 전송 규약으로 사용

- 인터넷 상에서 데이터를 주고 받기 위한 서버/클라이언트 구조의 프로토콜



HTTP request/response 모델



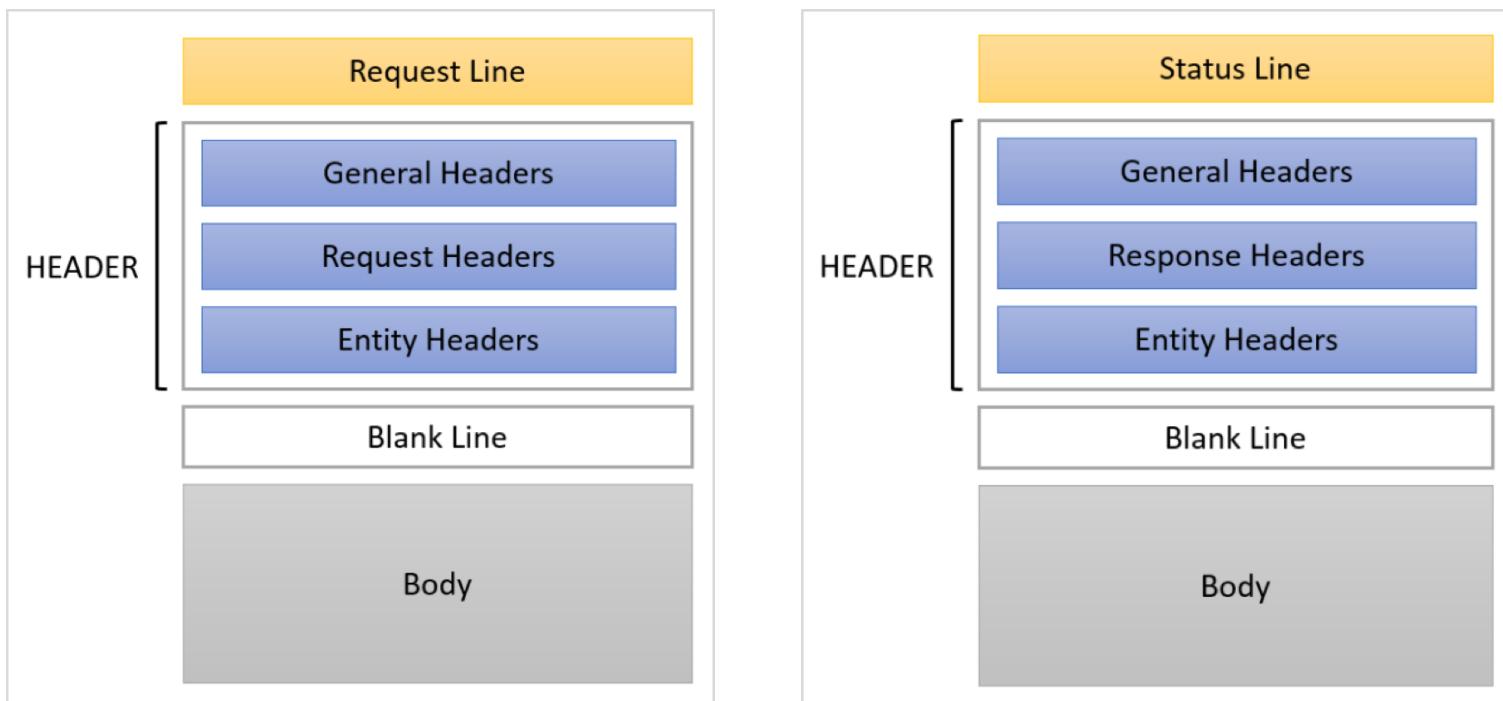
HTTP 메세지

헤더와 바디로 구성

Start Line	Method	공백	URL	공백	프로토콜버전	CR	LF
Header	Header Field Name	...	Value	CR	LF		
	Header Field Name	...	Value	CR	LF		
빈줄 (Blank Line)	CR	LF		
Body			Body				

HTTP 메세지

요청 메세지와 응답 메세지

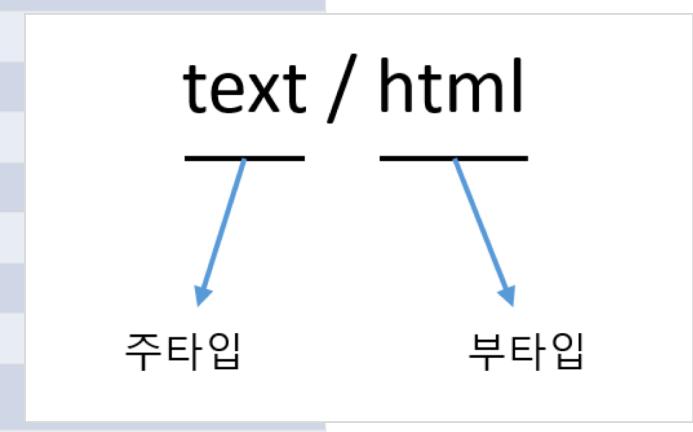


컨텐트 타입 (MIME 타입) 표현 - HTTP 헤더에 포함

- 인터넷 상에는 수 천 가지 종류의 컨텐츠가 존재

- 이러한 리소스에 효과적으로 접근하기 위해 컨텐츠에 대한 명세가 필요

타입/서브타입	의미
text/plain	플레인 텍스트
text/csv	CSV 형식 텍스트 (Comma Separated Values)
text/css	CSS 형식의 스타일시트
text/html	HTML 문서
text/xml	XML 문서(비추천)
image/jpeg	JPEG 이미지
image/png	PNG 이미지
application/xml	XML 문서
application/xhtml+xml	XHTML 문서
application/javascript	Javascript 파일
application/json	JSON 문서
application/msword	Word
application/vnd.ms-excel	Excel
application/pdf	PDF 파일
application/zip	ZIP 파일
application/x-shockwave-flash	Flash 오브젝트
application/x-www-form-urlencoded	HTML 폼형식



HTTP Methods

- ☑ 클라이언트 요청의 종류 - 서버는 이 정보를 통해 적당한 응답을 제공

Method	Description
GET	자원 요청
POST	Entity를 포함한 자원 요청
HEAD	HTTP Header 정보만 수신
TRACE	Request의 루프백 테스트
PUT	URL에 자원을 생성
DELETE	URL의 자원을 삭제
OPTIONS	응답 가능한 HTTP 메소드를 요청
CONNECT	터널링의 목적으로 연결 요청 (프록시에서 사용함)

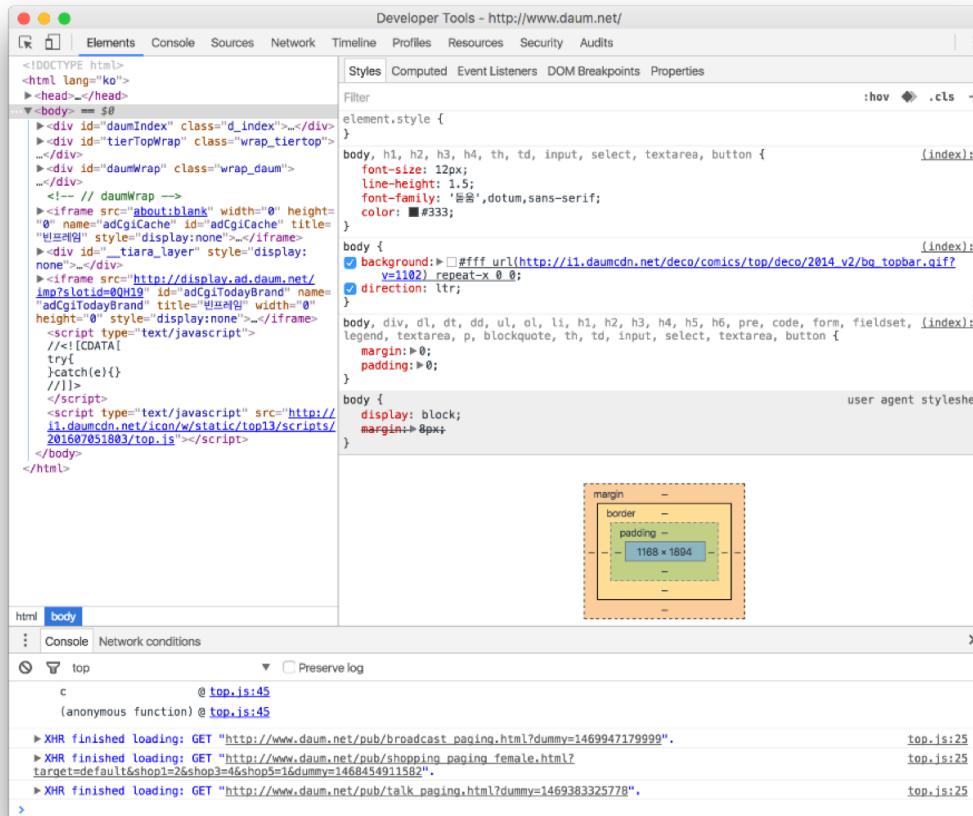
HTTP Status code

응답 상태 코드 타입

상태코드 타입	코드	내용
성공	200 - 226	응답이 성공적임
에러	400 - 499 (client) 500 - 599 (server)	400번 대 에러는 요청 에러 500번 대 에러는 서버 에러
리다이렉트	300 - 308	URL 리다이렉션에서 사용

브라우저 개발자 툴을 이용한 메세지 확인

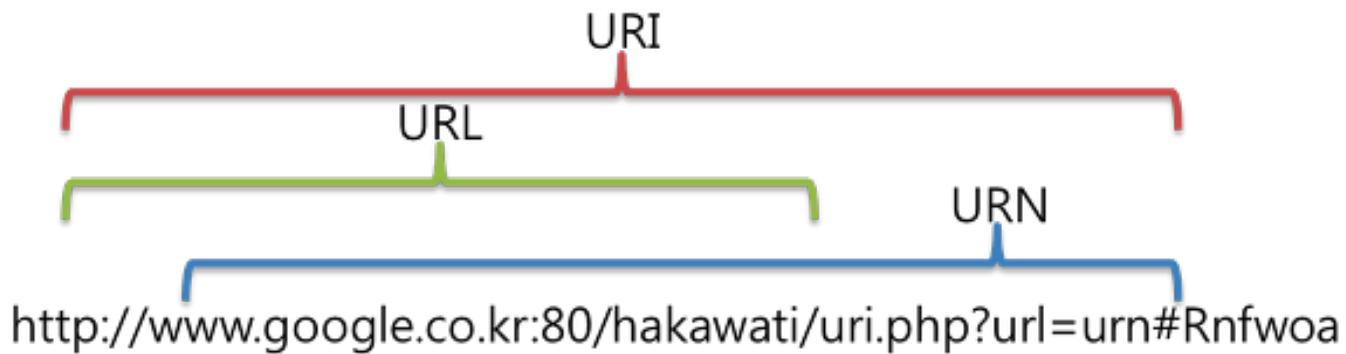
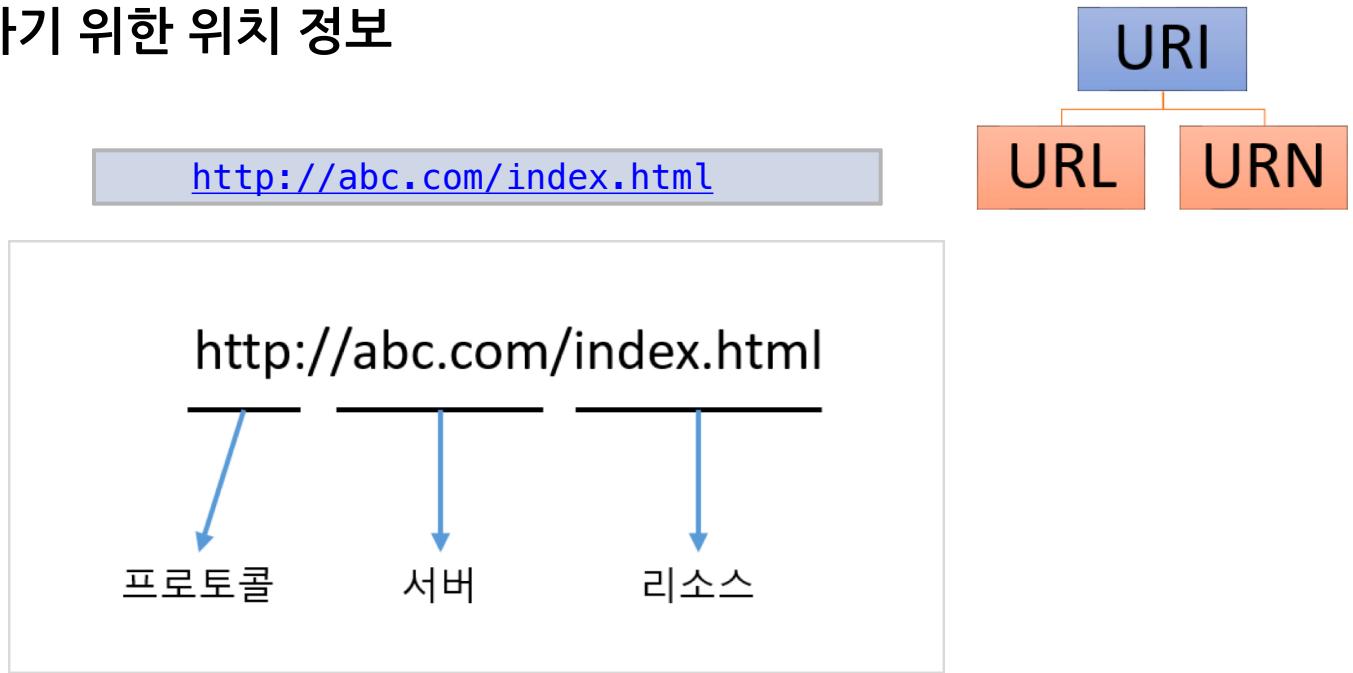
- ☑ 대부분의 브라우저는 개발자를 위해 HTTP 메세지 전송에 대한 정보 제공



Elements Console Sources Network Timeline Profiles Resources Security Audits

URI - 식별자로 사용

컨텐츠에 접근하기 위한 위치 정보



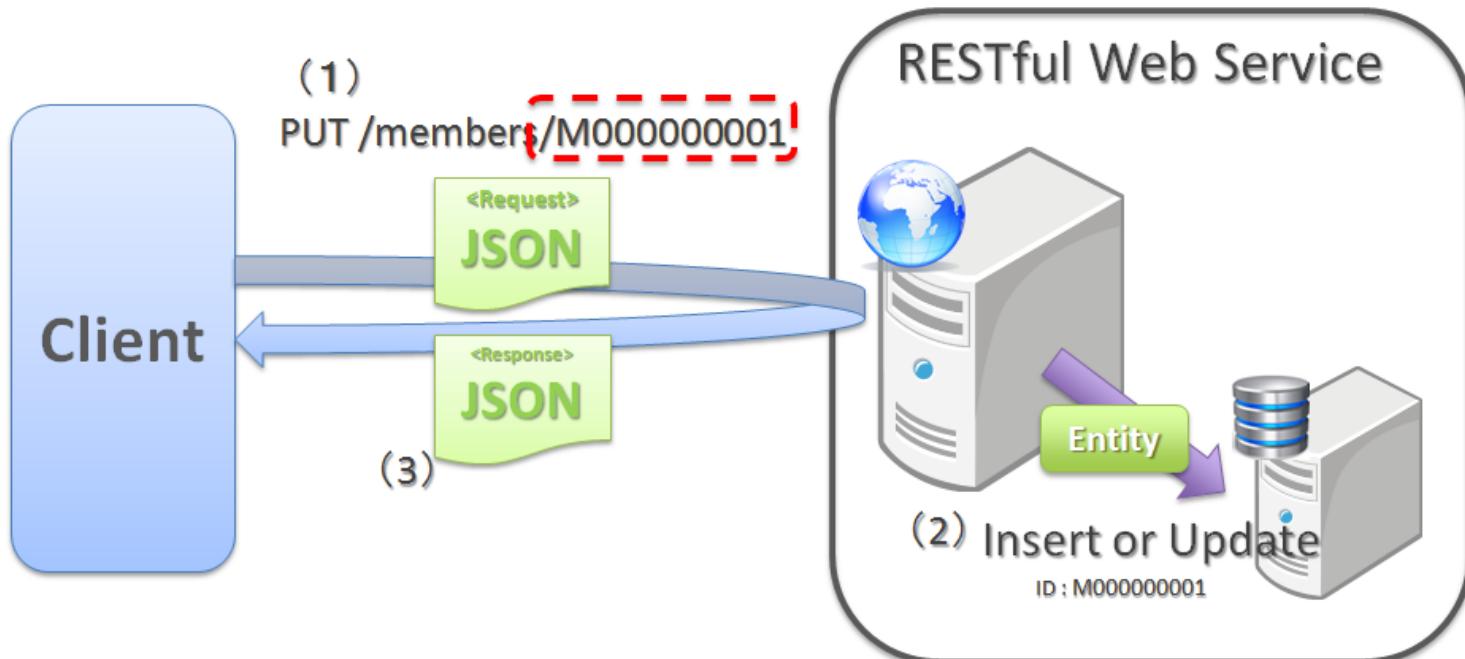
RESTful API

개요

RESTful 기반 웹 서비스

Roy Fielding 박사학위 논문에서 제안

- WEB 아키텍처가 WEB의 본래 설계의 우수성을 활용하지 못하므로 WEB의 장점을 최대한 활용할 수 있는 네트워크 기반의 아키텍처를 제안



데이터는 왕

- ☑ 소프트웨어는 대체되어도 수년간 쌓인 데이터는 대체할 수 없다

- ⌚ SOAP : 동작과 프로세싱에 집중
- ⌚ REST : 관심은 데이터 처리

스프링 3.0 부터 REST 작업에 대한 최고 수준의 지원을 제공 3.1, 3.2 4.0 에도 계속 됨

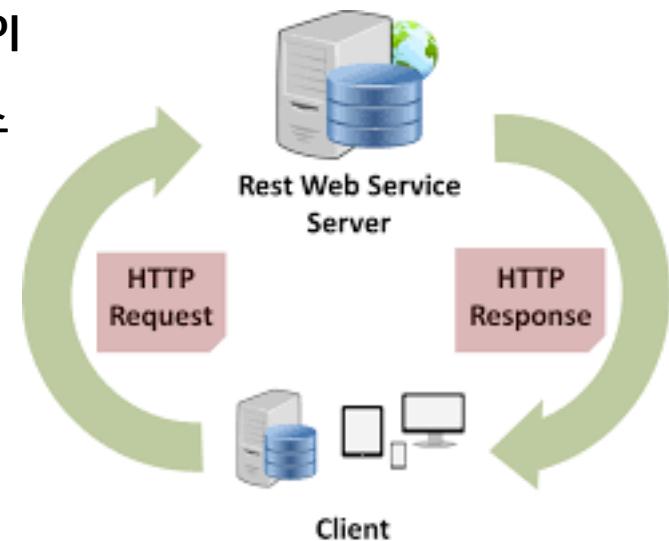
RESTful 기반 웹 서비스

RESTful 웹 서비스 (Representational State Transfer, 2000년)

- HTTP 프로토콜로 데이터를 전달하는 프레임워크
- 핵심은 웹에 개발된 리소스 이용

REST 아키텍처 스타일에 따라 정의되고 이용됨

- REST API - 소유자의 **자원**에 접근할 수 있는 API
- RESTful 하다 - REST API를 제공하는 웹 서비스

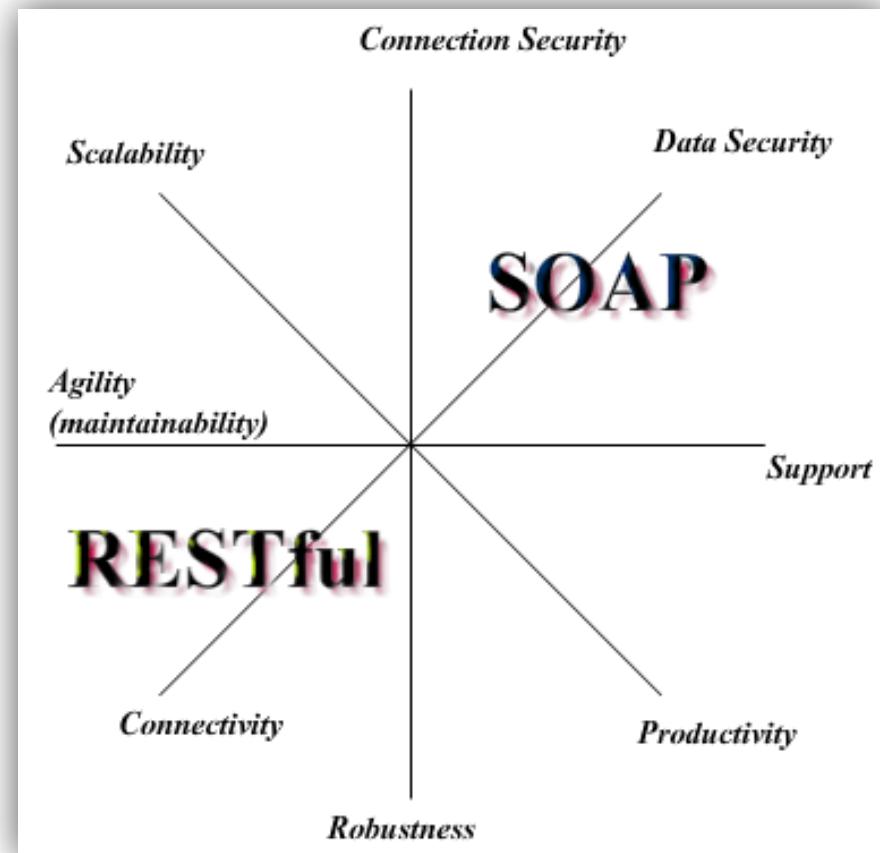


Getting REST

SOAP vs. REST

- SOAP - 많은 어플리케이션에게 부담
- REST - 더 간편한 대안 제공

REST에 대한 정확한 이해가 필요



REST 란 무엇인가?

- 2000년에 HTTP 프로토콜 저자 중 한명인 Roy Fielding 박사 논문에 소개
- 기존 웹 서비스들이 HTTP를 적극적으로 활용하지 못한 문제를 해결하기 위해 제안됨
- HTTP를 보다 HTTP답게 만들기 위한 방법론으로 볼수 있음
- 철저히 Resource 중심적 설계를 중요시하고
 - ⌚ Create - POST
 - ⌚ Read - GET
 - ⌚ Update - PUT
 - ⌚ Delete - DELETE 와 같이 HTTP 4가지 메소드를 용도에 맞게 사용
- 오늘날 Google, Amazon, Flickr, Tweeter와 같은 Open Source 업체들의 주도 하에 적극 적용되어, De-facto Standard로 자리잡음

REST 원리 및 원칙 (제약조건)

REST

- 확장성있는 웹 서비스를 위한 소프트웨어 아키텍처 적인 접근

- Client-Server

- Stateless

- 클라이언트의 상태를 서버에 저장하지 않음 (확장성)

- Cacheable

- Layered system

- Code on demand (optional)

- Uniform interface

- REST의 코어인 리소스는 URI에 의해 식별되어짐

- 데이터 구조(data representation)는 강제하지 않음 (주로 XML 또는 JSON)



REST API 사례 - 카카오 API

<https://developers.kakao.com/>

The screenshot shows the Kakao Developers homepage with a dark blue background. At the top left is the "kakao developers" logo. On the right, there's a navigation bar with links for "내 애플리케이션", "제품", "문서", "도구", "포럼", and a user account link "soongon@miramnsnc.com". A large teal circle is positioned on the right side of the header.

The main banner features a large teal circle on the left containing the text "Build with Kakao Developers" and a "시작하기" button. To the right of the banner are three smaller graphic elements: a grid of blue dots, a network graph of purple dots connected by lines, and a large teal circle.

Below the banner, the page is divided into sections. The first section is titled "추천 제품" (Recommended Products) and contains four cards:

- 카카오 로그인**: Includes a "Login" button and a brief description: "카카오계정 하나로 간편하게 여러분의 서비스에 로그인 할 수 있도록 하는 서비스입니다. 누구나 가지고 있는 카카오 계정으로 빠르고 안전하게 고객을 만들어 보세요." Buttons for "제품소개" and "문서보기".
- 친구**: Includes a user icon and a brief description: "카카오톡 친구들의 닉네임, 사진 정보를 활용할 수 있습니다. 고객에게 개인화된 서비스를 제공하세요." Buttons for "제품소개" and "문서보기".
- 메시지**: Includes a message icon and a brief description: "사용자들이 카카오톡으로 여러분의 콘텐츠를 공유할 수 있습니다. 카카오링크와 서비스 내에서 앱 전환 없는 메시지 API를 활용하여 여러분의 콘텐츠를 넓리 확산해보세요." Buttons for "제품소개" and "문서보기".
- 지도/로컬**: Includes a location pin icon and a brief description: "언제나 최신 정보를 제공하는 카카오맵의 API를 통해 지도, 위치, 장소와 관련된 정보를 다양한 위치 기반 서비스에 활용해 보세요." Buttons for "제품소개" and "문서보기".

At the bottom, there are two additional dark blue bars with white text: "카카오로그인", "카카오톡", "카카오", and "카카오 지식나라".

REST API 사례

네이버 API

☞ <https://developers.naver.com>

공공 데이터 포털 (정부 공공 API)

☞ <https://www.data.go.kr/>

구글 API

☞ <https://developers.google.com/>

API 마켓플레이스

☞ <https://rapidapi.com/>

마블 API

☞ <https://developer.marvel.com/>

뉴스 API

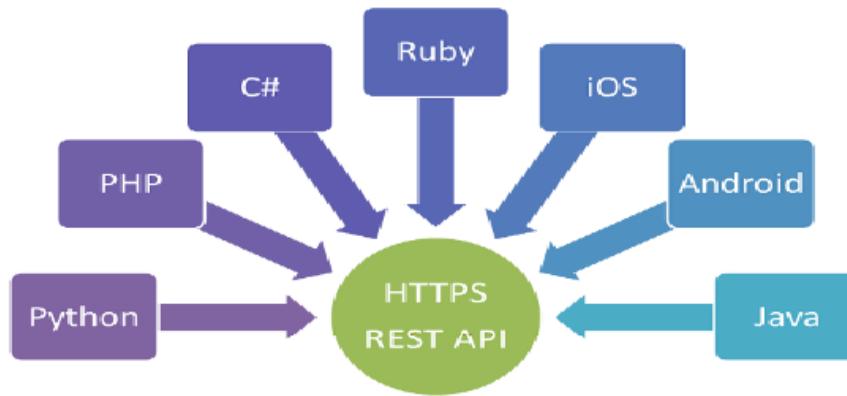
☞ <https://newsapi.org/>

날씨 API

☞ <https://openweathermap.org/>

Uniform Interface

- ✓ REST 디자인 constraints에서 가장 기본적인 제약
- ✓ 통일된 인터페이스를 통해 기술과 플랫폼에 상관없이 사용 가능



URL

```
GET /v2/search/web HTTP/1.1
Host: dapi.kakao.com
Authorization: KakaoAK {app_key}
```

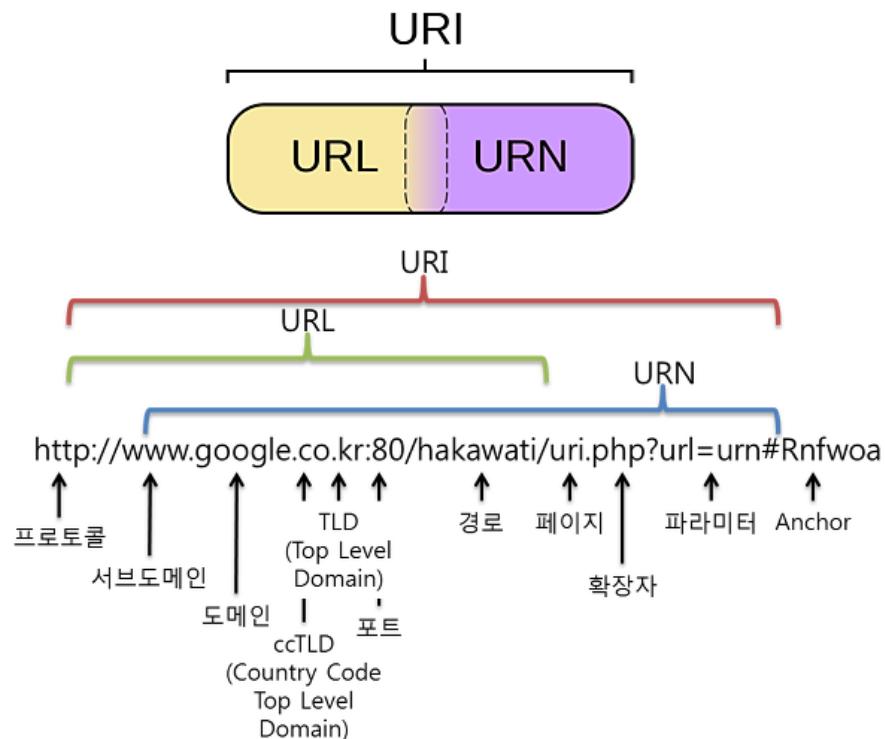
Uniform Interface - principles

- ☑ Uniform Interface 는 다음 4가지로 설명 되어짐
 - ⌚ Resource-based
 - ⌚ Representations 으로 resources 를 처리
 - ⌚ Self-Descriptive Message
 - ⌚ HATEOAS (Hypermedia As The Engine Of Application State)

Uniform Interface - 첫번째 원칙 : Resource-based

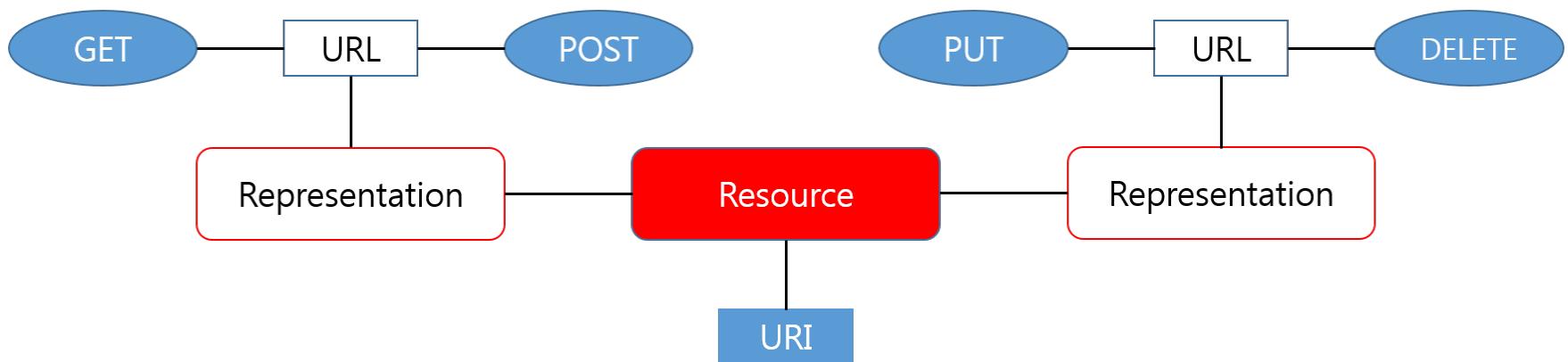
☒ 리소스는 URI 정의로 설계되어야 함

- URI 는 리소스를 unique 하게 해줌
- Resource 는 서비스나 정보를 제공하는 모든 것들이 될 수 있음
 - 주문, 송장, DB의 레코드, 검색 결과 등
 - 웹 상에서 주소를 통해 접근 될수 있음. 즉, 리소스는 URI를 반드시 가져야 함
 - URI 정의는 동사형이 아닌 명사형으로 정의하도록 권장됨



Uniform Interface - 두번째 원칙 : Representations

- Resource는 Unique ID로 하나 이상의 URI 뿐만 아니라, 다양한 방법으로 설명(대표)되는 Representation을 가질 수 있다.
 - Representation에 대한 접근을 위한 URL을 가지고
 - HTTP 4가지 메서드(POST/GET/PUT/DELETE)를 통해 CRUD 할 수 있다.



Uniform Interface - 두번째 원칙 : Representations (계속)

- ✓ 서버가 보내준 것은 Resource 가 아니라 Representation

```
GET https://example.org/greeting
Host: example.org
Accept: text/plain, text/html; q=0.9 *; q=0.1
Accept-Language: en, ko; q=0.9, *; q=0.1
```

```
HTTP/1.1 200 OK
Content-Length: 6
Date: Sun, 19 Mar 2017 10:20:47 GMT
Last-Modified: Sun, 19 Mar 2017 08:00:00 GMT
Content-Type: text/plain
Content-Language: en

hello
```

HTTP GET 요청과 응답

Uniform Interface - HTTP 에서의 Representation

GET 메서드 정의

The GET method requests transfer of a current selected representation for the target resource.

- Target resource에 대한 현재의 선택된 representation 하나를 반환한다.
- 리소스는 HTTP 요청의 대상이지만, 리소스의 개념을 제한하지 않는다.

Representation 이란?

- 어떤 리소스의 특정 시점의 상태를 반영하고 있는 정보
- 다음 두 가지로 구성됨
 - Representation data
 - Representation metadata

Uniform Interface - Representation 사례

영어 사용자와 한국어 사용자를 위한 representation

Content-Type: text/plain

Content-Language: en

hello

Content-Type: text/plain

Content-Language: ko

안녕하세요

Content-Type: text/html; charset=UTF-8

Content-Language: en

<html><body>hello</body></html>

Content-Type: text/html; charset=UTF-8

Content-Language: ko

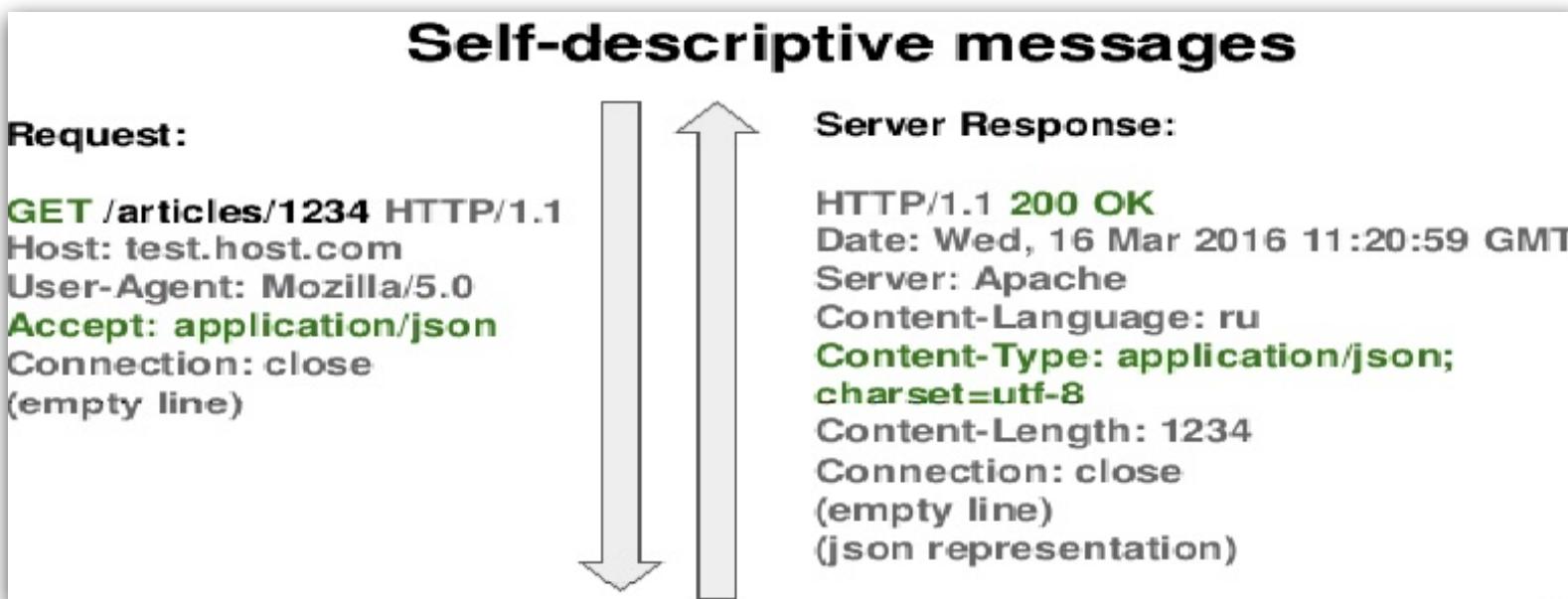
<html><body>안녕하세요</body></html>

- 클라이언트와 서버 간의 내용 협상(Content negotiation)을 통해 선택됨
- 일반적으로 사전 협상(proactive negotiation)에 의해 서버가 선택
- 적절한 representation이 존재하지 않는다면 406 Not Acceptable로 응답
- URI 가 동일하다면 같은 리소스임

Uniform Interface - 세번째 원칙 : Self-Descriptive message & API

REST 는 Stateless

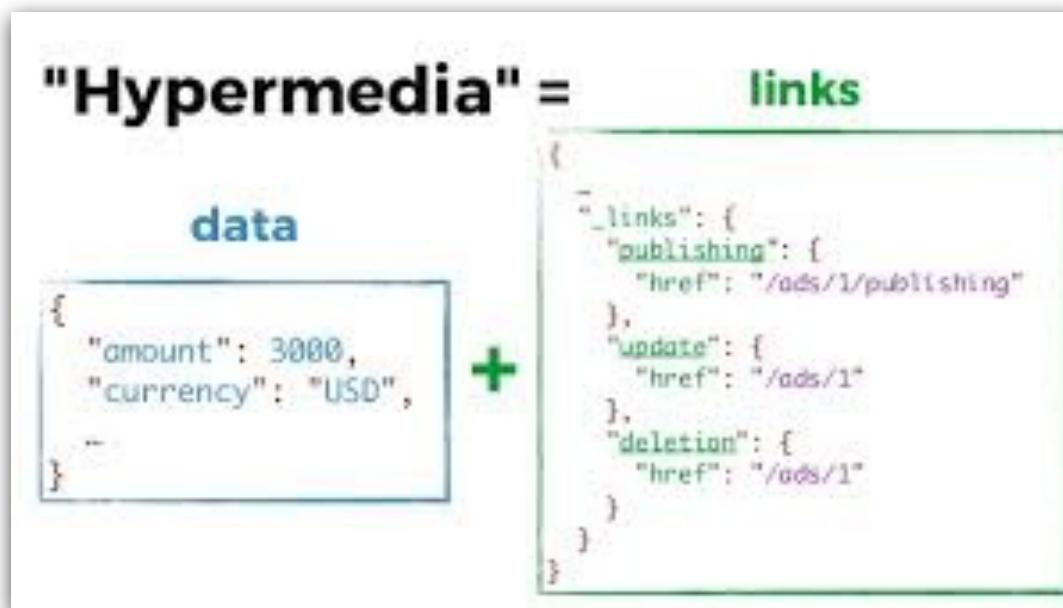
- 클라이언트 - 서버 간 충분한 설명적인 메세지가 필요
 - HTTP 메서드
 - HTTP 상태 코드
 - HTTP 헤더



Uniform Interface - 네번째 원칙 : HATEOAS

Hypermedia As The Engine Of Application State

- HATEOAS 메세지를 보내는 것은 어플리케이션의 상태를 변화 시킴
- HTTP 응답에 다음 Action이나 관계되는 리소스에 대한 HTTP Link 를 함께 리턴
 - 페이지 처리의 경우 리턴시, 전후 페이지에 대한 링크를 제공
 - 연관된 리소스에 대한 디테일한 링크를 표시



Uniform Interface - 네번째 원칙 : HATEOAS

Without HATEOAS:

```
1 Request:  
2 [Headers]  
3 user: jim  
4 roles: USER  
5 GET: /items/1234  
6 Response:  
7 HTTP 1.1 200  
8 {  
9   "id" : 1234,  
10  "description" : "FooBar TV",  
11  "image" : "fooBarTv.jpg",  
12  "price" : 50.00,  
13  "owner" : "jim"  
14 }
```

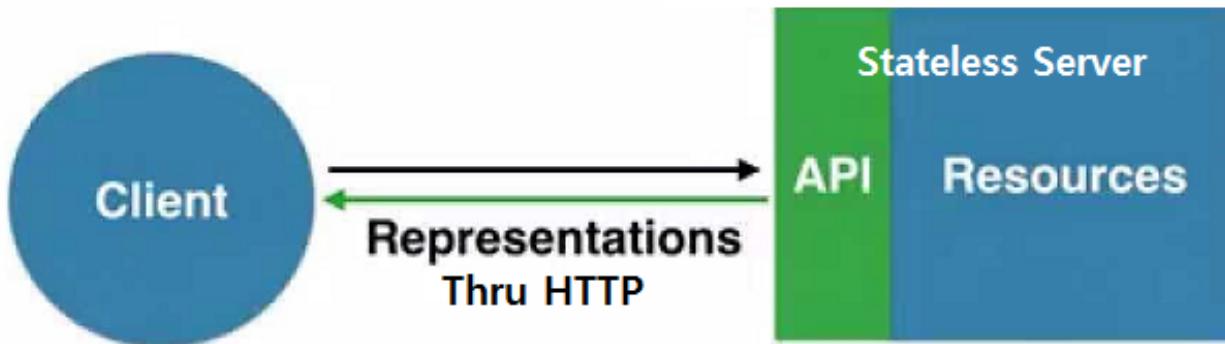
With HATEOAS:

```
1 Request:  
2 [Headers]  
3 user: jim  
4 roles: USER  
5 GET: /items/1234  
6 Response:  
7 HTTP 1.1 200  
8 {  
9   "id" : 1234,  
10  "description" : "FooBar TV",  
11  "image" : "fooBarTv.jpg",  
12  "price" : 50.00,  
13  "links" : [  
14    {  
15      "rel" : "modify",  
16      "href" : "/items/1234"  
17    },  
18    {  
19      "rel" : "delete",  
20      "href" : "/items/1234"  
21    }  
22  ]  
23 }  
24 }
```

Client-Server

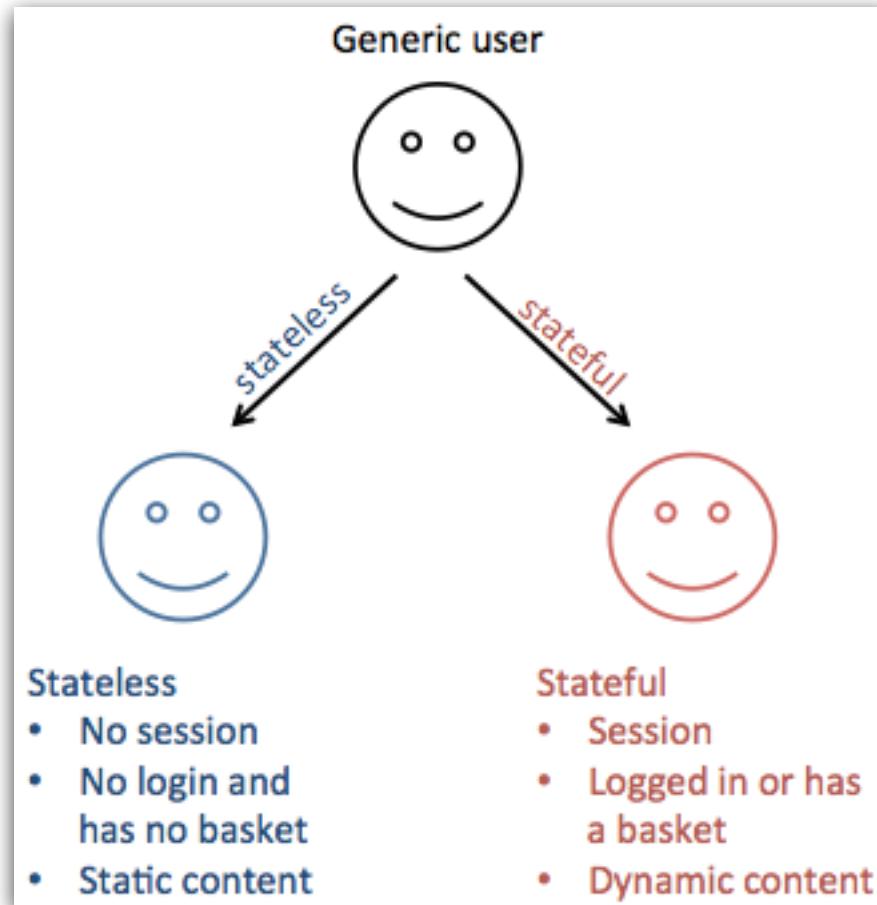
Client 를 Server 로 부터 분리

- ⌚ View 와 Data를 분리
- ⌚ Portability(이동성) 를 향상
- ⌚ REST 서버는 Resource 를 관리하는 API 를 제공
 - 클라이언트는 사용자 인증이나 Context(Session, Login 정보) 직접 관리
 - 서버는 UI나 User State에 상관없이 Server 기능에 집중 (단순화, 확장성)



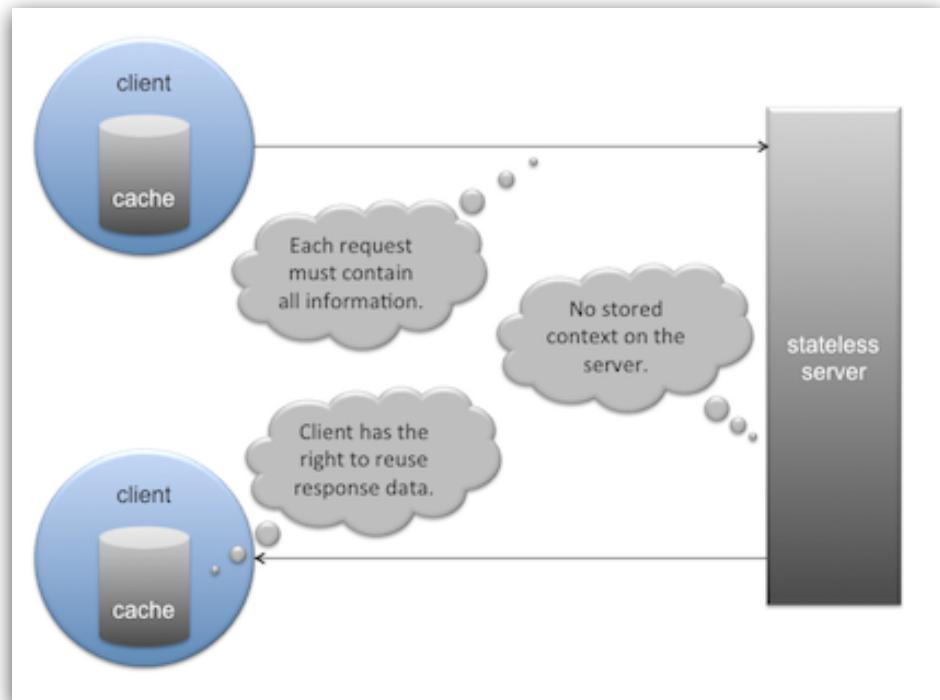
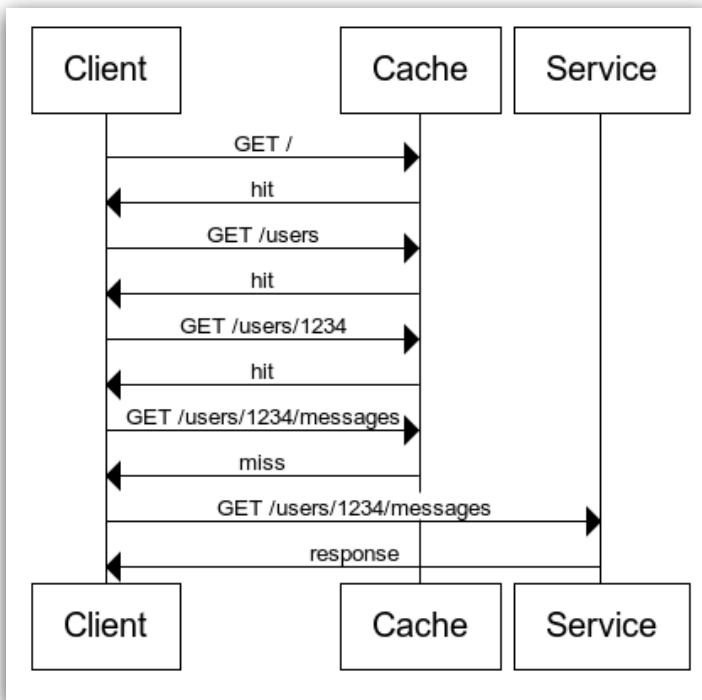
Stateless

- 세션정보와 같은 context를 저장할 필요가 없음
- 클라이언트는 자신을 구분하기 위해 서버에게 충분한 정보를 전달해야 함
 - API Key 또는 Token



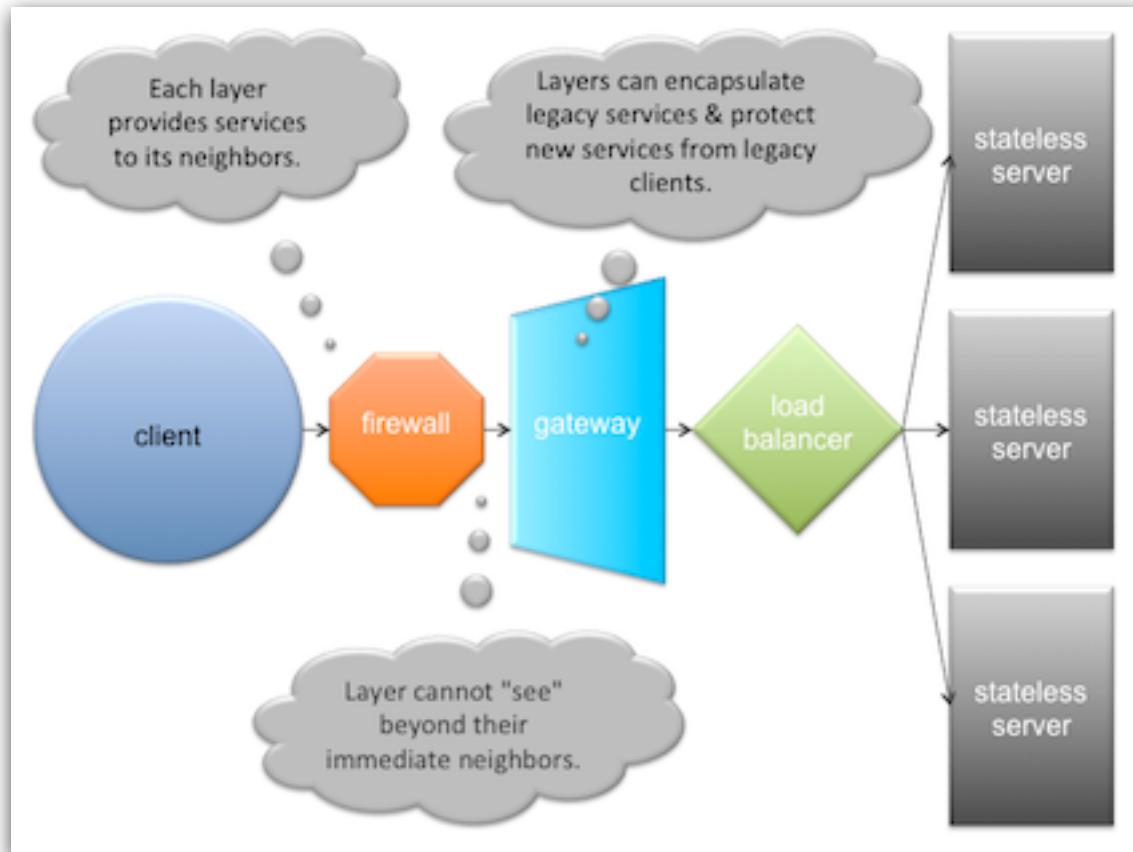
Cacheable

- HTTP 프로토콜의 Caching 기능을 적용할 수 있음
 - 웹 서비스의 60~80% 가 GET 방식의 요청
 - 구현은 HTTP 프로토콜 표준인 “Last-Modified” 태그나 E-Tag 를 이용



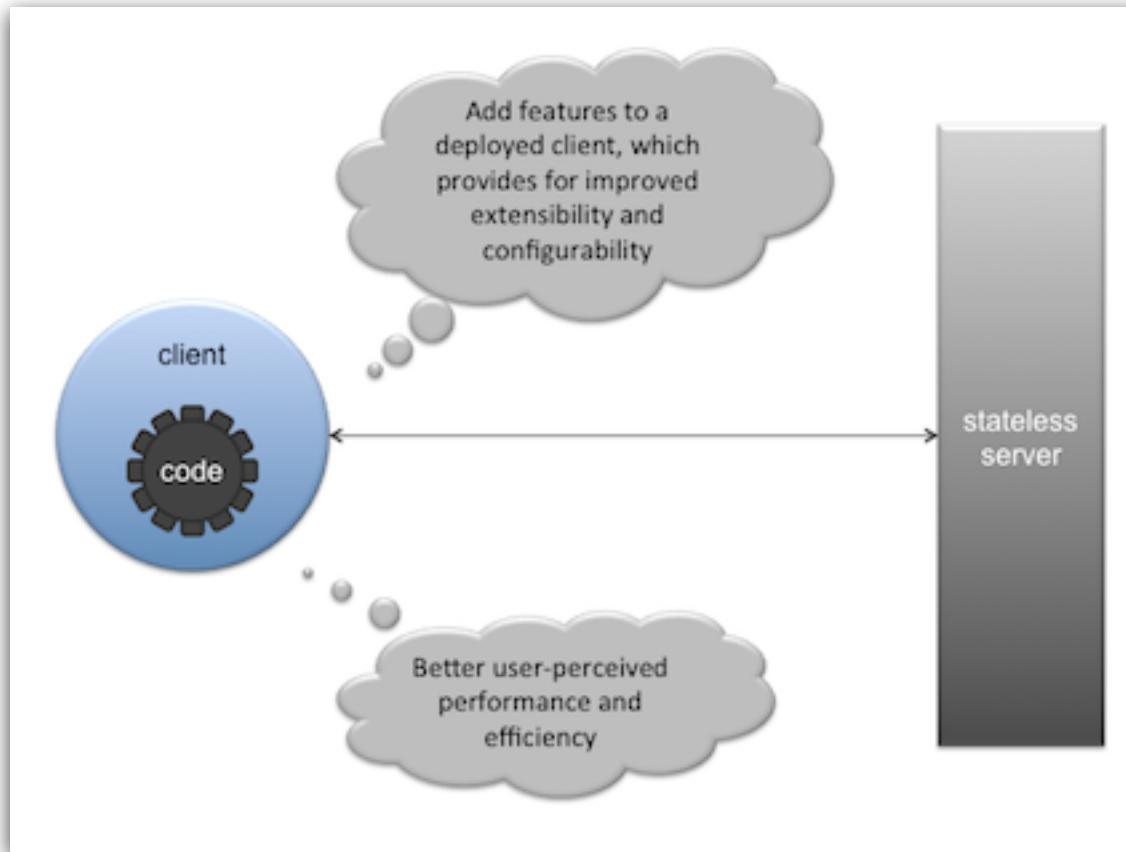
Layered System

- ✓ 시스템 스케일링을 위해 Gateway, Proxy Server, Firewall 과 같은 또 다른 Layer 들을 수용할 수 있음



Code on Demand (Optional)

- 서버는 실행 가능한 코드를 클라이언트에게 전송
 - 클라이언트의 기능을 일시적으로 확장하거나 커스터마이징 가능
 - Java Applet, JavaScript 가 대표적 예



Richardson maturity model

Level 3: - HATEOAS

Level 2: HTTP verbs

Level 1: Resources

Level 0: The Swamp of Plain Old XML

참고 자료

마이크로소프트 API 개발 가이드라인

- ☞ <https://docs.microsoft.com/ko-kr/azure/architecture/best-practices/api-design>

스프링 부트 - RESTful API 구현

Layered Architecture

Layered Architecure - 특징

계층화 아키텍처

- 🔊 효율적인 개발과 유지보수를 위해 계층화 하여 개발
- 🔊 대부분의 중/대규모 어플리케이션에서 적용
- 🔊 각 레이어는 독립된 R&R 을 가짐

Layered Architecture - 특징

프레젠테이션 영역

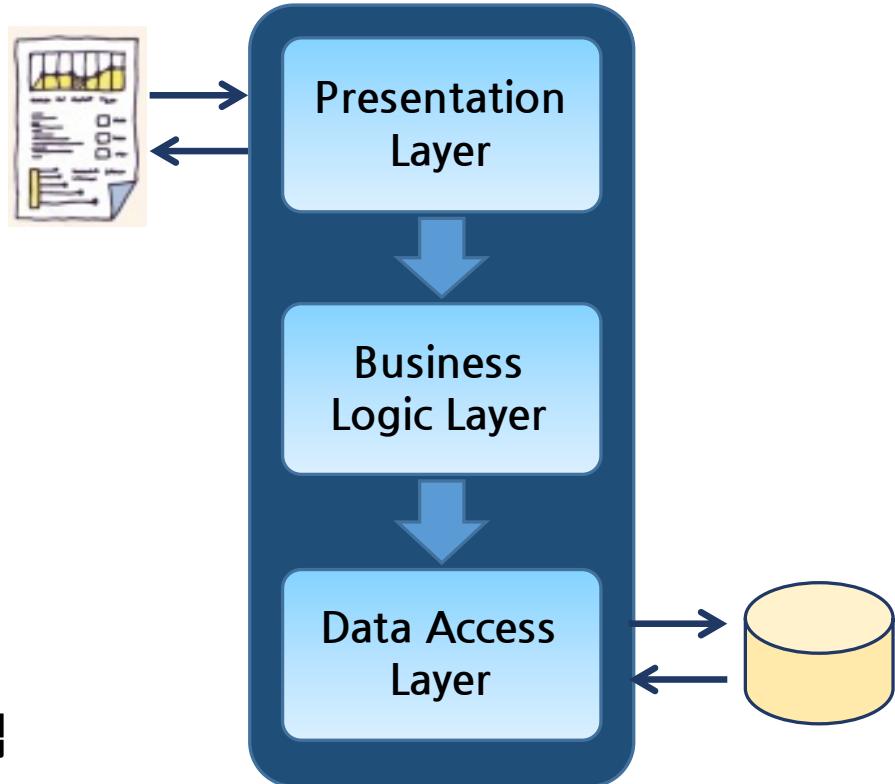
- 사용자와 상호작용을 담당
- 사용자의 요청을 분석/응답

비즈니스 영역

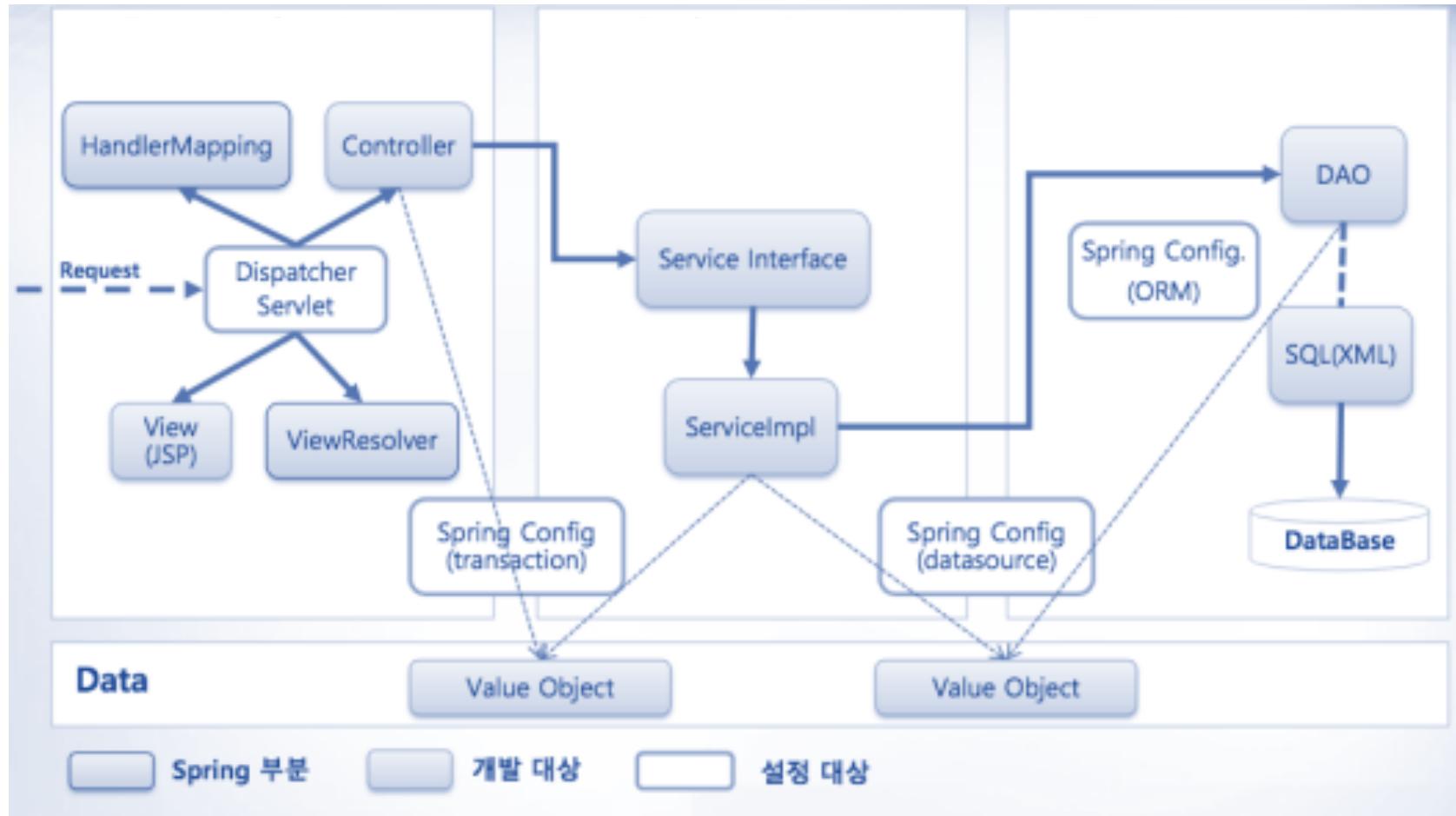
- 기능을 수행
- 트랜잭션 수행

데이터 영역

- 데이터의 저장과 조회를 담당
- 주로 데이터베이스와 연동하여 작업



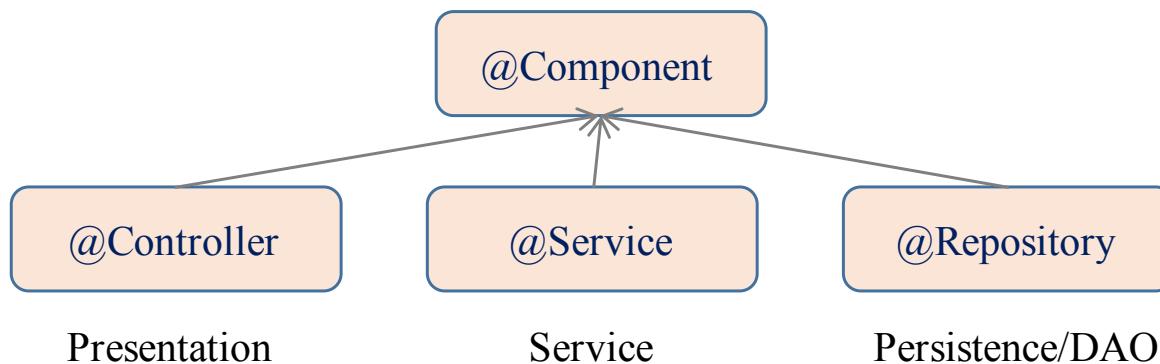
사례 : 전자정부 프레임워크



컴포넌트 자동등록

어노테이션(Annotation) 사용

- `@Component` 와 하위 어노테이션 사용



• `@Autowired`

- Component 간의 의존관계는 `Autowired`라는 Annotation 으로 적용

RESTful API 패턴 - part 1

RESTful API 디자인 패턴

Part 1

REST 서비스 디자인 패턴 :: Part 1

- REST 요청/응답 패턴
- Statelessness
- Content negotiation
- URI 템플리트
- 페이징 처리
- 유니코드

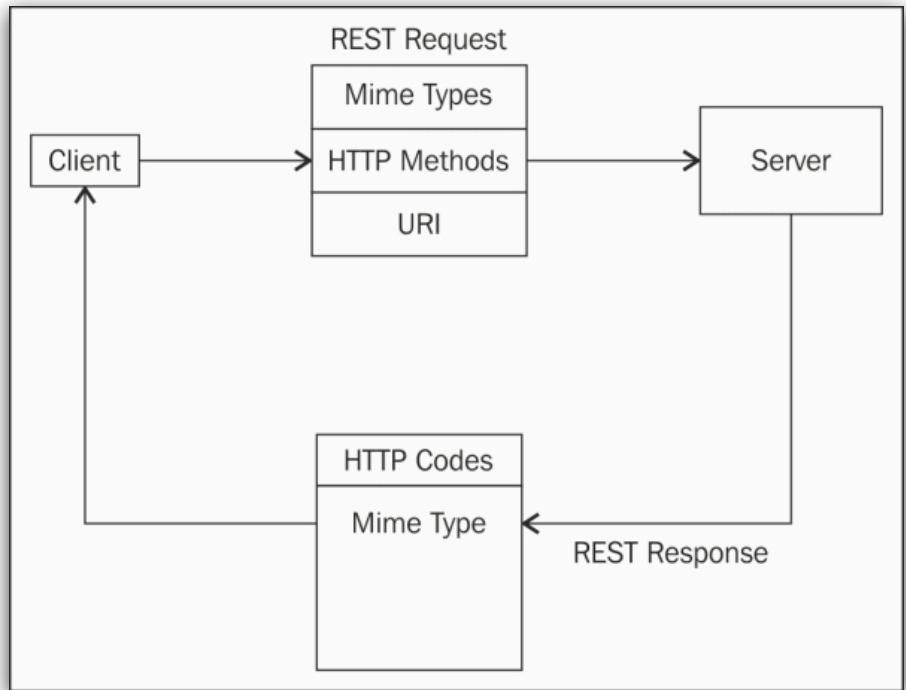
REST 요청/응답 패턴

클라이언트 REST 요청 생성

- 표준 HTTP 메서드
- Mime 타입
- URI

서버가 요청을 처리 응답 수행

- 표준 HTTP 응답 코드
- Mime 타입



스프링 부트 요청/응답 코드 실습

Statelessness

- ☑ 서버와 호출하는 클라이언트 간 정보(state)를 저장하지 않음
 - ⌚ 서버 세션을 사용하지 않음
 - ⌚ 이전 거래 정보나 현재 컨텍스트 저장하지 않음
- ☑ 클라이언트는 매번 호출 시 서버에게 충분한 정보를 제공해야 함
- ☑ 세션 상태는 클라이언트에서 전적으로 유지해야 함
- ☑ 서버는 클라이언트의 인증 정보도 저장해서는 안됨
 - ⌚ 클라이언트는 매번 요청마다 인증정보 포함
- ☑ 모든 요청/응답은 독립적임!

Statelessness :: 스프링 부트 코드

“상태없음” 을 유지하게 코드를 작성하려면

- URI에서 정보를 획득, 다른 특별한 구현을 하지 않는다. (세션 처리 등..)

아래 코드는 특정 투자자의 주식 목록을 요청하는 코드

```
@GetMapping(path = "/investors/{investorId}/stocks")  
public List<Stock> fetchStocksByInvestorId(@PathVariable String investorId,  
    @RequestParam(value="offset", defaultValue="0") int offset,  
    @RequestParam(value="limit", defaultValue="5") int limit) {  
    return investorService.fetchStocksByInvestorId(investorId, offset, limit);  
}
```

Statelessness 장점

확장성, Scalability

- 다중 서버에 디플로이 시 세션을 관리 하지 않아도 됨

복잡성이 줄어듬

- 서버 사이드 상태 동기화가 필요없음

캐시 사용 가능, 성능 증대

- 요청이 같으면 응답도 같다.

추적성 (Traceability)

- 클라이언트 요청은 매번 필요한 모든 정보를 포함한다.

HTTP/HTTPS 에 적합

- HTTP/HTTPS 프로토콜은 stateless, REST 구현은 HTTP를 적극적으로 이용

Content negotiation

- 다른 클라이언트는 다른 representations 을 요청
 - XML or JSON ..
- REST 서비스는 HTTP 헤더를 사용해 구현됨
 - 클라이언트 Accept 헤더로 원하는 형태의 리소스를 요청
 - 서버에서 결정하여 응답해 줌
- <http://www.w3.org/StyleSheets/TR/logo-REC>

The screenshot shows a browser's developer tools Network tab with the following details:

- Name:** logo-REC
- Headers** tab is selected.
- Request Headers:**
 - :authority: www.w3.org
 - :method: GET
 - :path: /StyleSheets/TR/logo-REC
 - :scheme: https
 - accept: text/html, application/xhtml+xml, application/xml;q=0.9, image/webp, image/apng, */*;q=0.8
 - accept-encoding: gzip, deflate, br
- Response Headers:**
 - access-control-allow-origin: *
 - cache-control: max-age=2592000
 - content-location: logo-REC.png
 - content-security-policy: upgrade-insecure-requests
 - content-type: image/png; qs=0.7

At the bottom of the Network tab, it says 1 / 2 requests | 479 B / 479 B tran.

Content negotiation :: HTTP 헤더를 이용한 사례

✓ Postman에서 헤더 세팅

REST-API-example-1- (●) REST-API-Example-2-I X + ... No Environment

▶ REST-API-Example-2-POST-NewStock

POST http://localhost:9090/investors/invr_1/stocks Params Send

Headers (2)

Key	Value	Description	...	Bulk Edit
Accept	application/json			
Content-Type	application/json			
New key	Value	Description		

✓ XML 응답하기 위한 라이브러리 추가

```
<dependency>
    <groupId>com.fasterxml.jackson.dataformat</groupId>
    <artifactId>jackson-dataformat-xml</artifactId>
</dependency>
```

Content negotiation :: HTTP 헤더를 이용한 사례 (계속)

✓ 컨트롤러에서 @GetMapping 설정

```
@GetMapping(path="/investors/{investorId}/stocks/{symbol}",  
produces={MediaType.APPLICATION_JSON_VALUE, MediaType.APPLICATION_XML_VALUE})
```

The image shows two side-by-side Postman requests for the URL `http://localhost:9090/investors/invr_1/stocks/exa`.

Left Request (XML Response):

- Method: GET
- Headers: `Accept: application/xml`
- Body: XML response (highlighted with an orange box)

```
1 <Stock>  
2   <symbol>EXA</symbol>  
3   <numberOfSharesHeld>200</numberOfSharesHeld>  
4   <tickerPrice>20.0</tickerPrice>  
5 </Stock>
```
- Preview tab shows the XML structure.

Right Request (JSON Response):

- Method: GET
- Headers: `Accept: application/json`
- Body: JSON response (highlighted with an orange box)

```
1 {  
2   "symbol": "EXA",  
3   "numberOfSharesHeld": 200,  
4   "tickerPrice": 20  
5 }
```
- Preview tab shows the JSON object.

Content negotiation :: 다른 방식 두 가지

- 리소스 URI 의 확장자를 사용하는 방식

<https://xxx.api.com/v1/students/course.xml>

<https://xxx.api.com/v1/students/courses.json>

- 파라미터를 활용하는 방식

<https://xxx.api.com/v1/students/course?format=xml>

<https://xxx.api.com/v1/students/courses?format=json>

Content negotiation :: 기타

클라이언트 우선순위

```
Accept:application/json,application/xml;q=0.9,*/*;q=0.8  
(json 또는 xml 지원, q -> 우선순위_생략시 1.0)
```

압축 지원

```
# Request      Accept-Encoding: gzip  
# Response     Content-Encoding: gzip Vary: Accept-Encoding
```

협상 실패 시

- Accept: application/json,*/*;q=0.0 (JSON 이외는 처리 않겠다)
- 서버는 406 Not Acceptable 전송

URI 템플리트

- URI 가 유사한 리소스를 식별하는 데 사용
- @PathVariable 어노테이션 사용

```
@GetMapping("/investors/{investorId}/stocks")  
@GetMapping("/investors/{investorId}/stocks/{symbol}")  
@DeleteMapping("/investors/{investorId}/stocks/{symbol}")
```

```
.....  
public ResponseEntity<Void> updateAStockOfTheInvestorPortfolio(  
    @PathVariable String investorId,  
    @PathVariable String symbol,  
    @RequestBody Stock stockTobeUpdated)  
{  
    Stock updatedStock =  
        investorService.updateAStockByInvestorIdAndStock(  
            investorId, symbol, stockTobeUpdated);  
    .....  
}
```

Pagination, 페이징 처리

- 대량의 목록 조회 시 페이지 단위로 데이터를 전송

- 성능에 영향 효과

- 페이지 정보는 URI 쿼리 파라미터로 전달

`xxx.api.com/stocks?page=2`

```
@GetMapping(path = "/investors/{investorId}/stocks")
public List<Stock> fetchStocksByInvestorId(
    @PathVariable String investorId,
    @RequestParam (value="offset", defaultValue="0") int offset,
    @RequestParam (value="limit", defaultValue="5") int limit) {

    return investorService.fetchStocksByInvestorId(investorId, offset, limit);
}
```

Pagination, 페이징 처리

✓ 결과 화면

The screenshot shows two parallel API requests in the Postman interface, demonstrating a pagination example.

Request 1 (Left):

- Method: GET
- URL: `http://localhost:9090/investors/invr_1/stocks?offset=0&limit=2`
- Status: 200
- Body (Pretty):

```
1 [  
2 {  
3   "symbol": "EXA",  
4   "numberOfSharesHeld": 200,  
5   "tickerPrice": 20  
6 },  
7 {  
8   "symbol": "EXB",  
9   "numberOfSharesHeld": 100,  
10  "tickerPrice": 60  
11 }  
12 ]
```

Request 2 (Right):

- Method: GET
- URL: `http://localhost:9090/investors/invr_1/stocks?offset=1&limit=5`
- Status: 200
- Body (Pretty):

```
1 [  
2 {  
3   "symbol": "EXB",  
4   "numberOfSharesHeld": 100,  
5   "tickerPrice": 60  
6 },  
7 {  
8   "symbol": "EX5",  
9   "numberOfSharesHeld": 200,  
10  "tickerPrice": 20  
11 },  
12 {  
13  "symbol": "EX6",  
14  "numberOfSharesHeld": 200,  
15  "tickerPrice": 20  
16 }  
17 ]
```

Pagination, 페이징 처리 :: 세 가지 방식

옵셋 기반

- 페이지 넘버와 개수를 기반으로 함

```
GET /investor/{id}/stocks?offset=2&limit=5  
(returns stocks 2 부터 7 까지 )
```

시간 기반

- 특정 시간을 기반으로 함
- 주로 시계열 데이터에서 사용 가능

```
GET /investor/{id}/stocks?since=xxxxxx&until=yyyyy  
(returns stocks 주어진 날짜 사이)
```

커서 기반

- 데이터 접근 커서를 기반

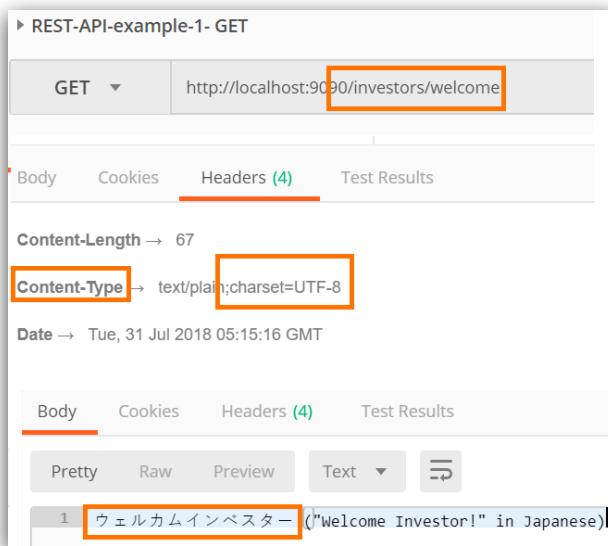
```
GET slack.com/api/users.list?limit=2&token=xoxp-1234-5678-90123
```

유니코드, Unicode

다국적 문자 세트를 지원하는 엔코딩 표준

- 중국어, 일본어, 한국어, 아랍어 ..
- 유니코드 임을 헤더에 포함하여 전송

```
@GetMapping(value="/investors/welcome", produces="text/plain; charset=UTF-8")
public String responseProducesConditionCharset() {
    return " ウエルカムインベスター ("Welcome Investor!" in Japanese);
```



REST 에서 CRUD 구현

HTTP 메소드

- GET
- POST
- PUT
- DELETE

Operation	HTTP method
Create	POST
Read	GET
Update	PUT
Delete	DELETE

CRUD 설계

UserController 생성 : /users

麦克风图标 GET

- public Map<String, Object> getAllUsers();
- public Map<String, Object> getUser(@PathVariable("userid") Integer userid);

麦克风图标 POST

- public Map<String, Object> createUser();

麦克风图标 PUT

- public Map<String, Object> updateUser();

麦克风图标 DELETE

- public Map<String, Object> deleteUser();

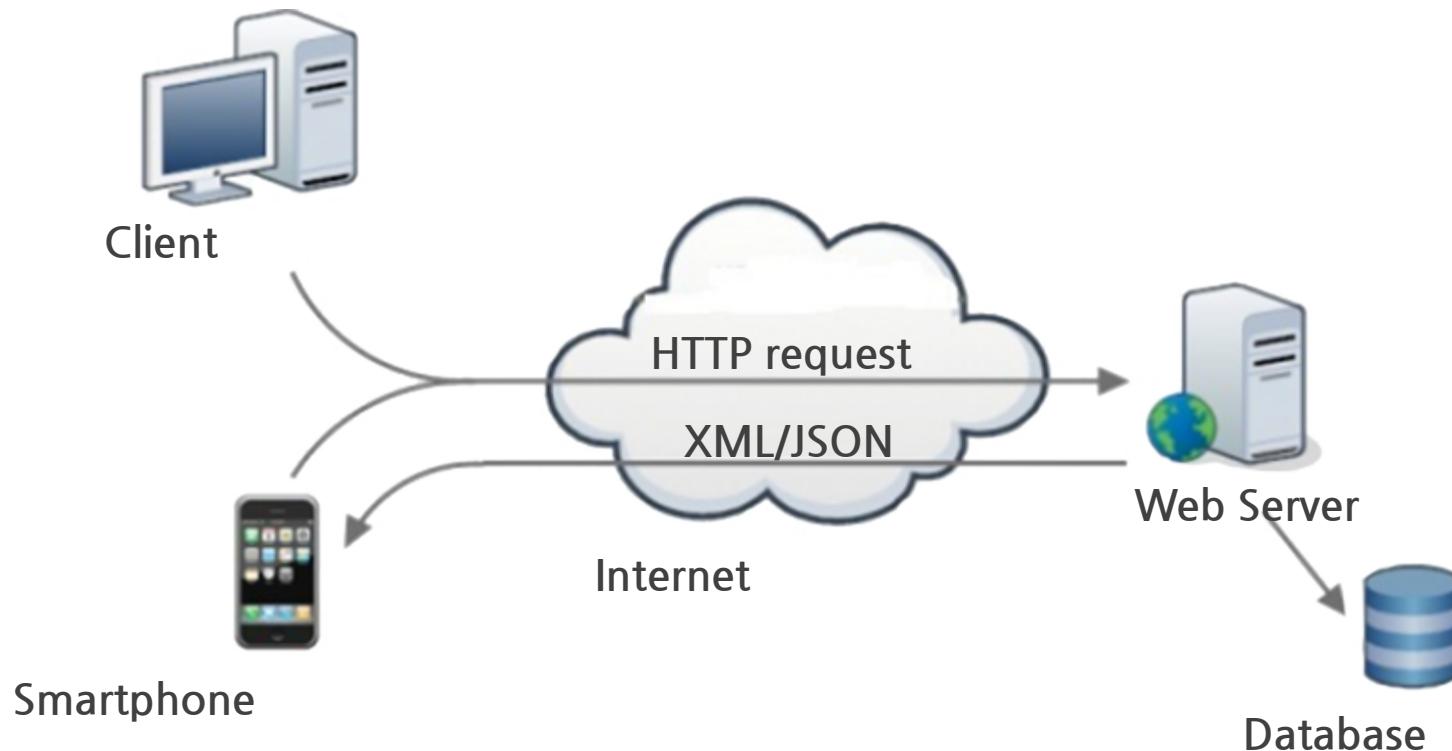
Data Transfer Object 디자인 패턴

- ☑ 데이터를 각 서버/클라이언트 간 전달하기 위한 목적으로 만든 객체
 - ⌚ java.io.Serializable 인터페이스 구현 - 객체직렬화 지원
 - ⌚ Immutable 객체로 만듬 - setter 포함하지 않음
 - ⌚ 도메인(모델) 객체를 래핑(wrapping)

```
public class User {  
    private Integer userid;  
    private String username;  
    public User(Integer userid, String username){  
        this.userid = userid;  
        this.username = username;  
    }  
    // getter and setter methods  
}
```

JSON

RESTful 웹서비스와 JSON/XML



JSON

JSON(JavaScript Object Notation)이란?

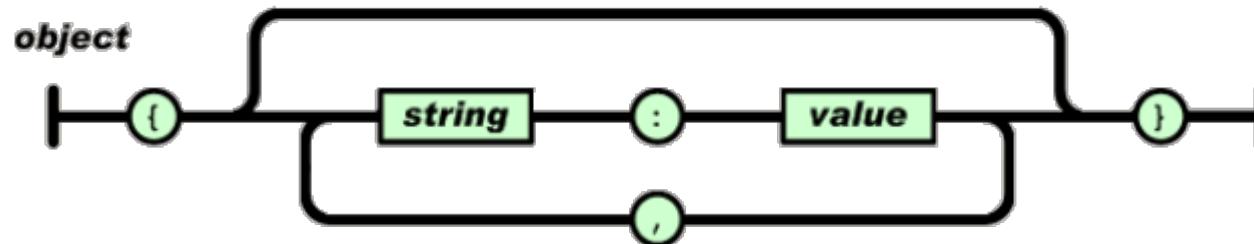
- ◎ <http://www.json.org>
- ◎ JSON은 경량(lightweight)의 DATA-교환 형식
- ◎ Javascript에서 객체를 만들 때 사용하는 표현식을 의미한다.
- ◎ JSON 표현식은 사람과 기계 모두 이해하기 쉬우며 용량이 작아서,
최근에는 JSON이 XML을 대체해서 데이터 전송 등에 많이 사용한다.
- ◎ 특정 언어에 종속되지 않으며, 대부분의 프로그래밍 언어에서
JSON 포맷의 데이터를 핸들링 할 수 있는 라이브러리를 제공하고 있다.

JSON

JSON(JavaScript Object Notation) 형식

1. name-value 형식의 쌍(pair)

: 여러 가지 언어들에서 object, hashtable, struct로 실현되었다.



예시

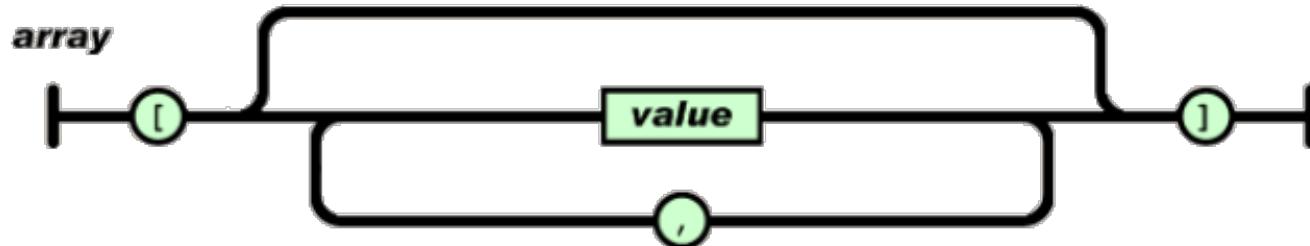
```
{  
    "firstName": "Brett",  
    "lastName": "McLaughlin",  
    "email": "brett@newInstance.com"  
}
```

JSON

JSON(JavaScript Object Notation) 형식

2. 값들의 순서화된 리스트 형식

: 여러 가지 언어들에서 array, list로 실현되었다.



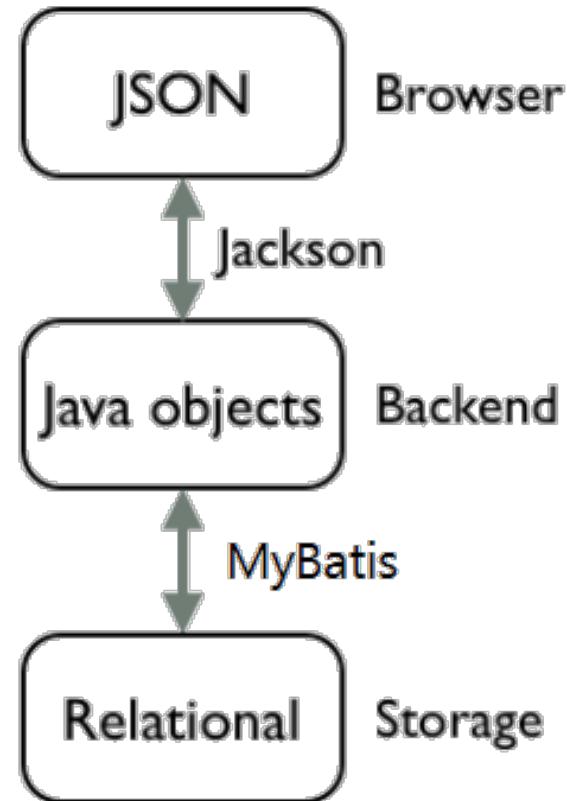
예시

```
{  
    "firstName": "Brett",  
    "lastName": "McLaughlin",  
    "email": brett@newInstance.com,  
    "hobby": ["puzzles", "swimming"]  
}
```

JSON

JSON 라이브러리 - Jackson

- ◎ <http://jackson.codehaus.org>
- ◎ High-Performance JSON Processor!
- ◎ Jackson은 JSON 형태를 Java 객체로, Java 객체를 JSON 형태로 변환해주는 Java 용 JSON 라이브러리이다.
- ◎ 가장 많이 사용하는 JSON 라이브러리이다.



API 응답 포맷

요청에 대한 응답의 결과를 상태와 데이터로 구분

```
{  
  "status": "OK",  
  "data": {...}  
}
```

에러 메세지

```
{  
  "status": "ERROR",  
  "data": null,  
  "error": {  
    "errorCode": 999,  
    "description": "Email address is invalid"  
  }  
}
```

API 응답 포맷 - 계속

페이징 처리

```
{  
    "status": "OK",  
    "data": [...],  
    "error": null,  
    "pageNumber": 1,  
    "nextPage": "http://localhost:8080/users?page=2",  
    "total": 13  
}
```

Jackson 라이브러리를 통해

- JSON <→ 자바 객체

UserController : getAllUsers() 구현

UserService

```
@Service
public class UserServiceImpl implements UserService {
    @Override
    public List<User> getAllUsers() {
        return this.users;
    }
    // Dummy users
    public static List<User> users;
    public UserServiceImpl() {
        users = new ArrayList<>();
        users.add(new User(100, "kim"));
        users.add(new User(101, "lee"));
        users.add(new User(102, "park"));
    }
}
```

UserController

```
@Autowired
UserService userSevice;
@ResponseBody
@GetMapping("")
public List<User> getAllUsers(){
    return userSevice.getAllUsers();
}
```

@RequestMapping

Class Level Mapping vs. Method Level Mapping

요청방식 선택

- ▶ GET, POST, PUT, DELETE

```
@RestController
@RequestMapping("/users")
public class UserController {

    @RequestMapping("/list")
    public String list() {
        return "bbs/list";
    }

    @RequestMapping(value="/write", method=RequestMethod.POST)
    public String doWrite() {
        return "bbs/write_ok";
    }
}
```

▶ Spring4.3 이후

- @GetMapping, @PostMapping, @PutMapping, @DeleteMapping

UserController : getUser() 구현

UserService

```
@Override  
public User getUser(Integer userid) {  
    return users.stream()  
        .filter(x -> x.getUserid() == userid)  
        .findAny()  
        .orElse(new User(0, "Not Available"));  
}
```

UserController

```
@ResponseBody  
@GetMapping("/{userid}")  
public User getUser(@PathVariable("userid") Integer userid){  
    return userSevice.getUser(100);  
}
```

<http://localhost:8080/user/100>

```
{  
    userid: 100,  
    username: "kim"  
}
```

@PathVariable

요청 URL을 파라미터로 사용 (경로변수)

- 예) <http://cafe.daum.net/sq99/2hq/68443>
- 요청 URL : /users/{userid}
- 파라미터로 처리 : @PathVariable("userid") String userid

```
@RestController
@RequestMapping("/users")
public class UserController {

    @Autowired
    private UserService userService;

    @RequestMapping("/users/{userid}")
    public User getUser(@PathVariable("userid") String userid) {
        return userService.getUser(userid);
    }
}
```

UserController : createUser() 구현

UserService

```
@Override  
public void createUser(Integer userid, String username) {  
    User user = new User(userid, username);  
    this.users.add(user);  
}
```

UserController

```
@ResponseBody  
@PostMapping("")  
public Map<String, Object> createUser(  
    @RequestParam(value="userid") Integer userid,  
    @RequestParam(value="username") String username) {  
    Map<String, Object> map = new HashMap<>();  
    userSevice.createUser(userid, username);  
    map.put("result", "added");  
    return map;  
}
```

@RequestParam

The screenshot shows the Postman application interface. At the top, there is a header bar with a URL input field containing "http://localhost:8080/", a "No Environment" dropdown, and several icons. Below the header is a main panel for a POST request to "http://localhost:8080/user". The "Body" tab is selected, showing two parameters: "userid" with value "104" and "username" with value "Kevin". There are also tabs for "Authorization", "Headers (1)", "Pre-request Script", and "Tests". The "Tests" tab is currently empty, displaying the message "No tests available". At the bottom, there are tabs for "Body", "Cookies", "Headers (3)", and "Test Results", with "Test Results" being the active tab. Status information at the bottom right indicates "Status: 200 OK" and "Time: 127 ms".

http://localhost:8080/ ● + ...

No Environment ▼ eye ⚙

POST ▼ http://localhost:8080/user Params Send ▼ Save ▼

Authorization Headers (1) Body Pre-request Script Tests Code

form-data x-www-form-urlencoded raw binary

	Key	Value	Description	...	Bulk Edit
<input checked="" type="checkbox"/>	userid	104			X
<input checked="" type="checkbox"/>	username	Kevin			
	New key	Value	Description		

Body Cookies Headers (3) Test Results Status: 200 OK Time: 127 ms

No tests available

UserController : updateUser() 구현

UserService

```
@Override  
public void updateUser(Integer userid, String username) {  
    users.stream()  
        .filter(x -> x.getUserid() == userid)  
        .findAny()  
        .orElseThrow(() -> new RuntimeException("Item not found"))  
        .setUsername(username);  
}
```

UserController

```
@ResponseBody  
@PutMapping("")  
public Map<String, Object> updateUser(  
    @RequestParam(value="userid") Integer userid,  
    @RequestParam(value="username") String username) {  
    Map<String, Object> map = new LinkedHashMap<>();  
    userSevice.updateUser(userid, username);  
    map.put("result", "updated");  
    return map;  
}
```

UserController : deleteUser() 구현

UserService

```
@Override  
public void deleteUser(Integer userid) {  
  
    users.removeIf((User u) -> u.getUserid() == userid);  
  
}
```

UserController

```
@ResponseBody  
@DeleteMapping("/{userid}")  
public Map<String, Object> deleteUser(  
    @PathVariable("userid") Integer userid) {  
    Map<String, Object> map = new LinkedHashMap<>();  
    userSevice.deleteUser(userid);  
    map.put("result", "deleted");  
    return map;  
}
```

@RequestBody

- ☑ @RequestBody - 클라이언트에서 JSON 으로 데이터 요청
- ☑ Model Class - 클라이언트 폼이나 파라미터로 데이터 요청

RESTful API 패턴

Part 2

REST 서비스 디자인 패턴 :: Part 2

- API Versioning
- Uniform contract
- Entity endpoint
- Endpoint redirection
- 응답코드와 REST 패턴

API versioning

버전 넘버

- Major.Minor (v1.1) 또는 Major 만 사용 (v1)

버전 변경 시점

- Major version : 이전 버전과 하위호환을 깨는 큰 변화가 생겼을 경우
 - API 삭제 또는 API URL 변경된 경우
 - API 파라미터가 삭제되거나, 파라미터명이 변경된 경우
 - API의 동작이 변경된 경우
 - 반환하던 에러 코드가 변경된 경우
 - 공통적으로, 클라이언트 코드 변경 필요
- Minor version : Major 변경 이외에 다양한 상황이 존재 할 수 있음

API versioning :: 세 가지 방식

URI 경로에 버전정보 포함

```
http://api.foo.com/coffees/1234
```

```
http://api.foo.com/v2/coffees/1234
```

• V2 가 최신 버전, 과거 버전으로 접근했을 때 에러 코드

- 301 Moved permanently : 다른 URI 로 영구적으로 대체됨
- 302 Found : 요청된 리소스가 일시적으로 위치가 변경됨 (요청된 URI 사용 가능)

요청 쿼리 파라미터에 버전정보 포함

```
http://api.foo.com/coffees/1234?version=v2
```

헤더에 버전정보 포함

```
Accept: application/vnd.github.v3+json
```

API versioning :: URI 경로에 포함

URI 경로에 버전정보 포함

```
@GetMapping({"/v1/investors", "/v1.1/investors", "/v2/investors"})  
public List<Investor> fetchAllInvestors()  
{  
    return investorService.fetchAllInvestors();  
}
```

The screenshot shows a Postman interface with the following details:

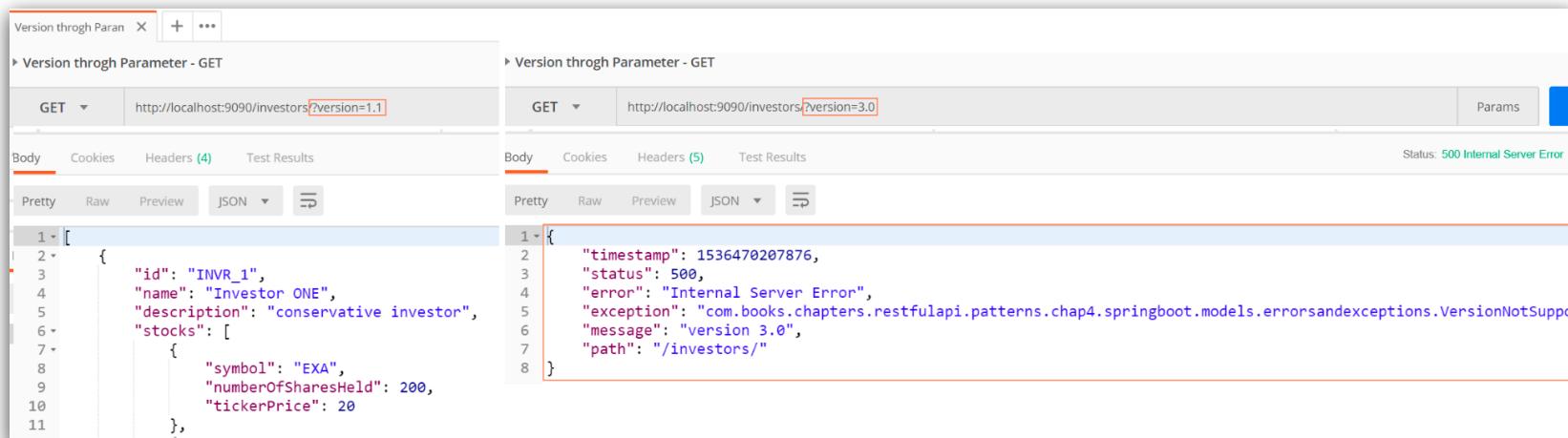
- Method:** GET
- URL:** `http://localhost:9090/v2/investors/`
- Headers:** Authorization (Key: Value)
- Body:** (Empty)
- Tests:** (Empty)
- Responses:**
 - Status: 200 OK
 - Time: 34 ms
 - Size: 836 B
- Preview:** Shows a JSON response with 11 items, each representing an Investor object. The first item has the following structure:

```
1 [  
2 {  
3     "id": "INVR_1",  
4     "name": "Investor ONE",  
5     "description": "conservative investor",  
6     "stocks": [  
7         {  
8             "symbol": "EXA",  
9             "numberOfSharesHeld": 200,  
10            "tickerPrice": 20  
11        },
```

API versioning :: 쿼리 파라미터

요청 쿼리 파라미터에 버전정보 포함

```
@GetMapping("/investors")  
public List<Investor> fetchAllInvestorsForGivenVersionAsParameter(  
    @RequestParam("version") String version)  
    throws VersionNotSupportedException {  
  
    if (!(version.equals("1.1") || version.equals("1.0"))){  
        throw new VersionNotSupportedException("version " + version);  
    }  
  
    return investorService.fetchAllInvestors();  
}
```



The screenshot shows two separate API requests in the Postman interface. Both requests are GET requests to the URL `http://localhost:9090/investors`.

The first request is labeled "Version through Parameter - GET" and has a query parameter `version=1.1`. The response body is a JSON array containing one investor object:

```
1 [  
2   {  
3     "id": "INVR_1",  
4     "name": "Investor ONE",  
5     "description": "conservative investor",  
6     "stocks": [  
7       {  
8         "symbol": "EXA",  
9         "numberOfSharesHeld": 200,  
10        "tickerPrice": 20  
11      },  
12    ]  
13  ]
```

The second request is also labeled "Version through Parameter - GET" and has a query parameter `version=3.0`. The response body is a JSON object representing an error response:

```
1 {  
2   "timestamp": 1536470207876,  
3   "status": 500,  
4   "error": "Internal Server Error",  
5   "exception": "com.books.chapters.restfulapi.patterns.chap4.springboot.models.errorsandexceptions.VersionNotSupportedException",  
6   "message": "version 3.0",  
7   "path": "/investors/"  
8 }
```

Both requests have the "Body" tab selected in the Postman interface.

API versioning :: 커스텀 헤더

헤더에 버전정보 포함

- Accept 헤더 사용 (content-negotiation)
- x-resource-version 같은 커스텀 헤더도 가능

```
@GetMapping("/investorsbychversion")
public List<Investor> fetchAllInvestors...
    @RequestHeader("x-resource-version")
        String version)
    throws VersionNotSupportedException {
    return getResultsAccordingToVersion(version);
}
```

The screenshot shows two API requests in Postman:

Top Request (Successful):

- Method: GET
- URL: `http://localhost:9090/investorsbycustomheaderversion/`
- Headers (1):
 - `x-resource-version` (checked, value: 1.1)
- Body: [JSON Response]

```
[{"id": "INVR_1", "name": "Investor ONE", "description": "conservative investor", "stocks": [{"symbol": "EXA", "numberofSharesHeld": 200, "tickerPrice": 20}]}]
```
- Status: 200 OK, Time: 41 ms, Size: 836 B

Bottom Request (Failed):

- Method: GET
- URL: `http://localhost:9090/investorsbycustomheaderversion/`
- Headers (1):
 - `x-resource-version` (unchecked, value: 1.1)
- Body: [JSON Response]

```
{"timestamp": 1536474707513, "status": 400, "error": "Bad Request", "exception": "org.springframework.web.bind.ServletRequestBindingException", "message": "Missing request header 'x-resource-version' for method parameter of type String", "path": "/investorsbycustomheaderversion/"}
```
- Status: 400 Bad Request, Time: 28 ms

Uniform contract :: 일관된 주소경로

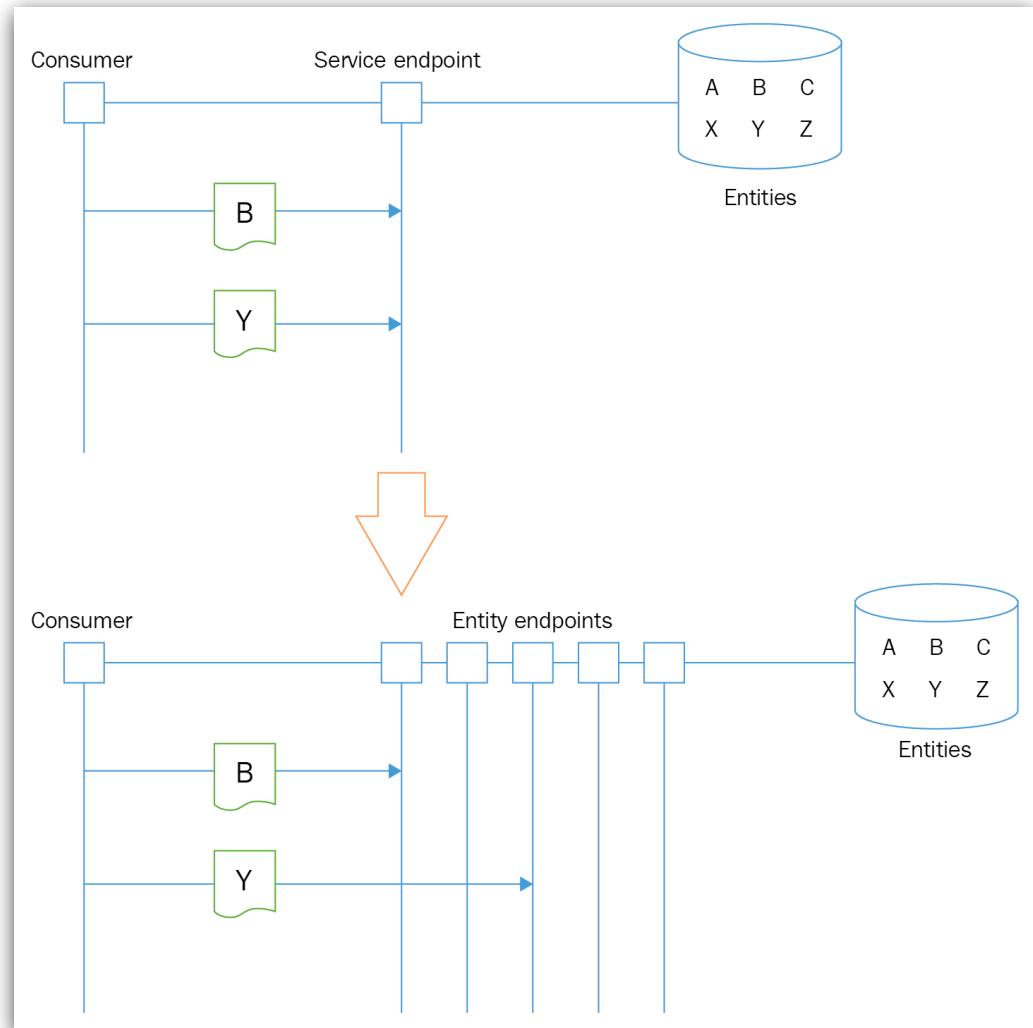
- API 서비스는 계속해서 수정, 발전, 버그픽스, 보안패치 등.. 변화한다.
 - ☞ 클라이언트(Consumer)는 동일한 엔드포인트와 구현으로 접근 가능해야 함

- REST 원칙 상 엔드포인트는 HTTP verbs 만 사용해야 함
 - ☞ GET
 - ☞ POST
 - ☞ PUT
 - ☞ PATCH
 - ☞ DELETE

Entity endpoints

엔티|E| : investors, stocks, articles, teams, players ..

- /investors/investorId
- investor/stockId
- /articles
- /articles/articleId
- /teams
- /teams/teamId
- /teams/teamId/players



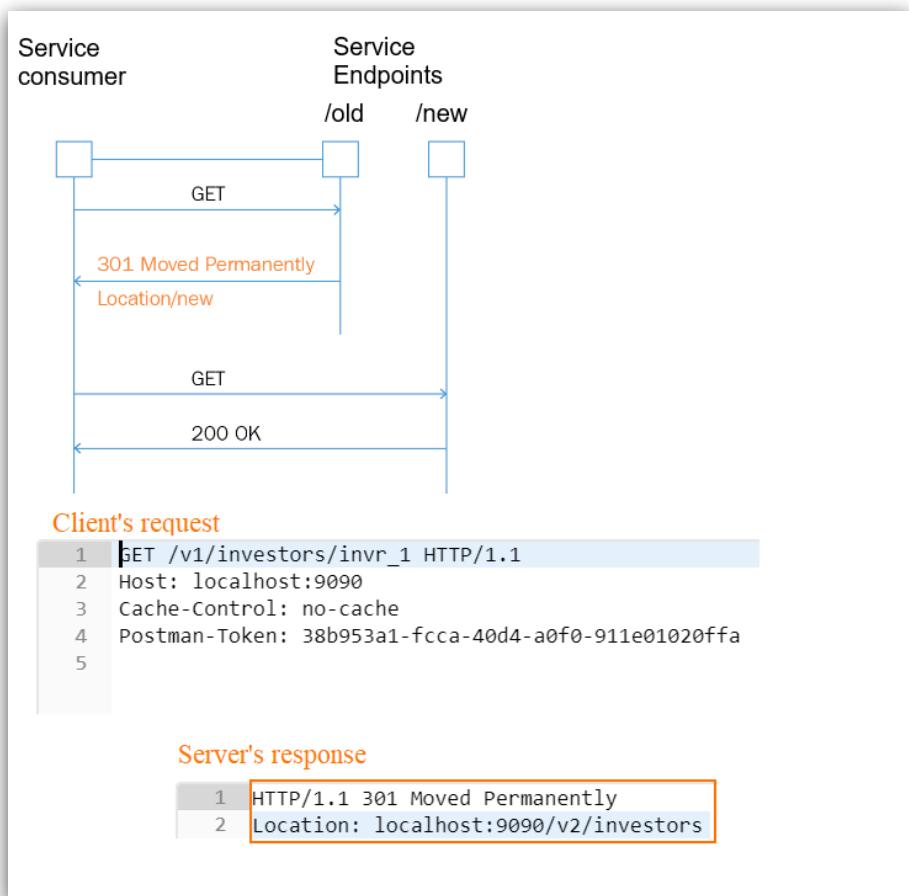
Endpoint redirection

엔드포인트 변경 시 클라이언트에게 통보 하는 방법

- 엔드포인트 변경은 가능한 최소화하는 것이 좋음
- Location 헤더와 함께 표준 HTTP 응답 코드 사용 (300번 대)
 - 301 Moved permanently
 - 307 Temporary Redirect

HATEOAS 구현 사용

- 엔드포인트 리다이렉션
을 대체할 수 있음



응답코드와 REST 패턴

HTTP의 표준 응답코드

• 200 번대 : 응답 성공

- 200 OK : create, update, delete (PUT, POST, DELETE) 동작
응답의 일부로 컨텐츠를 반환
- 201 Created : PUT으로 리소스를 생성 했을 때
Location 헤더가 반드시 포함되어야 함
- 204 No Content : DELETE, POST, PUT 동작에서 사용됨
응답되는 컨텐츠가 없음
- 202 Accepted : 비동기 작업에서 사용됨, 아직 동작이 완료되지 않았음
Location 헤더를 반환해야 하며, 요청을 모니터링 할수 있음

• 300 번대 : 리다이렉션

- 301 Permanent : 해당 URL 이 영구적으로 새로운 URL로 변경되었음
- 302 Found : 요청한 리소스가 임시적으로 새로운 URL로 이동했음(Temporarily moved), 컨텐츠만 새로운 URL에서 제공

응답코드와 REST 패턴

HTTP의 표준 응답코드

400 번대 : 클라이언트 요청 에러

- 401 Unauthorized : 인증이 되지않아 응답이 거부됨
 - 404 Not Found : 요청한 리소스가 잘못되었거나 없음, 인가 문제로도 사용됨
 - 406 Not Acceptable : Accept 헤더에 대한 응답이 없을 때
 - 415 Unsupported Media Type : Content-Type 헤더에 대한 응답이 없을 때

500 번대 : 서버 에러

Security and Traceability

보안과 추적성

Traceability :: 로깅과 예외처리

- REST API 로깅
- RESTful 서비스 예외처리

REST API 로깅

- ☑ 복잡한 분산 어플리케이션은 에러가 발생할 포인트도 다양
 - ⌚ 문제를 발견하고 수정하기 어려움
 - ⌚ 문제 발생 즉시 확인하고 수정/처리 될 때 비용이 낮아질 수 있음

- ☑ 분산된 어플리케이션으로 부터 로그를 모으고 분석 필요
 - ⌚ 예) Splunk (<http://www.splunk.com/>)

REST API 로깅 :: Servlet Filter 예제

필터를 이용한 간단한 로깅 예제

```
@WebFilter(filterName = "LoggingFilter", urlPatterns = {"/*"})
public class LoggingFilter implements Filter {
    static final Logger logger = Logger.getLogger(LoggingFilter.class);
    @Override
    public void doFilter(ServletRequest servletRequest, ServletResponse servletResponse,
        FilterChain filterChain) throws IOException, ServletException {
        HttpServletRequest httpServletRequest = (HttpServletRequest) servletRequest;
        logger.info("request" +httpServletRequest.getPathInfo().toString());
        filterChain.doFilter(servletRequest, servletResponse);
    }
}
```

REST API 로깅 :: Best practices

운영 로그에서 포함해야 할 기본 내용

- ⌚ 날짜와 시간
- ⌚ 로깅 레벨
- ⌚ 쓰레드 명
- ⌚ 간단한 로거 이름
- ⌚ 에러내용에 대한 메세지

운영로그에서 개인정보나 민감한 정보는 포함해서는 안됨

- ⌚ PII (Personally identifiable information)
- ⌚ 데이터 마스킹

API 호출자에 대한 정보 포함

- ⌚ 다양한 클라이언트에 대한 정보 : 모바일, 웹, 다른 서비스 ..

REST API 로깅 :: Spring boot Logging system

Logback framework 사용

로깅 레벨

- ⌚ TRACE
- ⌚ DEBUG
- ⌚ INFO
- ⌚ WARN
- ⌚ ERROR

로그 레벨 설정

```
# spring framework logging
logging.level.org.springframework = ERROR

# local application logging
logging.level.com.acomp.hello = INFO
```

Logger 사용

```
package com.example.restapp;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
// other imports

@RestController
public class HomeController {
    private static final Logger logger =
        LoggerFactory.getLogger(HomeController.class);

    @GetMapping("/")
    public Map<String, Object> test() {
        Map<String, Object> map = new HashMap<>();
        map.put("result", "Aloha");
        logger.trace("{test} trace");
        logger.debug("{test} debug");
        logger.info("{test} info");
        logger.warn("{test} warn ");
        logger.error("{test} error");
        return map;
    }
}
```

예외처리

Exception Handling

스프링 예외처리

스프링 예외처리 방법

- ⌚ 전역처리 Global level - `@ControllerAdvice`
- ⌚ 컨트롤러단에서 처리 Controller level - `@ExceptionHandler`
- ⌚ 메소드 단위 처리 Method level - `try/catch`

@ControllerAdvice - 전역처리

- ☑ **@ControllerAdvice는 스프링 어플리케이션의 모든 예외를 처리**
 - 💡 **@RestController** 는 컨트롤러
 - 💡 **@ExceptionHandler** 어노테이션을 사용하여 예외를 처리할 클래스를 정의

```
@ControllerAdvice  
@RestController  
public class GlobalExceptionHandler {  
  
    @ExceptionHandler(BaseException.class)  
    @ResponseStatus(HttpStatus.BAD_REQUEST)  
    public String handleBaseException(BaseException e) {  
        return e.getMessage();  
    }  
  
    @ExceptionHandler(value = Exception.class)  
    @ResponseStatus(403)  
    public String handleException(Exception e) {  
        return e.getMessage();  
    }  
}
```

@ExceptionHandler 를 사용한 컨트롤러 단 예외처리

- ✓ HomeController.java 파일 내에 다음 코드가 있으면, 해당 컨트롤러의 모든 NumberFormatException 을 잡아서 처리한다.

```
@ExceptionHandler(value = NumberFormatException.class)
public String nfeHandler(NumberFormatException e){
    return e.getMessage();
}
```

비즈니스 예외 예제

명시적 예외처리가 필요없는 RuntimeException 타입으로 작성

```
public class ClientException extends RuntimeException {  
  
    private final int errorCode;  
    private final String errorDescription;  
  
    public ClientException(ApiError error) {  
        super(error.getErrorCode() + ": " + error.getDescription());  
        this.errorCode = error.getErrorCode();  
        this.errorDescription = error.getDescription();  
    }  
  
    public int getErrorCode() {  
        return errorCode;  
    }  
  
    public String getDescription() {  
        return errorDescription;  
    }  
}
```

에러 처리와 로깅

☒ 에러 시 에러코드 전송 전략 실제 사례

APIs	HTTP code	샘플 메세지	비고
Facebook	200	{"type":"OAUTH exception","message":"...."}	200 성공을 의미. API 는 에러메세지를 제공
Twilio	401	{"type":"OAUTH exception","message":"...."}	기존 HTTP 코드를 사용
네이버	401	Authentication failed (인증 실패하였습니다.)	기존 HTTP 코드를 사용

Authentication and authorization

인증과 인가

인증 방식의 변화

과거

- 기업내 통합 인증 방식, SSO(Single Sign On)

SOA 기반 아키텍처

- 기업은 파트너사 또는 외부 서비스에서도 API 사용을 허가해야 함
- 다양한 어플리케이션과 플랫폼에서 사용 가능하게 단순한 인증방식 필요

Spring Security and JWT

- 스프링 시큐리티
- JSON Web Token (JWT)
- 웹 서비스에서 JWT 생성하기 및 다루기

Spring Security

스프링 시큐리티 란?

- 脆弱한 인증과 권한 프레임워크
- REST API 를 인증되고 권한이 있는 요청에만 허락해 줌

JWT

- 보안(인증과 권한)에 많이 사용됨
- JWT 토큰 형태로 사용됨
 - URL-safe, 웹브라우저 호환, SSO 지원

주요 인증 방식

로그인 기반 인증 (크레덴셜 기반 인증)

- Credential-based authentication
- 토큰 기반 인증



인증정보를 다른 어플리케이션으로 전달

- 제 3자가 인증을 처리하는 방식
- OAuth2
- 페이스북/구글 같은 소셜 계정들을 이용하여 로그인

2단계 인증

- Two-factor authentication

하드웨어 인증

- Hardware authentication

서버 기반 인증 시스템 문제점

세션

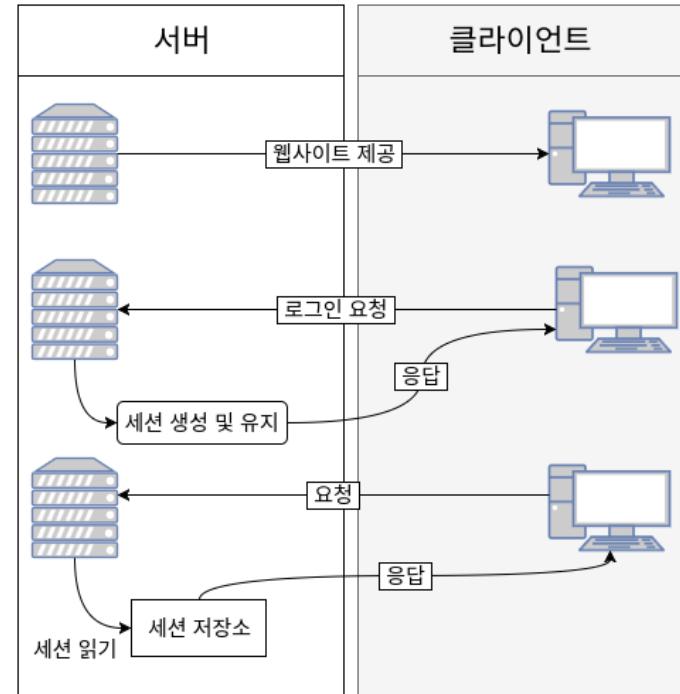
- 유저가 인증할 때 이 기록을 서버에 저장
- 메모리에 저장 혹은 데이터베이스 시스템에 저장
- 유저의 수(동시 접속)가 많으면 서버나 DB에 무리를 줌

확장성

- 클러스터링 구성 시 세션 정보도 같이 공유해야 함
- 서버 구성이 복잡해짐

CORS (Cross-Origin Resource Sharing)

- 쿠키를 여러 도메인에서 관리하는 것이 번거로움

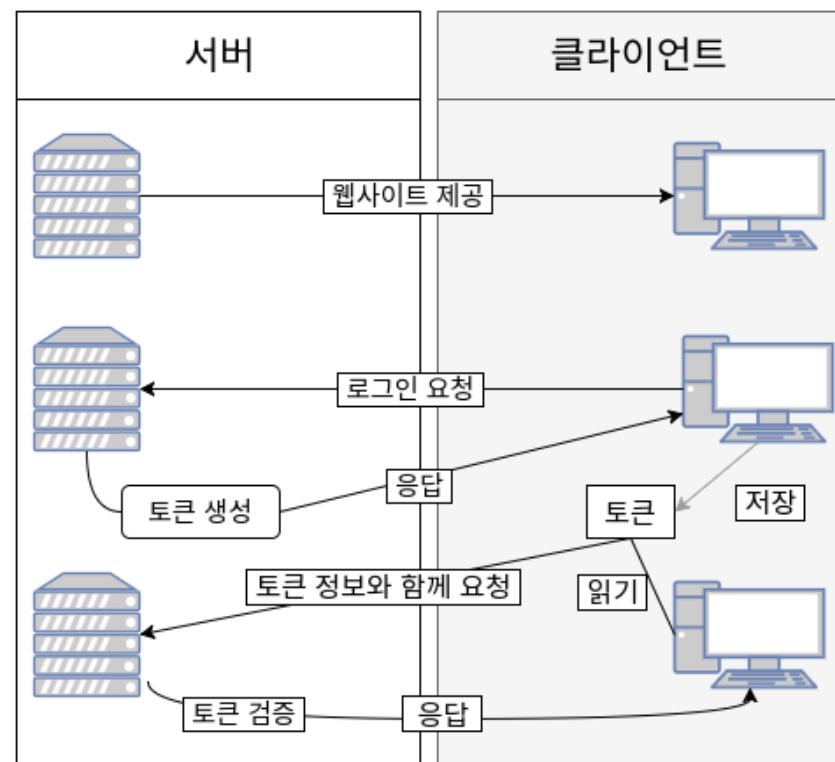


토큰 기반 인증 시스템 작동 원리

작동원리

- 유저가 아이디와 비밀번호로 로그인 수행
- 서버측에서 해당 정보 검증
- 계정정보가 정확하다면 서버측에서 유저에게 signed 토큰을 발급
- 클라이언트에서는 토큰을 저장해 두고 요청마다 토큰을 서버에 함께 전달
- 서버에서 토큰을 검증하고 요청에 응답

토큰은 *HTTP* 헤더에 포함시켜서 전달



토큰 기반 인증 시스템 - 토큰의 장점

무상태(Stateless)이며 확장성(Scalability)이 있음

보안성

- 쿠키를 사용하지 않음
- 토큰환경에서도 취약점은 존재함

Extensibility

- 서버 확장이 아닌 기능확장 가능

여러 플랫폼 및 도메인

- CORS - 아무 도메인에서나 토큰만 유효하면 요청이 정상적으로 처리됨

웹 표준 기반

- JWT - 토큰기반 인증 시스템 구현체 (RFC 7519)

JSON Web Token

JWT 특징

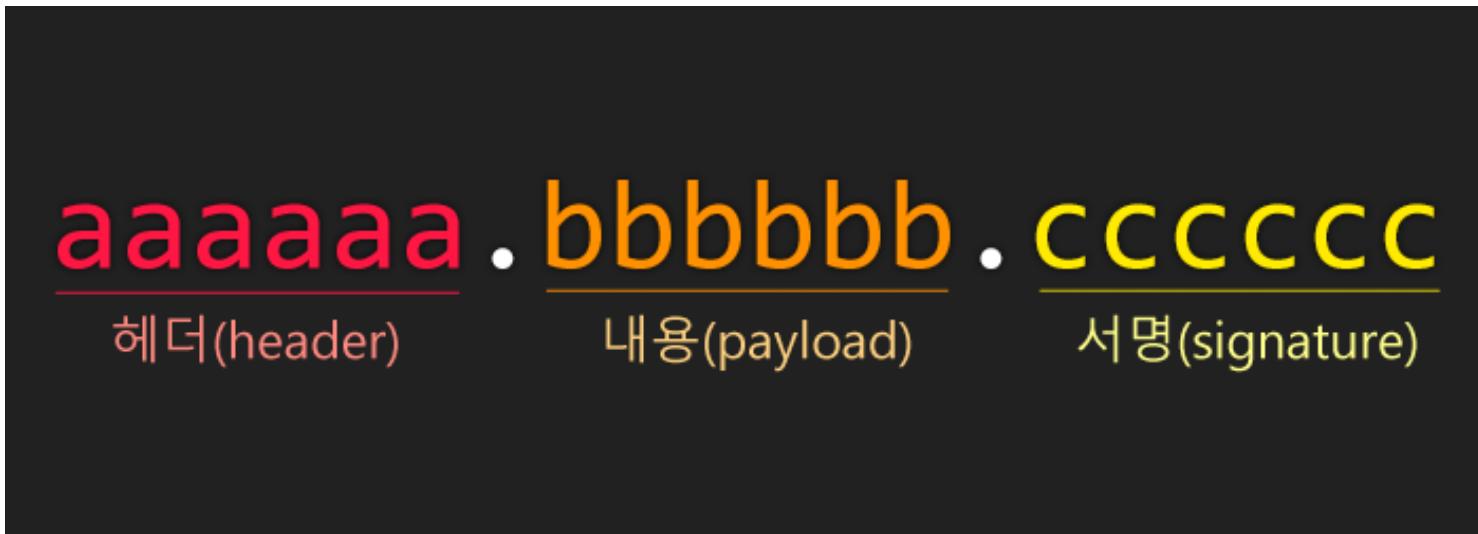
- 웹표준으로 다양한 프로그래밍 언어에서 지원 (C, Java, Python, C++, R, JS, GO, Swift ..)
- Self-contained - 필요한 모든 정보를 자체적으로 가지고 있음
- 웹 서버의 경우 HTTP 헤더에 넣어서 전달 또는 URL 파라미터로 전달 가능

JWT 사용되는 상황

- 회원인증
- 정보 교환

JWT 구조

- ✓ JWT는 . 을 구분자로 3가지 문자열로 구성



JWT 구조

헤더 (Header) - 두 가지 정보를 포함

- typ : 토큰의 타입을 지정 (JWT)
- alg : 해싱 알고리즘 지정 (HMAC SHA256, RSA ..)

```
{  
  "typ": "JWT",  
  "alg": "HS256"  
}
```

JWT 구조

정보 (payload)

- ▶ 토큰에 담을 정보가 포함 (클레임이라고 함, name/value 쌍으로 구성)
- ▶ 클레임(claim)은 다음 세 분류로 나뉨
 - 등록된 (registered) 클레임
 - iss, sub, aud, exp, nbf, sat, jti
 - 공개 (public) 클레임
 - 충돌 방지 이름이 필요 (주로 URI 형식으로 네이밍)
 - 비공개 (private) 클레임
 - 클라이언트와 서버 간의 협의하에 사용되는 클레임 이름들

```
{  
    "iss": "example.com",  
    "exp": "1485270000000",  
    "https://example.com/jwt\_claims/is\_admin    "userId": "11028373727102",  
    "username": "kim"  
}
```

JWT 구조

서명 (signature)

- 헤더의 인코딩값과 정보의 인코딩값을 합친 후 주어진 비밀키로 해싱하여 생성

```
eyJhbGciOiJIUzI1NilsInR5cCI6IkpXVCJ9.eyJzdWliOilxMjM0NT  
Y3ODkwliwibmFtZSI6IkpvaG4gRG9IiwiWF0IjoxNTE2MjM5MD  
IyfQ.SflKxwRJSMeKKF2QT4fwpMeJf36POk6yJV_adQssw5c
```

<https://jwt.io/>

JWT 토큰 만들기

디펜던시 설정

```
<dependency>
    <groupId>javax.xml.bind</groupId>
    <artifactId>jaxb-api</artifactId>
    <version>2.3.1</version>
</dependency>
<dependency>
    <groupId>io.jsonwebtoken</groupId>
    <artifactId>jjwt</artifactId>
    <version>0.9.1</version>
</dependency>
```

JWT 토큰 구현

SecurityService

```
public interface SecurityService {  
    String createToken(String subject, long ttlMillis);  
    // other methods  
}
```

SecurityServiceImpl

```
private static final String secretKey= "4C8kum4LxyKWYLM78sKdXrzbBjDCFyfX";  
@Override  
public String createToken(String subject, long ttlMillis) {  
    if (ttlMillis <= 0) {  
        throw new RuntimeException("Expiry time must be greater than Zero :["+ttlMillis+"] ");  
    }  
    // The JWT signature algorithm we will be using to sign the token  
    SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;  
    byte[] apiKeySecretBytes = DatatypeConverter.parseBase64Binary(secretKey);  
    Key signingKey = new SecretKeySpec(apiKeySecretBytes, signatureAlgorithm.getJcaName());  
    JwtBuilder builder = Jwts.builder()  
        .setSubject(subject)  
        .signWith(signatureAlgorithm, signingKey);  
    long nowMillis = System.currentTimeMillis();  
    builder.setExpiration(new Date(nowMillis + ttlMillis));  
    return builder.compact();  
}
```

JWT 토큰 구현

HomeController

```
@Autowired  
Private SecurityService securityService;  
  
@GetMapping("/security/generate/token")  
public Map<String, Object> generateToken(@RequestParam(value="subject") String subject) {  
    String token = securityService.createToken(subject, (2 * 1000 * 60));  
    Map<String, Object> map = new LinkedHashMap<>();  
    map.put("result", token);  
    return map;  
}
```

토큰 생성

```
http://localhost:8080/security/generate/token?subject=one
```

JWT 토큰에서 subject 추출

SecurityServiceImpl

```
@Override  
public String getSubject(String token) {  
    Claims claims = Jwts.parser()  
        .setSigningKey(DatatypeConverter.parseBase64Binary(secretKey))  
        .parseClaimsJws(token).getBody();  
    return claims.getSubject();  
}
```

HomeController

```
@ResponseBody  
@GetMapping("/security/get/subject")  
public Map<String, Object> getSubject(@RequestParam("token") String token) {  
    String subject = securityService.getSubject(token);  
    Map<String, Object> map = new LinkedHashMap<>();  
    map.put("result", subject);  
    return map;  
}
```

<http://localhost:8080/security/get/subject?token=eyJhbGciOiJIUzI1NiJ9.eyJ...>

RESTful 웹서비스 테스트

- JUnit 테스트
- MockMvc (Controller mocking)

JUnit

JUnit

- Java에서 독립된 단위테스트(Unit Test)를 지원해주는 프레임워크

단위테스트(Unit Test)란?

- 소스 코드의 특정 모듈이 의도된 대로 정확히 작동하는지 검증하는 절차, 즉 모든 함수와 메소드에 대한 테스트 케이스(Test case)를 작성하는 절차
- jUnit은 보이지 않고 숨겨진 단위 테스트를 끌어내어 정형화시켜 단위테스트를 쉽게 해주는 테스트 지원 프레임워크

Service 테스트

UserService

```
@RunWith(SpringRunner.class)
@SpringBootTest
public class UserTests {
    @Autowired
    private UserService userService;

    @Test
    public void testAllUsers(){
        List<User> users = userService.getAllUsers();
        assertEquals(3, users.size());
    }

    // other methods
    @Test
    public void testSingleUser(){
        User user = userService.getUser(100);
        assertTrue(user.getUsername().contains("kim"));
    }
}
```

MockMvc

MockMvc 를 이용하여 **localhost:8080/** 테스트

```
@SpringBootTest
@RunWith(SpringJUnit4ClassRunner.class)
public class UserMockMVCTests {
    @Autowired
    private WebApplicationContext ctx;
    private MockMvc mockMvc;
    @Before
    public void setUp() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.ctx).build();
    }
    @Test
    public void testBasicMVC() throws Exception {
        MvcResult result = mockMvc
            .perform(MockMvcRequestBuilders.get("/"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("result", is("Aloha"))))
            .andReturn();
        String content = result.getResponse().getContentAsString();
        System.out.println("{testBasicMVC} response : " + content);
    }
}
```

MockMvc

MockMvc 를 이용하여 **localhost:8080/users/100** 테스트

```
@SpringBootTest
@RunWith(SpringJUnit4ClassRunner.class)
public class UserMockMVCTests {
    @Autowired
    private WebApplicationContext ctx;
    private MockMvc mockMvc;
    @Before
    public void setUp() {
        this.mockMvc = MockMvcBuilders.webAppContextSetup(this.ctx).build();
    }
    @Test
    public void testSingleUser() throws Exception {
        MvcResult result = mockMvc
            .perform(MockMvcRequestBuilders.get("/users/100"))
            .andExpect(status().isOk())
            .andExpect(jsonPath("userid", is(100)))
            .andExpect(jsonPath("username", is("kim")))
            .andReturn();
        String content = result.getResponse().getContentAsString();
        System.out.println("{testSingleUser} response : " + content);
    }
}
```

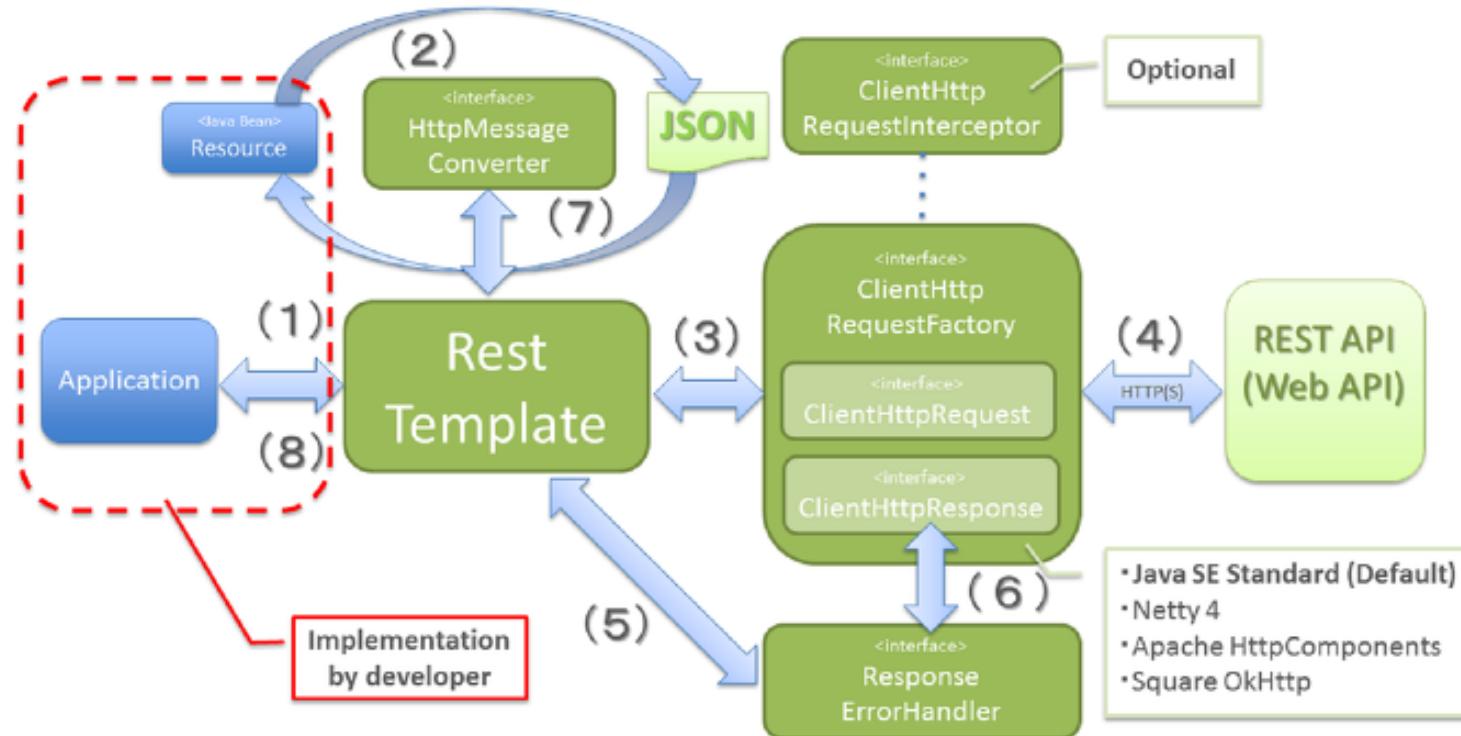
REST 클라이언트 작성

REST Client 작성

- 다른 REST API 서비스를 호출 해야 될 때가 있음
 - ⌚ 결재 API, SMS API, 공공 API 등
- Spring 3.0부터 지원하고, 스프링에서 제공하는 http 통신에 사용
 - ⌚ HTTP 서버와의 통신을 단순화하고 RESTful 원칙을 지킨다.
 - ⌚ 기계적이고 반복적인 코드를 최대한 줄여준다.
 - ⌚ json, xml 를 쉽게 응답받는다.
- RestTemplate 사용
 - ⌚ HTTP 클라이언트 역할
 - ⌚ GET, POST, PUT, DETETE (추가로 OPTIONS, HEAD) 사용 가능

RestTemplate 동작원리

- ✓ org.springframework.http.client 패키지에 포함
 - HttpClient를 추상화하여 제공
 - 내부통신에서는 Apache HttpComponents 를 사용



RestTemplate 의존성 추가

- Apache httpClient 에 대한 의존성이 있음
- 스프링 4 버전 이후부터 Apache httpClient 4.3 이상 버전 필요

```
<dependency>
    <groupId>org.apache.httpcomponents</groupId>
    <artifactId>httpclient</artifactId>
    <version>4.5.6</version>
</dependency>
```

RestTemplate 빈 등록

✓ RestTemplate 을 빈으로 등록

- @Autowired 어노테이션으로 모든 클래스에서 사용 가능

```
@Configuration
public class AppConfig {
    @Bean
    public RestTemplate restTemplate() {
        HttpComponentsClientHttpRequestFactory factory =
            new HttpComponentsClientHttpRequestFactory();
        factory.setReadTimeout(5000); // 읽기시간초과, ms
        factory.setConnectTimeout(3000); // 연결시간초과, ms.
        HttpClient httpClient = HttpClientBuilder.create()
            .setMaxConnTotal(100) // connection pool 적용.
            .setMaxConnPerRoute(5) // connection pool 적용.
            .build();
        factory.setHttpClient(httpClient); // 동기실행에 사용될 HttpClient 세팅

        return new RestTemplate(factory);
    }
}
```

RestTemplate - GET 요청

getForObject 메서드 사용

```
String content = template.getForObject("http://localhost:8080/", String.class);
```

```
User user = template.getForObject("http://localhost:8080/user/100", User.class);
```

```
String result = restTemplate.getForObject(  
    "http://example.com/hotels/{hotel}/bookings/{booking}", String.class, "42", "21");
```

```
Map<String, String> vars = new HashMap<String, String>();  
vars.put("hotel", "42");  
vars.put("booking", "21");  
String result = restTemplate.getForObject(  
    "http://example.com/hotels/{hotel}/bookings/{booking}", String.class, vars);
```

REST 클라이언트

POJO로 응답 받기 - GET 방식

- 응답 JSON을 객체로 매팅

```
public class User implements Serializable {  
    private long id;  
    private String name;  
    // standard getters and setters  
}
```

- RestTemplate 클래스 :: getForObject 메소드 사용

```
RestTemplate restTemplate = new RestTemplate();  
User user = restTemplate  
    .getForObject("http://localhost:8080/users/" + 1, User.class);  
  
assertThat(user.getName(), notNullValue());  
assertThat(user.getId(), is(1));
```

RestTemplate :: getForObject

```
@RestController
@RequestMapping("/client")
public class ClientController {
    private final Logger log = LoggerFactory.getLogger(this.getClass());

    @Autowired
    private RestTemplate template;
    @ResponseBody
    @RequestMapping("/test")
    public Map<String, Object> test() {
        Map<String, Object> map = new HashMap<>();
        String content = template.getForObject("http://localhost:8080/", String.class);
        map.put("result", content);
        return map;
    }
}
```

```
@ResponseBody
@RequestMapping("/test/user")
public Map<String, Object> test GetUser() {
    Map<String, Object> map = new HashMap<>();
    User user = template.getForObject("http://localhost:8080/users/100", User.class);
    map.put("result", user);
    return map;
}
```

User 클래스는 반드시 *Serializable* 해야 함!

User 클래스 수정

```
public class User implements Serializable {
    private static final long serialVersionUID = 3453281303625368221L;

    private Integer userid;
    private String username;

    public User() { }
    public User(Integer userid, String username) {
        this.userid = userid;
        this.username = username;
    }
    public Integer getUserId() {
        return userid;
    }
    public void setUserId(Integer userid) {
        this.userid = userid;
    }
    public String getUsername() {
        return username;
    }
    public void setUsername(String username) {
        this.username = username;
    }
    @Override
    public String toString() {
        return "User [userid=" + userid + ", username=" + username + "]";
    }
}
```

RestTemplate - POST 요청

postForLocation

```
uri = "http://example.com/hotels/{id}/bookings";  
  
RestTemplate template = new RestTemplate();  
  
Booking booking = // create booking object  
  
URI location = template.postForLocation(uri, booking, "1");
```

postForObject

```
public <T> T postForObject(String url,  
                           Object request,  
                           Class<T> responseType,  
                           Map<String,?> uriVariables)  
throws RestClientException
```

REST 클라이언트

- 리소스 생성 - POST 방식
 - ⌚ postForObject 메소드

```
ClientHttpHttpRequestFactory requestFactory = getClientHttpHttpRequestFactory();
RestTemplate restTemplate = new RestTemplate(requestFactory);

HttpEntity<User> request = new HttpEntity<>(new User(1, "kim"));
User user = restTemplate.postForObject(fooResourceUrl, request, User.class);

assertThat(user, notNullValue());
assertThat(user.getName(), is("kim"));
```

- ⌚ postForLocation 메소드
 - 생성된 리소스의 URI만 반환받음

```
HttpEntity<User> request = new HttpEntity<>(new User(1, "kim"));
URI location =
    restTemplate.postForLocation(fooResourceUrl, request);

assertThat(location, notNullValue());
```

REST 클라이언트

리소스 생성 - POST 방식

exchange 메소드

```
RestTemplate restTemplate = new RestTemplate();
HttpEntity<User> request = new HttpEntity<>(new User(1, "kim"));
ResponseEntity<User> response = restTemplate
    .exchange(resourceUrl, HttpMethod.POST, request, User.class);

assertThat(response.getStatusCode(), is(HttpStatus.CREATED));

User user = response.getBody();

assertThat(user, notNullValue());
assertThat(user.getName(), is("kim"));
```

REST 클라이언트 - HTTP Basic 인증

인증 정보를 미리 설정

• getInterceptors 메소드 사용

```
template = new RestTemplate();
String credentials =
    Base64.getEncoder()
        .encodeToString((username + ":" + password).getBytes());
template.getInterceptors().add((request, body, execution) -> {
    request.getHeaders().add("Authorization", "Basic " + credentials);
    return execution.execute(request, body);
});
```

에러처리

Exception Handling

REST 어플리케이션 예외처리

스프링 예외처리 방법

- ⌚ 전역처리 Global level - `@ControllerAdvice`
- ⌚ 컨트롤러단에서 처리 Controller level - `@ExceptionHandler`
- ⌚ 메소드 단위 처리 Method level - `try/catch`

@ControllerAdvice - 전역처리

- ☑ **@ControllerAdvice는 스프링 어플리케이션의 모든 예외를 처리**
 - 💡 **@RestController** 는 컨트롤러
 - 💡 **@ExceptionHandler** 어노테이션을 사용하여 예외를 처리할 클래스를 정의

```
@ControllerAdvice  
@RestController  
public class GlobalExceptionHandler {  
  
    @ResponseStatus(HttpStatus.BAD_REQUEST)  
    @ExceptionHandler(value = BaseException.class)  
    public String handleBaseException(BaseException e) {  
        return e.getMessage();  
    }  
  
    @ExceptionHandler(value = Exception.class)  
    public String handleException(Exception e) {  
        return e.getMessage();  
    }  
}
```

@ExceptionHandler 를 사용한 컨트롤러 단 예외처리

- ✓ HomeController.java 파일 내에 다음 코드가 있으면, 해당 컨트롤러의 모든 NumberFormatException 을 잡아서 처리한다.

```
@ExceptionHandler(value = NumberFormatException.class)
public String nfeHandler(NumberFormatException e){
    return e.getMessage();
}
```

비즈니스 예외 예제

명시적 예외처리가 필요없는 RuntimeException 타입으로 작성

```
public class ClientException extends RuntimeException {  
  
    private final int errorCode;  
    private final String errorDescription;  
  
    public ClientException(ApiError error) {  
        super(error.getErrorCode() + ": " + error.getDescription());  
        this.errorCode = error.getErrorCode();  
        this.errorDescription = error.getDescription();  
    }  
  
    public int getErrorCode() {  
        return errorCode;  
    }  
  
    public String getDescription() {  
        return errorDescription;  
    }  
}
```

RESTful 웹 서비스 문서화

Swagger

RESTful 웹 서비스 사용 설명서 만들기

- 서비스 entry point
- 각 리소스에 대한 경로
- 각 리소스에 대한 HTTP 메서드 - GET, POST, PUT, DELETE 같은
- 파라미터 목록
- 요청과 응답에 대한 데이터 양식
 - ⌚ JSON, XML 이나 TEXT
- API 에 의해 응답되는 상태코드나 에러메세지
- 기타 추가적인 REST API 사용 설명서
 - ⌚ 보안, 비즈니스 로직 등

API 문서화 툴

ApiDOC - <http://apidocjs.com/>

- 소스코드에 어노테이션으로 작성
- [Demo](#) 보기

Swagger - <https://swagger.io/>

- 어노테이션 기반으로 문서화 지원
- 자바와 Node 모두 지원

Doc 템플리트 for 웹서비스

- 구글독으로 제공되는 문서 템플리트
- [링크](#)

Swagger 설치

라이브러리 추가

```
<!-- Swagger -->
<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger2</artifactId>
    <version>2.9.2</version>
</dependency>

<dependency>
    <groupId>io.springfox</groupId>
    <artifactId>springfox-swagger-ui</artifactId>
    <version>2.9.2</version>
</dependency>
```

Swagger 설치

✓ Swagger Config 파일 생성

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {

    @Bean
    public Docket api() {
        return new Docket(DocumentationType.SWAGGER_2)
            .apiInfo(apiInfo())
            .select()
            .apis(RequestHandlerSelectors
                .basePackage("com.example.controller"))
            .paths(PathSelectors.any())
            .build();
    }
    private ApiInfo apiInfo() {
        return new ApiInfoBuilder()
            .title("user management")
            .description("swagger2 demo")
            .build();
    }
}
```

Swagger 접속

<http://localhost:8080/swagger-ui.html>

Swagger 어노테이션

컨트롤러 클래스 어노테이션

- `@Api(value = "main", description = "main controller")`

컨트롤러 메서드 어노테이션

- `@ApiOperation(value = "게시판 등록")`
- `@ApiImplicitParams({`
 `@ApiImplicitParam(name = "title", value = "제목", required = true,`
 `dataType = "string", paramType = "query", defaultValue = ""),`
 `@ApiImplicitParam(name = "content", value = "내용", required = true,`
 `dataType = "string", paramType = "query", defaultValue = "")`
})

모델 클래스 프러퍼티

- `@ApiModelProperty(notes = "파라미터1", required = true)`

Performance

HTTP 압축

Content negotiation

- ☞ 서버로 데이터를 요청할 때 클라이언트는 다양한 representation을 받을 수 있음
 - 데이터 타입 representation : DOC / PDF / HTML / TEXT
 - 언어 representation : 한글 / 영어 / 중국어
- ☞ 클라이언트와 서버 간의 동의가 필요
- ☞ 컨텐트 협상의 종류 두 가지
 - 서버 주도 (proactive content negotiation)
 - 클라이언트(에이전트) 주도 (reactive content negotiation)

HTTP 압축

Accept-Encoding

- ☞ 서버에게 리소스를 압축해서 전달 가능한지 물어봄
 - 주로 gzip 과 deflate 엔코딩 타입 사용

```
Accept-Encoding: gzip, deflate
```

```
Accept-Encoding: compress, gzip
```

```
Accept-Encoding:
```

```
Accept-Encoding: *
```

```
Accept-Encoding: compress;q=0.5, gzip;q=1.0
```

```
Accept-Encoding: gzip;q=1.0, identity; q=0.5, *;q=0
```

- ☞ 클라이언트가 요청한 압축 알고리즘을 제공할 수 없을 때
 - 406 (Not Acceptable)

HTTP 압축

Content-Encoding

- 서버에서 클라이언트로 전달되는 데이터의 압축 형식 표현
- 클라이언트에게 해당 형식으로 바디가 압축 되었다고 알려 줌

```
// Single Encoding option  
Content-Encoding: gzip  
Content-Encoding: compress
```

```
// Multiple Encoding options  
Content-Encoding: gzip, identity  
Content-Encoding: deflate, gzip
```

HTTP 캐싱

- ☑ 클라이언트는 같은 리소스(representation)을 여러번 요청할 수 있음

- ⌚ 리소스 재사용으로 성능 향상 기대
- ⌚ 캐싱은 웹 어플리케이션 성능향상의 가장 우선순위
- ⌚ 캐싱은 모든 레이어에 적용 가능

- ☑ HTTP 캐싱

- ⌚ 모든 브라우저에 HTTP 캐시 기능 포함

Spring Webflux

스프링5에서 소개되는 비동기 reactive 웹 프레임워크

Spring Webflux 소개

소개

- 기존의 스프링 프레임워크는 Servlet API와 Servlet 컨테이너 기반
- 새로운 Webflux 는 non-blocking, reactive stream 지원

왜 새로운 프레임워크가 필요한가?

- 효율적으로 동작하는 고성능 웹 애플리케이션 개발
 - Non-blocking 웹스택
- 함수형 프로그래밍
 - 자바5의 어노테이션과 자바8의 람다 표현식
 - WebFlux 는 자바8 기준 어노테이션 컨트롤러와 함수형 웹 엔드포인트 방식 지원
- 서비스 간 호출이 많은 마이크로서비스 아키텍처에 적합

Reactive : 무엇이고 왜 써야 하는가?

Reactive 시스템이란?

- 복수개의 서비스로 이루어진 분산 시스템이 정상 상황 뿐만 아니라 장애 상황에 서도 일관된 동작을 보장해주는 시스템
- Microservice 가 지향하는 시스템

이벤트에 반응하는 방식

- 네트워크 컴포넌트가 IO 이벤트에 반응
- Non-blocking 으로 처리하는 방식이 가능해짐

Reactive API

- **Reactor** : 스프링 WebFlux 에서 사용
- RxJava : Reactive-X 라이브러리의 자바 버전

Reactive Programming 이란?

비동기 데이터 스트림으로 프로그래밍 하는 것

Programming with asynchronous data streams

WebFlux 프로그래밍 모델

스프링 WebFlux 프로그래밍 모델 두 가지

• Annotated Controller

- 기존 스프링 MVC 모델 방식과 동일
- 다른 점은 WebFlux 는 반응형 @RequestBody 파라미터 지원

• Functional Endpoints

- 람다식 기반의 가벼운 함수형 모델
- 가벼운 라우팅 기능과 요청을 처리하는 라이브러리
- 콜백형태로 요청이 있을 때만 호출됨

새로운 요청 / 응답

- 서블릿 API 는 리액티브 함수형 스타일에 적합하지 않음
 - HttpServletRequest / HttpServletResponse
- ServerRequest / ServerResponse

WebFlux 구조

@Controller, @RequestMapping

Router Functions

spring-webmvc

spring-webflux

Servlet API

HTTP / Reactive Streams

Servlet Container

Tomcat, Jetty, Netty, Undertow

WebFlux 프로그래밍 모델

Annotated Controller

- ▶ 스프링 MVC 모델 기반의 기존 spring-web 모듈과 같은 방식으로 구성 가능
- ▶ 스프링 MVC에서 제공하는 어노테이션 그래도 사용 가능

Functional Endpoints

- ▶ 자바 8 람다 스타일 routing 과 handling 방식
- ▶ 가벼운 routing 기능과 request 처리 라이브러리
- ▶ Callback 형태로 요청이 있을 때만 호출 됨

Flux and Mono

Spring5 Reactor

스프링 5 WebFlux :: Reactor

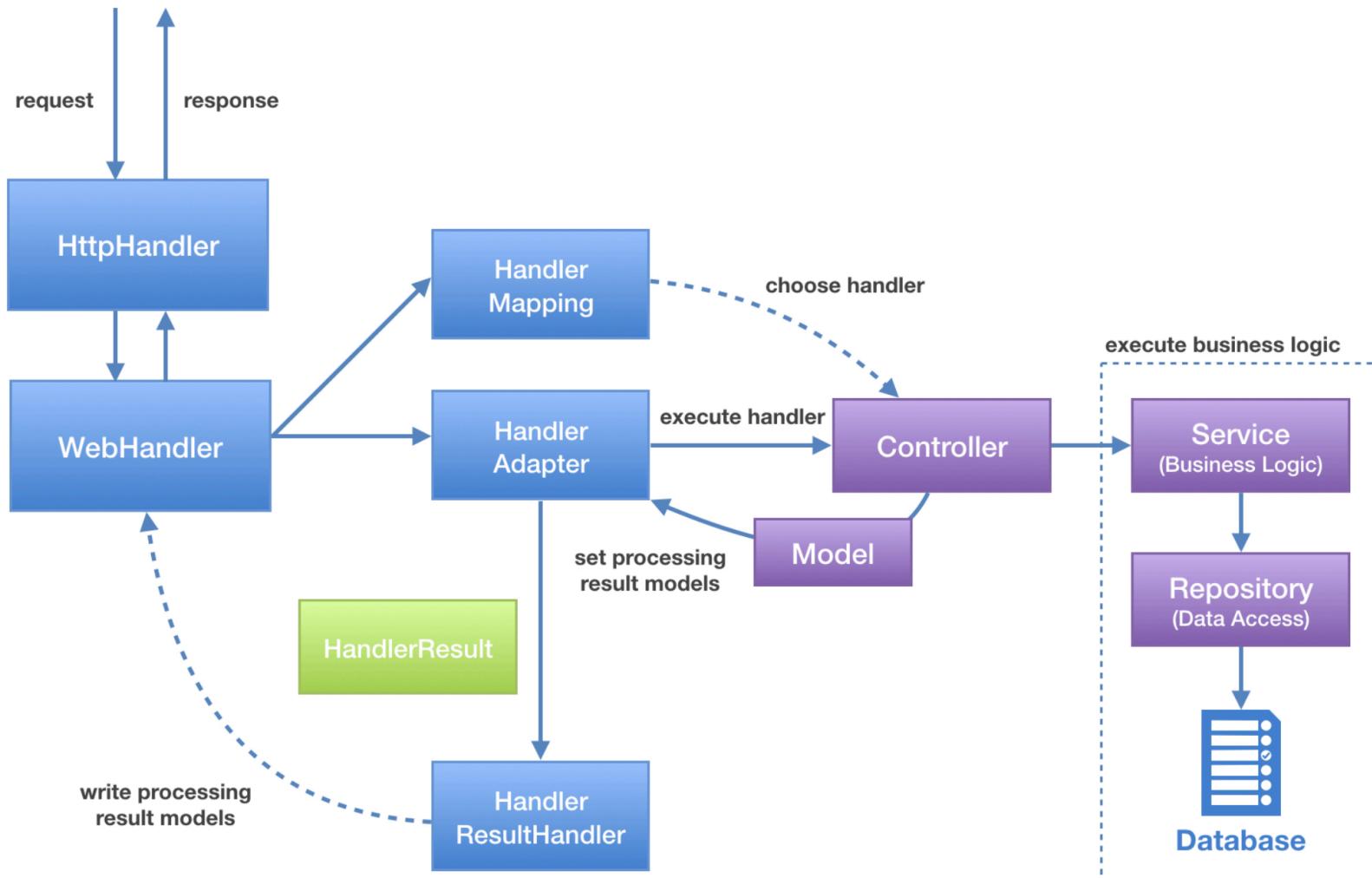
Reactor

- Pivotal 의 오픈소스 프로젝트로, JVM 위에서 동작하는 논블럭킹 애플리케이션 작성을 위한 리액티브 라이브러리
- RxJava 2 와 함께 Reactive Stream의 구현체
- 스프링 프레임워크 5 부터 리액티브 프로그래밍을 위해 지원하는 라이브러리
- 최소 자바8 이상에서 지원

Reactor 의 두 가지 타입 - Publisher 구현체

- Flux : 0..N 개인 여러 개의 결과를 처리하는 객체
- Mono : 0..1 개의 결과만을 처리하는 객체

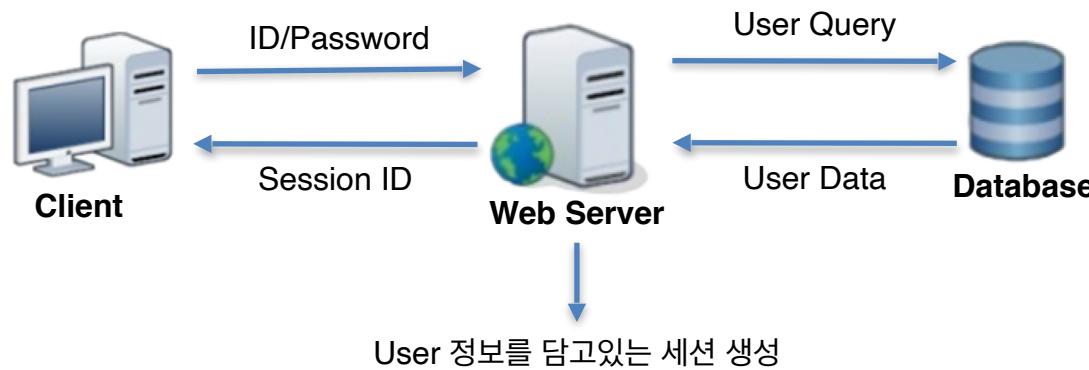
WebFlux 동작흐름



OAuth 2.0 인증

기존 인증체계와 RESTful API의 인증

☑ 기존 Web Application 의 인증 체계



기존 인증체계와 RESTful API의 인증

기존 Web Application 의 인증 체계의 단점

- ⌚ ID / Password가 노출되기 쉽다.
- ⌚ 해킹 또는 개인 정보 유출을 막기 위해 정기적인 비밀번호 갱신이 필요하다.
- ⌚ ID / Password 만으로 Application의 거의 모든 권한을 가진다.

기존 인증체계와 RESTful API의 인증

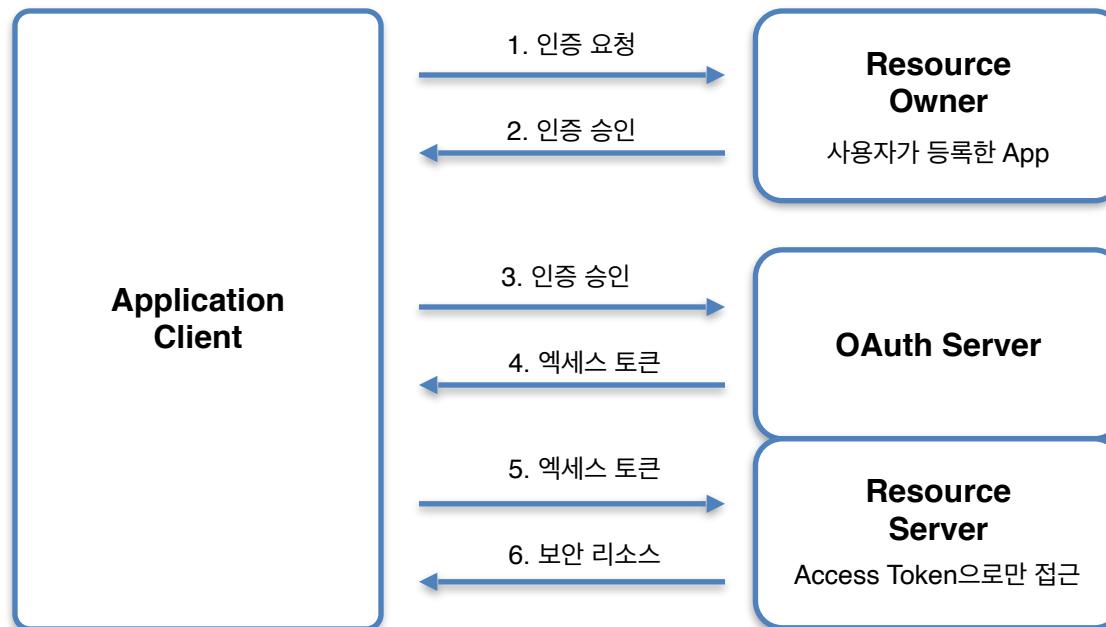
기존 Web Application 의 인증 체계의 단점 제거

- ID / Password가 노출되기 쉽다.
 - → ID / Password를 만들지 않는다.
 - → 사용자별로 Token을 임시 생성.
- 해킹 또는 개인 정보 유출을 막기 위해 정기적인 비밀번호 갱신이 필요하다.
 - → Password를 만들지 않아, 비밀번호의 갱신이 필요 없다.
- ID / Password 만으로 Application의 거의 모든 권한을 가진다.
 - → Application의 사용을 위한 최소한의 권한을 부여한다.
- Resource 혹은 API에 접근하기 위해 HTTP Header를 통한 2중 인증을 사용한다.

RESTful API → OAuth 2.0

OAuth 2.0

- ✓ Pivotal, Google, Amazon, Facebook, Twitter 등의 회사에서 채택한 인증에 관련된 “공개 표준 명세”.
 - ID / Password 대신, Client ID와 Secret Key로 접근 권한을 부여함.
 - Resource에 대한 접근이 필요할 때, Access Token을 발급받아 접근함.



MSA - Micro Service Architecture

- Monolithic 아키텍처
- 마이크로서비스 아키텍처
- 마이크로서비스 컴포넌트와 툴

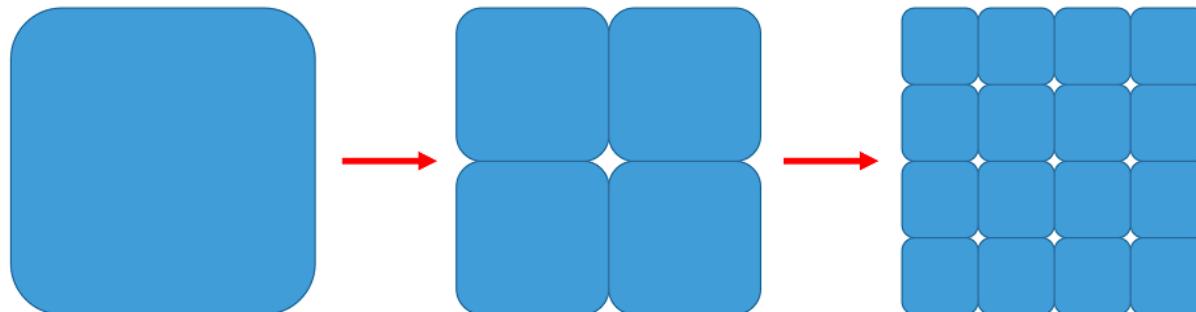
Monolithic 서비스의 단점

- 코드 변경시 전체 어플리케이션 배포 필요
- 성능 이슈 발생 시 발견 및 수정이 어려움
- 시스템 재 기동시 느림
- 의존 라이브러리 충돌이 있을 수 있음
- 스캐일링의 어려움
- 어플리케이션 성장 시 복잡성 제어가 어려움
- 각 서비스가 강결합 되어 있어 하나의 서비스 업그레이드가 어려움

MSA from Wikipedia

- 마이크로서비스는 어플리케이션을 느슨하게 결합된 서비스들의 모음으로 구조화 하는 서비스지향 아키텍처(SOA) 구조화 방식의 변형이다.

- 서비스들의 느슨한 결합
 - 서비스 독립성
 - 배포 독립성
 - 데이터 독립성



マイクロサービス

각 서비스가 독립적

- 서비스 별 독립 개발 및 독립 디플로이
- 서비스 업그레이드 용이 - 개별 업그레이드
- 서비스들이 약결합되어 스캐일링이 쉬움

에러에 강함

자동화

- CI / CD 가 쉬워져서 빠른 업그레이드 보장

상태를 가지지 않음

マイクロ 서비스 구성 요소

Microservice Components

Configuration server

- 모든 서버들의 설정을 가지고 있음, 설정 변경 시 재 기동없이 즉시 반영됨

Load balancer

- Service discovery에게 정보를 받아 전 서비스의 부하를 분산 관리
- Netflix Ribbon

Service discovery

- 실시간 스캐일링 되는 서비스 인스턴스에 대한 정보를 가지고 있음

Circuit breaker

- 특정 서비스 장애 시 다른 서비스의 장애를 보호해 주는 역할을 함

Edge server

- API Gateway 가 발전된 형태
- 외부 세계와 서비스들의 경계
- Netflix Zuul

マイクロ 서비스 관련 라이브러리

Netflix Eureka

マイクル サービス ライブリ
 マイティア ロード バランサー

Netflix Zuul

Spring Cloud Config Server

Netflix Ribbon

Spring Cloud Netflix

Spring Security OAuth2

Netflix Hystrix and Turbine

Eclipse Microprofile

API 설계 유의점

RESTful API 는 de-facto standard
효과적인 설계를 위한 가이드라인을 알아보자!

문자열과 구조 규칙

/ 를 이용하여 계층 구조를 표현

- 슬래시는 리소스의 위치에 대한 계층적인 정보를 표현할 때 사용

```
http://www.abc.com/text/readme.html
```

/ 는 URI 마지막에 붙이지 않음

- URI 가 슬래시로 끝나게 하지 않음
- 아래 두 URI는 웹 브라우저에서는 같다고 인식 하지만 엄연히 다름

```
http://www.abc.com/text/  
http://www.able.com/text
```

문자열과 구조 규칙

- 하이픈은 가독성을 높이지만 언더라인은 사용하지 않음
 - 두 단어 이상된 문자열은 공백을 하이픈으로 대체하여 사용

```
http://www.abe.com/big-text/book1.html (o)
```

```
http://www.abc.com/big_text/book1.html (x)
```

- 문자열은 소문자를 사용하고 확장자는 사용하지 않음
 - URI의 문자열은 대소문자를 구분함 (스키마 정보와 호스트 제외)
 - Camel 표기법 같은 대소문자 혼합형 네이밍은 적합하지 않음

```
http://abc.com/BigText/book1.json
```

```
http://abc.com/bigText/book1.json
```

```
http://abc.com/bigtext/book1
```

리소스 설계 규칙 - 리소스의 종류

Document - 데이터베이스의 레코드와 유사한 개념

```
http://api.api.com/leagues/seattle  
http://api.api.com/leagues/seattle/teams/trebuchet  
http://api.api.com/leagues/seattle/teams/korea/players/mike
```

Collection - 서버에서 관리하는 디렉토리 구조 (복수형으로 표현)

```
http://api.abc.com/books  
http://api.abc.com/books/categories  
http://api.abc.com/categories/it/books
```

Store - 클라이언트에서 관리하는 리소스 저장소

```
PUT /users/1234/books/iot
```

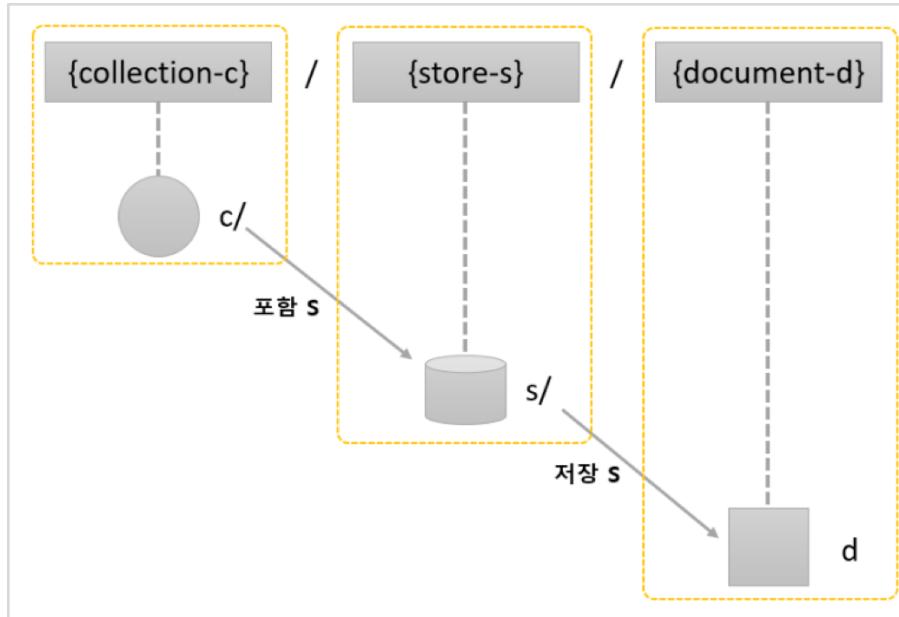
Controller - 실행 가능한 개념 (입력과 출력 값을 가짐)

```
POST /alerts/245743/resend
```

리소스 설계 규칙 - 경로 규칙

경로 규칙

- 奴隶(/)로 구분된 각 URI 경로 부분은 다양한 형태로 설계 가능
- 경로 각 부분에 의미있는 값들을 부여, 계층적 구조를 표현



URI 경로 설계 시 유의 해야하는 규칙

- 도큐먼트 이름은 단수를 사용한다.

```
http://api.abc.com/categories/it/books/apache
```

- 컬렉션은 복수를 사용한다.

```
http://api.abc.com/categories/it/books/apache/chapters
```

- 스토어의 이름은 동사를 사용한다.

```
http://api.abc.com/categories/move  
http://api.abc.com/categories/reindex  
http://api.abc.com/categories/run-test
```

URI 경로 설계 시 유의 해야하는 규칙

- ☑ 경로상의 변화되는 부분은 유일한 값을 사용한다.
 - 💡 변수를 실제 값으로 대체

```
http://api.abc.com/categories/{categoryId}/books/{bookId}
```

- ☑ CRUD 기능은 URI에 사용하지 않는다.
 - 💡 URI는 리소스 식별 용도로만 사용됨
 - 💡 HTTP 메서드로 CRUD 기능을 표현할 수 있음

```
DELETE /users/1234
```

```
GET /deleteUser?id=1234  
GET /selectUser?id=1234  
DELETE /deleteUser=1234  
POST /users/1234/delete
```

잘못된 표현

Query String 규칙

- RFC 3986 - URI 쿼리는 선택사항이고, Fragment 사이에 위치

```
scheme “://” authority “/” path [ “?” query ] [ “#” fragment ]
```

```
http://api.abc.com/students/morgan/send-sms  
http://api.abc.com/students/morgan/send-sms?text=hello
```

- URI의 쿼리로 컬렉션, 스토어를 필터링

```
GET /users  
GET /users?role=admin
```

- 컬렉션이나 스토어의 결과를 페이지ing하여 사용

```
GET /users?pageSize=25&pageStartIndex=50
```

```
POST /users/search
```

응답 상태 코드

구분	설명
1xx	정보 전송 프로토콜 수준의 정보를 교환
2xx	성공 클라이언트 요청이 성공적으로 수행되었을 경우 반환된다.
3xx	재전송 클라이언트는 요청을 완료하기 위해 추가적인 행동을 취해야 하는 경우에 반환된다.
4xx	클라이언트 오류 클라이언트의 잘못된 요청일 경우 반환된다.
5xx	서버오류 서버 쪽의 오류로 인한 메시지 반환인 경우

API 문서화

Swagger

swagger Select a spec v1 ▾

catalog service 1.0.0

[Base URL: localhost:5000]
</swagger/v1/swagger.json>

Schemes
HTTP ▾

Artist

GET /api/artists

POST /api/artists

GET /api/artists/{id}

GET /api/artists/{id}/items

Genre

Items

ItemsHateoas

Users

API 테스팅

Postman

Postman 을 사용한 서비스 테스트

- API 테스트(query)를 위한 최고의 툴
- 개발자와 관련 팀에서 모두 사용 되어짐

📢 서비스 개발 , 테스트, 모니터링 용

The screenshot shows the Postman application interface. On the left, there's a sidebar with a 'Collections' tab selected, displaying a tree structure of API endpoints. A collection named 'Artist Get By Id' is expanded, showing a single GET request for 'Artist Get By Id' with a base URL of {{baseUrl}}/api/artists/:id. The 'Params' tab is active, showing a parameter 'id' with a value of 8f33f15a-a5db-45f7-1fc9-08d682a5dfd6. The 'Body' tab shows the response body in Pretty JSON format:

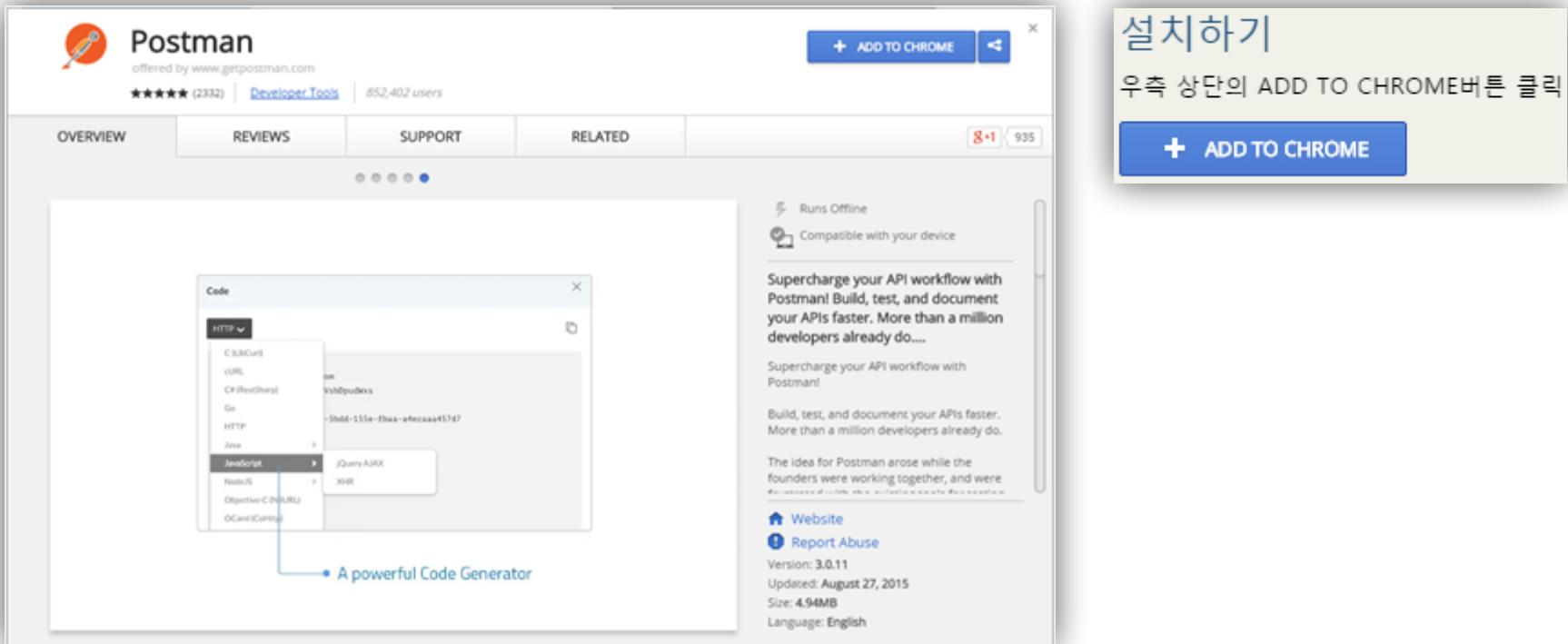
```
1 = [{  
2   "artistId": "8f33f15a-a5db-45f7-1fc9-08d682a5dfd6",  
3   "artistName": "myArtist"  
4 }]
```

The main interface includes tabs for 'Send', 'Save', and 'Download', and status information at the bottom: Status: 200 OK, Time: 335 ms, Size: 248 B.

Postman :: REST 클라이언트

Postman: REST API 테스트하는 Chrome 확장 프로그램 설치

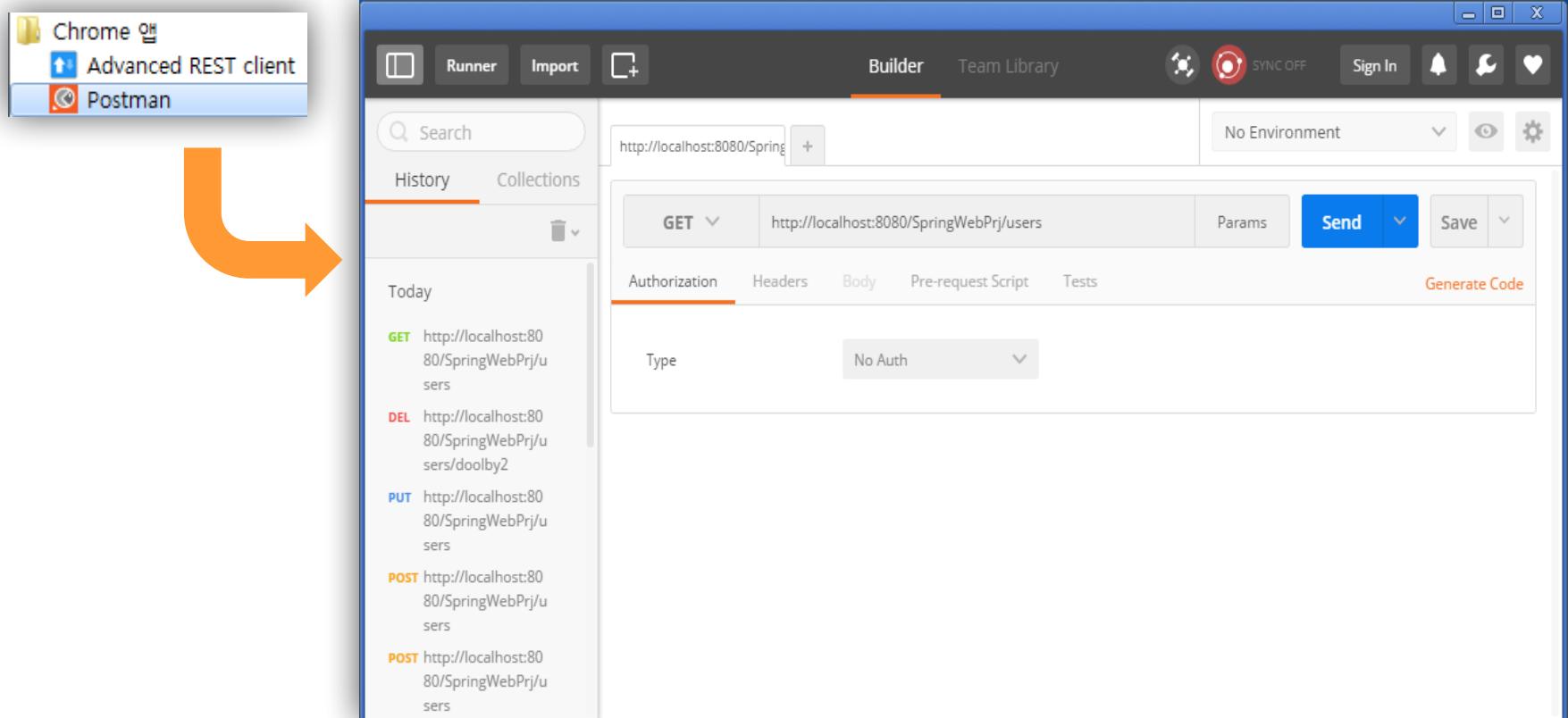
<https://chrome.google.com/webstore/detail/postman/fhbjgbiflinjbdggehcdcbncddomop>



The screenshot shows the Google Chrome Web Store page for the Postman extension. The page includes the Postman logo, developer information (offered by www.getpostman.com), user reviews (4.5 stars, 2332 reviews), developer tools, and related extensions. A large central image displays the Postman interface, specifically a code generator window for generating various API requests (e.g., C URLs, Python, Go, Java, JavaScript, Node.js, Objective-C, OCaml). A callout points to this window with the text "A powerful Code Generator". To the right of the main image is a sidebar with the title "설치하기" (Install) and the instruction "우측 상단의 ADD TO CHROME버튼 클릭" (Click the ADD TO CHROME button at the top right). A prominent blue "ADD TO CHROME" button is located in this sidebar.

Postman :: REST 클라이언트

Postman 실행





THANK YOU