



# Spring Boot Spring Testing RESTful API





Somkiat Puisungnoen

Search

Somkiat | Home

Update Info 1 View Activity Log 10+ ...

Timeline About Friends 3,138 Photos More

When did you work at Opendream? X

... 22 Pending Items

Post Photo/Video Live Video Life Event

What's on your mind?

Public Post

Intro

Software Craftsmanship

Software Practitioner at สยามชานาญกิจ พ.ศ. 2556

Agile Practitioner and Technical at SPRINT3r

Somkiat Puisungnoen 15 mins · Bangkok · ...

Java and Bigdata



Facebook somkiat.cc

Page Messages Notifications 3 Insights Publishing Tools Settings Help ▾

somkiat.cc  
@somkiat.cc

Home Posts Videos Photos

Liked Following Share ... + Add a Button



**[https://github.com/up1/  
course-springboot-2024](https://github.com/up1/course-springboot-2024)**



# Spring Boot



# Agenda

REST (REpresentational State Transfer)

Introduction to Spring Boot

Goals of Spring Boot

Better project structure of Spring Boot

Develop RESTful APIs

Error handling

Layer in Spring Boot project

Working with Spring Data (JPA and JDBC)



# Agenda

Manage transaction

Secure service with Spring Security

Observable service

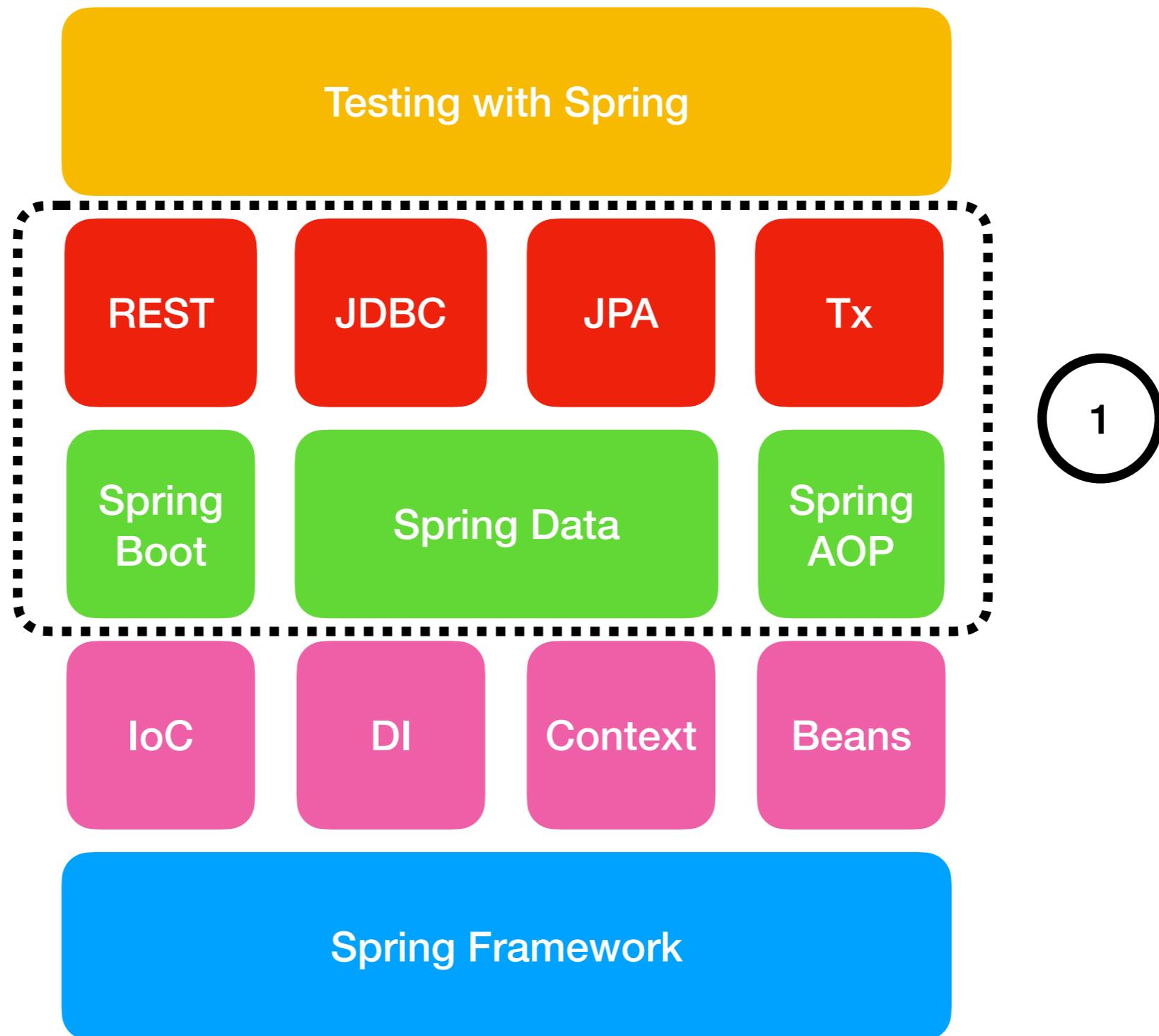
Application metrics

Distributed tracing

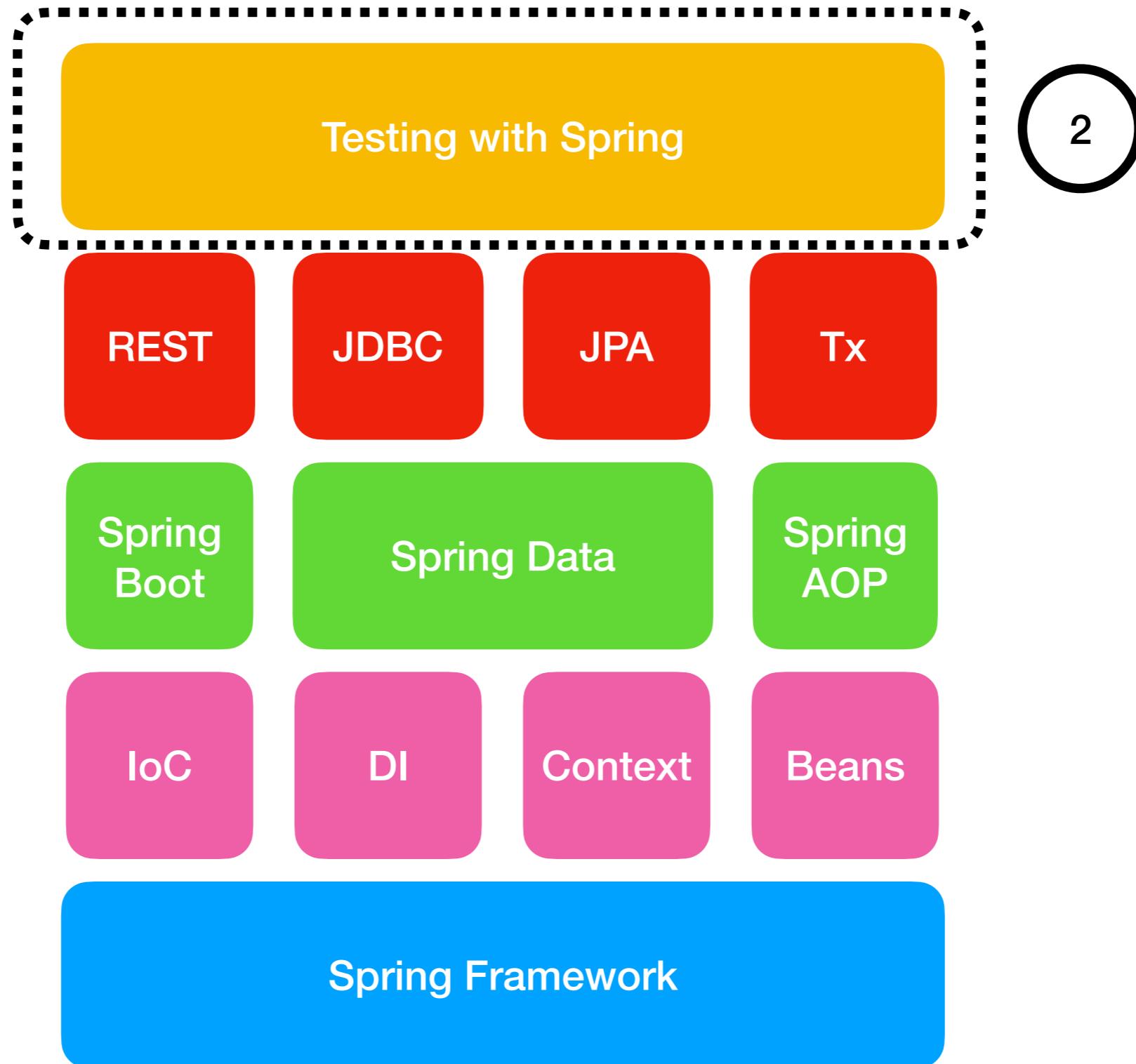
Centralized logging



# Workshop



# Workshop



# My Rule

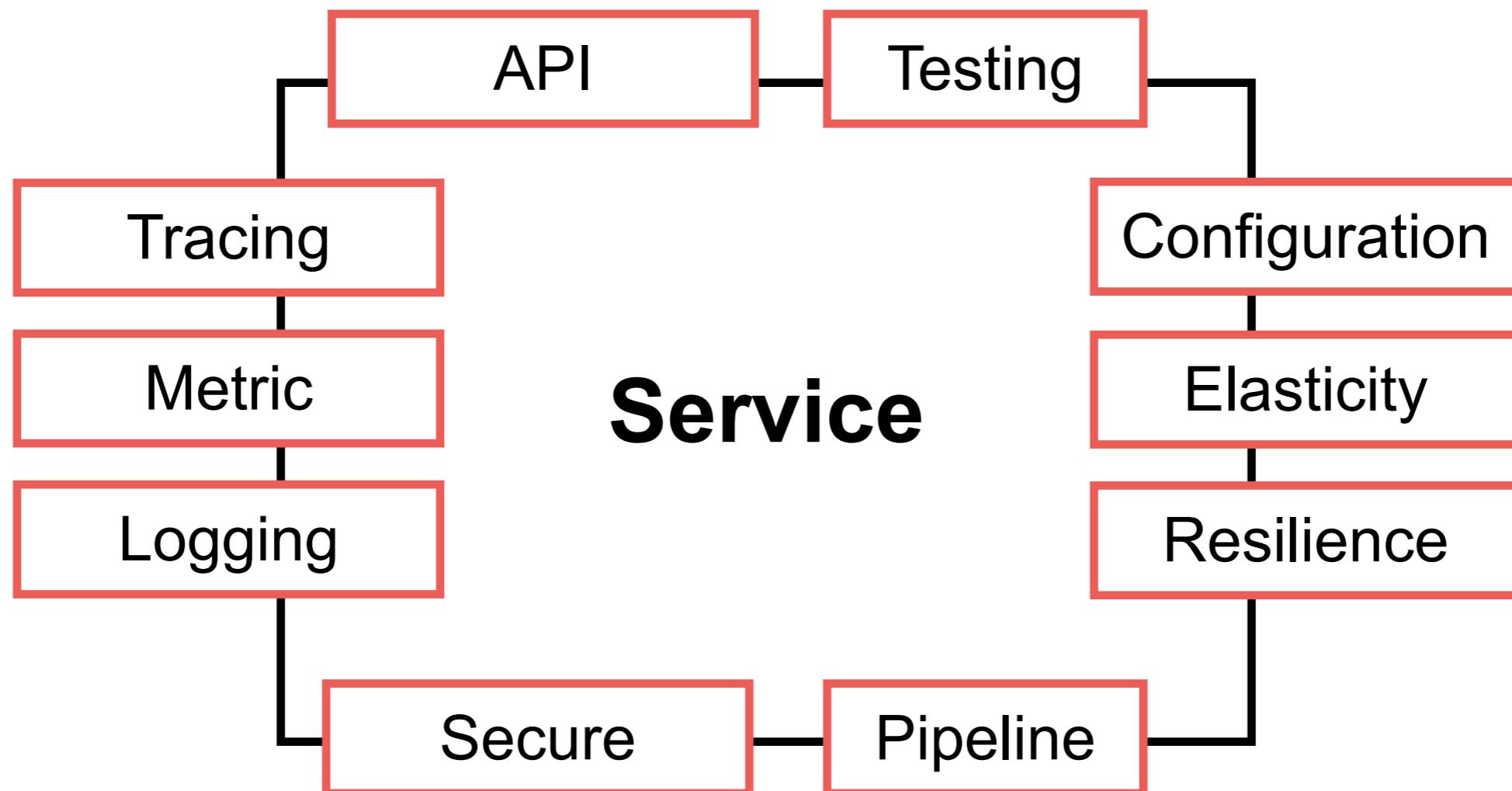
**Always write tests**



# Let's start



# Properties of Service



# APIs

## Application Programming Interface

**REST API**

gRPC

Messaging



# Develop REST API



<https://spring.io/projects/spring-boot>



# Spring Framework



# Spring Boot



# Create Spring Boot project

The screenshot shows the Spring Initializr web interface for creating a new Spring Boot project. The configuration is as follows:

- Project:** Maven (selected)
- Language:** Java (selected)
- Spring Boot:** 3.3.2 (selected)
- Project Metadata:**
  - Group: com.example
  - Artifact: demo
  - Name: demo
  - Description: Demo project for Spring Boot
  - Package name: com.example.demo
  - Packaging: Jar (selected)
  - Java: 21 (selected)
- Dependencies:**
  - Spring Web** (selected): Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.
  - Spring Data JPA**: Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.
  - H2 Database**: Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

<https://start.spring.io/>



# REST

REpresentation State Transfer

The style of **software architecture** behind  
RESTful services

Defined in 2000 by Roy Fielding

[https://www.ics.uci.edu/~fielding/pubs/dissertation/rest\\_arch\\_style.htm](https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm)



# Goals

Scalability

Generality of interfaces

Independent deployment of components



# REST properties

Performance

Scalability

Simplicity

Modification

Visibility

Portability

Reliability



# REST constraints

Client-server

Stateless

Cacheable

Layer system

Uniform  
interface

Code on  
demand

eg. JavaScript



# **RESTful service**

Implementation from REST



# REST Request Messages

RESTful request is typically in form of  
**Uniform Resource Identifiers (URI)**



# REST Request Messages

RESTful request is typically in form of  
**Uniform Resource Identifiers (URI)**

Structure of URI depend on specific service



# REST Request Messages

RESTful request is typically in form of  
**Uniform Resource Identifiers (URI)**

Structure of URI depend on specific service

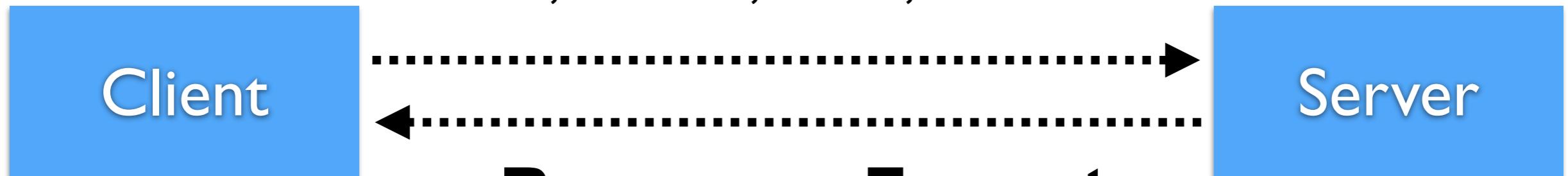
Request can include parameter and data in  
body of request as XML, JSON etc.



# REST Request & Response

**Request Method**

GET, POST, PUT, DELETE



**Response Format**

XML or JSON



# Status Code

Code	Description
1xx	Information
2xx	Successful
3xx	Redirection
4xx	Client error
5xx	Server error

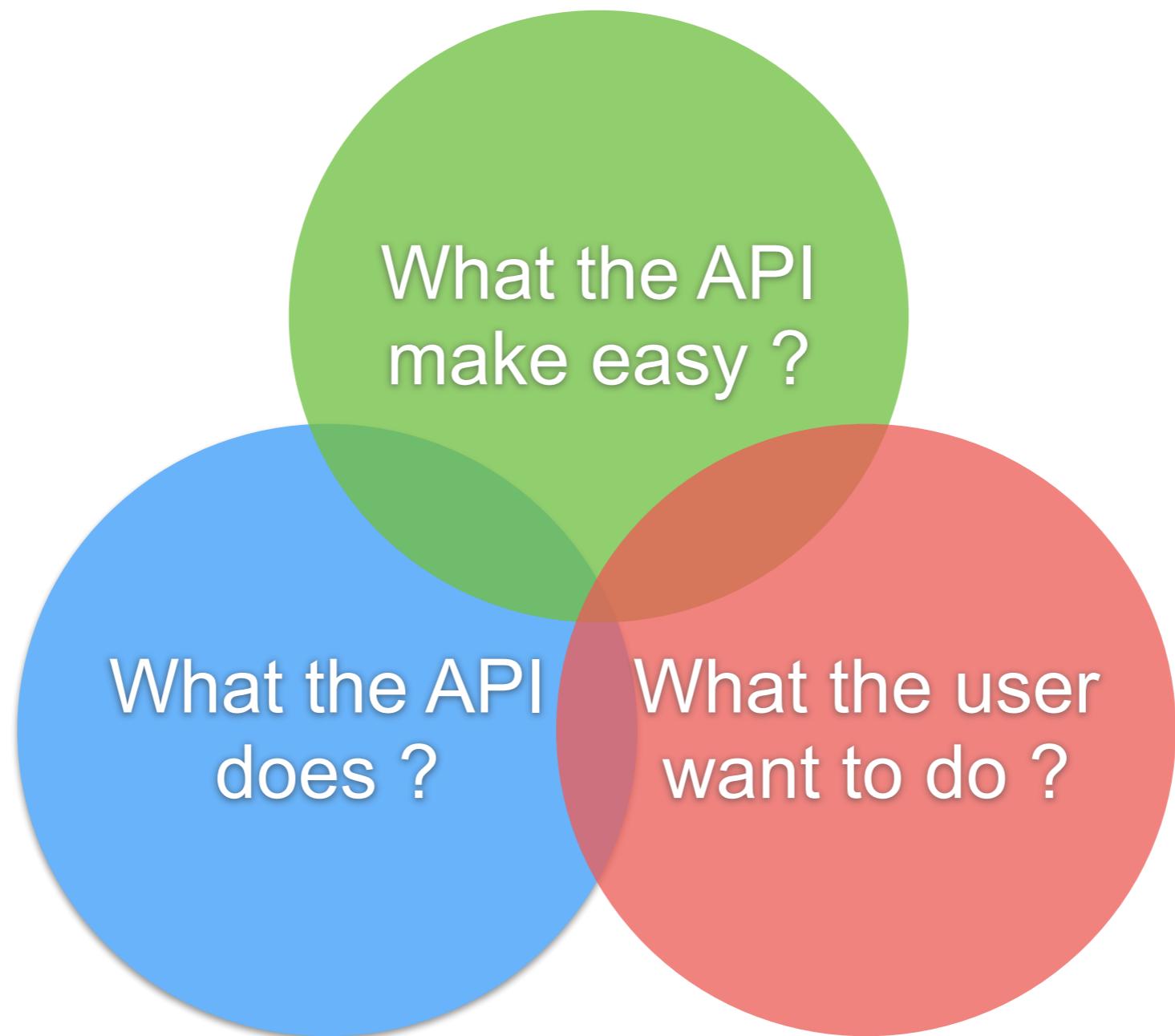
<https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>



# Response format ?



# Good APIs ?



# Principles of API Design



# Principles of API Design

Consistency

Statelessness

Resource-oriented design

Use standard  
HTTP methods

Versioning

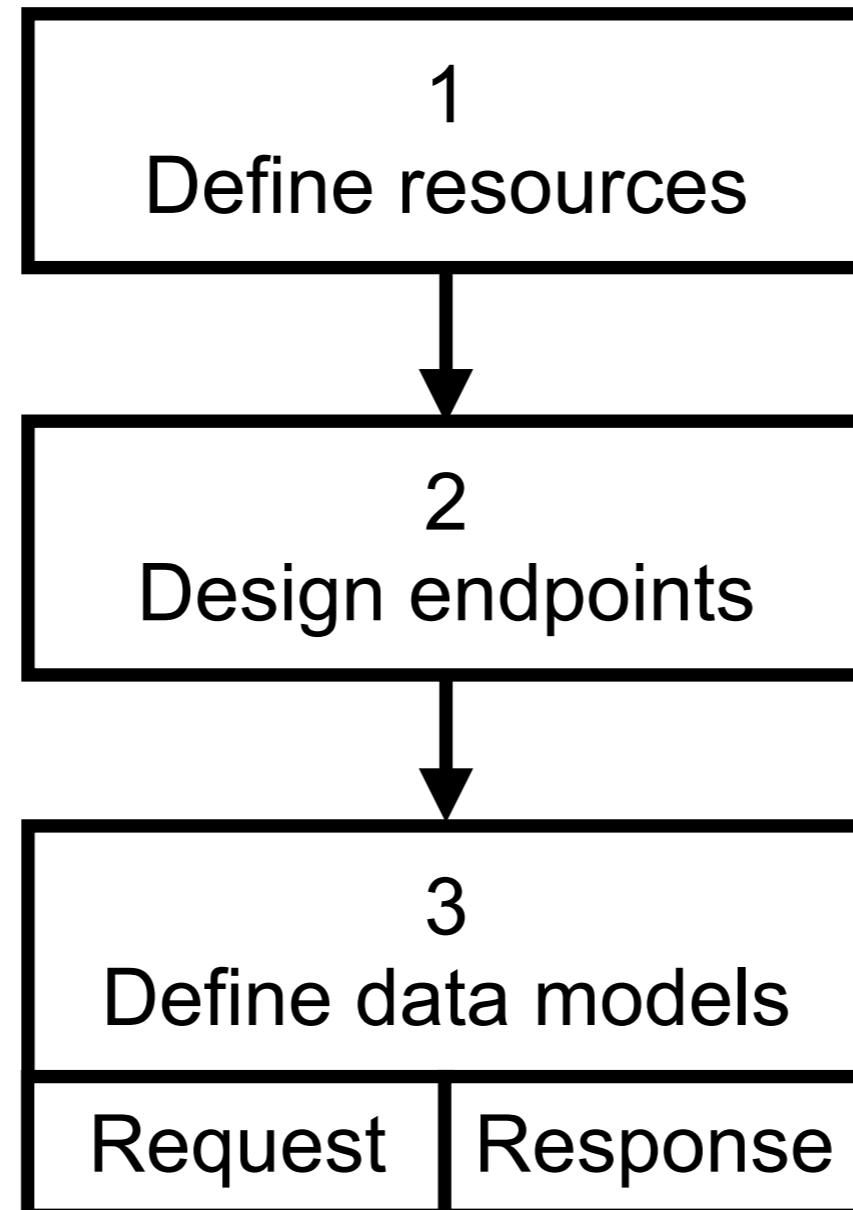


# HTTP Methods meaning

Method	Meaning
GET	Read data
POST	Create/Insert new data
PUT/PATCH	Update data or insert if a new id
DELETE	Delete data



# Steps to Design APIs



# More !!

Authentication  
Authorization

Pagination and  
filtering

Rate Limiting

Pagination and  
filtering

Error handling

Documentation

Testing

Monitoring and  
Observability



# Implementation

Coding

Testing



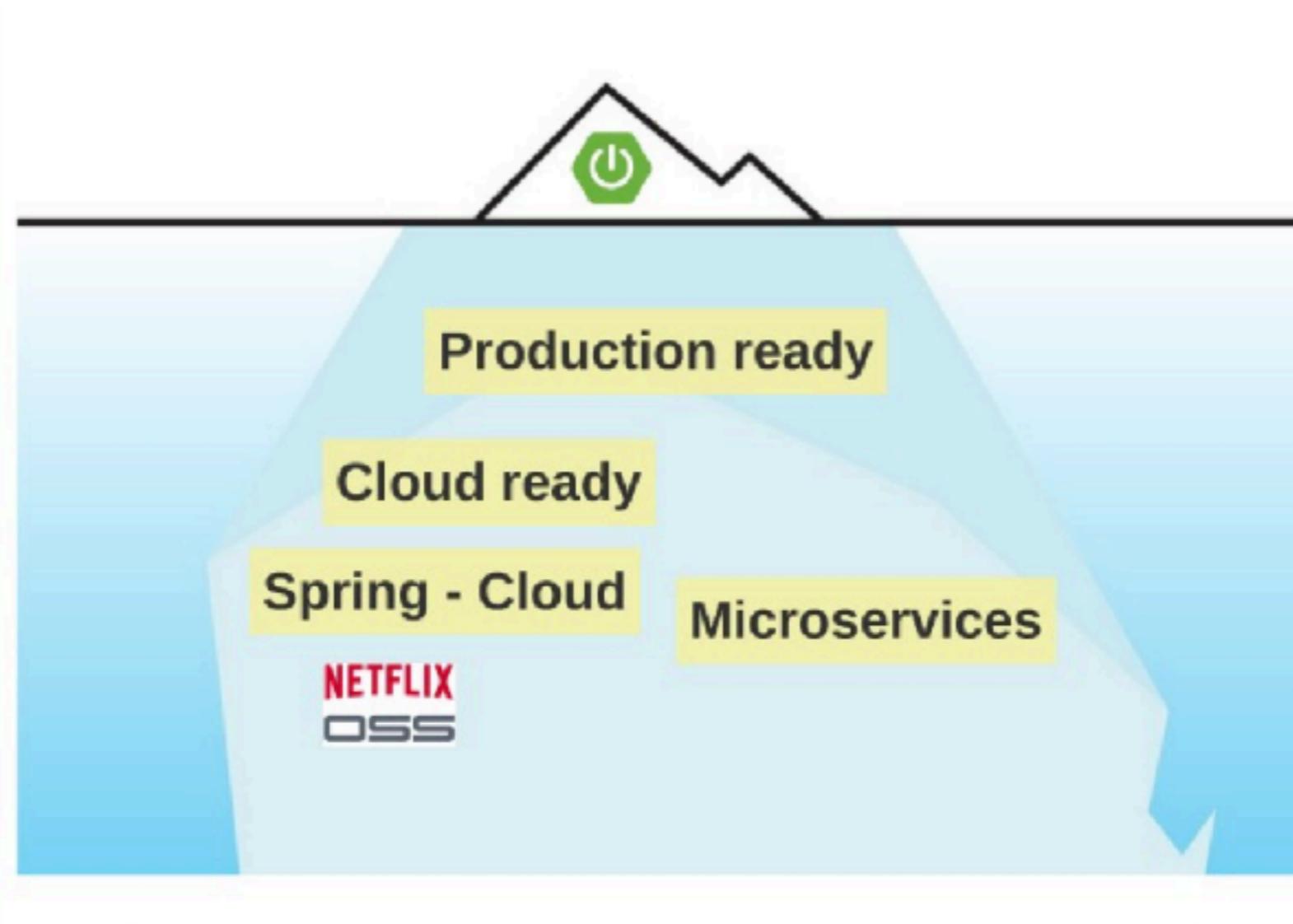
# **Spring Boot**

## **3.3.2 (current)**



# Why Spring Boot ?

Application skeleton generator  
Reduce effort to add new technologies



# What ?

Embedded application server

Integration with tools/technologies (starter)

Production tools (monitoring, health check)

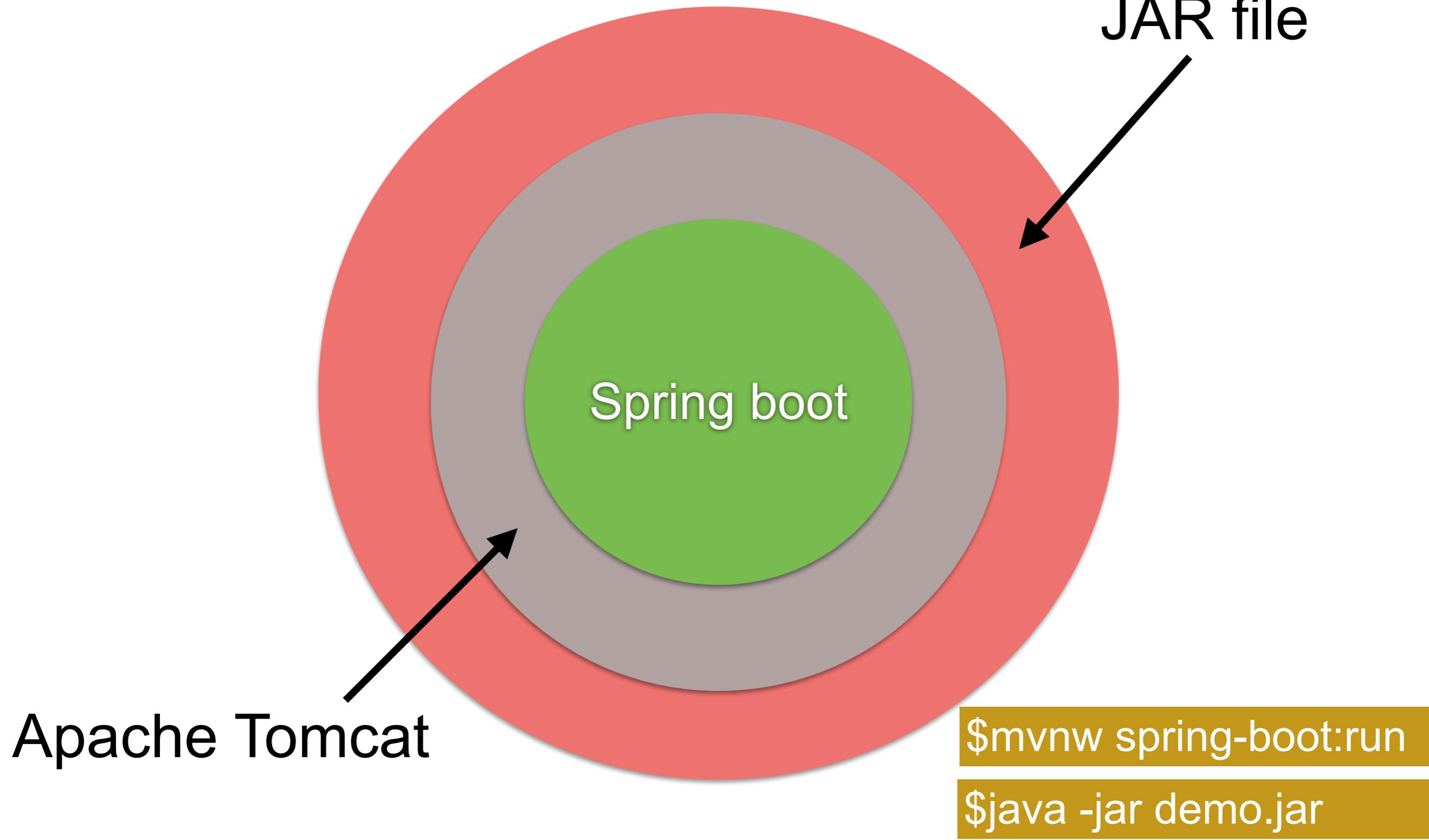
Configuration management

Dev tools

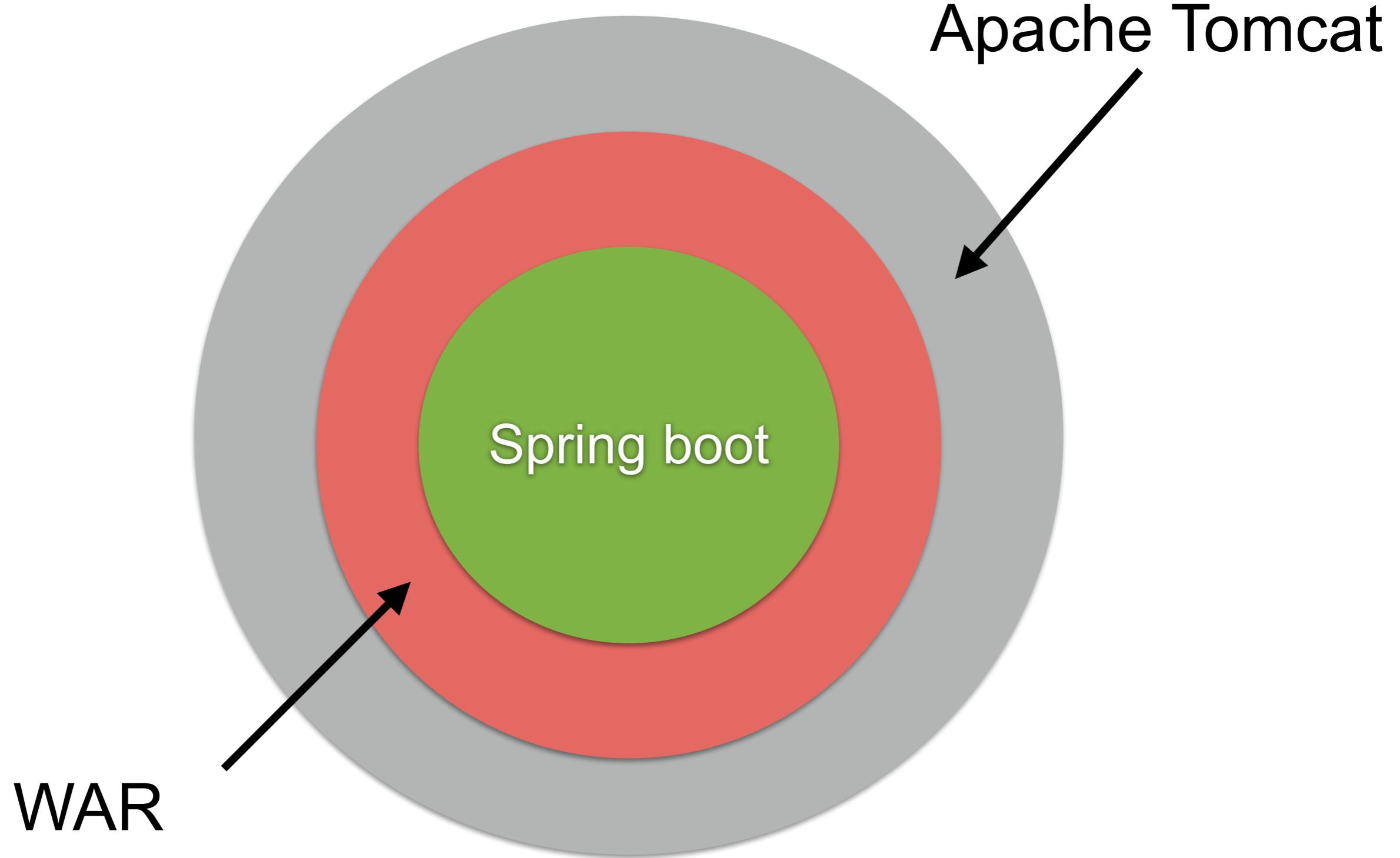
No source code generation, no XML



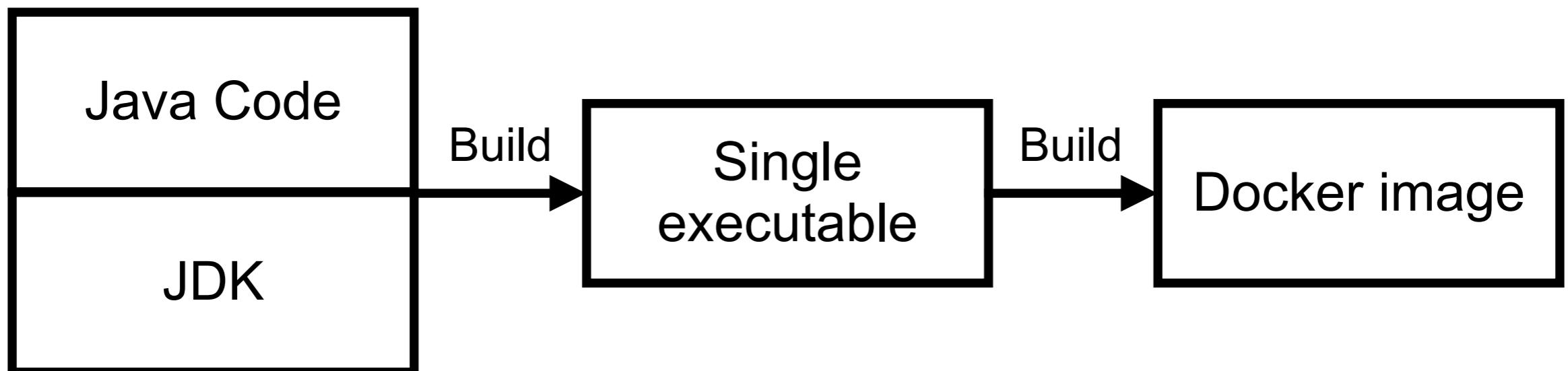
# How ?



# Traditional Deployment



# GraalVM Native Support



<https://www.graalvm.org/>



# Add to Spring Boot project



The screenshot shows the Spring Initializr web interface. On the left, there are sections for Project (Gradle - Groovy selected), Language (Java selected), and Spring Boot (3.1.4 selected). On the right, a red dashed box highlights the 'Dependencies' section, which includes 'GraalVM Native Support' (selected) under 'DEVELOPER TOOLS'. Below it, a note says 'Support for compiling Spring applications to native executables using the GraalVM native-image compiler.' At the bottom, the title '1. GraalVM support' is displayed in large red text.

Project

- Gradle - Groovy
- Gradle - Kotlin
- Maven

Language

- Java
- Kotlin
- Groovy

Spring Boot

- 3.2.0 (SNAPSHOT)
- 3.2.0 (M3)
- 3.1.5 (SNAPSHOT)
- 3.1.4
- 3.0.12 (SNAPSHOT)
- 3.0.11
- 2.7.17 (SNAPSHOT)
- 2.7.18

Project Metadata

Group: com.example

Artifact: demo

Name: demo

Description: Demo project for Spring Boot

Package name: com.example.demo

Packaging:  Jar  War

Java:  21  17  11  8

Dependencies

**GraalVM Native Support** DEVELOPER TOOLS

Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

## 1. GraalVM support

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



# **Building RESTful API with Spring Boot**



# Create new project



Project  
 Gradle - Groovy  Gradle - Kotlin  
 Maven

Language  
 Java  Kotlin  Groovy

Spring Boot  
 3.2.0 (SNAPSHOT)  3.2.0 (M2)  3.1.4 (SNAPSHOT)  3.1.3  
 3.0.11 (SNAPSHOT)  3.0.10  2.7.16 (SNAPSHOT)  2.7.15

Project Metadata  
**Group**: com.example  
**Artifact**: demo  
**Name**: demo  
**Description**: Demo project for Spring Boot  
**Package name**: com.example.demo  
**Packaging**:  Jar  War  
**Java**:  20  17  11  8

## Dependencies

[ADD DEPENDENCIES...](#) 36 + 3

### Spring Web WEB

Build web, including RESTful applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

### H2 Database SQL

Provides a fast in-memory database that supports JDBC API and H2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

### Spring Data JPA SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

1. Spring Web
2. Spring Data JPA
3. H2 Database



# Run project (Dev mode)

**./mvnw spring-boot:run**

```
2018-06-07 13:03:30.412  INFO 12828 --- [  
oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
oApplication           : No active profile set, fall
```



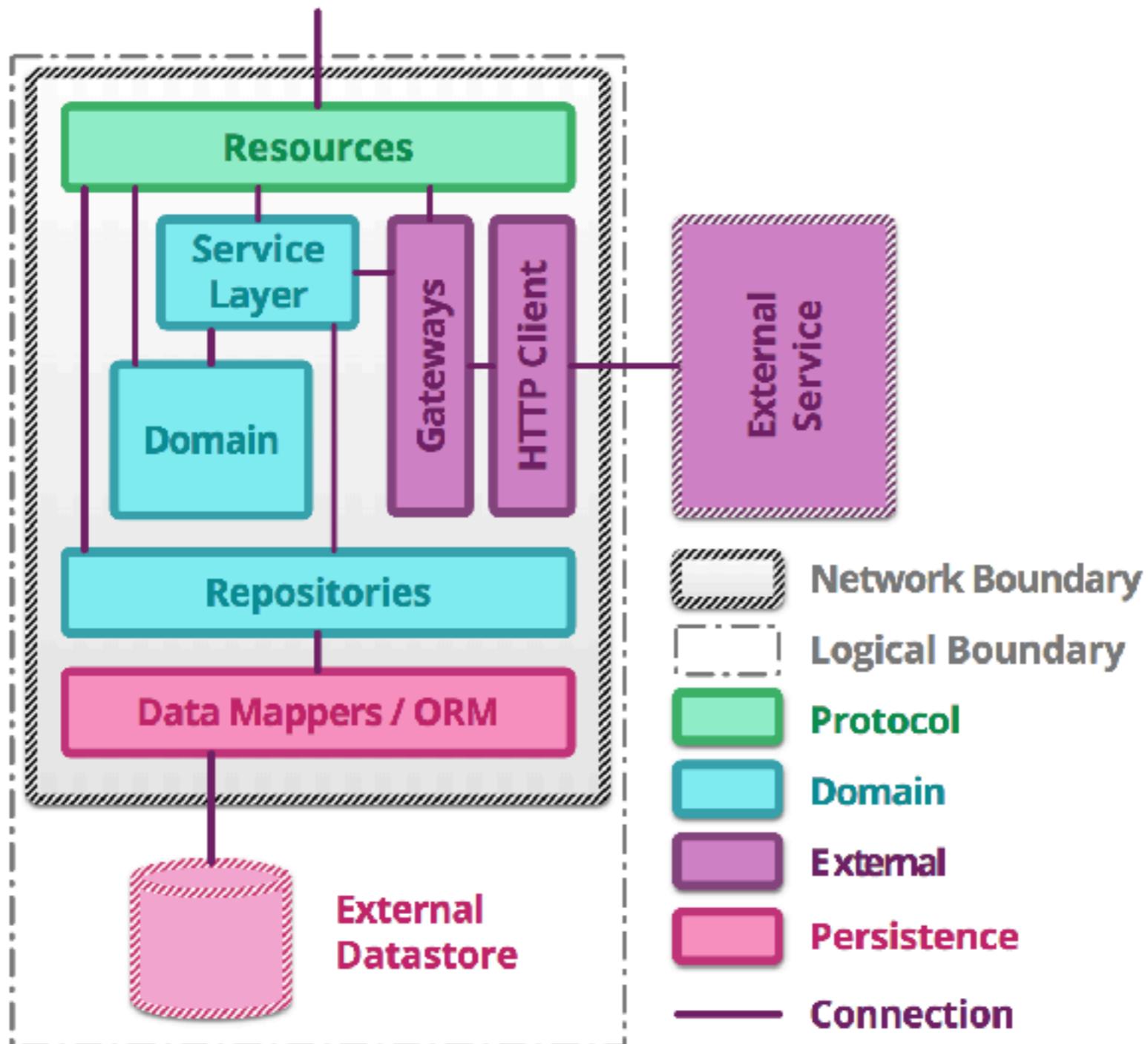
# Run project (production mode)

```
./mvnw package  
$java -jar target/<file name>.jar
```

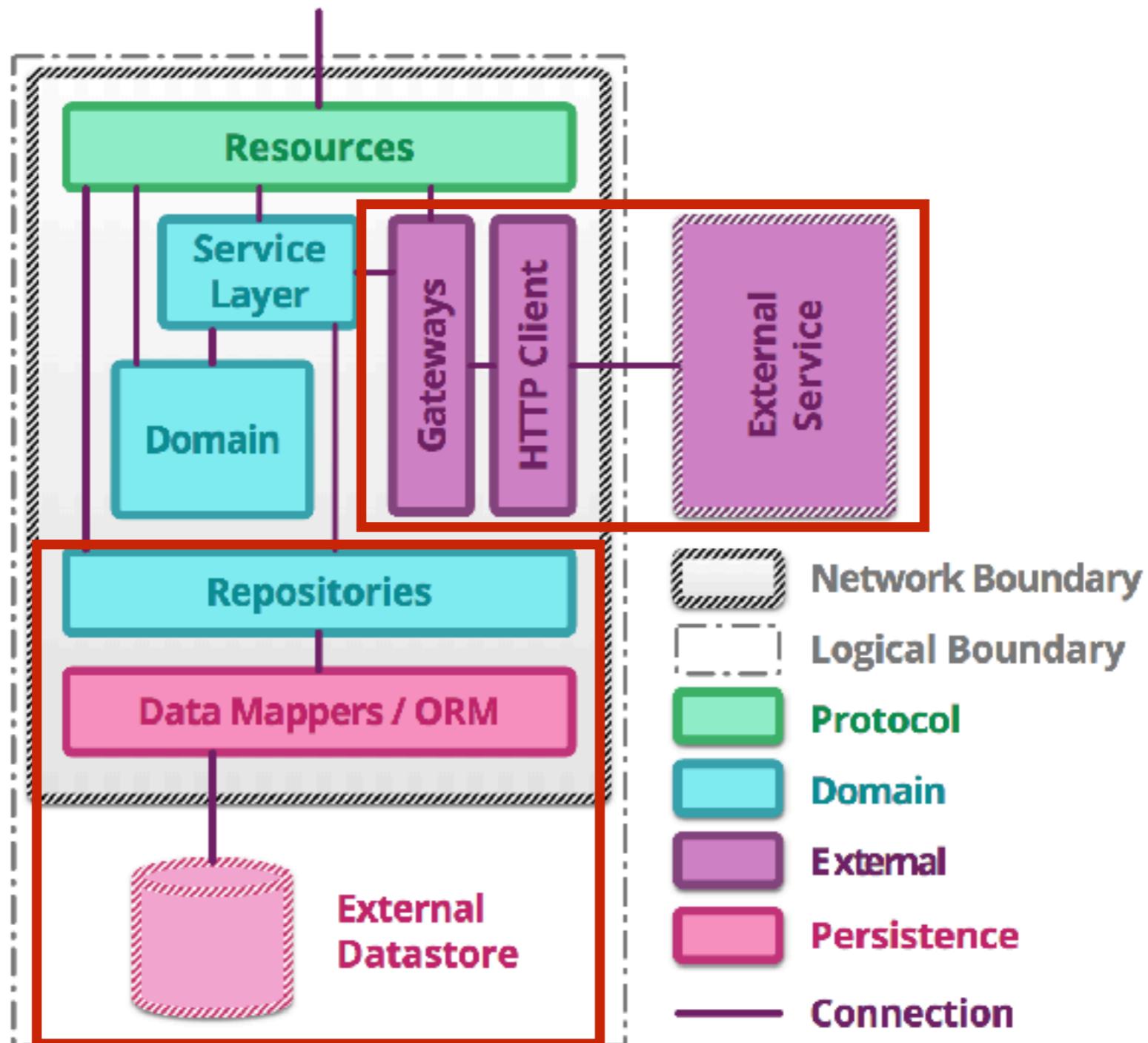
```
2018-06-07 13:03:30.412  INFO 12828 --- [  
oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
oApplication           : No active profile set, fall
```



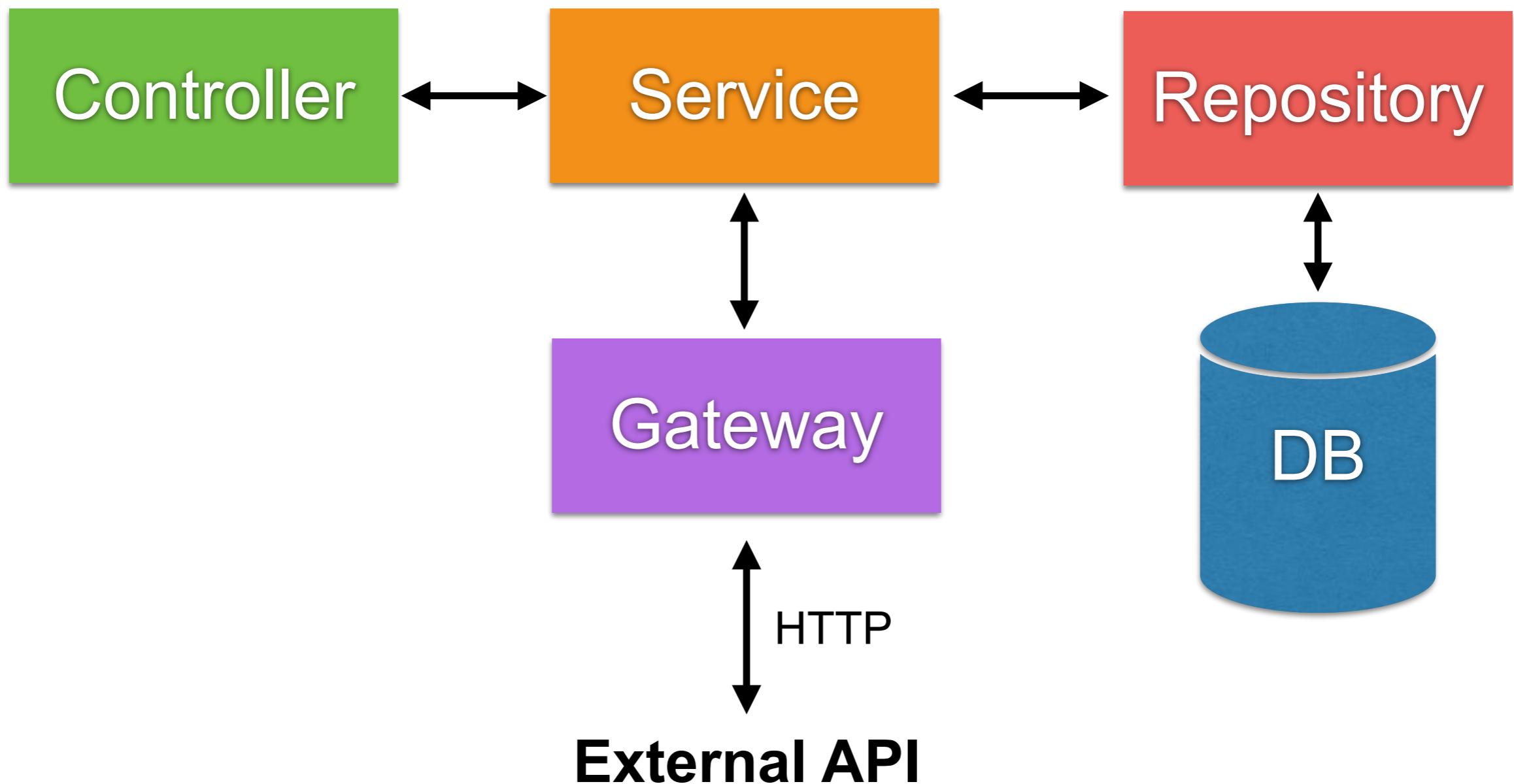
# Project Structure



# Service Structure



# Structure of Spring Boot project



# Controller

Request and Response  
Validation input

Delegate to others classes such as service and repository



# Service

Control the flow of main business logic  
Manage database transaction  
Don't reuse service



# Repository

Manage data in data store such as RDBMS and  
NoSQL



# Gateway

Call external service via network such as  
WebServices and RESTful APIs



# Spring Boot Structure (1)

Separate by function/domain/feature

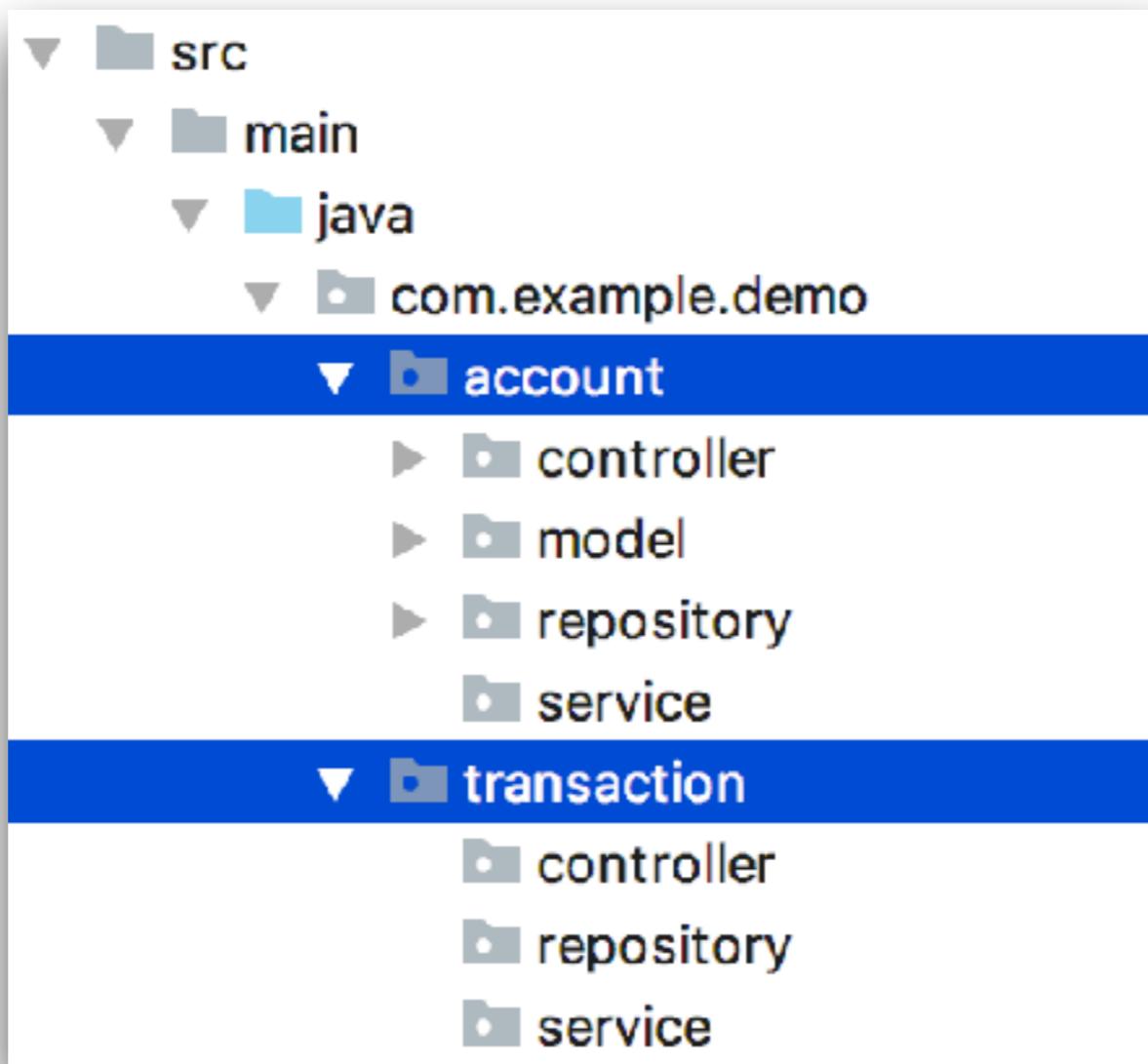
- feature1**
  - controller
  - service
  - repository
- feature2**
  - controller
  - service
  - repository

<https://docs.spring.io/spring-boot/reference/using/structuring-your-code.html>



# Spring Boot Structure (2)

Separate by function/domain/feature



# Configurations

A twelve-factor app should externalize all such configurations that vary between deployments

## File application.properties

```
spring.datasource.url=jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/movies  
spring.datasource.username=${MYSQL_USER}  
spring.datasource.password=${MYSQL_PASSWORD}
```

## Usage

```
set MYSQL_HOST=localhost  
set MYSQL_PORT=3306  
set MYSQL_USER=movies  
set MYSQL_PASSWORD=password
```

<https://docs.spring.io/spring-boot/reference/features/external-config.html>



# Convert from properties to YAML

## Yaml to properties / Properties to Yaml converter

[\*\*<< to yaml\*\*](#) [\*\*to properties >>\*\*](#)

**Yaml**

```
spring:  
  h2:  
    console:  
      enabled: 'true'  
  profiles: test  
  kafka:  
    streams:  
      bootstrap-servers: localhost:9092,  
      192.168.0.1:9092
```

**Properties**

```
spring.profiles=test  
spring.h2.console.enabled=true  
spring.kafka.streams.bootstrap-  
servers=localhost:9092, 192.168.0.1:9092
```

[\*\*convert >>\*\*](#)

[\*\*<< convert\*\*](#)

[\*\*<< convert\*\*](#) [\*\*convert >>\*\*](#)

**YAML**

<https://mageddo.com/tools/yaml-converter>



# Convert in IntelliJ IDEA

The screenshot shows the IntelliJ IDEA plugin marketplace interface. A red dashed box highlights the search results for 'convert YAML'. The results list includes:

- Convert YAML and Properties File** by chencn, 111.9K downloads, 3.49 rating, Install button.
- Properties to YAML Converter** by Jakub Kubrynski, 103.4K downloads, 3.50 rating, Install button.
- convert yaml to properties** by zhang min, 11.3K downloads, 1.90 rating, Install button.
- JSON-YAML-XML Converter** by Syncro Soft, 2.6K downloads, 3.57 rating, Paid status, Install button.
- Format-converter** by Dmitrii\_Priporov, 596 downloads, 4.47 rating, Install button.

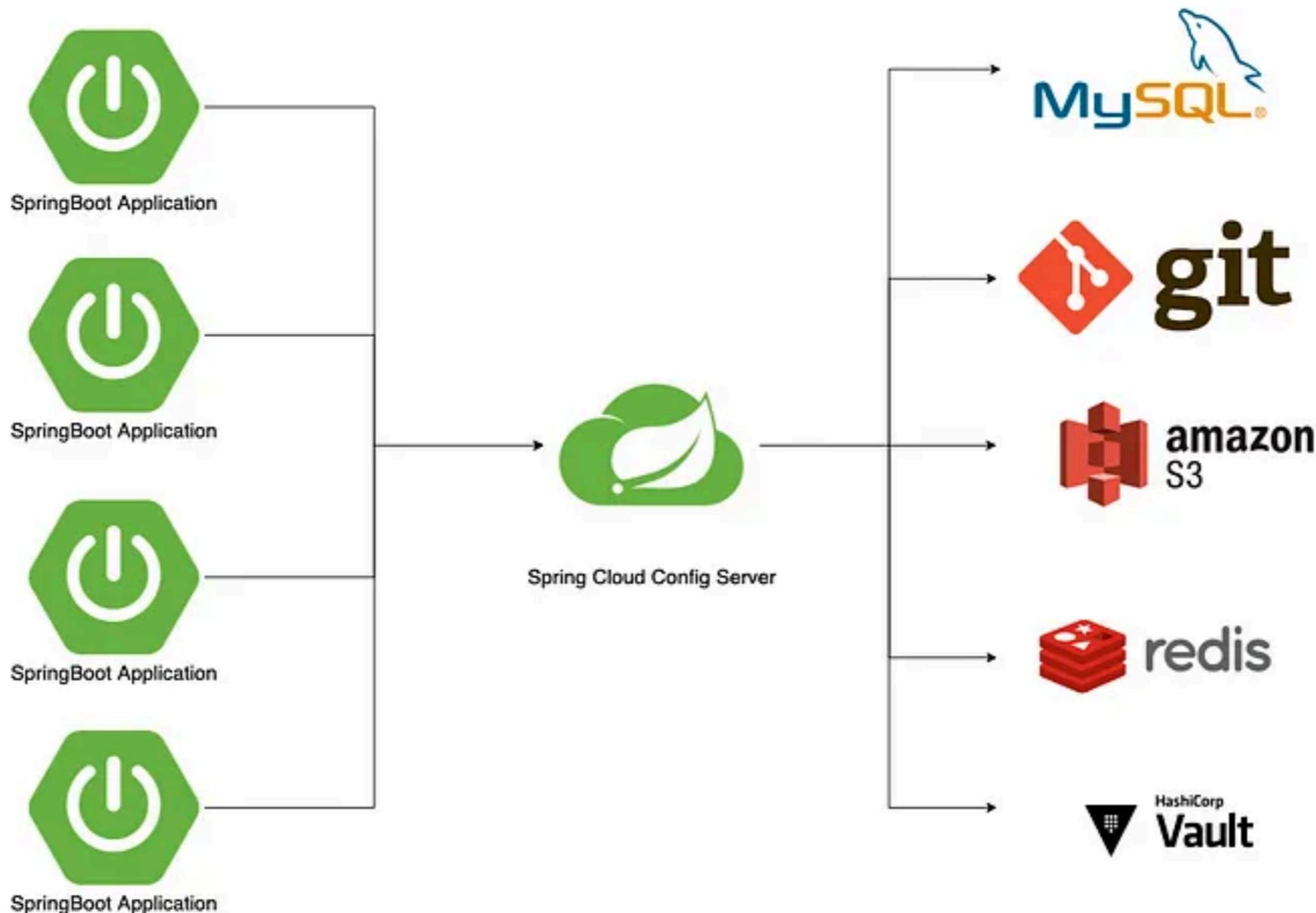
A second red dashed box highlights the details page for the 'Convert YAML and Properties File' plugin. The page includes:

- Code Tools** section.
- Convert YAML and Properties File** title.
- chencn** link to plugin homepage.
- Install** button and version **1.0.5**.
- Reviews** and **Additional Info** sections.
- Overview** and **What's New** tabs.
- A screenshot of the plugin's interface showing file conversion.

<https://plugins.jetbrains.com/plugin/13804-convert-yaml-and-properties-file>



# Centralized Configuration



<https://spring.io/projects/spring-cloud-config>



# Design RESTful APIs



# Users

Method	Path	Description
GET	/users	Get all users
GET	/users/{id}	Get user by id
POST	/users	Create new user
PUT	/users/{id}	Update user
DELETE	/users/{id}	Delete user by id

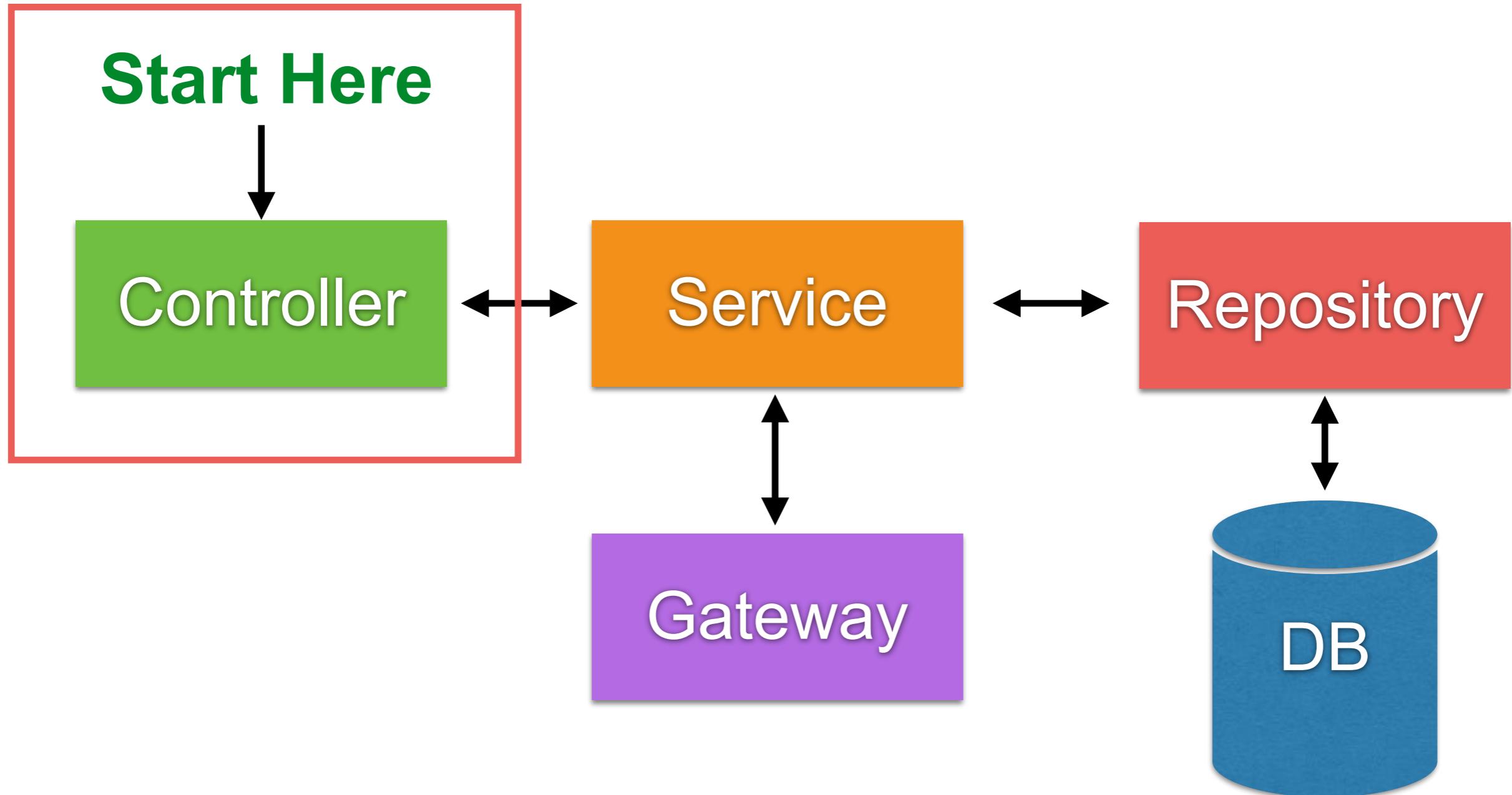


# HTTP status codes

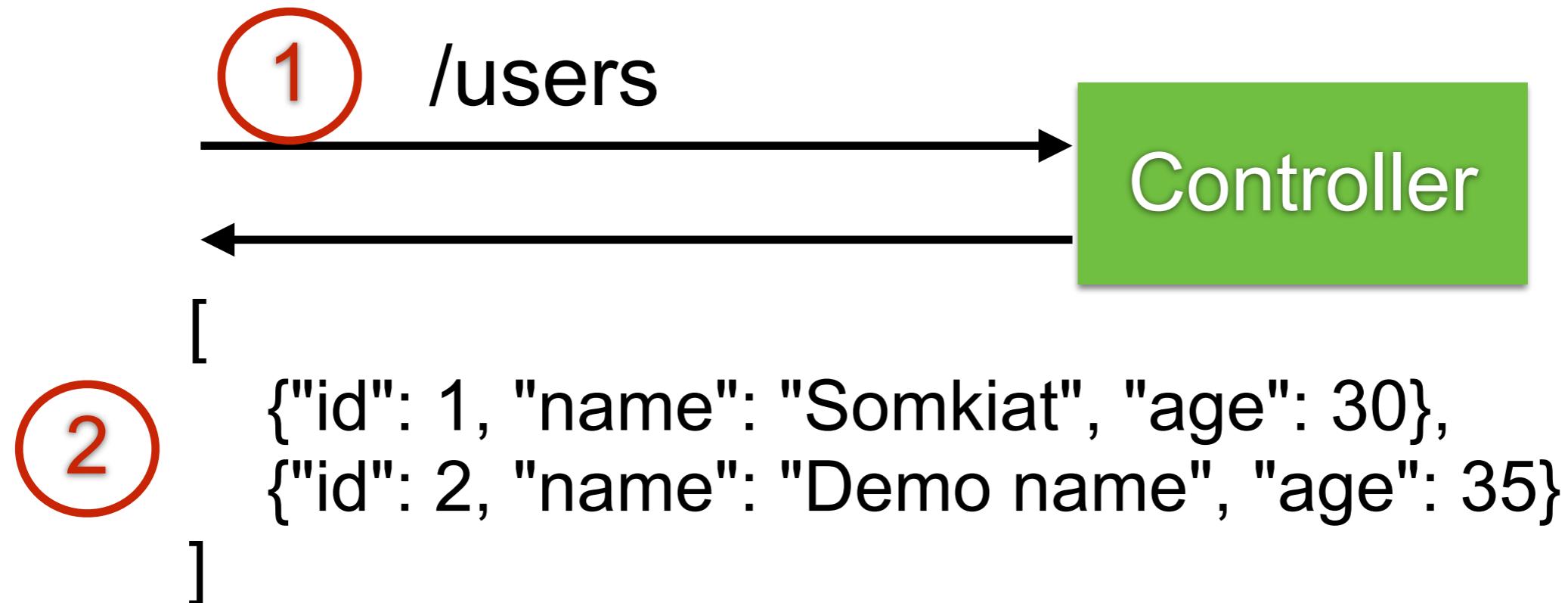
Code	Description	Example
1xx	Informational responses	100 = Continue 101 Switching protocols
2xx	Success	200 = OK 201 = Created
3xx	Redirection	300 = Multiple choices 301 = Moved permanently
4xx	Client errors	400 = Bad request 403 = Forbidden 404 = Not found
5xx	Server errors	500 = Internal server error 502 = Bad gateway 503 = Service unavailable



# Basic structure of Spring Boot



# Get all users



# 1. Create REST Controller

## UserController.java

```
@RestController
public class UserController {

    @GetMapping("/users")
    public List<UserResponse> getAllUsers() {
        List<UserResponse> userResponseList = new ArrayList<>();
        userResponseList.add(new UserResponse(1, "demo 1", 30));
        userResponseList.add(new UserResponse(2, "demo 2", 35));
        return userResponseList;
    }
}
```



# 2. Create model class

## UserResponse.java

```
public class UserResponse {  
    private int id;  
    private String name;  
    private int age;  
  
    public UserResponse() {}  
  
    public UserResponse(int id, String name, int age) {  
        this.id = id;  
        this.name = name;  
        this.age = age;  
    }  
}
```



# 3. Compile and Packaging

\$mvnw clean package



# 4. Run

```
$java -jar target/hello.jar
```



# 5. Open in browser

<http://localhost:8080/users>

```
[  
  {  
    "id": 1,  
    "name": "demo 1",  
    "age": 30  
  },  
  {  
    "id": 2,  
    "name": "demo 2",  
    "age": 35  
  }]  
]
```



# Create more APIs

Get user by id

Create a new user

Update user by id

Delete user by id



# Get user by id

GET /users/{id}

```
@GetMapping("/users/{id}")
public UserResponse getUserById(
    @PathVariable int id) {

    UserResponse userResponse = new UserResponse(id, "Demo", 40);
    return userResponse;
}
```



# Create UserResponse

**POJO**  
Setter/Getter

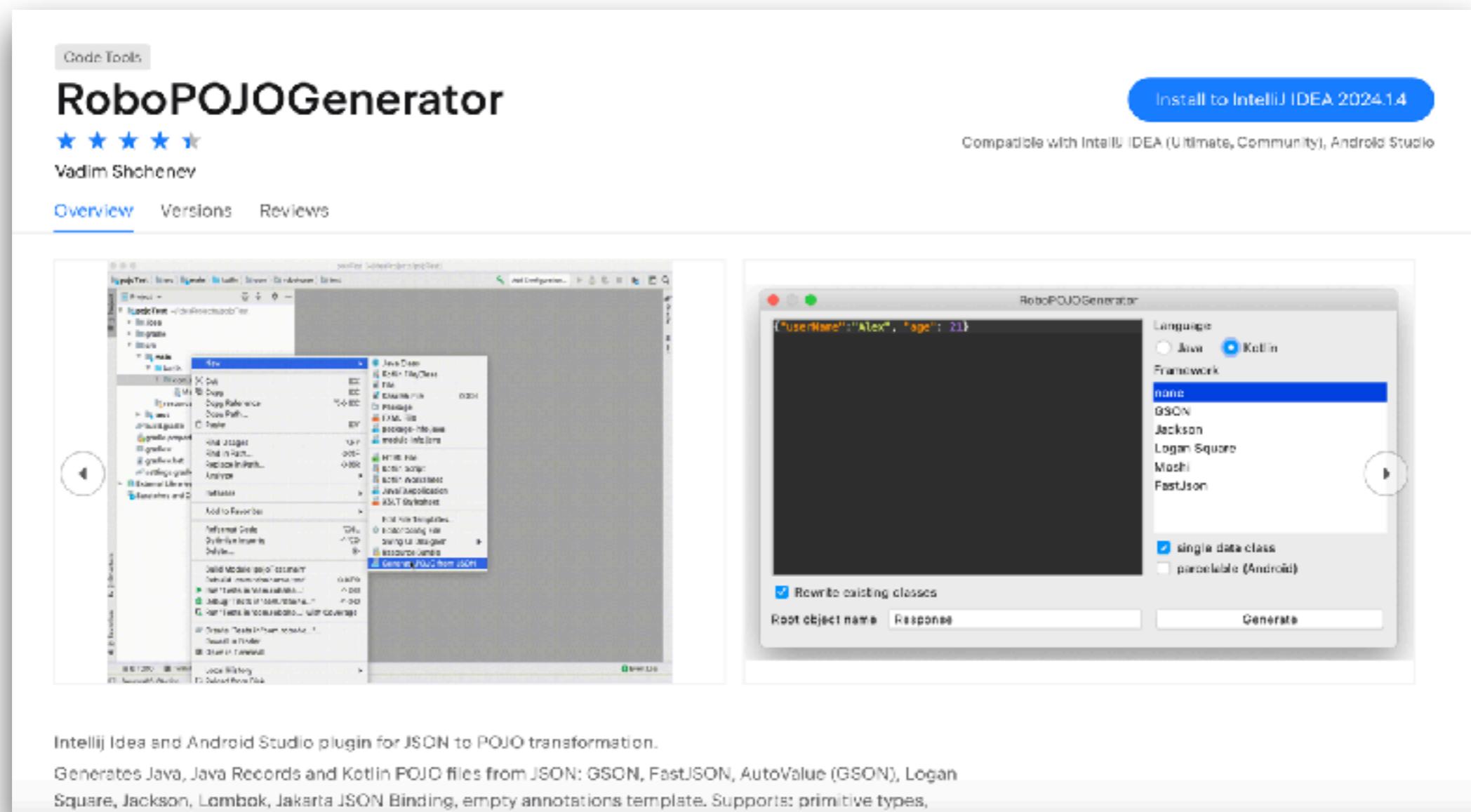
**Lombok**

**Record**  
Java 14+



# Convert JSON to POJO

## IntelliJ IDEA



<https://plugins.jetbrains.com/plugin/8634-robopojogenerator>



# Convert JSON to POJO

## VS Code

The screenshot shows the Visual Studio Marketplace page for the "quicktype" extension. The extension is titled "Paste JSON as Code" and is developed by "quicktype". It has 2,314,268 installs and a rating of 4.6 stars from 46 reviews. It is described as free. The description below the title states: "Copy JSON, paste as Go, TypeScript, C#, C++ and more." There are two buttons at the bottom: a green "Install" button and a "Trouble Installing?" link. Below the main section, there are tabs for "Overview" (which is selected), "Version History", "Q & A", and "Rating & Review". A large text block at the bottom lists supported languages and frameworks: "Supports C (cJSON), C#, C++, Crystal, Dart, Elm, Flow, Go, Haskell, JSON Schema, Java, JavaScript, JavaScript PropTypes, Kotlin, Objective-C, PHP, Pike, Python, Ruby, Rust, Scala3, Smithy, Swift, TypeScript, TypeScript Effect Schema and TypeScript Zod".

<https://marketplace.visualstudio.com/items?itemName=quicktype.quicktype>



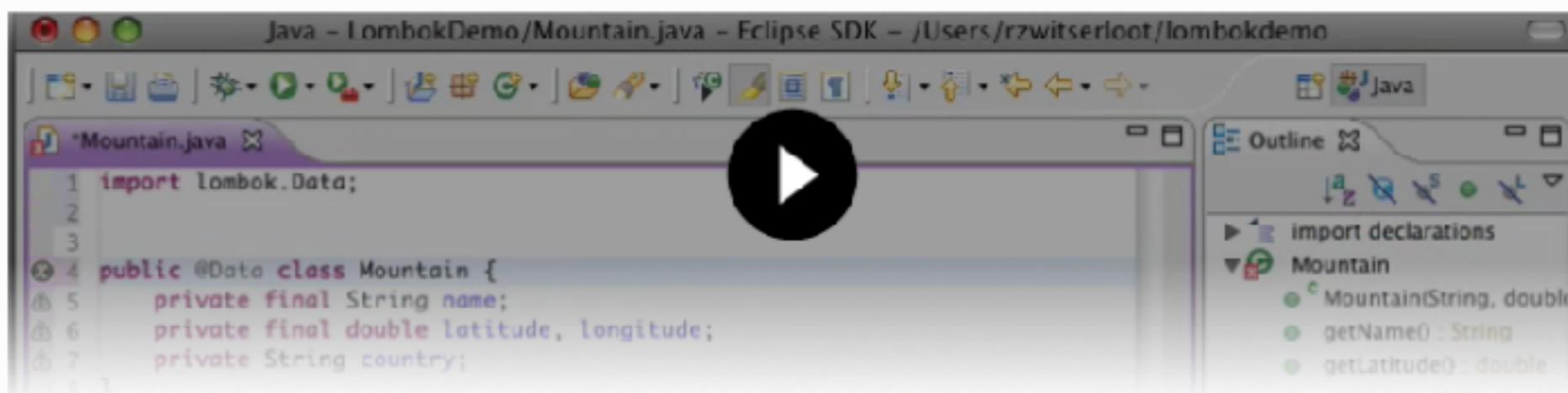
# Lombok



# Using Lombok

## Project Lombok

Project Lombok is a Java library that automatically plugs into your editor and build tools, spicing up your Java. Never write another getter or equals method again, with one annotation your class has a fully featured builder, Automate your logging variables, and much more.



Project Lombok is **powered by:**

<https://projectlombok.org/>



# IntelliJ IDEA plug-in

The screenshot shows the IntelliJ IDEA plugin marketplace interface. A red dashed box highlights the search results for 'lombok'. The first result, 'Lombok' by JetBrains s.r.o., is selected and has a checkmark next to its name. To the right, the plugin's details page is displayed, featuring the title 'Lombok' and a link to the 'Plugin homepage'. Below the title are buttons for 'Disable' and '232.9921.47'. A navigation bar at the bottom of the details page includes 'Overview' (which is underlined), 'What's New', 'Reviews', and 'Additional Info'. The 'Overview' section contains the heading 'IntelliJ Lombok plugin' and a description: 'A plugin that adds first-class support for Project Lombok Features'. A bulleted list of features follows: '@Getter and @Setter', '@FieldNameConstants', '@ToString', '@EqualsAndHashCode', '@AllArgsConstructor, @RequiredArgsConstructor and @NoArgsConstructor'.

Plugins Marketplace Installed 1 ⚙️ ← →

Search Results (36) Sort By: Relevance

Q: lombok

Lombok ✓  
↓ 14.6M ⭐ 4.04 232.9921.47 JetBrains s.r.o.

Lombok Builder Helper ✓  
↓ 197.4K 1.5.0 dorukguner

KFANG AGENT Kfang Agent Lombok Install

RoboPOJOGenerator Update ✓  
↓ 342.2K ⭐ 4.66 2.3.3 → 2.4.1 Vadim Shchenev

Lombok HermitCrab Install

GsonFormatPlus Install

Tools Integration

**Lombok**  
JetBrains s.r.o. [Plugin homepage](#)

Disable | 232.9921.47

Overview What's New Reviews Additional Info

## IntelliJ Lombok plugin

A plugin that adds first-class support for Project Lombok Features

- [@Getter and @Setter](#)
- [@FieldNameConstants](#)
- [@ToString](#)
- [@EqualsAndHashCode](#)
- [@AllArgsConstructor, @RequiredArgsConstructor and @NoArgsConstructor](#)

<https://plugins.jetbrains.com/plugin/6317-lombok>



# Use Lombok

```
@Data  
@Builder  
@NoArgsConstructor  
@AllArgsConstructor  
public class UserRequest {  
    private String email;  
    private String password;  
}
```



# Create more controller ..



# Create a new user

POST /users

```
@PostMapping("/users")
public UserResponse createNewUser(
    @RequestBody UserRequest newUser) {
    UserResponse newUserResponse = new UserResponse(
        1,
        newUser.getName(),
        newUser.getAge());
    return newUserResponse;
}
```



# Update user by id

PUT /users/{id}

```
@PutMapping("/users/{id}")
public UserResponse updateUser(@RequestBody UserRequest newUser,
                               @PathVariable int id) {
    // TODO
    // 1. find by id
    // 2. found => update user
    // 3. not found => ?? (create ? or throw error)
    UserResponse updatedUserResponse = new UserResponse(
        id,
        newUser.getName(),
        newUser.getAge());
    return updatedUserResponse;
}
```



# Delete user by id

DELETE /users/{id}

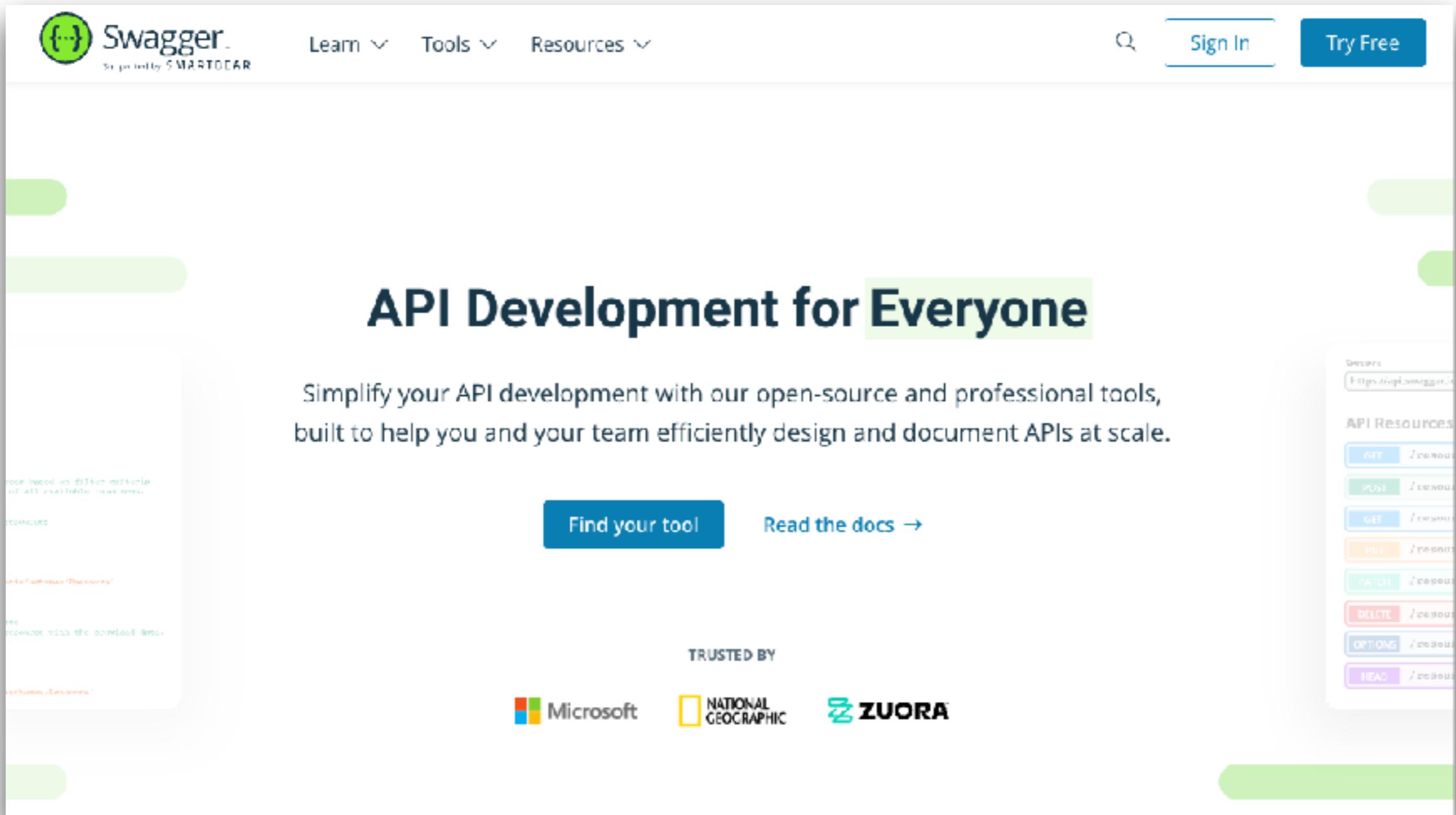
```
@DeleteMapping("/users/{id}")
public void deleteUser(@PathVariable int id) {
    // TODO
}
```



# API Documentation



# Swagger/ Open API



The screenshot shows the official Swagger website. At the top, there's a navigation bar with the Swagger logo (a green circle with white brackets), a search bar, and buttons for "Sign In" and "Try Free". Below the header, a large green banner reads "API Development for Everyone". Underneath, a sub-banner says "Simplify your API development with our open-source and professional tools, built to help you and your team efficiently design and document APIs at scale." To the right, there's a sidebar titled "API Resources" listing various HTTP methods: GET, POST, PUT, PATCH, DELETE, OPTIONS, and HEAD, each associated with a color-coded button. On the left, there's a sidebar with links like "TECHNIQUES", "TUTORIALS", and "SAMPLES". In the center, there are two buttons: "Find your tool" and "Read the docs →". Below these buttons, the text "TRUSTED BY" is followed by logos for Microsoft, National Geographic, and Zuora.

<https://swagger.io/>



# Example

The screenshot shows the Swagger Petstore API documentation. At the top, it displays the URL <https://petstore.swagger.io/v2/swagger.json>. The main title is "Swagger Petstore" with version 1.0.7 and OAS 2.0. Below the title, there's a note about the sample server and links to "Terms of service", "Contact the developer", "Apache 2.0", and "Find out more about Swagger". The "Schemes" dropdown is set to "HTTPS" and the "Authorize" button is visible. The "pet" resource is expanded, showing various operations:

- POST /pet/{petId}/uploadImage** - uploads an image
- POST /pet** - Add a new pet to the store
- PUT /pet** - Update an existing pet
- GET /pet/findByStatus** - Finds Pets by status
- GET /pet/findByTags** - Finds Pets by tags
- GET /pet/{petId}** - Find pet by ID
- POST /pet/{petId}** - Updates a pet in the store with form data.
- DELETE /pet/{petId}** - Deletes a pet



# Swagger + Spring Boot 3

## Spring Doc



<https://springdoc.org/>



# How to test UserController ?



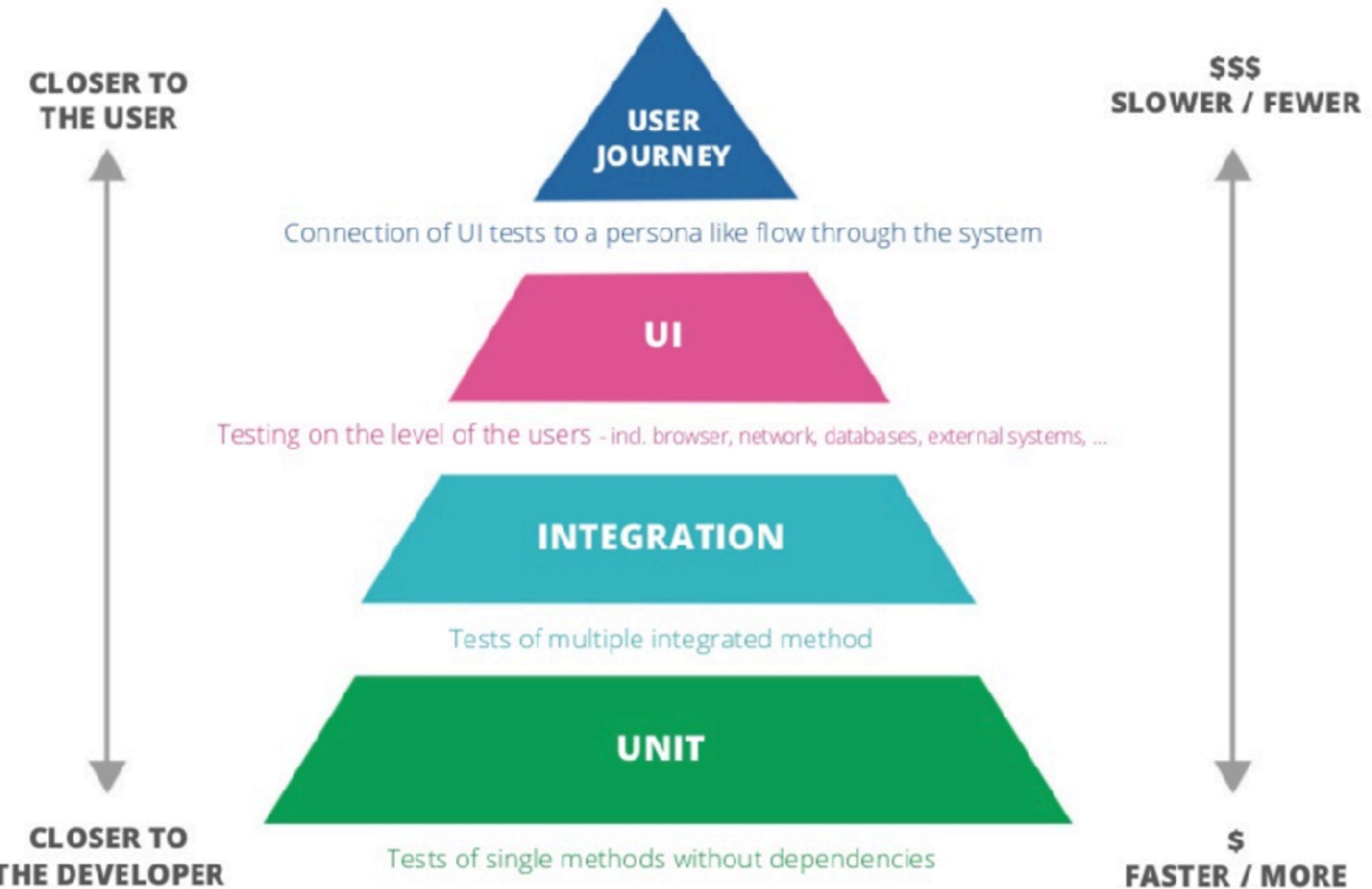
# Postman



POSTMAN

<https://www.postman.com/downloads/>





# Controller testing

How to testing with Spring Boot ?

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```

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



# Spring Boot Testing

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-test</artifactId>
    <scope>test</scope>
</dependency>
```



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



# Testing in Spring Boot

**@SpringBootTest**  
**@WebMVCTest**  
**@JsonTest**  
**@DataJpaTest**  
**@RestClientTest**



# Testing in Spring Boot

@SpringBootTest

@WebMVC Test

@JsonTest

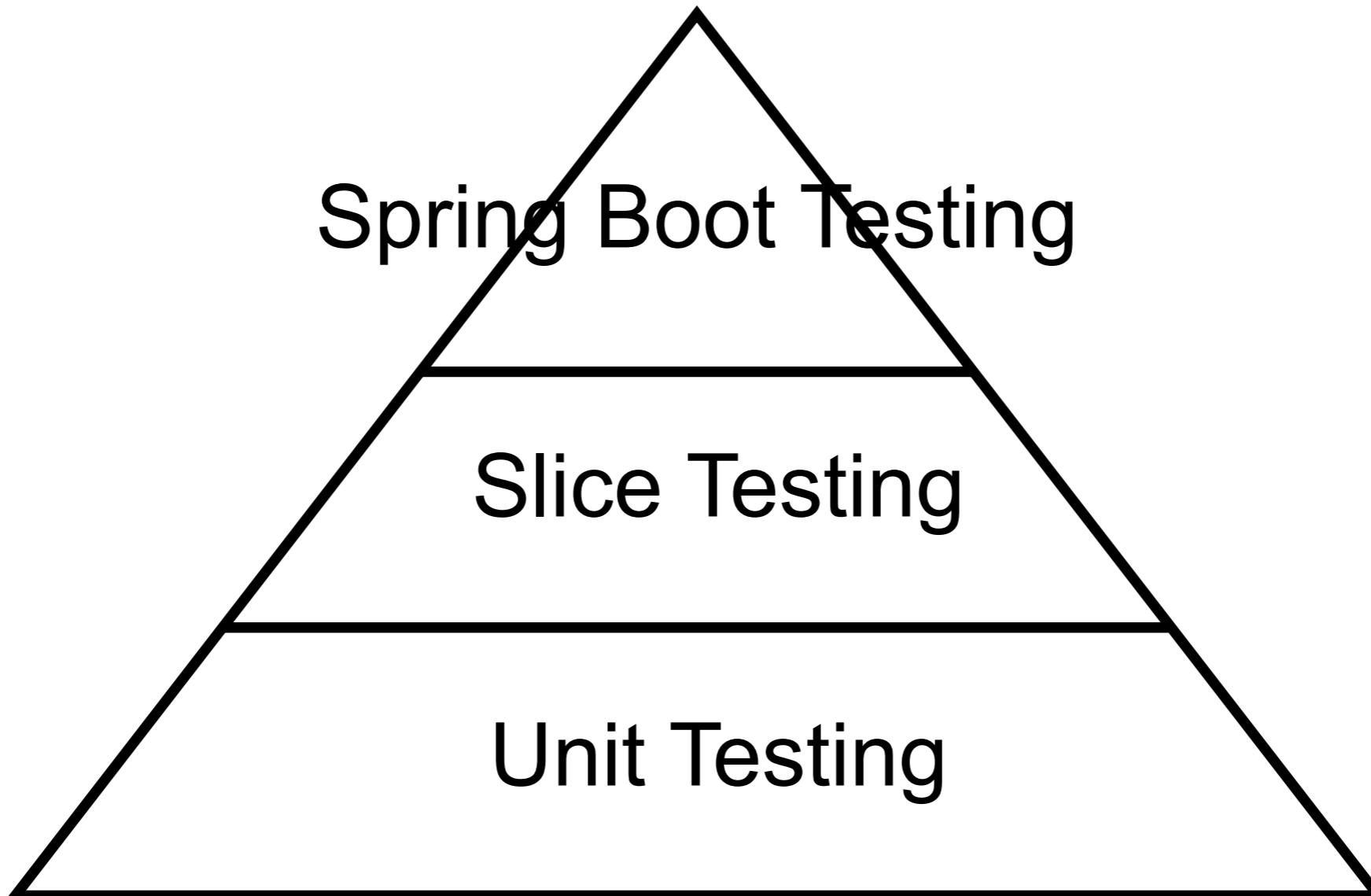
@DataJpaTest

@RestClientTest

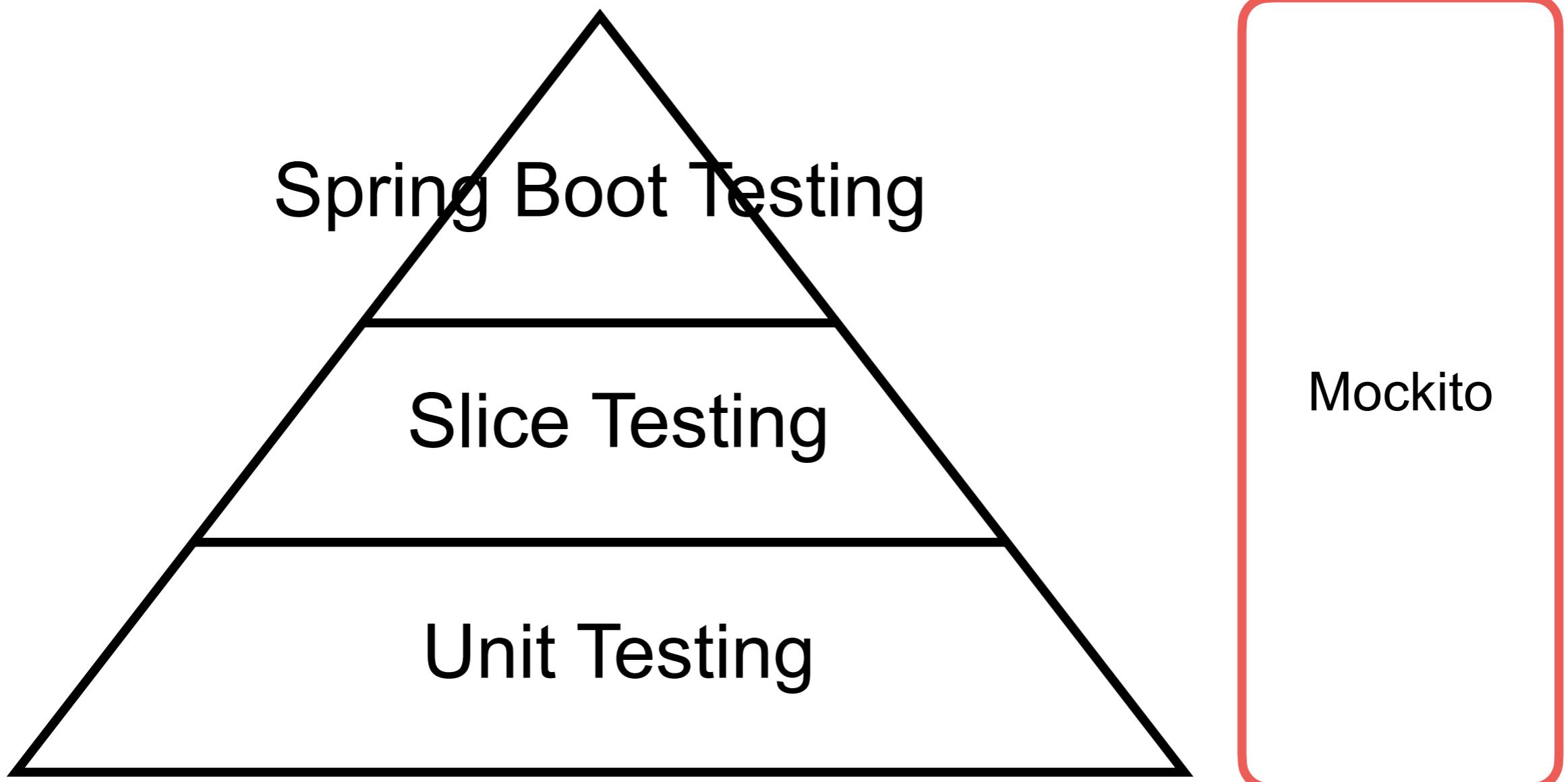
**Slice testing**



# Testing in Spring Boot



# Testing in Spring Boot



# Controller testing

1. Spring Boot Testing
2. Slice Testing with MockMvc
3. Unit Testing



# 1. SpringBootTest

## Application context

Controller Advice

Controllers

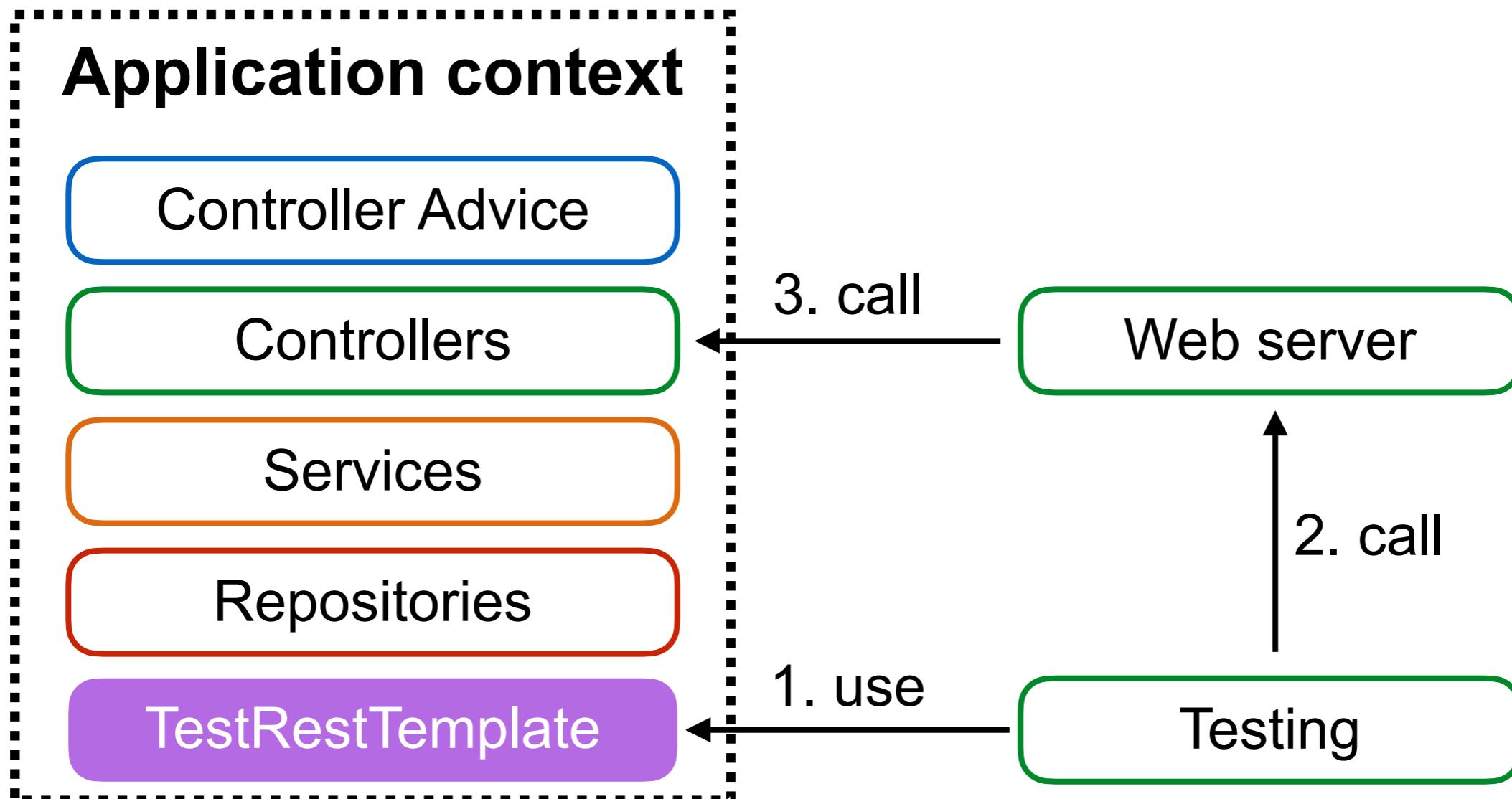
Services

Repositories

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#features.testing>



# 1. SpringBootTest



# SpringBootTest #1

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment
        = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    public void sayHi() {
        // Action :: Call controller
        Hello actualResult
            = testRestTemplate.getForObject("/hello/somkiat",
                                            Hello.class);

        // Assertion :: Check result with expected result
        assertEquals("Hello, somkiat", actualResult.getMessage());
    }
}
```



# SpringBootTest #2

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment
        = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    public void sayHi() {
        // Action :: Call controller
        Hello actualResult
            = testRestTemplate.getForObject("/hello/somkiat",
                Hello.class);

        // Assertion :: Check result with expected result
        assertEquals("Hello, somkiat", actualResult.getMessage());
    }
}
```



# SpringBootTest #3

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment
        = SpringBootTest.WebEnvironment.RANDOM_PORT)
public class HelloControllerTest {

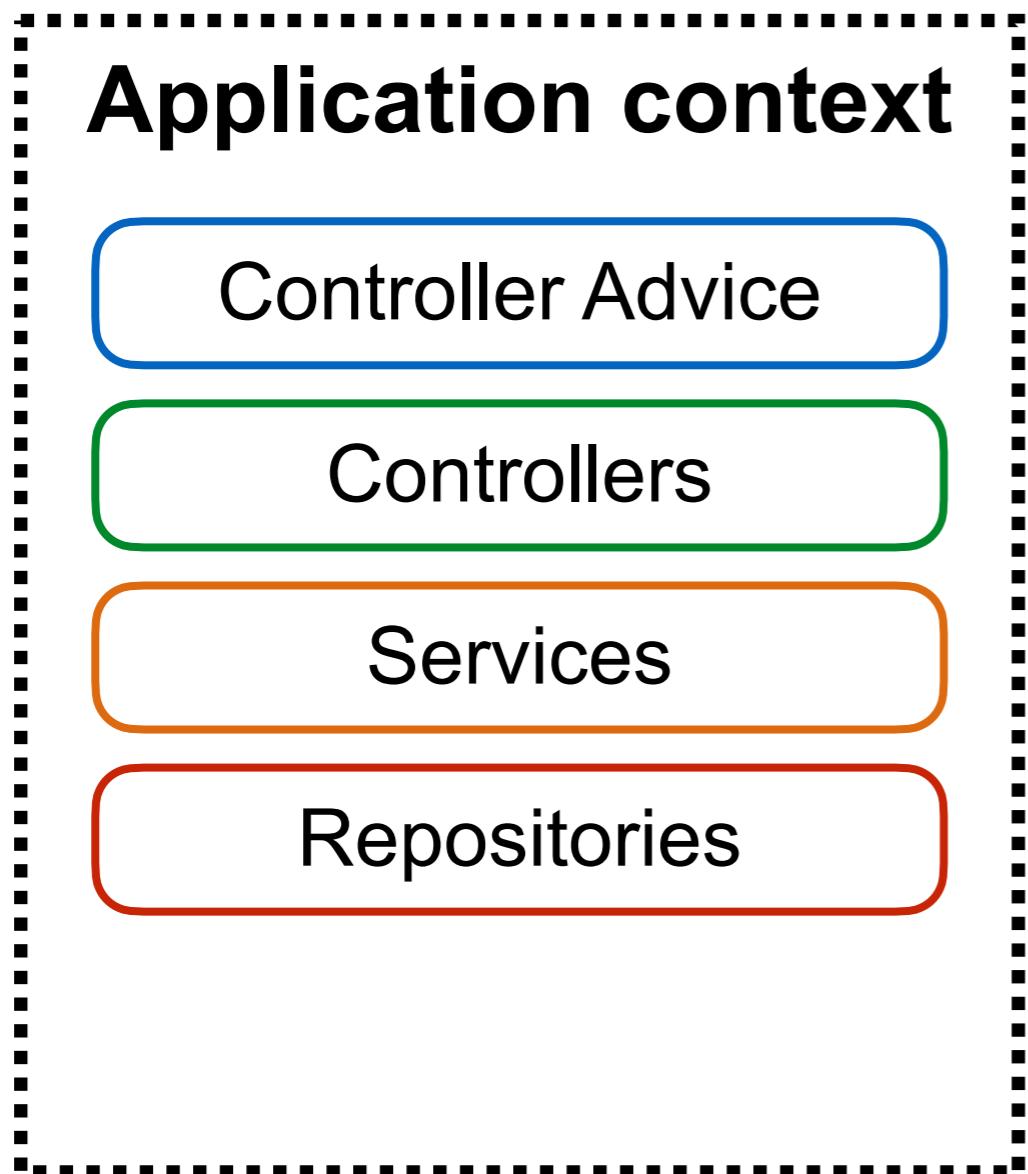
    @Autowired
    private TestRestTemplate testRestTemplate;

    @Test
    public void sayHi() {
        // Action :: Call controller
        Hello actualResult
                = testRestTemplate.getForObject("/hello/somkiat",
                                                Hello.class);

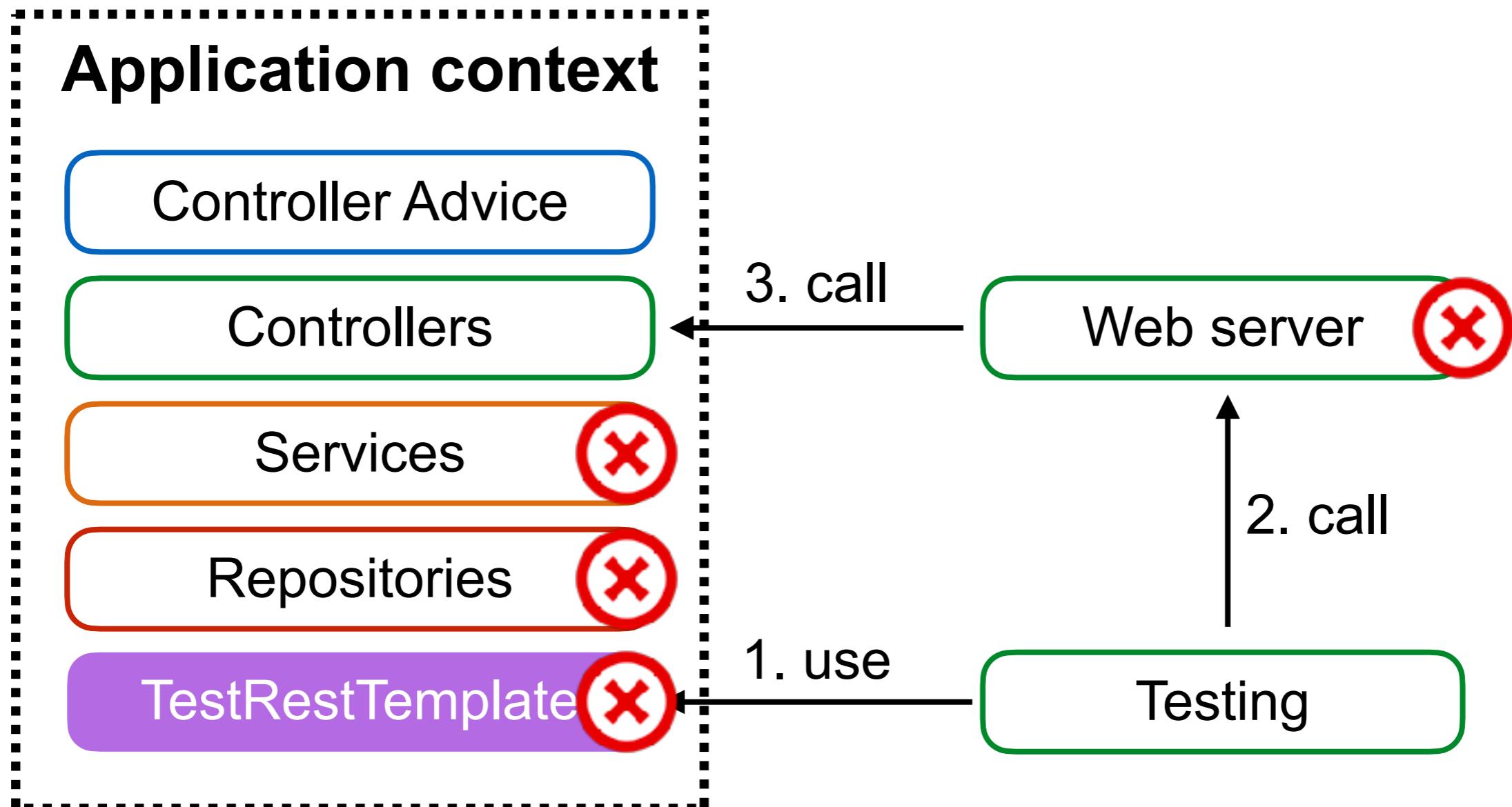
        // Assertion :: Check result with expected result
        assertEquals("Hello, somkiat", actualResult.getMessage());
    }
}
```



# 2. Slice Testing with MockMVC



# 2. Slice Testing with MockMVC



# MockMvcTest #1

```
@RunWith(SpringRunner.class)
@WebMvcTest(NumberController.class)
public class NumberControllerMockMvcTest {

    @MockBean
    private MyRandom stubRandom;

    @Autowired
    private MockMvc mvc;

    @Test
    public void success() throws Exception {
        NumberControllerResponse expected
            = new NumberControllerResponse("5555");

        // Stub
        given(stubRandom.nextInt(10)).willReturn(5555);
    }
}
```



# MockMvcTest #2

```
@RunWith(SpringRunner.class)
@WebMvcTest(NumberController.class)
public class NumberControllerMockMvcTest {

    @MockBean
    private MyRandom stubRandom;

    @Autowired
    private MockMvc mvc;

    @Test
    public void success() throws Exception {
        NumberControllerResponse expected
            = new NumberControllerResponse("5555");

        // Stub
        given(stubRandom.nextInt(10)).willReturn(5555);

        // ...
    }
}
```



# MockMvcTest #3

## Use ObjectMapper to convert JSON to object

```
// Call API HTTP response code = 200
String response =
    this.mvc.perform(get("/number"))
    .andExpect(status().isOk())
    .andReturn()
    .getResponse().getContentAsString();

// Convert JSON message to Object
ObjectMapper mapper = new ObjectMapper();
NumberControllerResponse actual =
    mapper.readValue(response,
        NumberControllerResponse.class);
```



# 3. Unit testing with Controller



# Unit test

Use Test Double

In java, use Mockito library



<http://site.mockito.org/>



# Unit testing with Mockito #1

```
@ExtendWith(MockitoExtension.class)
public class NumberControllerUnitTest {

    @Mock
    private MyRandom stubRandom;

    @Test
    public void success() throws Exception {
        NumberControllerResponse expected
            = new NumberControllerResponse("5555");

        // Stub
        given(stubRandom.nextInt(10)).willReturn(5555);
    }
}
```



# Unit testing with Mockito #2

```
@ExtendWith(MockitoExtension.class)
public class NumberControllerUnitTest {
    @Mock
    private MyRandom stubRandom;

    @Test
    public void success() throws Exception {
        NumberControllerResponse expected
            = new NumberControllerResponse("5555");

        // Stub
        given(stubRandom.nextInt(10)).willReturn(5555);
    }
}
```



# Unit testing with Mockito #3

```
@Test
public void success() throws Exception {
    NumberControllerResponse expected
        = new NumberControllerResponse("5555");

    // Stub
    given(stubRandom.nextInt(10)).willReturn(5555);

    // Call
    NumberController controller = new NumberController(stubRandom);
    NumberControllerResponse actual = controller.randomNumber();

    // Assert
    assertEquals("5555", actual.getValue());
    assertEquals(expected, actual);
}
```



# Error handling



# Working with Error Responses

**ErrorResponse**

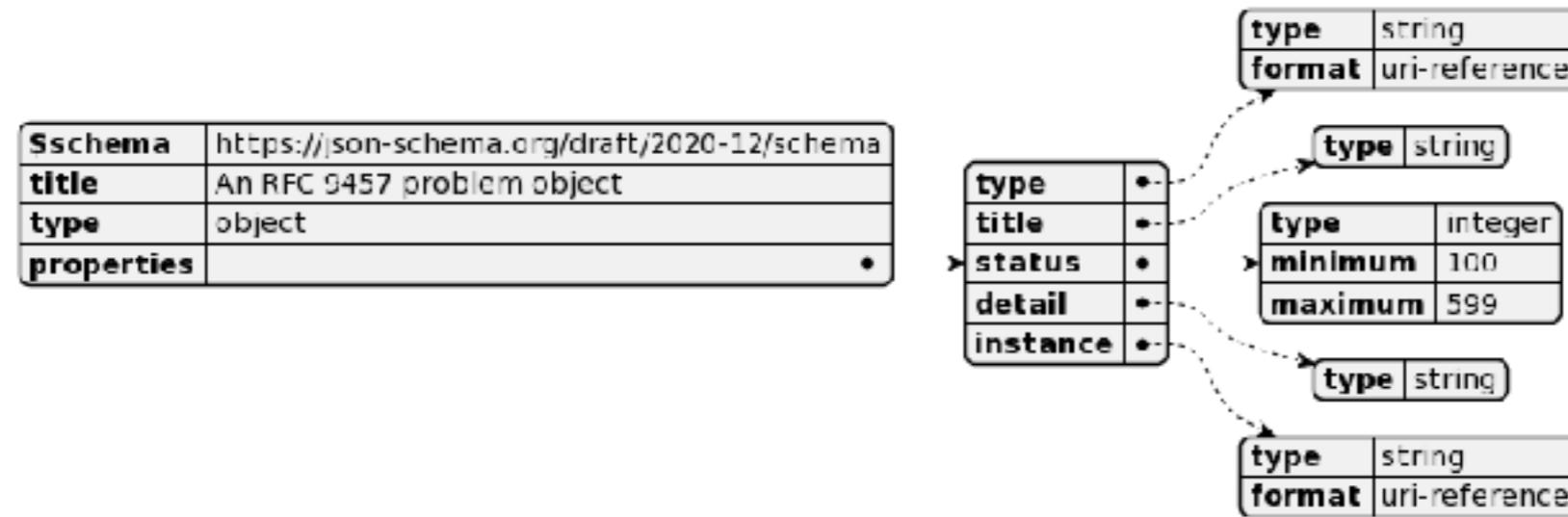
**ProblemDetail**

**Customization**

<https://docs.spring.io/spring-framework/reference/web/webmvc/mvc-ann-rest-exceptions.html>



# ProblemDetail



```
{  
  "type": "https://problems-registry.smartbear.com/missing-body-property",  
  "status": 400,  
  "title": "Missing body property",  
  "detail": "The request is missing an expected body property.",  
  "code": "400-09",  
  "instance": "/logs/registrations/d24b2953-ce05-488e-bf31-67de50d3d085",  
  "errors": [  
    {  
      "detail": "The body property {name} is required",  
      "pointer": "/name"  
    }  
  ]  
}
```

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



Spring Framework

© 2020 - 2024 Siam Chamnkit Company Limited. All rights reserved.

# ProblemDetail

Enable ProblemDetail for error by default

## application.properties

```
spring.mvc.problemdetails.enabled=true
```



# Example in ControllerAdvice

```
@RestControllerAdvice
class HelloControllerAdvice {

    @ExceptionHandler(InvalidInputException.class)
    public ProblemDetail handleInvalidInputException(
        InvalidInputException e, WebRequest request) {

        ProblemDetail problemDetail
            = ProblemDetail
                .forStatusAndDetail(HttpStatus.BAD_REQUEST, e.getMessage());
        problemDetail.setInstance(URI.create("demo-instance"));
        problemDetail.setTitle("demo");

        return problemDetail;
    }
}
```



# Error handling

```
@Service
public class UserService {

    private AccountRepository accountRepository;

    @Autowired
    public UserService(AccountRepository accountRepository) {
        this.accountRepository = accountRepository;
    }

    public Account getAccount(int id) {
        Optional<Account> account = accountRepository.findById(id);
        if(account.isPresent()) {
            return account.get();
        }
        throw new MyAccountNotFoundException(
            String.format("Account id=[%d] not found", id));
    }
}
```



# MyAccountNotFoundException

```
public class MyAccountNotFoundException  
    extends RuntimeException {
```

```
    public MyAccountNotFoundException(String message) {  
        super(message);  
    }
```

```
}
```



# Response Status

404 = Not Found

Status	Description
400	Request body doesn't meet API spec
401	Authentication/Authorization fail
403	User can't perform the operation
404	Resource does not exist
405	Unsupported operation
500	Error on server



# Handling error in Spring Boot

```
@RestControllerAdvice  
public class AccountControllerHandler {  
  
    @ExceptionHandler(MyAccountNotFoundException.class)  
    public ResponseEntity<ExceptionResponse> accountNotFound(  
        MyAccountNotFoundException exception) {  
  
        ExceptionResponse response =  
            new ExceptionResponse(exception.getMessage(),  
                "More detail");  
  
        return new ResponseEntity<ExceptionResponse>(response,  
            HttpStatus.NOT_FOUND);  
    }  
}
```

1



# Handling error in Spring Boot

```
@RestControllerAdvice  
public class AccountControllerHandler {  
  
    @ExceptionHandler(MyAccountNotFoundException.class)  
    public ResponseEntity<ExceptionResponse> accountNotFound(  
        MyAccountNotFoundException exception) {  
  
        ExceptionResponse response =  
            new ExceptionResponse(exception.getMessage(),  
                "More detail");  
  
        return new ResponseEntity<ExceptionResponse>(response,  
            HttpStatus.NOT_FOUND);  
    }  
}
```

2



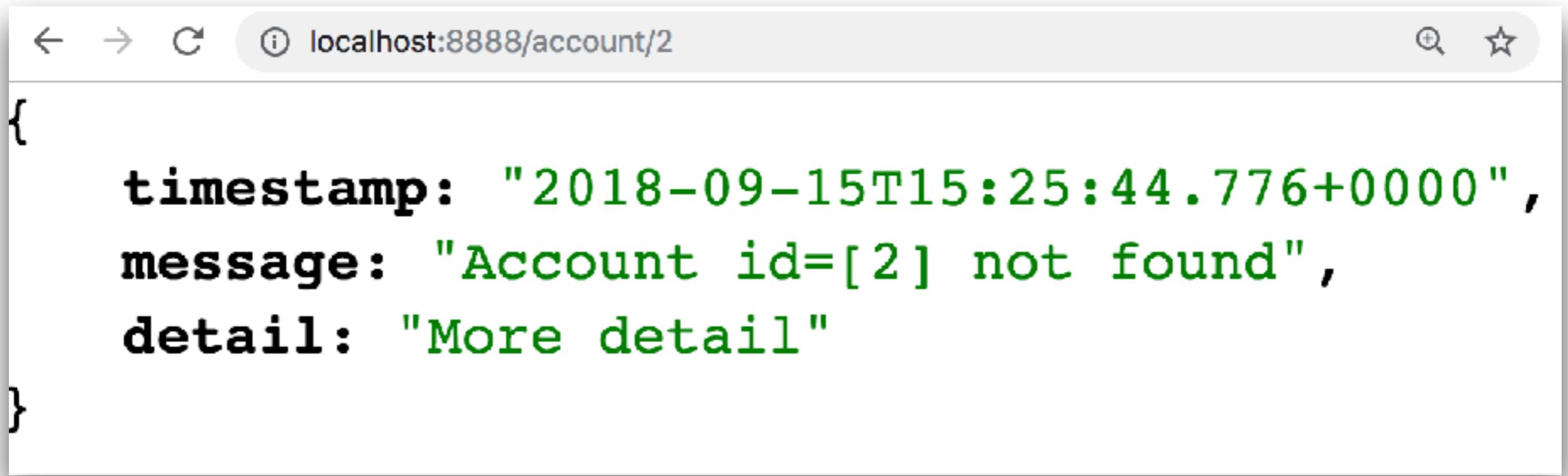
# ExceptionResponse

Response format of error

```
public class ExceptionResponse{  
  
    private Date timestamp = new Date();  
    private String message;  
    private String detail;  
  
    public ExceptionResponse(String message, String detail) {  
        this.message = message;  
        this.detail = detail;  
    }  
}
```



# Result of API



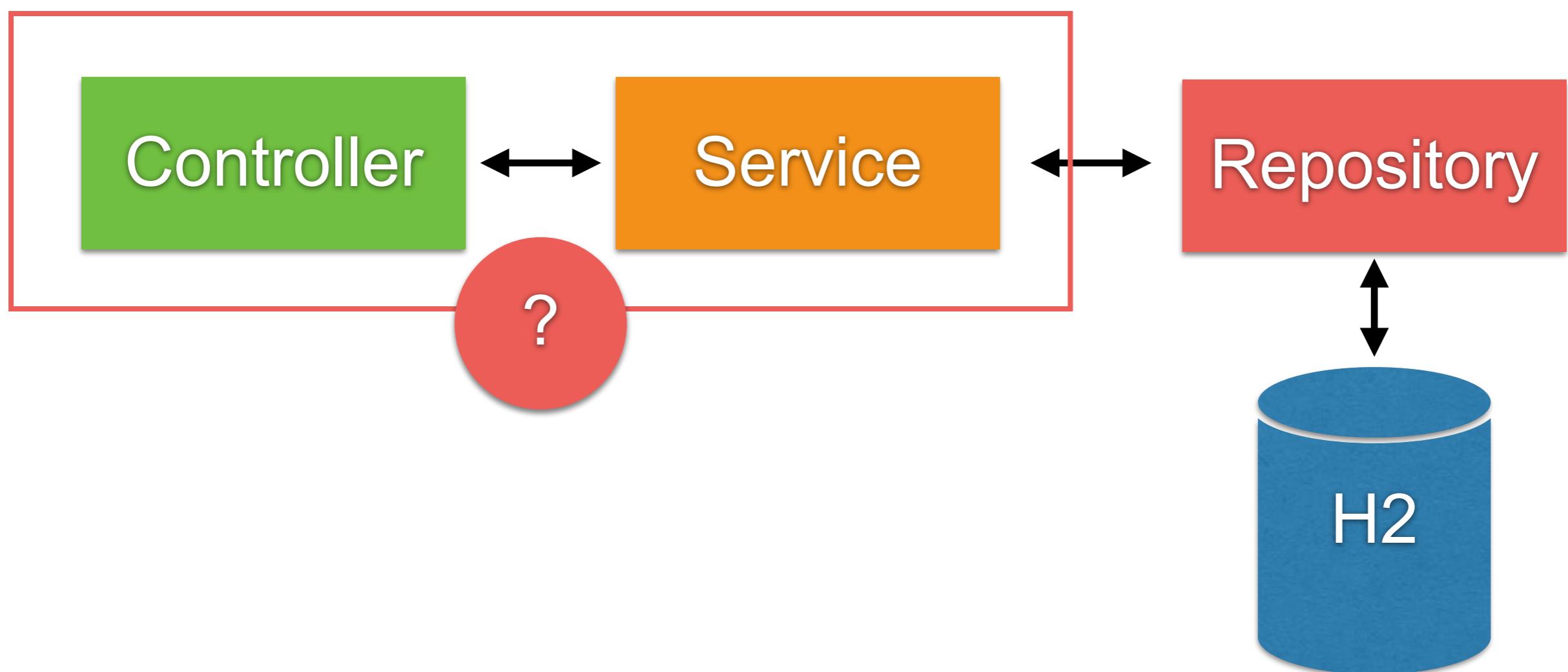
```
← → C ⓘ localhost:8888/account/2 🔍 ⭐  
{  
  timestamp: "2018-09-15T15:25:44.776+0000",  
  message: "Account id=[ 2 ] not found",  
  detail: "More detail"  
}
```



# How to test ?



# How to test with Error/Exception ?



# Testing with WebMvcTest and MockMvc

Try to check data in response

```
@Test  
public void getByIdWithNotFoundAccount() throws Exception {  
    // Stub  
    given(userService.getAccount(2))  
        .willThrow(new MyAccountNotFoundException("Not found"));  
  
    mockMvc.perform(  
        get("/account/2")  
            .accept(MediaType.APPLICATION_JSON))  
        .andExpect(status().isNotFound())  
        .andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8))  
        .andExpect(jsonPath("$.message", is("Not found")));  
}
```

1



# Testing with WebMvcTest and MockMvc

Try to check data in response

```
@Test  
public void getByIdWithNotFoundAccount() throws Exception {  
    // Stub  
    given(userService.getAccount(2))  
        .willThrow(new MyAccountNotFoundException("Not found"));  
  
    mockMvc.perform(  
        get("/account/2")  
            .accept(MediaType.APPLICATION_JSON))  
        .andExpect(status().isNotFound())  
        .andExpect(content().contentType(MediaType.APPLICATION_JSON_UTF8))  
        .andExpect(jsonPath("$.message", is("Not found")));  
}
```

2



# Testing with Service

Try to check exception

```
@ExtendWith(MockitoExtension.class)
public class UserServiceTest {

    @Mock
    private UserRepository userRepository;

    @Test
    public void user_not_found_with_exception() {
        given(userRepository.findById(1))
            .willReturn(Optional.empty());
        UserService userService = new UserService();
        userService.setRepository(userRepository);

        Assertions.assertThrows(RuntimeException.class, () -> {
            userService.getData(1);
        });
    }
}
```



# Compile with testing

\$mvnw clean test

```
[INFO]
[INFO] Results:
[INFO]
[INFO] Tests run: 2, Failures: 0, Errors: 0, Skipped: 0
[INFO]
[INFO]
```



# Code coverage



**"Code coverage can show the high risk areas in a program, but never the risk-free."**

Paul Reilly, 2018, Kotlin TDD with Code Coverage



# Code coverage

A tool to measure how much of your code is covered by tests that break down into classes, methods and lines.



# Code coverage

But 100% of code coverage **does not mean** that your code is 100% correct



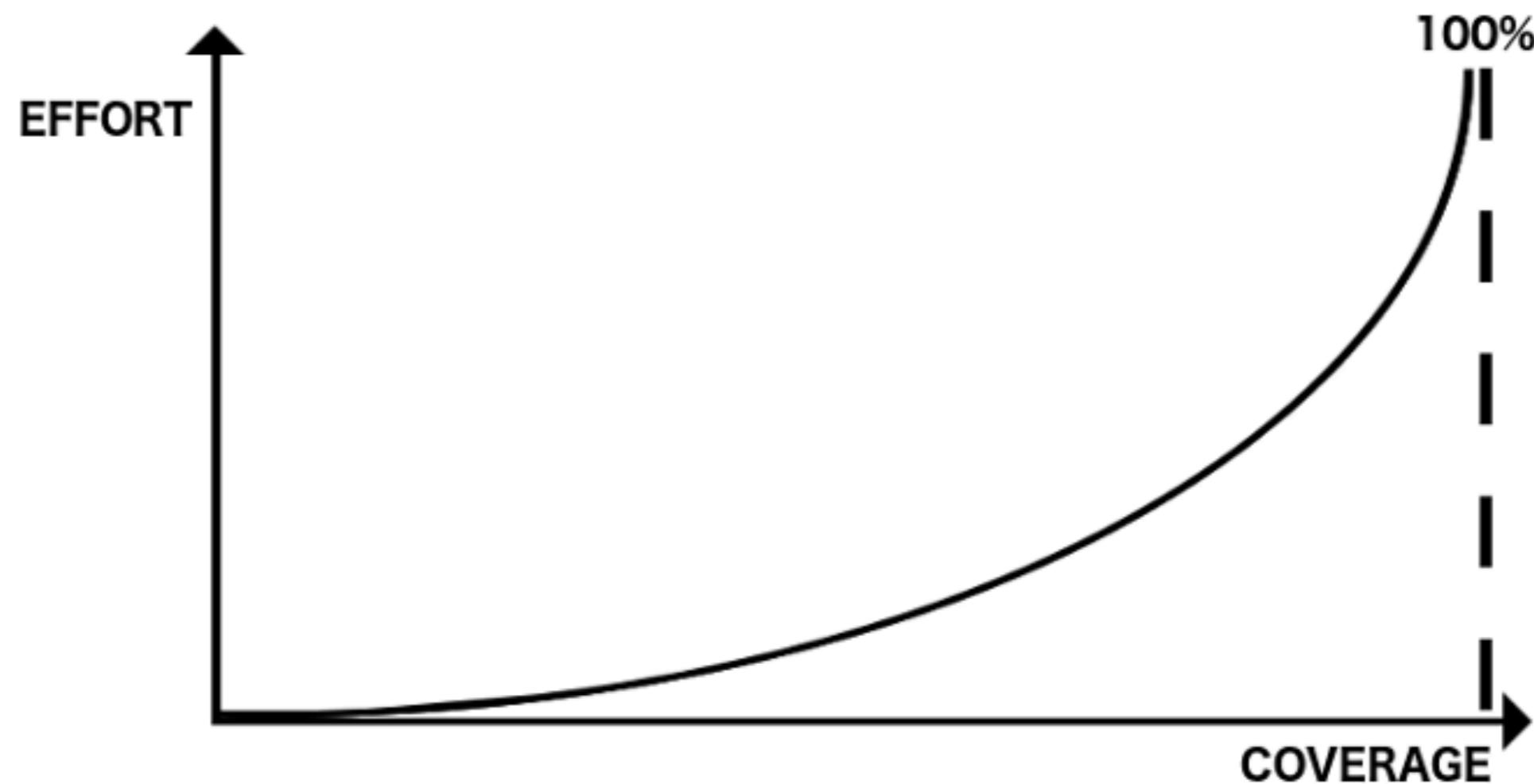
# Code coverage

Powerful tool to improve the quality of your code

**Code coverage != quality of tests**



# Code coverage 100% ?



# % of Code/Test coverage



# Code coverage with Java

Cobertura  
Jacoco

<http://bit.ly/2DIlsDeX>



# Run test again

\$mvnw clean package

Cobertura Report generation was successful.

Cobertura 2.1.1 - GNU GPL License (NO WARRANTY) - See COPYRIGHT file

Cobertura: Loaded information on 3 classes.

time: 125ms

Cobertura Report generation was successful.

---

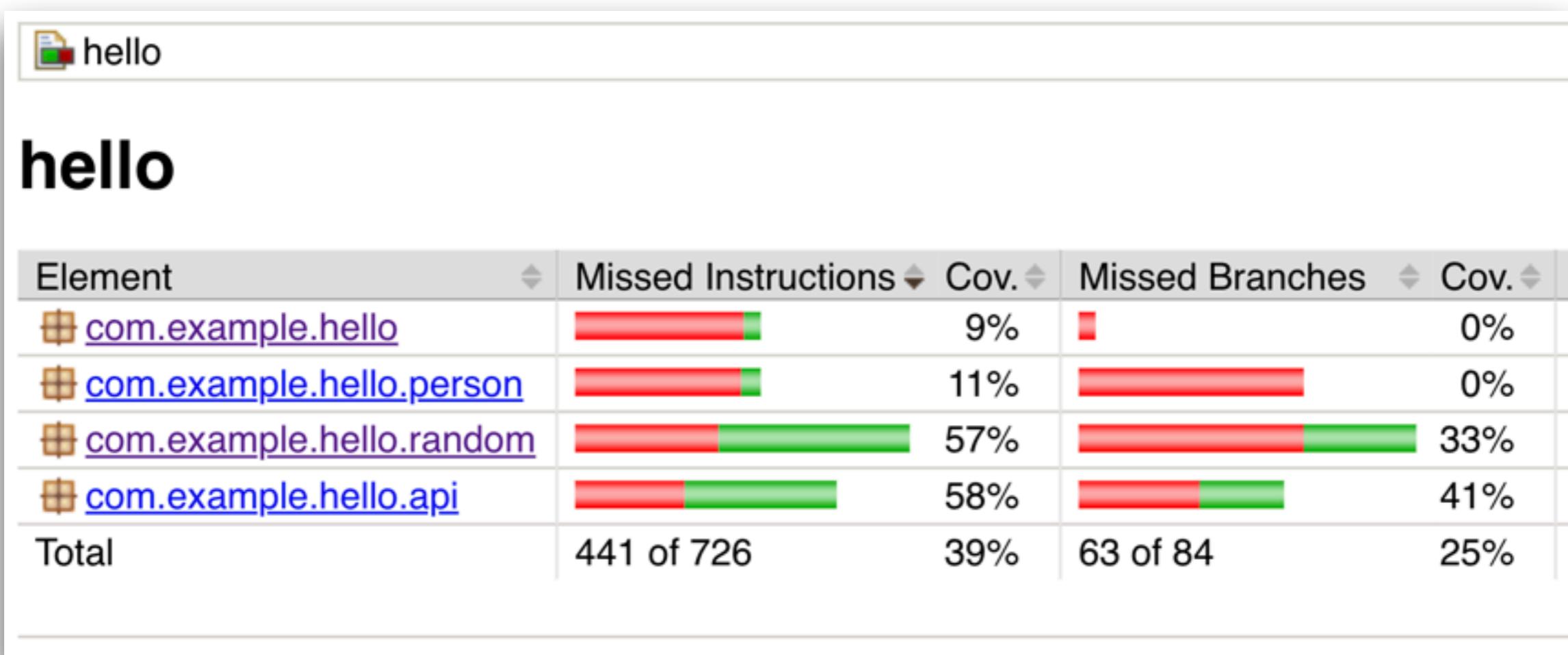
BUILD SUCCESS

---



# Coverage report

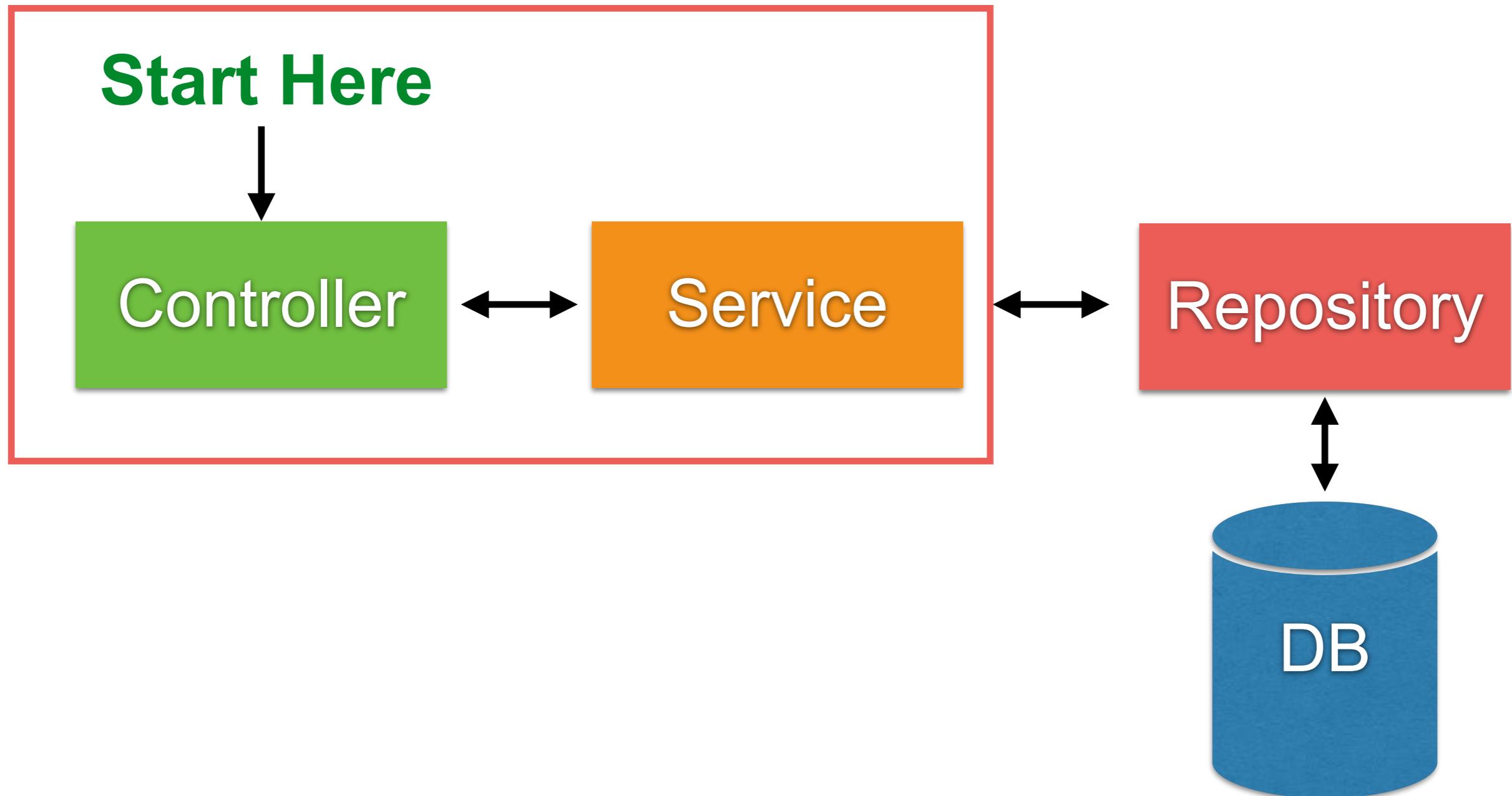
open target/site/jacoco/index.html



# Move business logic to service

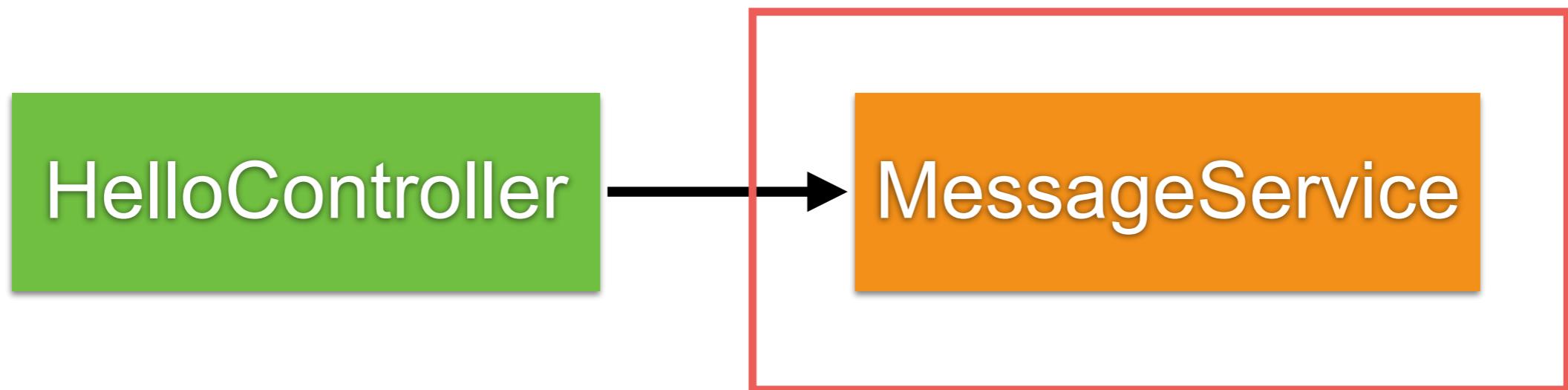


# Working with service



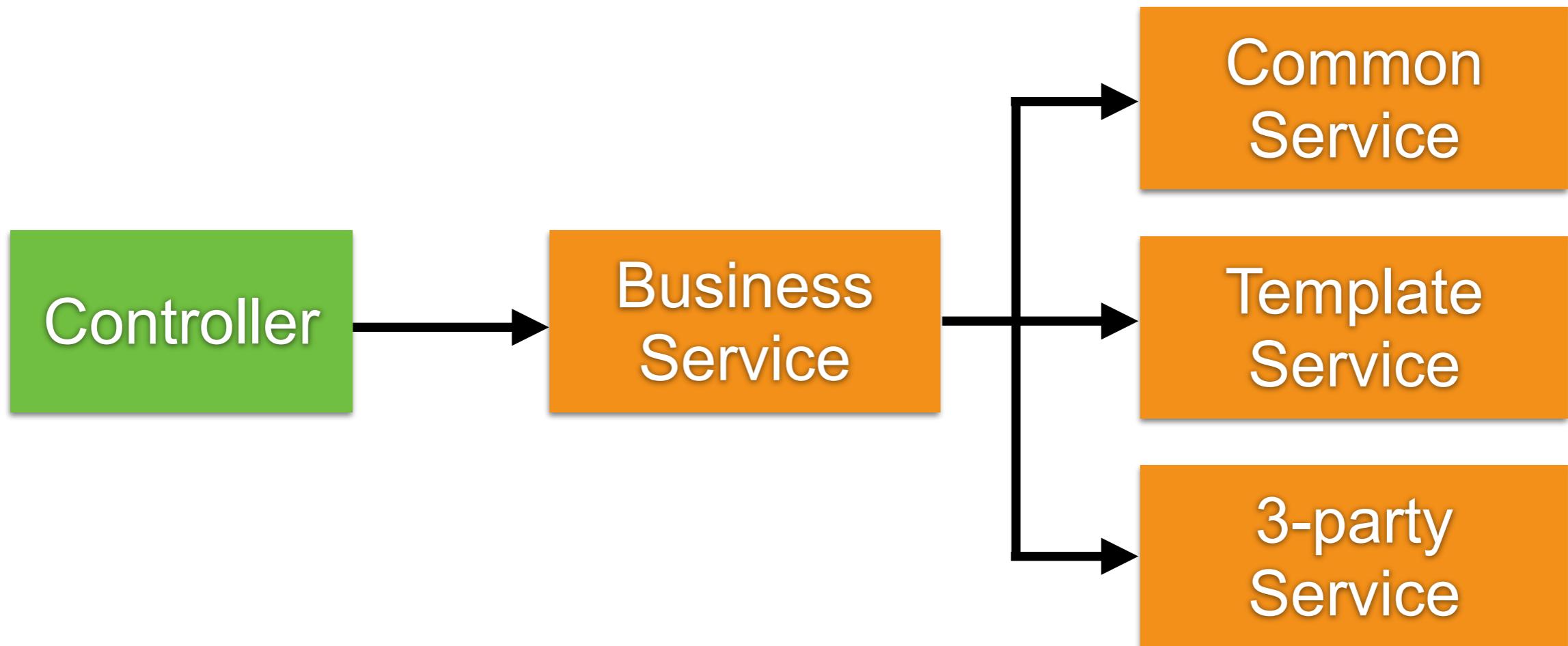
# Move business logic to service

Service class or interface ?

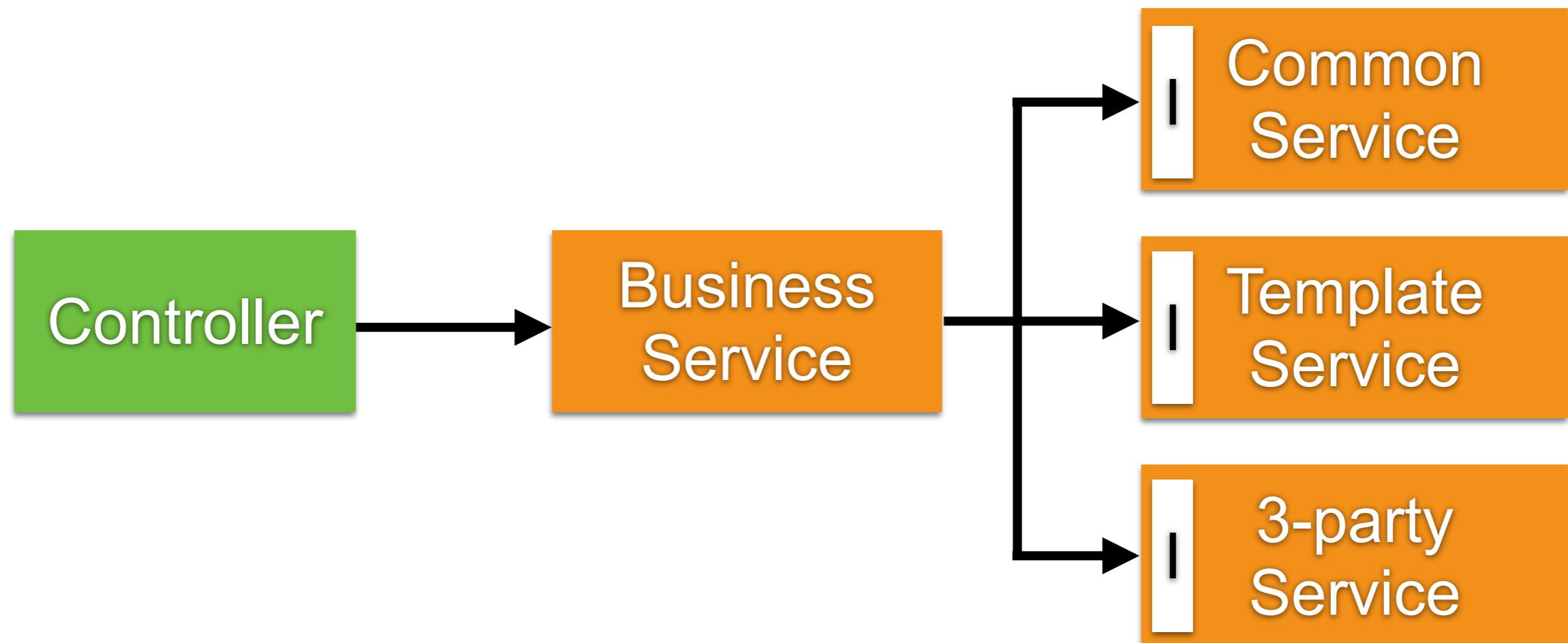


# Types of service

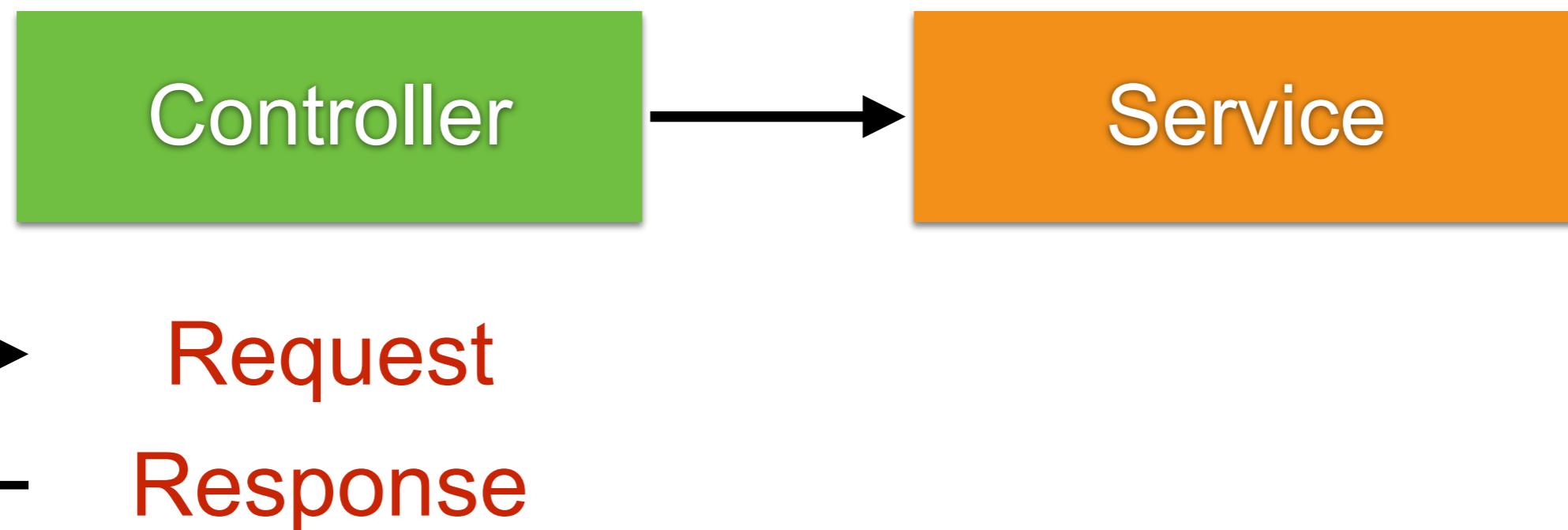
Business services (not reuse)  
Common/Template/3-party services



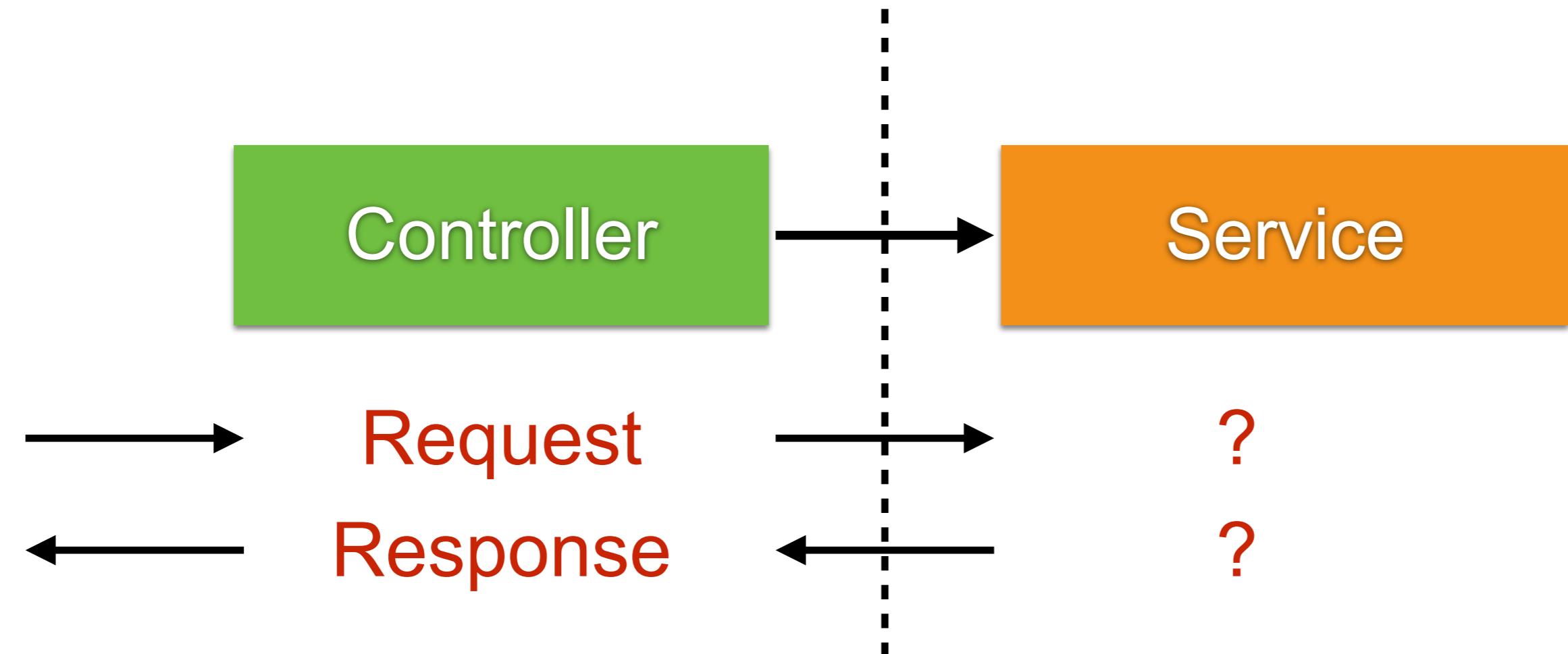
# Interface for common service



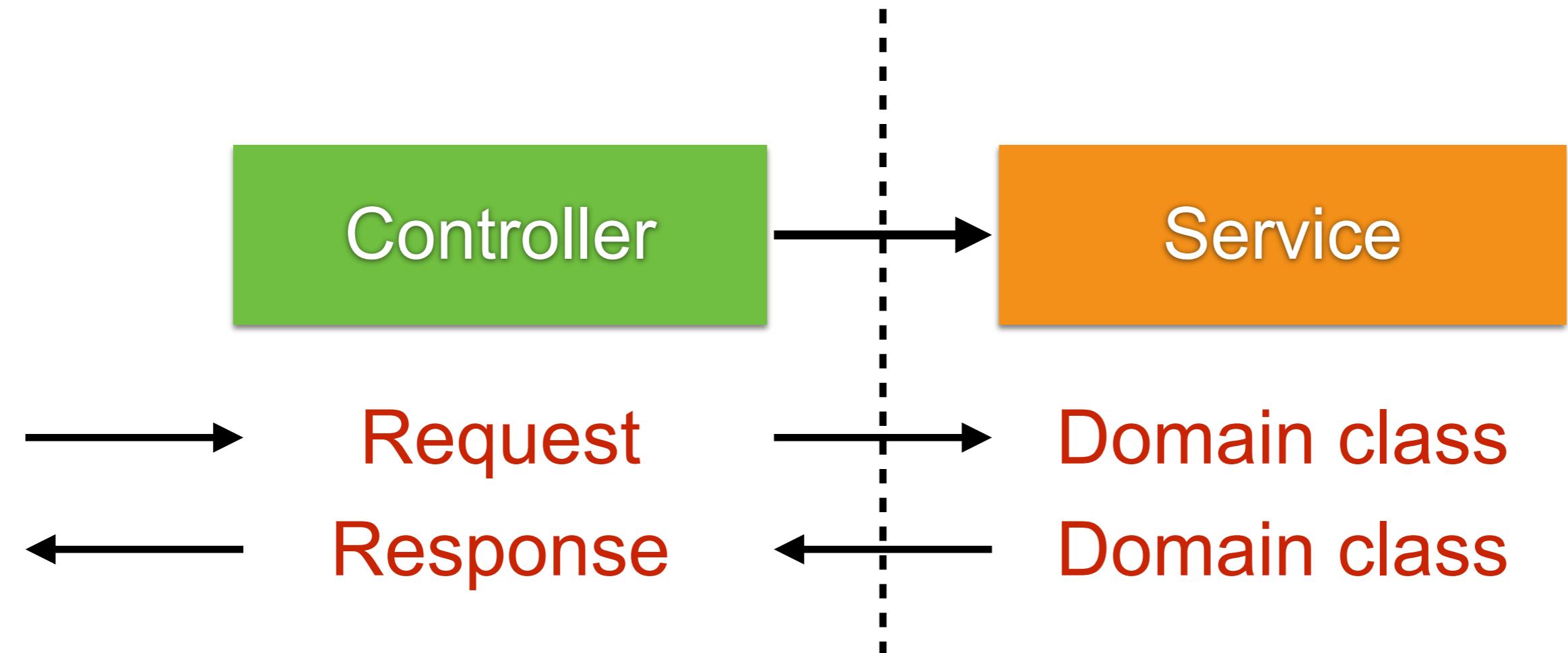
# Data Model for service ?



# Data Model ?

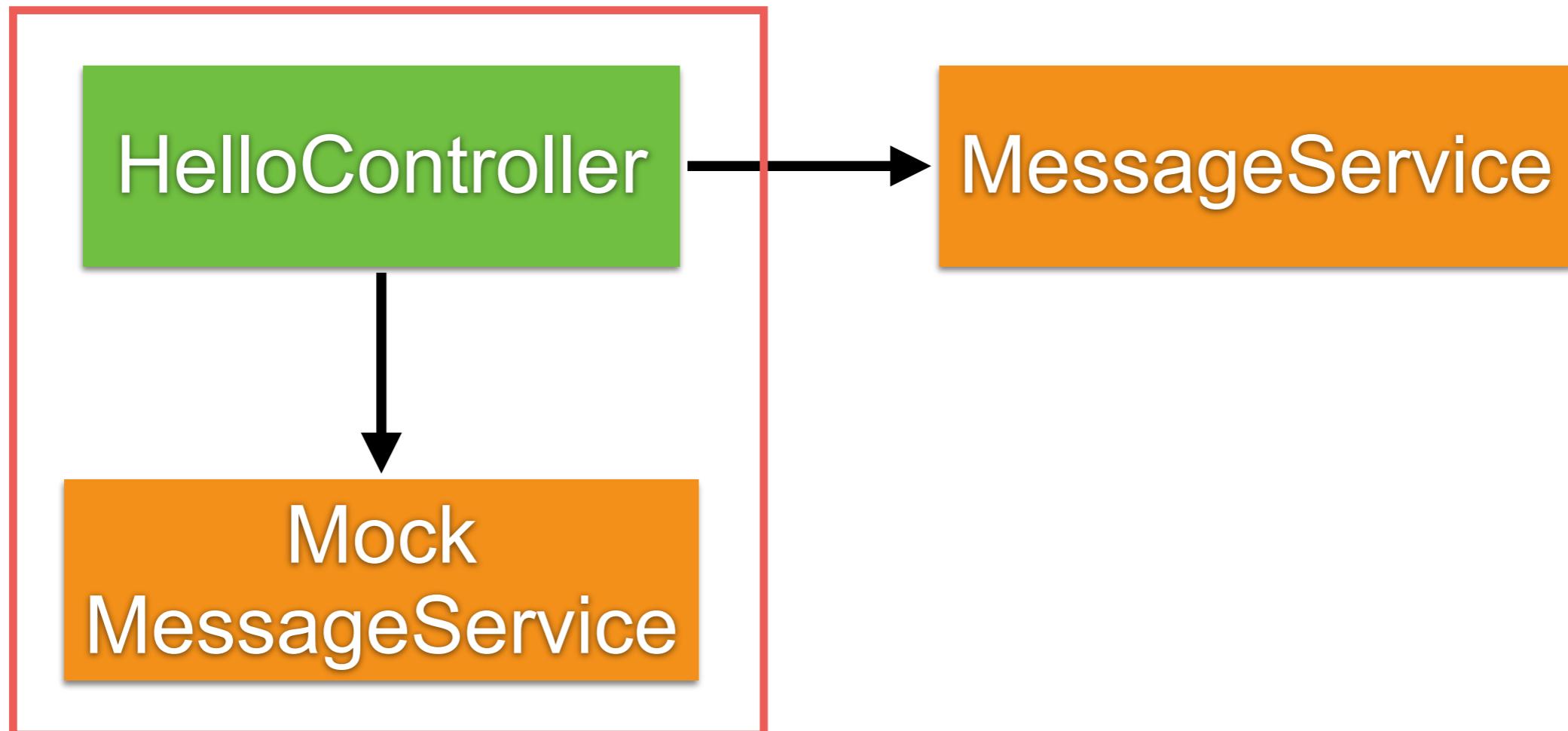


# Data Model ?



# Testing controller with service

Try to mocking service with Mockito



# Run all tests !!

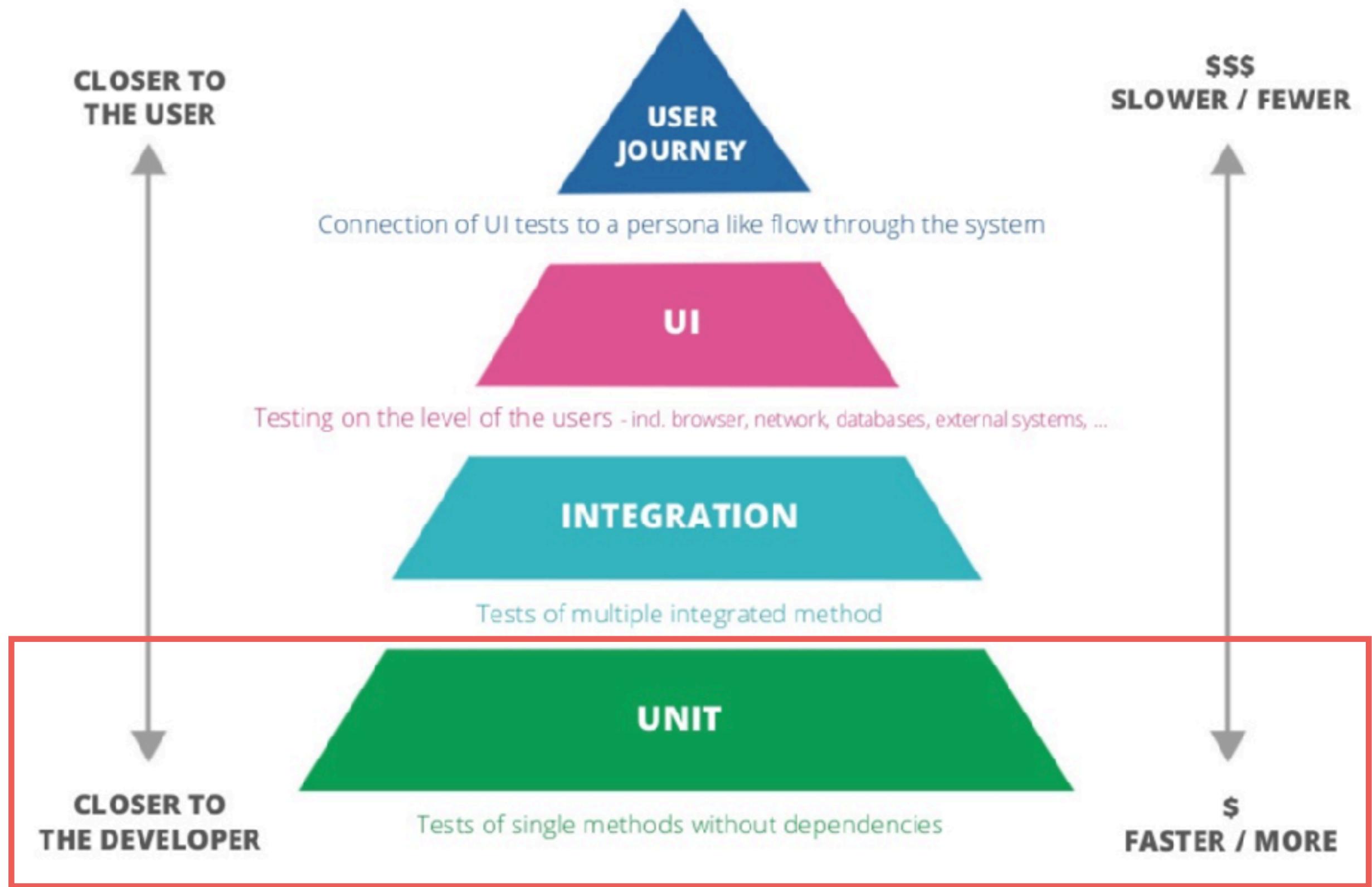
\$mvnw clean test

```
[INFO]
[INFO] Results:
[INFO]
[WARNING] Tests run: 10, Failures: 0, Errors: 0,
[INFO]
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 18.299 s
[INFO] Finished at: 2018-08-20T23:36:31+07:00
[INFO] -----
```

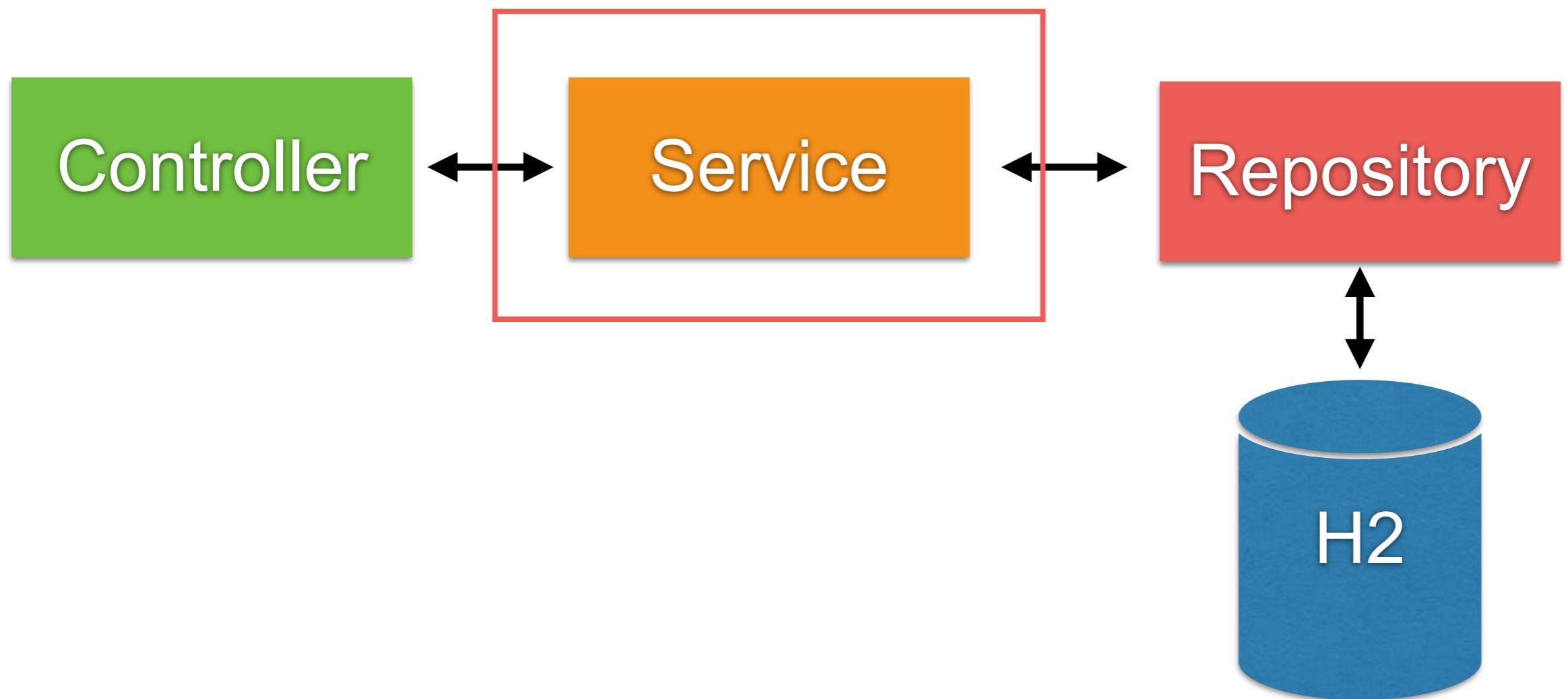


# How to improve the speed of testing ?





# Service Testing ?



# Service Testing

```
@ExtendWith(MockitoExtension.class)
public class UserServiceTest {

    @Mock
    private AccountRepository accountRepository;

    @Test
    public void getAccount() {
        // Stub
        Account account = new Account();
        account.setUserName("user");
        account.setPassword("pass");
        account.setSalary(1000);
        given(accountRepository.findById(1))
            .willReturn(Optional.of(account));

        UserService userService = new UserService(accountRepository);
        Account actualAccount = userService.getAccount(1);
        assertNotNull(actualAccount);
    }
}
```

1



# Service Testing

```
@ExtendWith(MockitoExtension.class)
```

```
public class UserServiceTest {
```

```
    @Mock
```

```
    private AccountRepository accountRepository;
```

```
    @Test
```

```
    public void getAccount() {
```

```
        // Stub
```

```
        Account account = new Account();
```

```
        account.setUserName("user");
```

```
        account.setPassword("pass");
```

```
        account.setSalary(1000);
```

```
        given(accountRepository.findById(1))
```

```
            .willReturn(Optional.of(account));
```



2

```
        UserService userService = new UserService(accountRepository);
```

```
        Account actualAccount = userService.getAccount(1);
```

```
        assertNotNull(actualAccount);
```

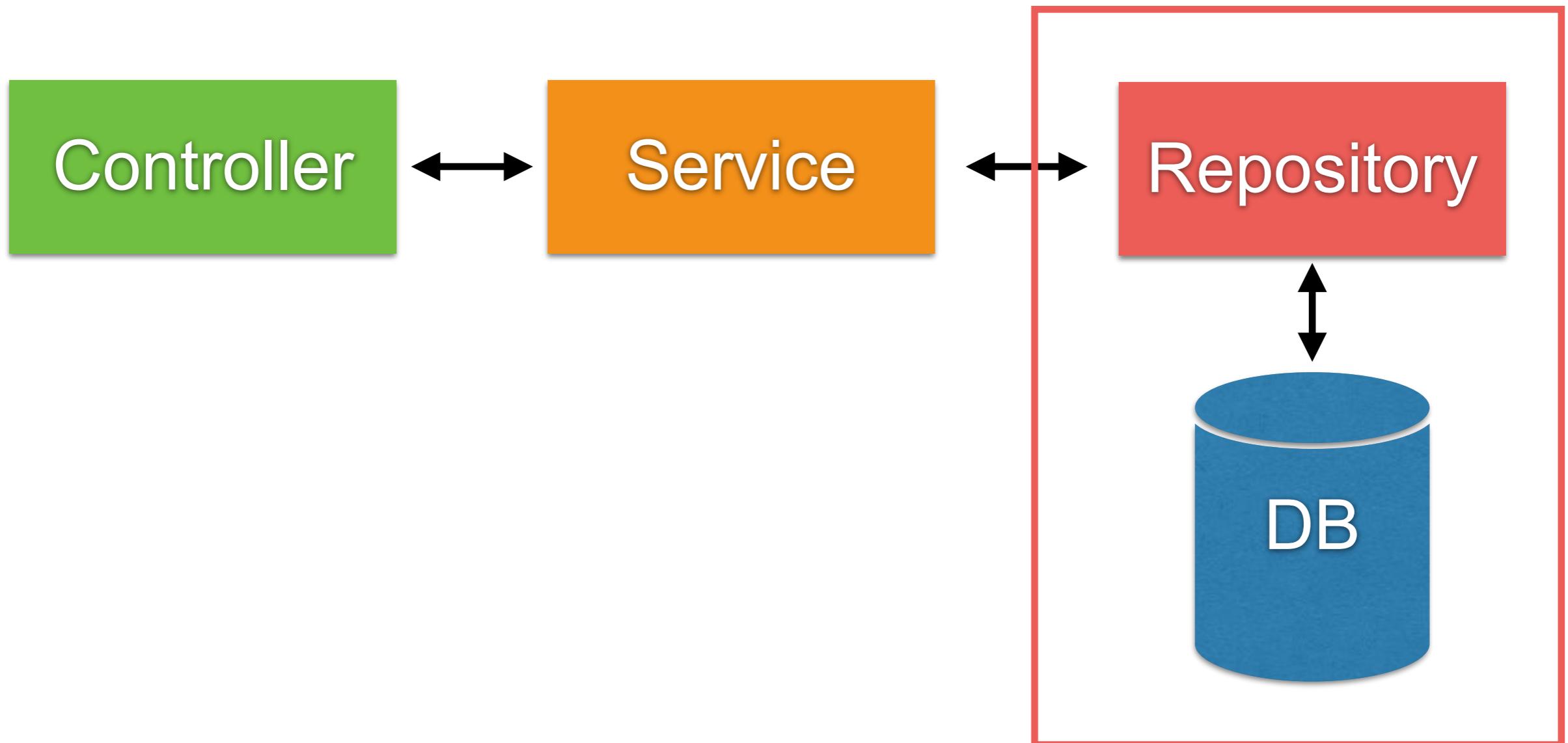
```
}
```



# Working with Repository



# Working with repository



# Basic of JDBC

```
// 1. Load jdbc driver
Class.forName("postgresql");

// 2. Create connection
Connection connection = DriverManager.getConnection("", "", "");

// 3. Prepared Statement
String sql = "SELECT * FROM TABLE WHERE name=?";
PreparedStatement pStmt = connection.prepareStatement(sql);

// 4. Query
ResultSet resultSet = pStmt.executeQuery();
while(resultSet.next()) {

}

// 5. Release resource
if(resultSet != null) {
    resultSet.close();
    resultSet = null;
}
```



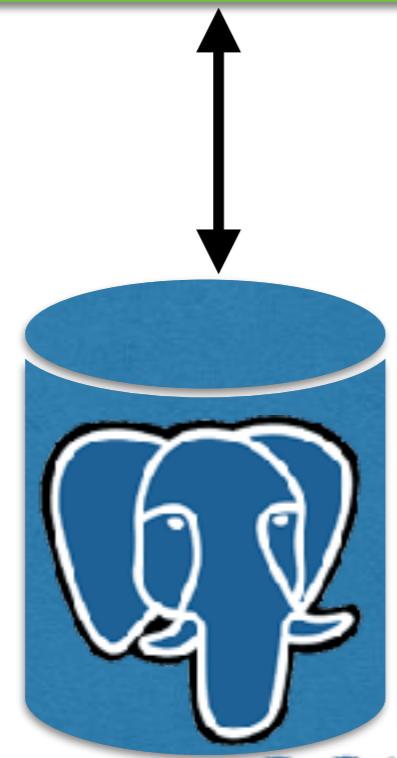
# Framework !!

```
// 1. Load jdbc driver  
Class.forName("postgresql");  
Manage by Framework  
// 2. Create connection  
Connection connection = DriverManager.getConnection("", "", "");  
  
// 3. Prepared Statement  
String sql = "SELECT * FROM TABLE WHERE name=?";  
PreparedStatement pStmt = connection.prepareStatement(sql);  
  
// 4. Query  
ResultSet resultSet = pStmt.executeQuery();  
while(resultSet.next()) {  
  
}  
  
// 5. Release resource  
if(resultSet != null) {  
    resultSet.close();  
    resultSet = null;  
}
```

**Manage by Framework**

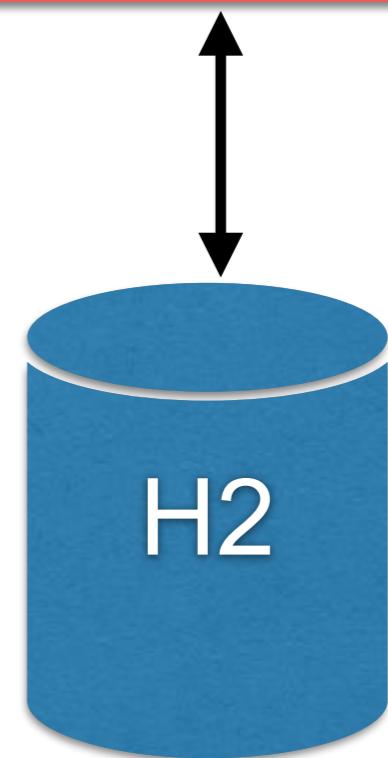
# Working with Database ?

Production



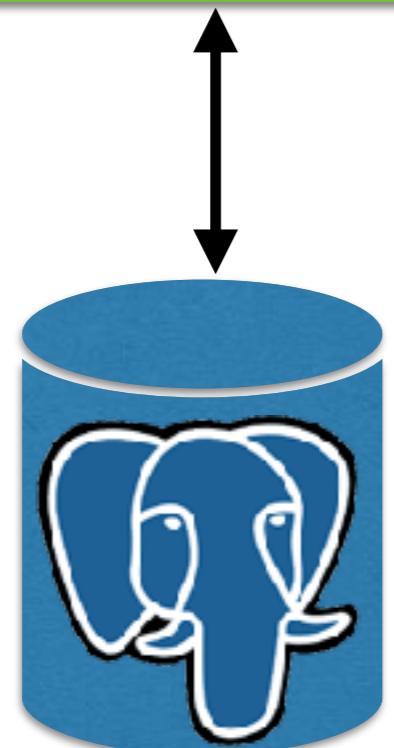
PostgreSQL

Testing



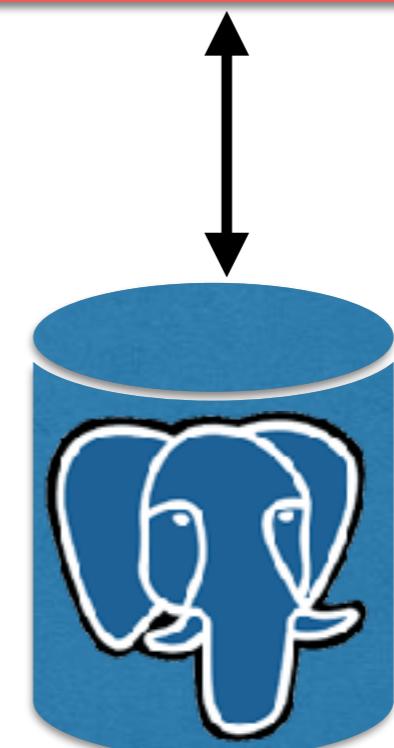
# Working with Database ?

Production



PostgreSQL

Testing



PostgreSQL



# Working with repository

We're using Spring Data



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



# Spring Data

JDBC

JPA

MongoDB

Redis

more ...

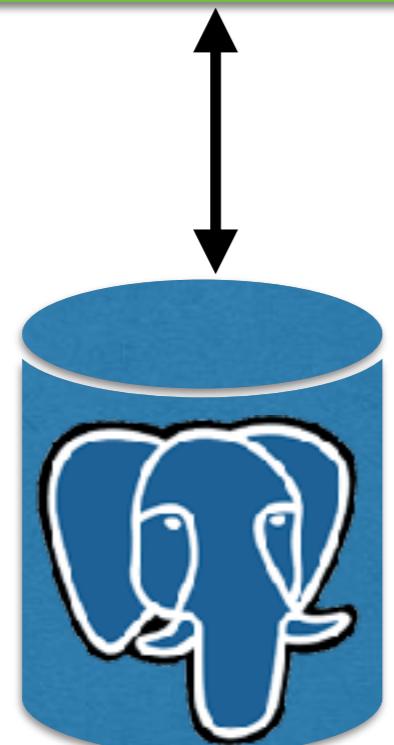


# Working with Spring Data JPA



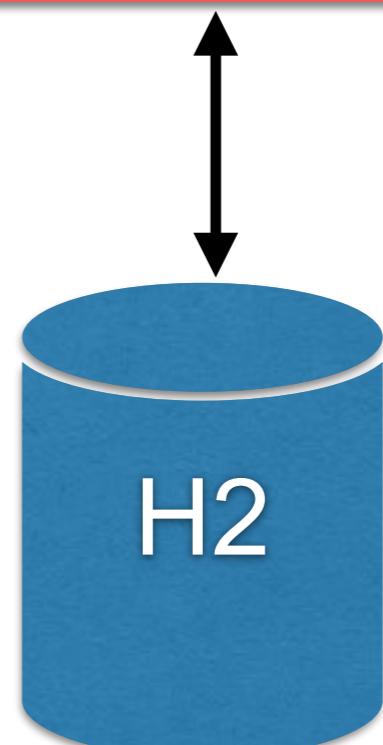
# Working with Database

Production



PostgreSQL

Testing



# Modify pom.xml

Add library of Spring Data JPA, PostgreSQL, H2

**Dependencies**

**ADD DEPENDENCIES... ⌂ + B**

---

**PostgreSQL Driver** SQL

A JDBC and R2DBC driver that allows Java programs to connect to a PostgreSQL database using standard, database independent Java code.

---

**H2 Database** SQL

Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

---

**Spring Data JPA** SQL

Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

<https://start.spring.io/>



# Modify pom.xml

## H2 for testing

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>
```

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>
```

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```



# Modify pom.xml

## PostgreSQL for production

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-data-jpa</artifactId>
</dependency>

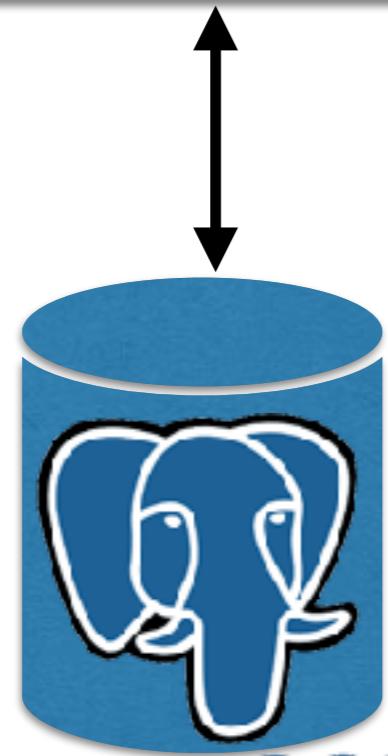
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>

<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency>
```



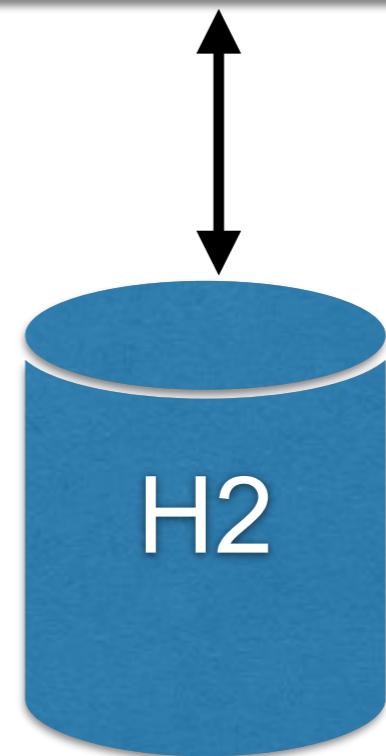
# Start in testing scope

Production

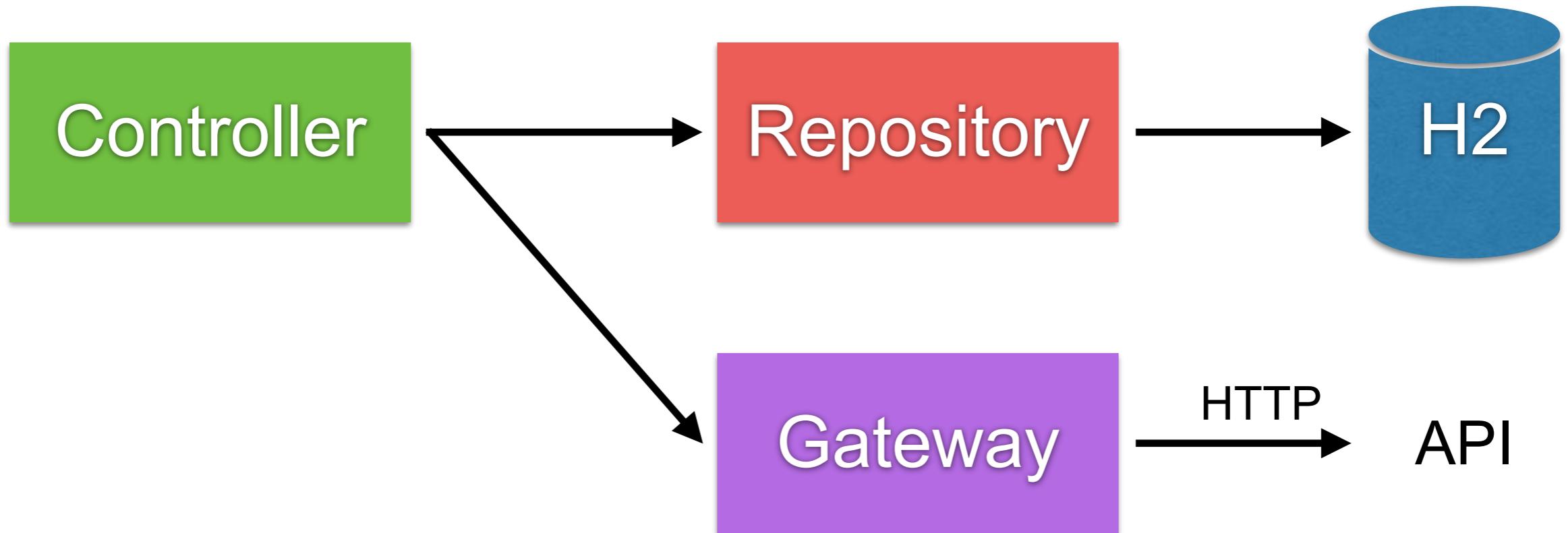


PostgreSQL

Testing

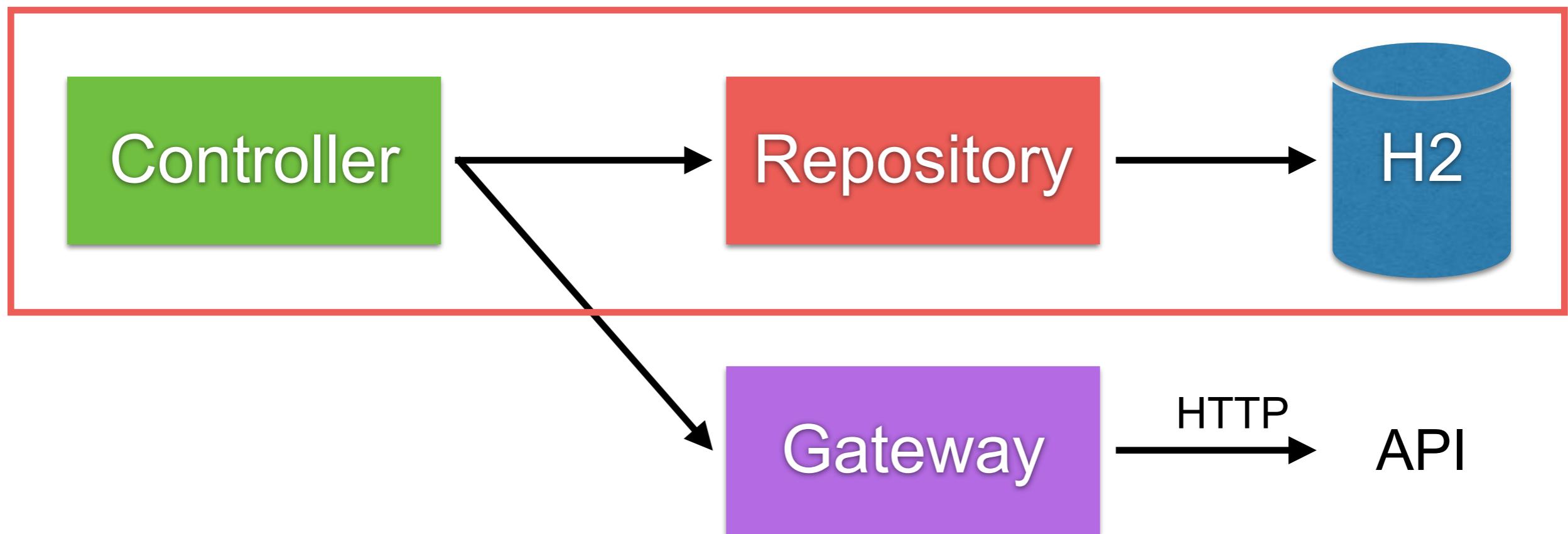


# Use cases



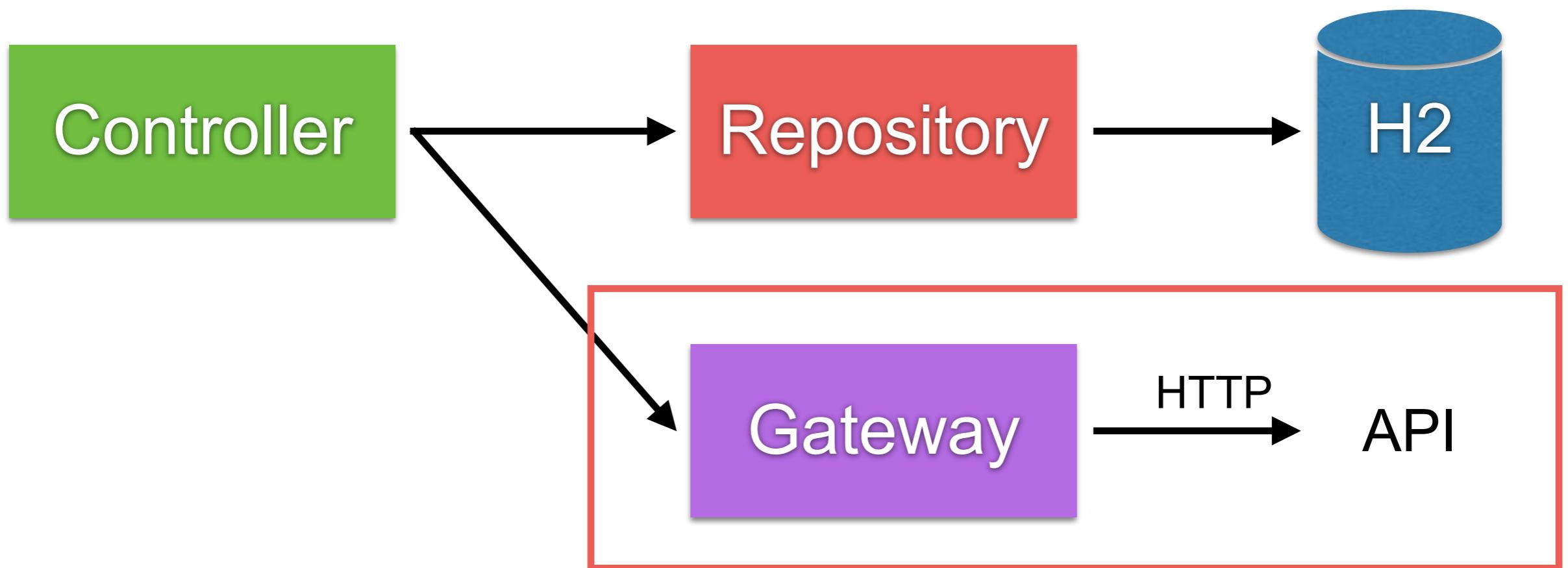
# Use case 1

## Working with repository



# Use case 2

## Working with API



# Use case 1

## Working with Database



# Use case 1

Working with repository

3. HelloController



2. PersonRepository



Controller

Repository

H2



4. PersonResponse



1. Person



# 1. Create Entity class

In package person

```
@Entity  
public class Person {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private long id;  
    private String firstName;  
    private String lastName;  
  
    public Person() {  
    }  
}
```



# 2. Create repository with JPA

## PersonRepository.java

```
import java.util.Optional;  
  
import org.springframework.data.repository.CrudRepository;  
  
public interface PersonRepository  
    extends CrudRepository<Person, Long> {  
  
    Optional<Person> findByLastName(String lastName);  
  
}
```



# 2. Create repository with JPA

## PersonRepository.java

```
import java.util.Optional;  
  
import org.springframework.data.repository.CrudRepository;  
  
public interface PersonRepository  
    extends CrudRepository<Person, Long> {  
  
    Optional<Person> findByLastName(String lastName);  
}
```

*SELECT \* FROM Person WHERE LastName=?*

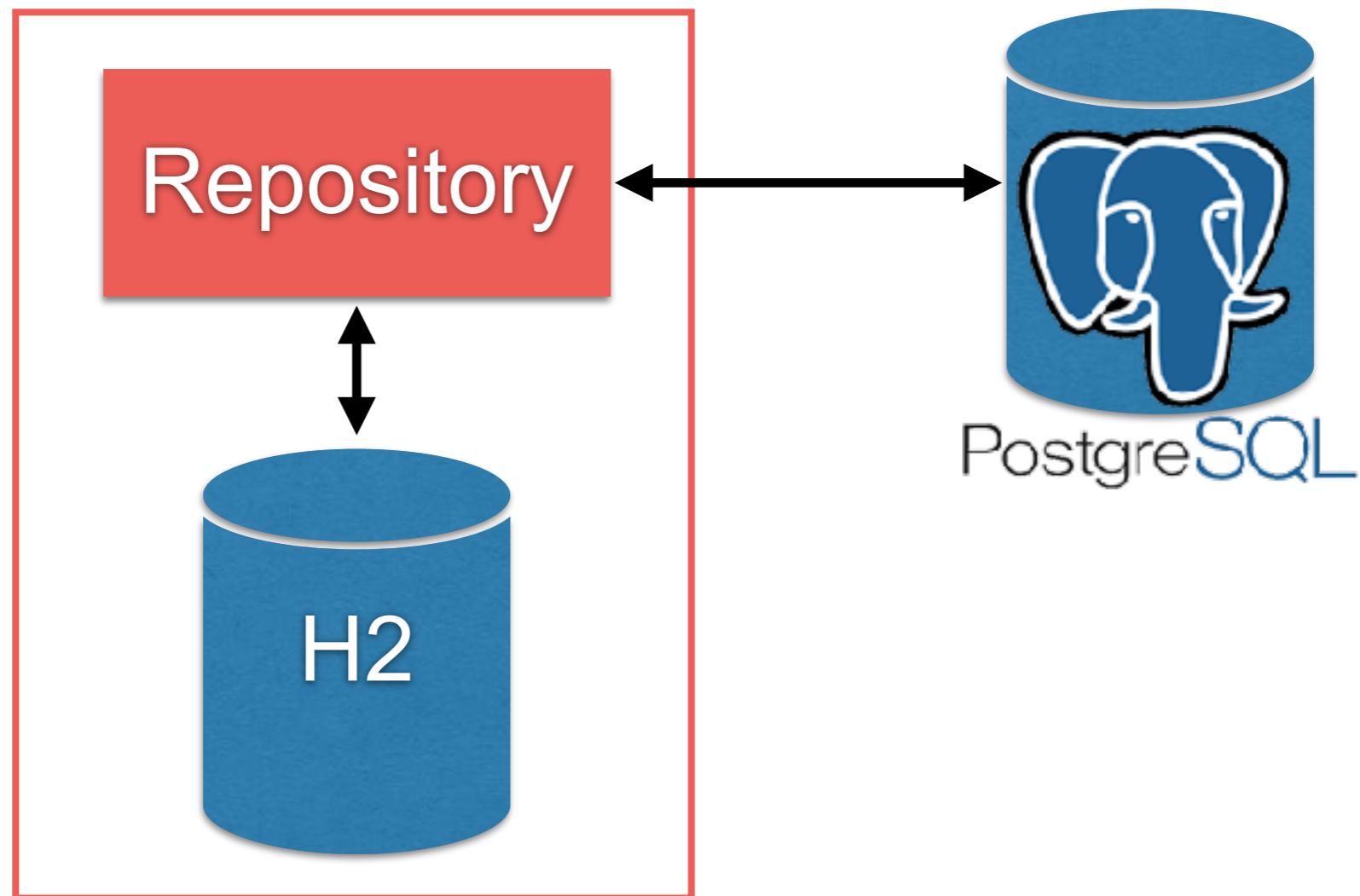


# How to testing repository ?



# Repository Testing

Using `@DataJpaTest` (slice testing)



Working with In-memory database



# Repository Testing #1

## Setup test with @DataJpaTest

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class PersonRepositoryTest {

    @Autowired
    private PersonRepository repository;

    @After
    public void tearDown() throws Exception {
        repository.deleteAll();
    }

}
```



# Repository Testing #2

## Auto wired repository for testing

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class PersonRepositoryTest {

    @Autowired
    private PersonRepository repository;

    @After
    public void tearDown() throws Exception {
        repository.deleteAll();
    }

}
```



# Repository Testing #3

Clear data in table after executed each test case

```
@RunWith(SpringRunner.class)
@DataJpaTest
public class PersonRepositoryTest {

    @Autowired
    private PersonRepository repository;

    @After
    public void tearDown() throws Exception {
        repository.deleteAll();
    }

}
```



# Repository Testing #3

Write your first test case

```
@Test  
public void should_save_fetch_a_person() {  
  
    Person somkiat = new Person("Somkiat", "Pui");  
    repository.save(somkiat);  
  
    Optional<Person> maybeSomkiat  
        = repository.findByLastName("Pui");  
  
    assertEquals(maybeSomkiat, Optional.of(somkiat));  
}
```



# Run test

\$mvnw clean test

Hibernate: drop table person if exists

Hibernate: drop sequence if exists hibernate\_sequence

Hibernate: create sequence hibernate\_sequence start with 1 increment by 1

Hibernate: create table person (id varchar(255) not null, first\_name varchar(255), last\_name varchar(255), primary key (id))

Insert data

Hibernate: call next value for hibernate\_sequence

Hibernate: insert into person (first\_name, last\_name, id) values (?, ?, ?)

Hibernate: select person0\_.id as id1\_0\_, person0\_.first\_name as first\_na2\_0\_, person0\_.last\_name as last\_nam3\_0\_ from person person0\_ where person0\_.last\_name=?

Hibernate: select person0\_.id as id1\_0\_, person0\_.first\_name as first\_na2\_0\_, person0\_.last\_name as last\_nam3\_0\_ from person person0\_

2

Query data

Hibernate: drop table person if exists

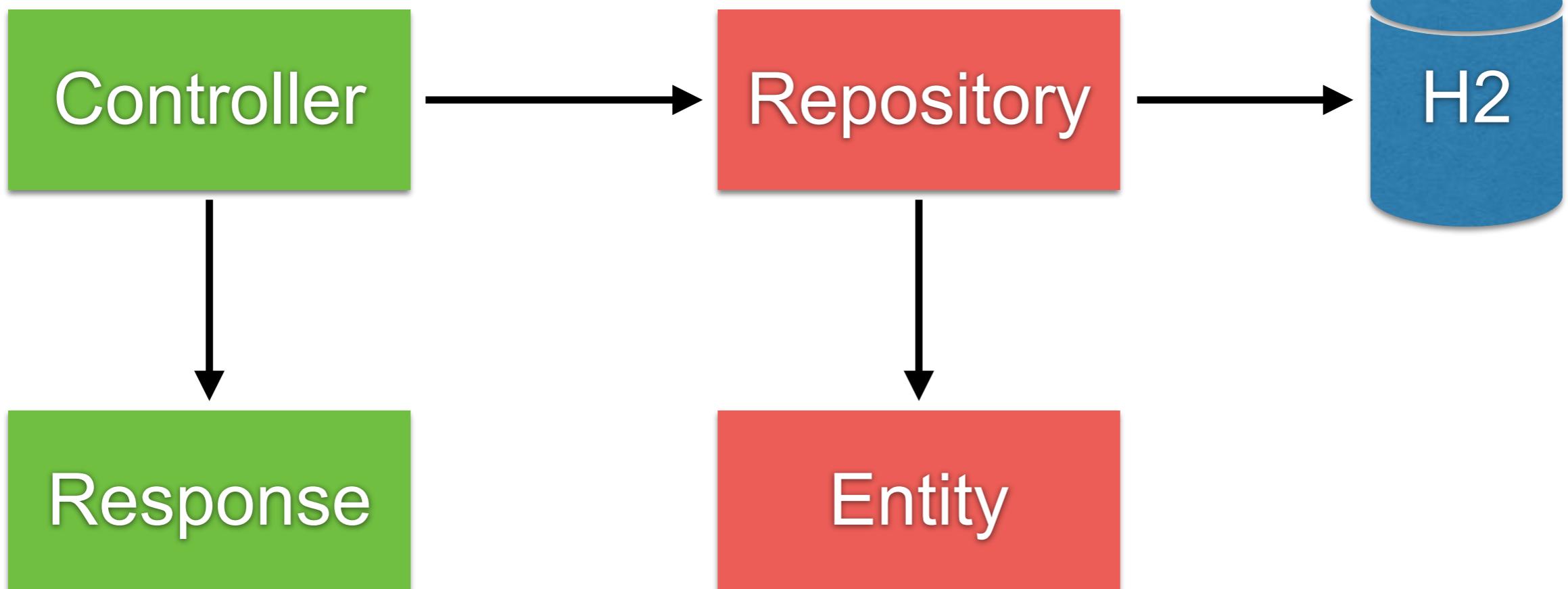
Hibernate: drop sequence if exists hibernate\_sequence



# Use case 1

Integrate repository with controller

3. HelloController      2. PersonRepository



1. Person

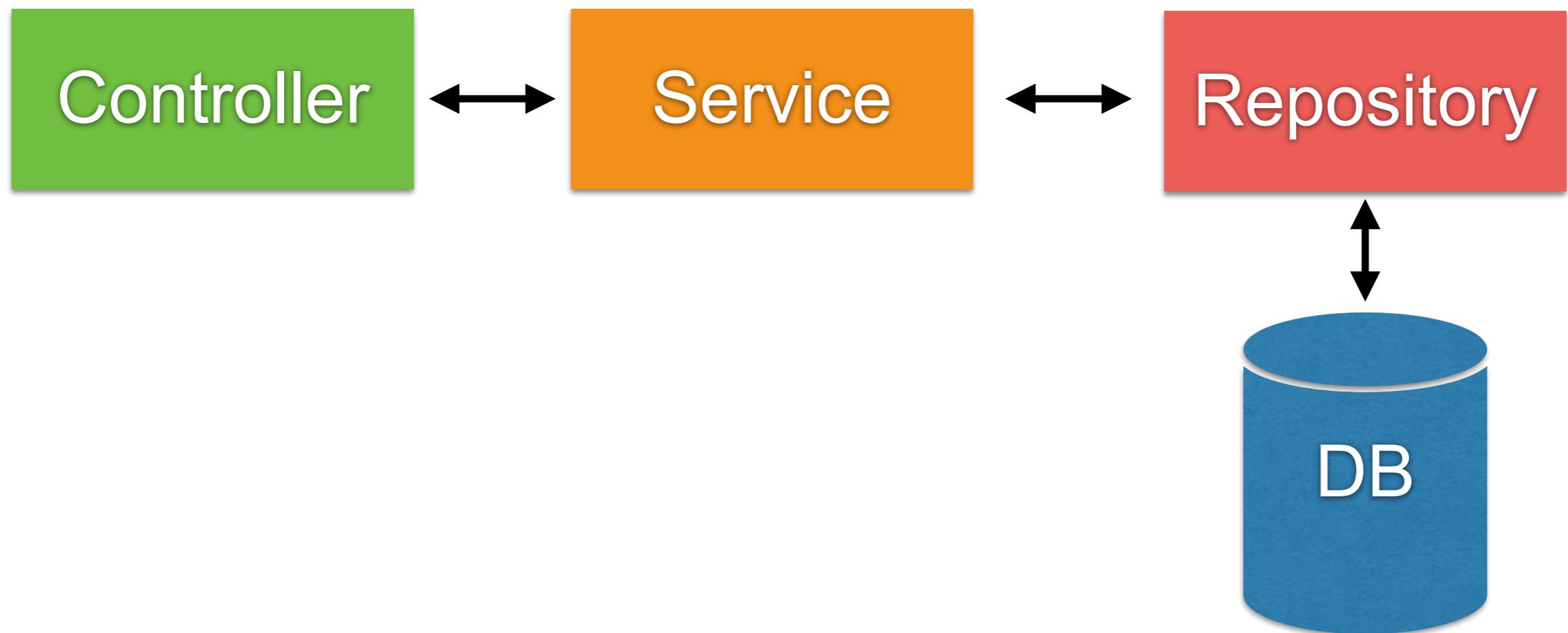
4. PersonResponse



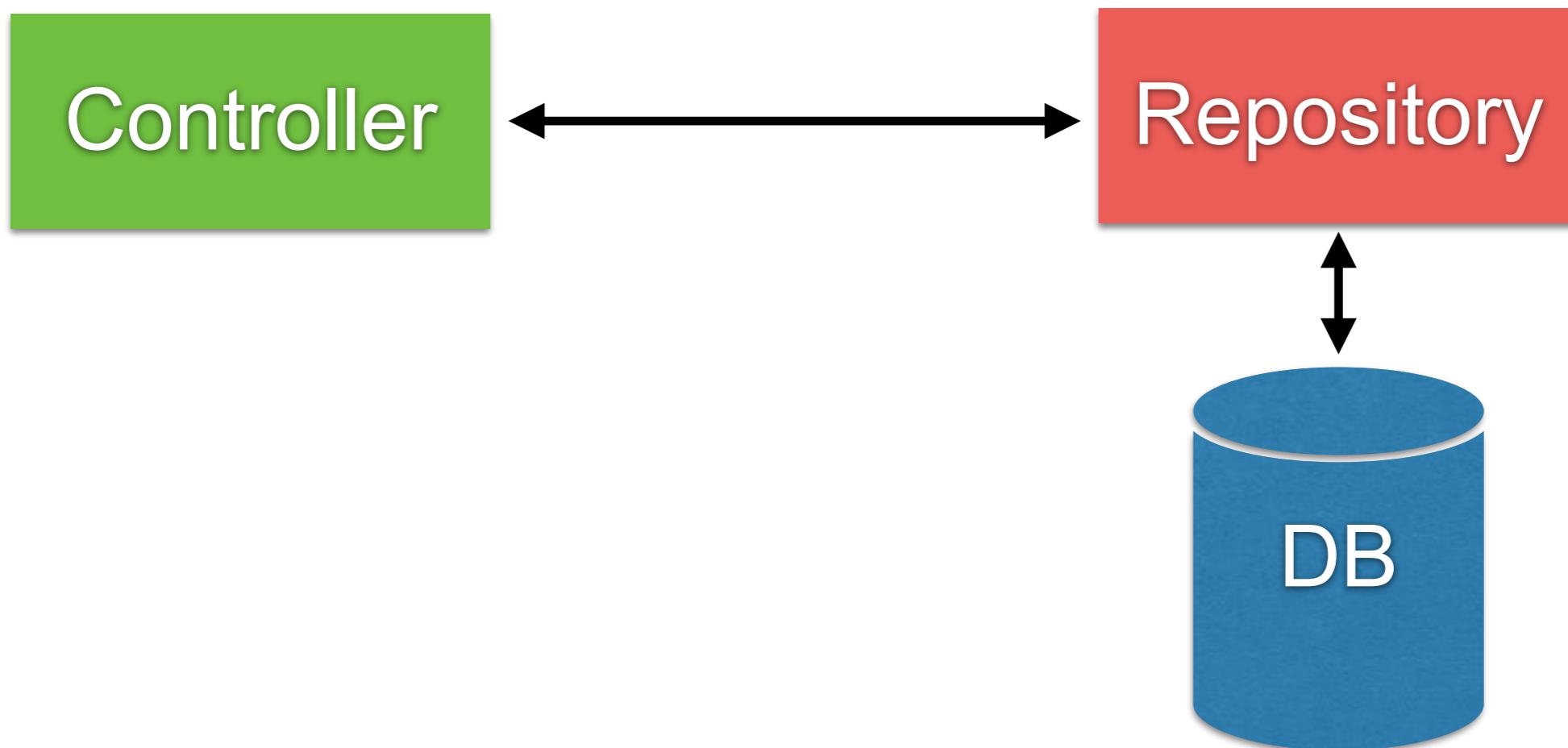
# Integrate repository with service/controller



# Service use repository ?



# Controller use repository ?



# Controller call repository

## Create HelloController.java

```
@RestController
public class HelloController {

    private final PersonRepository personRepository;

    @Autowired
    public HelloController(final PersonRepository personRepository) {
        this.personRepository = personRepository;
    }
}
```



# Controller call repository

## Create HelloController.java

```
@GetMapping("/hello/{lastName}")
public HelloResponse hello(@PathVariable final String lastName) {

    Optional<Person> foundPerson
        = personRepository.findByLastName(lastName);

    return foundPerson
        .map(person ->
            new HelloResponse(person.getFirstName(),
                person.getLastName()))
        .orElseThrow(() -> new RuntimeException());
}
```



# Run spring boot

\$mvnw spring-boot:run



# Fix !!!

## Modify src/main/resources/application.properties

server.port=8088

spring.datasource.url=jdbc:postgresql://127.0.0.1:15432/postgres

spring.datasource.username=testuser

spring.datasource.password=password

spring.datasource.platform=POSTGRESQL

spring.jpa.show-sql=true

spring.jpa.hibernate.ddl-auto=create-drop

spring.jpa.database-platform=org.hibernate.dialect.PostgreSQLDialect

## Start database server !!



# Fix !!!

Modify pom.xml  
Delete or comment postgresql dependency

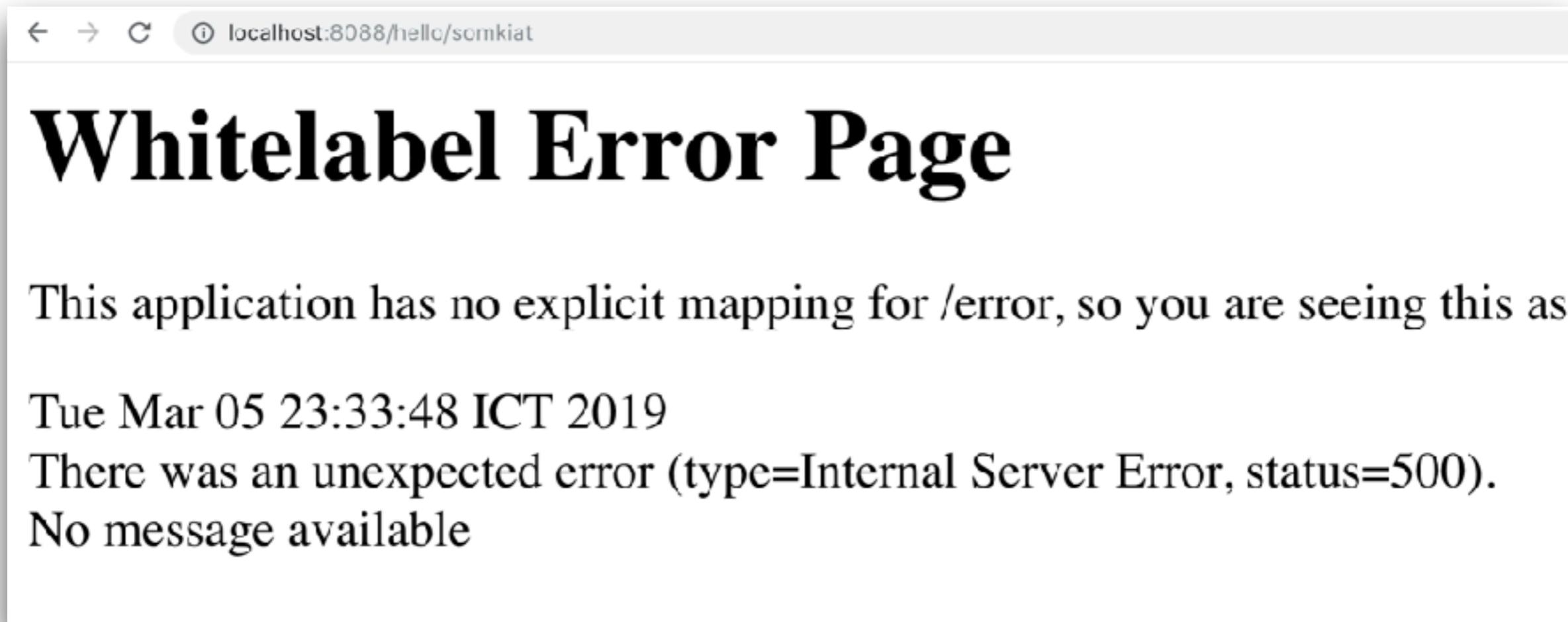
```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- <dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <scope>runtime</scope>
</dependency> -->
```



# Run spring boot

\$mvnw spring-boot:run



# Initial data in database



# Initial database #1

## Using @Bean and CommandLineRunner

```
@Bean
public CommandLineRunner initData(MessageRepository repository) {
    return new CommandLineRunner() {

        @Override
        public void run(String... args) throws Exception {
            repository.save(new Message("somkiat1"));
            repository.save(new Message("somkiat2"));
        }
    };
}
```



# Initial database #2

## Using @PostConstruct

```
@PostConstruct  
public void initData() {  
    Account account1 = new Account();  
    account1.setAccountId("01");  
    accountRepository.save(account1);  
    Account account2 = new Account();  
    account2.setAccountId("02");  
    accountRepository.save(account2);  
}
```



# Initial database #3

**Schema** (resources/schema.sql)

**Data** (resources/data.sql)

## Schema.sql

```
CREATE TABLE account(
    id BIGINT AUTO_INCREMENT PRIMARY KEY,
    account_Id VARCHAR(16) NOT NULL UNIQUE,
    mobile_No VARCHAR(10),
    name VARCHAR(50),
    account_Type CHAR(2)
);
```

## Data.sql

```
INSERT INTO account (account_Id) VALUES ('01');
INSERT INTO account (account_Id) VALUES ('02');
```



# Initial database 2

Disable auto generate DDL from JPA in file application.yml

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none
```



# Initial database 2

## Problem with naming strategy !!

```
spring:  
  jpa:  
    show-sql: true  
    hibernate:  
      ddl-auto: none  
      naming:  
        physical-strategy:  
          org.springframework.boot.orm.jpa.hibernate.SpringPhysicalNamingStrategy  
        implicit-strategy:  
          org.springframework.boot.orm.jpa.hibernate.SpringImplicitNamingStrategy
```

<https://github.com/spring-projects/spring-boot/tree/master/spring-boot-project/spring-boot/src/main/java/org/springframework/boot/orm/jpa/hibernate>



# Run and see from logging

Execute file schema.sql and data.sql

```
.datasource.init.ScriptUtils      : Executing SQL script from URL [file:/Users/somki...  
.datasource.init.ScriptUtils      : Executed SQL script from URL [file:/Users/somki...  
.datasource.init.ScriptUtils      : Executing SQL script from URL [file:/Users/somki...  
.datasource.init.ScriptUtils      : Executed SQL script from URL [file:/Users/somki...
```



# Run spring boot

\$mvnw spring-boot:run



# **Write controller testing ?**

**\$mvnw clean test**



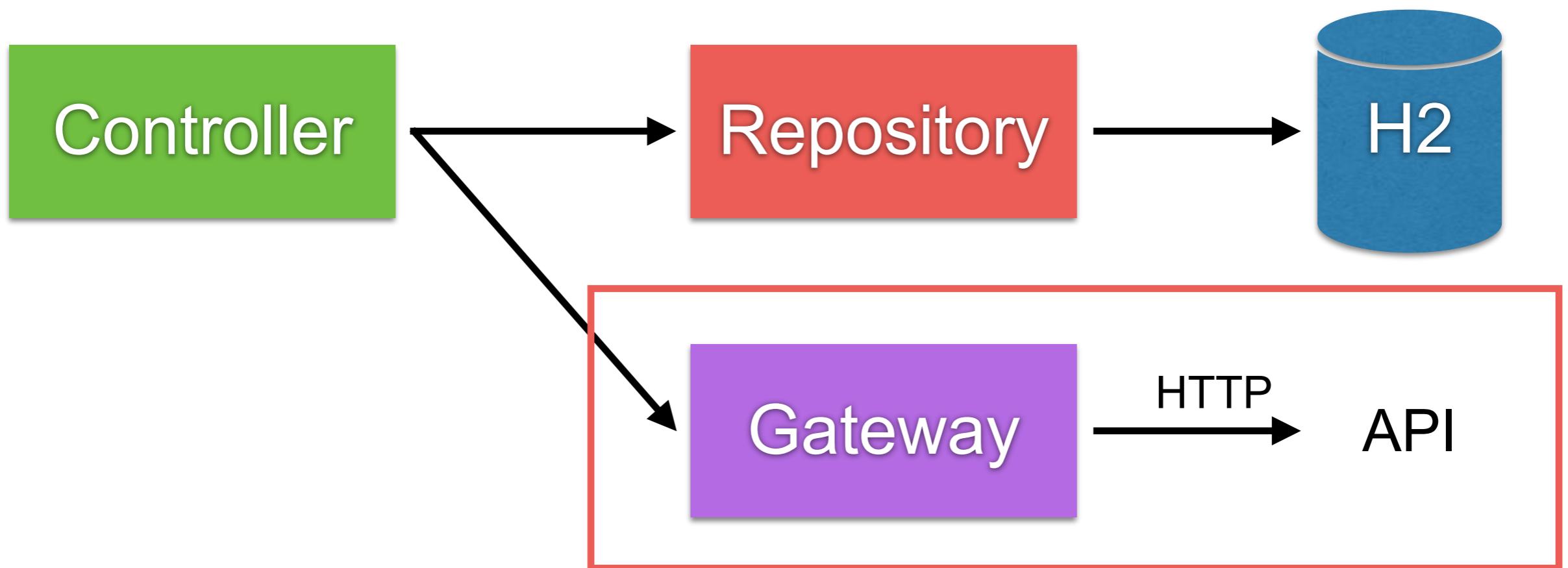
# Use case 2

## Working with API



# Use case 2

## Working with API



# JSON Place Holder

<https://jsonplaceholder.cypress.io/posts/1>

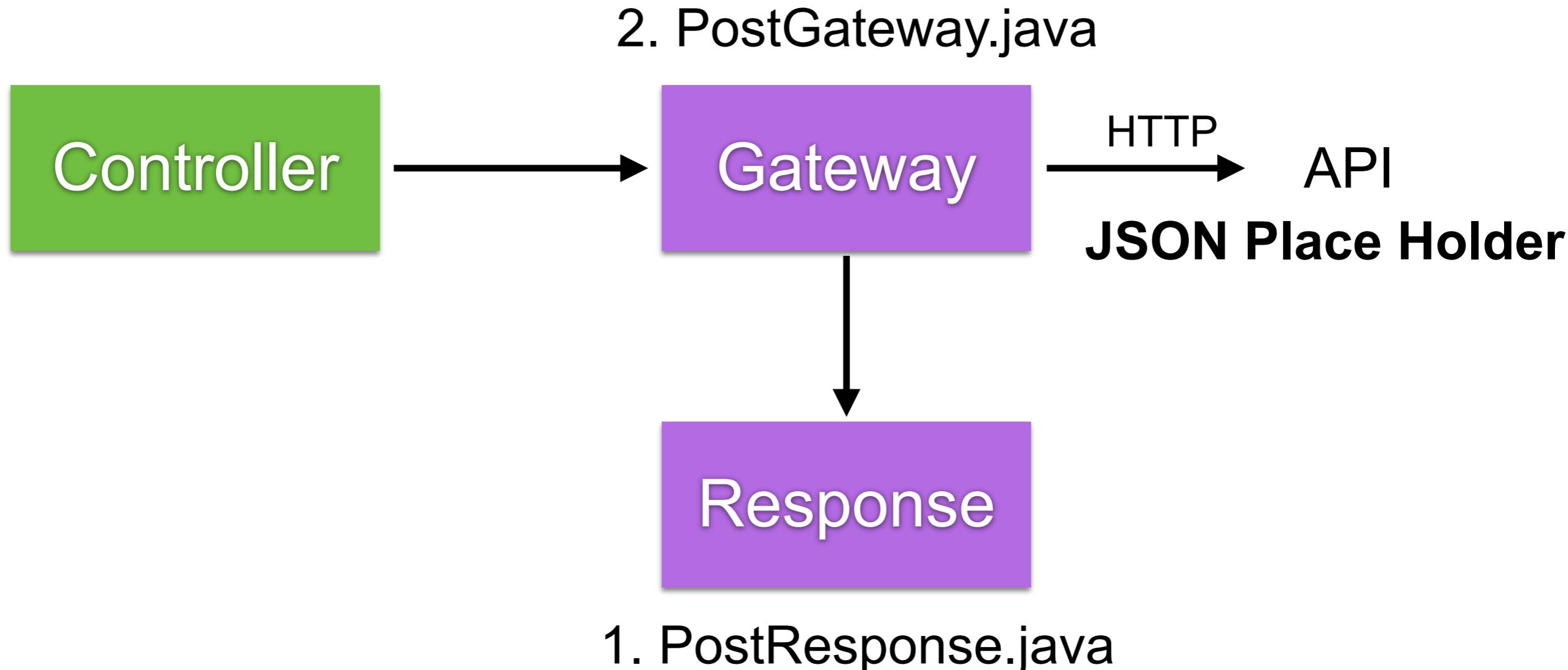
The screenshot shows a web-based API testing interface for JSONPlaceholder. At the top, the title "JSONPlaceholder" is displayed in large, bold letters. Below it, a subtitle reads "Fake Online REST API for Testing and Prototyping" and "Powered by [JSON Server + LowDB](#)". A code snippet in a light gray box illustrates a JavaScript fetch call:

```
fetch('https://jsonplaceholder.cypress.io/todos/1')
  .then(response => response.json())
  .then(json => console.log(json))
```

At the bottom of the interface is a blue button labeled "Try it".



# Working with API



# Library to call external APIs

Rest Template

OpenFeign

HTTP Interface

Spring framework 6



# 1. Create Response class

In package post

```
public class PostResponse {  
    private int id;  
    private int userId;  
    private String title;  
    private String body;
```



# 2. Create PostGateway class #1

In package post

```
@Component
public class PostGateway {

    private final RestTemplate restTemplate;
    private final String postApiUrl;

    @Autowired
    public PostGateway(final RestTemplate restTemplate,
                       @Value("${post.api.url}") final String postApiUrl) {
        this.restTemplate = restTemplate;
        this.postApiUrl = postApiUrl;
    }
}
```



# 2. Create PostGateway class #1

In package post

```
@Component
public class PostGateway {

    private final RestTemplate restTemplate;
    private final String postApiUrl;

    @Autowired
    public PostGateway(final RestTemplate restTemplate,
        @Value("${post.api.url}") final String postApiUrl) {
        this.restTemplate = restTemplate;
        this.postApiUrl = postApiUrl;
    }
}
```

Configuration ?



# Configuration

Configuration in file application.properties

```
post.api.url=https://jsonplaceholder.cypress.io
```



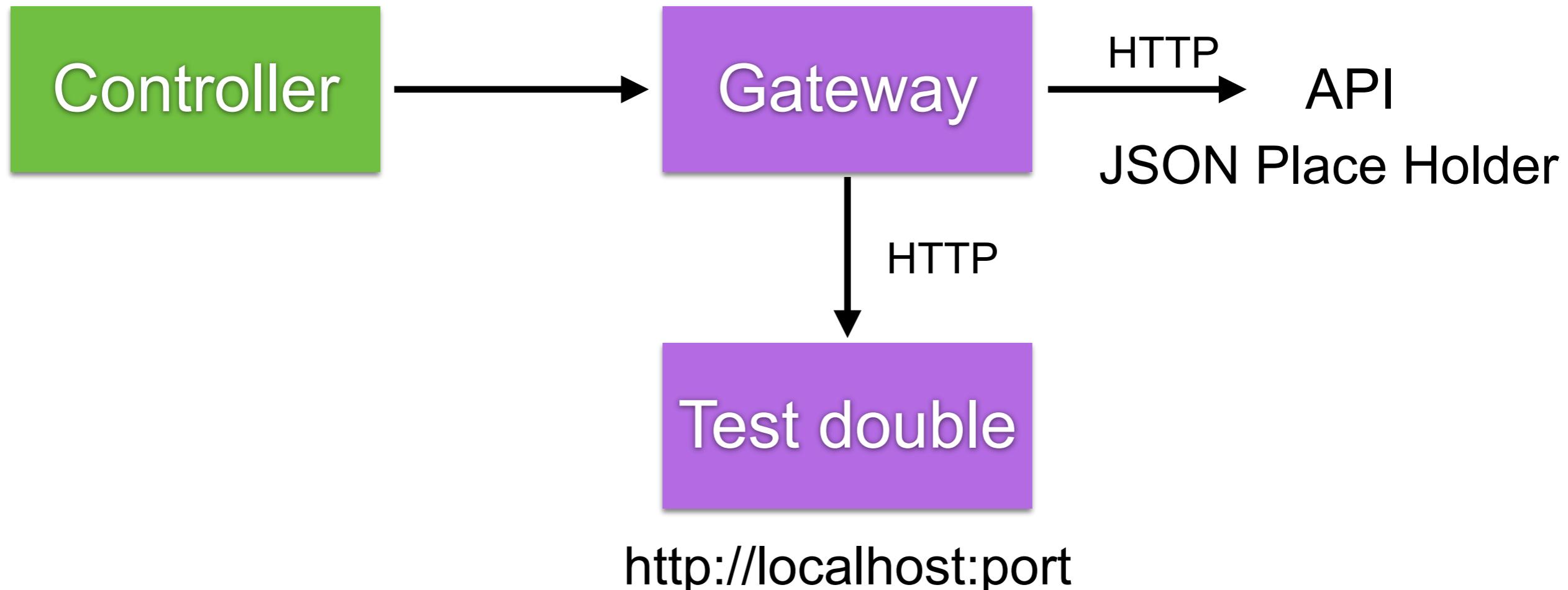
# 2. Create PostGateway class #2

Get data from API

```
public Optional<PostResponse> getPostById(int id) {  
    String url = String.format("%s/posts/%d", postApiUrl, id);  
  
    try {  
        return Optional.ofNullable(  
            restTemplate.getForObject(url, PostResponse.class));  
    } catch (RestClientException e) {  
        return Optional.empty();  
    }  
}
```



# Testing with API



# Testing with API

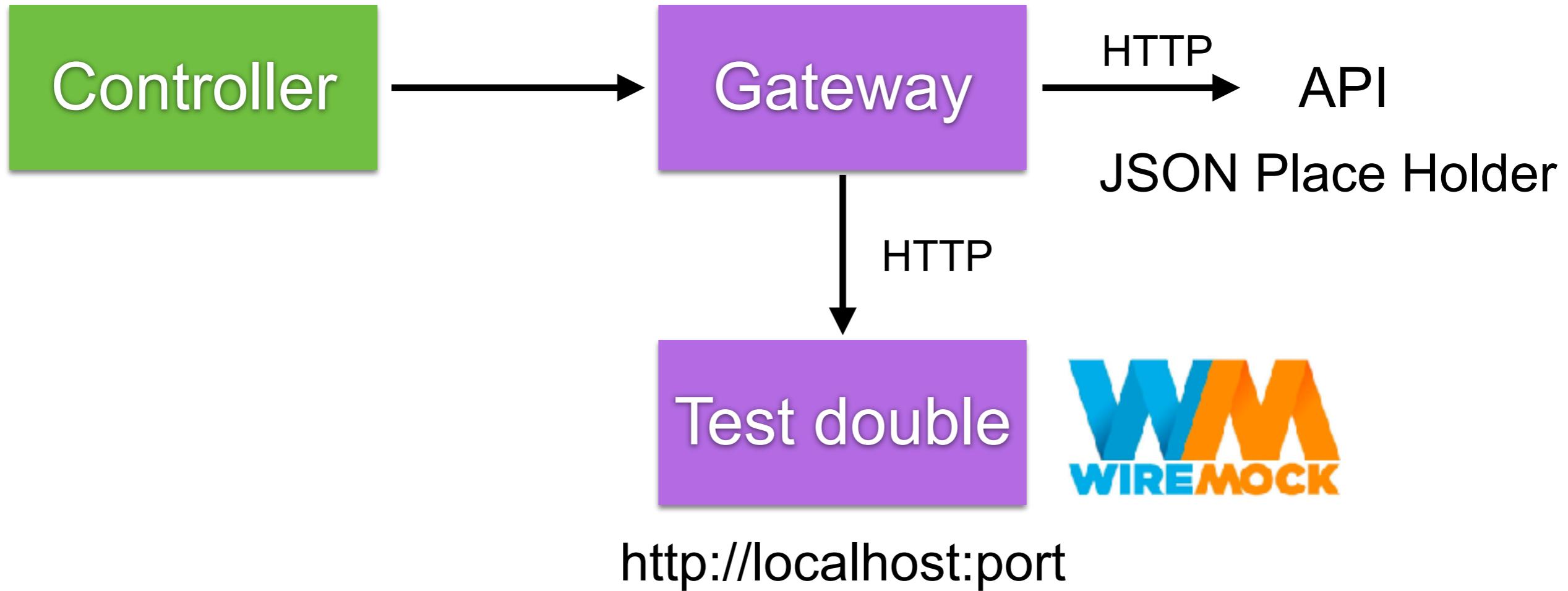
Unit testing with Mockito

**Component testing with WireMock**

Consumer testing with Pact



# Component testing with WireMock



**/src/test/resources/application.properties**

post.api.url=http://localhost:9999



# Component testing #1

## Working with WireMock

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@AutoConfigureWireMock(port = 9999)
public class PostGatewayComponentTest {

    @Autowired
    private PostGateway postGateway;
```

*“Default port = 9999”*



# Component testing #2

## Success case

```
@Test  
public void getPostById() throws IOException {  
    stubFor(get(urlPathEqualTo("/posts/1"))  
        .willReturn(aResponse()  
            .withBody(read("classpath:postApiResponse.json"))  
            .withHeader(CONTENT_TYPE, MediaType.APPLICATION_JSON_VALUE)  
            .withStatus(200));  
  
    Optional<PostResponse> postResponse = postGateway.getPostById(1);  
    assertEquals(11, postResponse.get().getId());  
    assertEquals(11, postResponse.get().getUserId());  
    assertEquals("Test Title", postResponse.get().getTitle());  
    assertEquals("Test Body", postResponse.get().getBody());  
}
```

Stub response



# Component testing #3

## Read data from resources folder

```
public static String read(String filePath) throws IOException {  
    File file = ResourceUtils.getFile(filePath);  
    return new String(Files.readAllBytes(file.toPath()));  
}
```



# Component testing #4

## File postApiResponse.json

```
{  
  "userId": 11,  
  "id": 11,  
  "title": "Test Title",  
  "body": "Test Body"  
}
```



# **Write controller testing ?**

**\$mvnw clean test**



# Separate tests with jUnit



<https://semaphoreci.com/community/tutorials/how-to-split-junit-tests-in-a-continuous-integration-environment>



# Separate test with jUnit



# Working with jUnit 4 Category

Create interface in each category

```
package com.lotto.lotto.category;  
  
public interface UnitTest {  
}
```

```
package com.lotto.lotto.category;  
  
public interface SlicingTest {  
}
```

```
package com.lotto.lotto.category;  
  
public interface IntegrationTest {  
}
```



# Add category in each test class

```
@RunWith(SpringRunner.class)
@SpringBootTest(webEnvironment = WebEnvironment.RANDOM_PORT)
@Category(IntegrationTest.class)
public class AccountControllerTest {

    @Autowired
    private TestRestTemplate testRestTemplate;

    @MockBean
    private UserService userService;
```



# Run test by category

```
$mvnw clean test  
-Dgroups="com.lotto.lotto.category.UnitTest"
```



# jUnit 5 Tagging and Filtering

```
import org.junit.jupiter.api.Tag;
import org.junit.jupiter.api.Test;

@Tag("fast")
@Tag("model")
class TaggingDemo {

    @Test
    @Tag("taxes")
    void testingTaxCalculation() {
    }

}
```

<https://junit.org/junit5/docs/current/user-guide/#writing-tests-tagging-and-filtering>



# Run test by tag

```
$mvnw clean test  
-Dgroups="fast, model"
```



# Aspect-Oriented Programming

## AOP



# Aspects ?

Reusable blocks of code that are injected into application at runtime

Powerful tools for adding behavior

Solve cross-cutting concerns in one place



# Common Applications of Aspects

Logging

Transaction management

Caching

Security

Execute time



# Why use Aspects ?

Reduce code duplication

DRY (Don't Repeat Yourself)

Maintain application logic



# Example of Execute time



# Working with transaction

<https://github.com/up1/course-springboot-2022/wiki/Transaction-Management>



# Important Quality Service



# Important Quality Service

Security  
Observability  
Configurability



# Secure Service

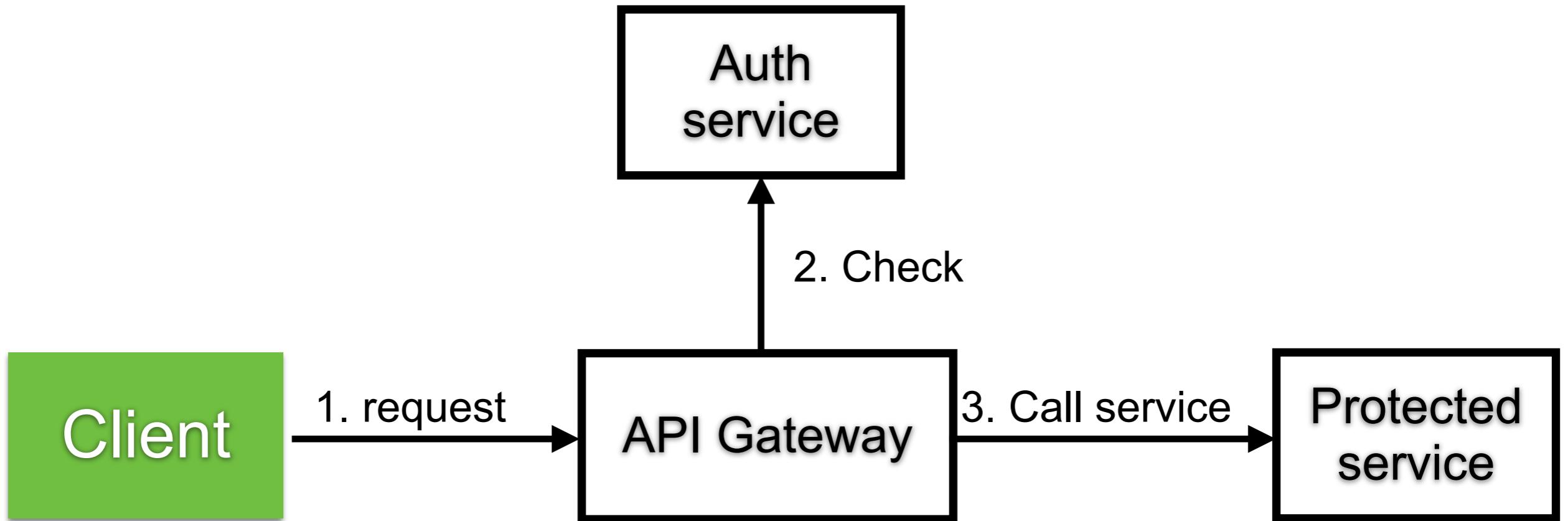


# Secure Service

API Gateway  
Within service



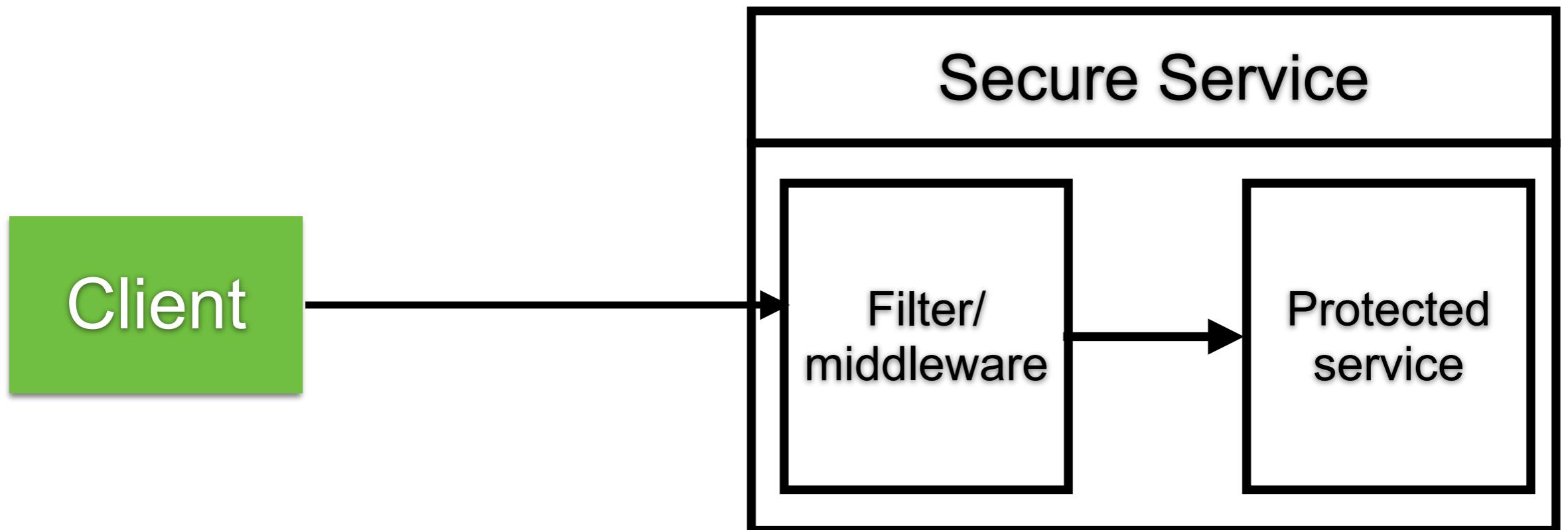
# API Gateway



Kong



# Within service



<https://github.com/up1/demo-spring-security>



# Secure Service with Spring

Working with Spring Security  
Support authentication and authorization

Spring Boot 3

Spring Security 6

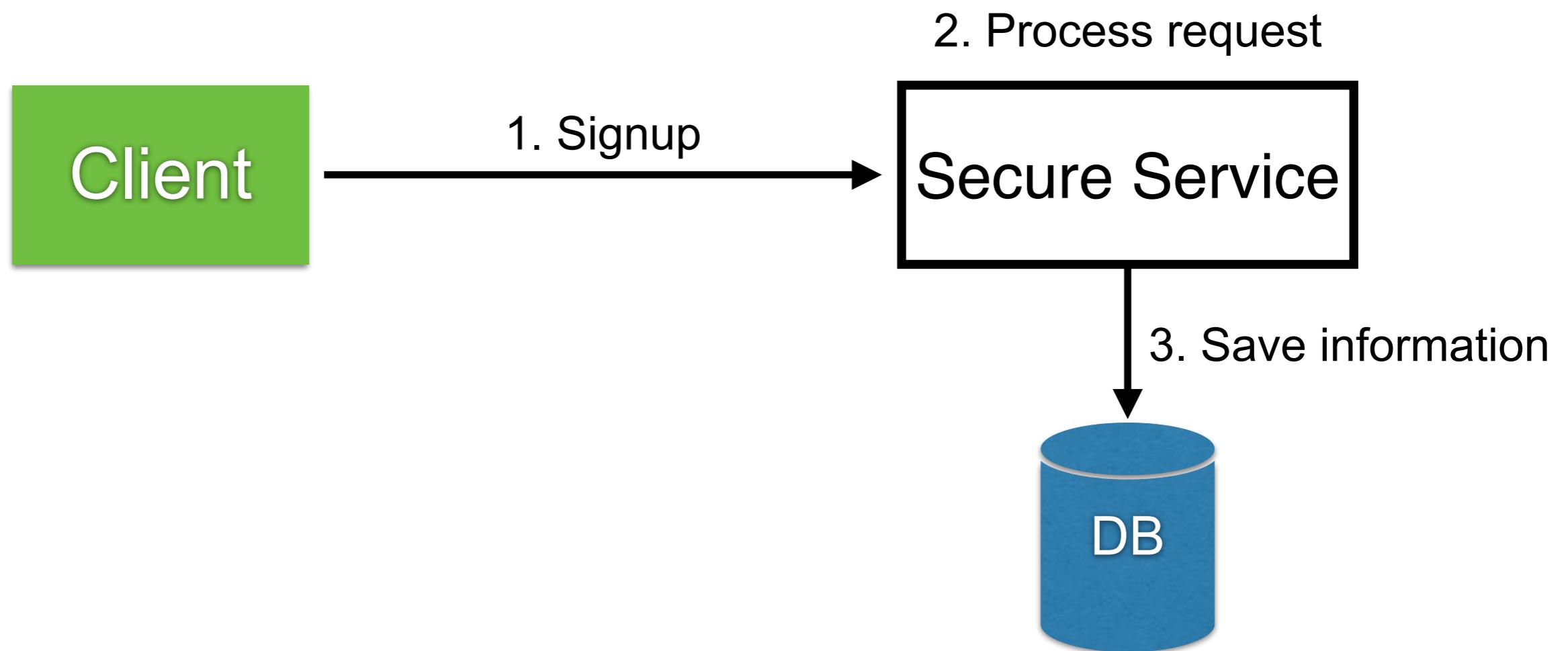
JDK 17

<https://spring.io/projects/spring-security>

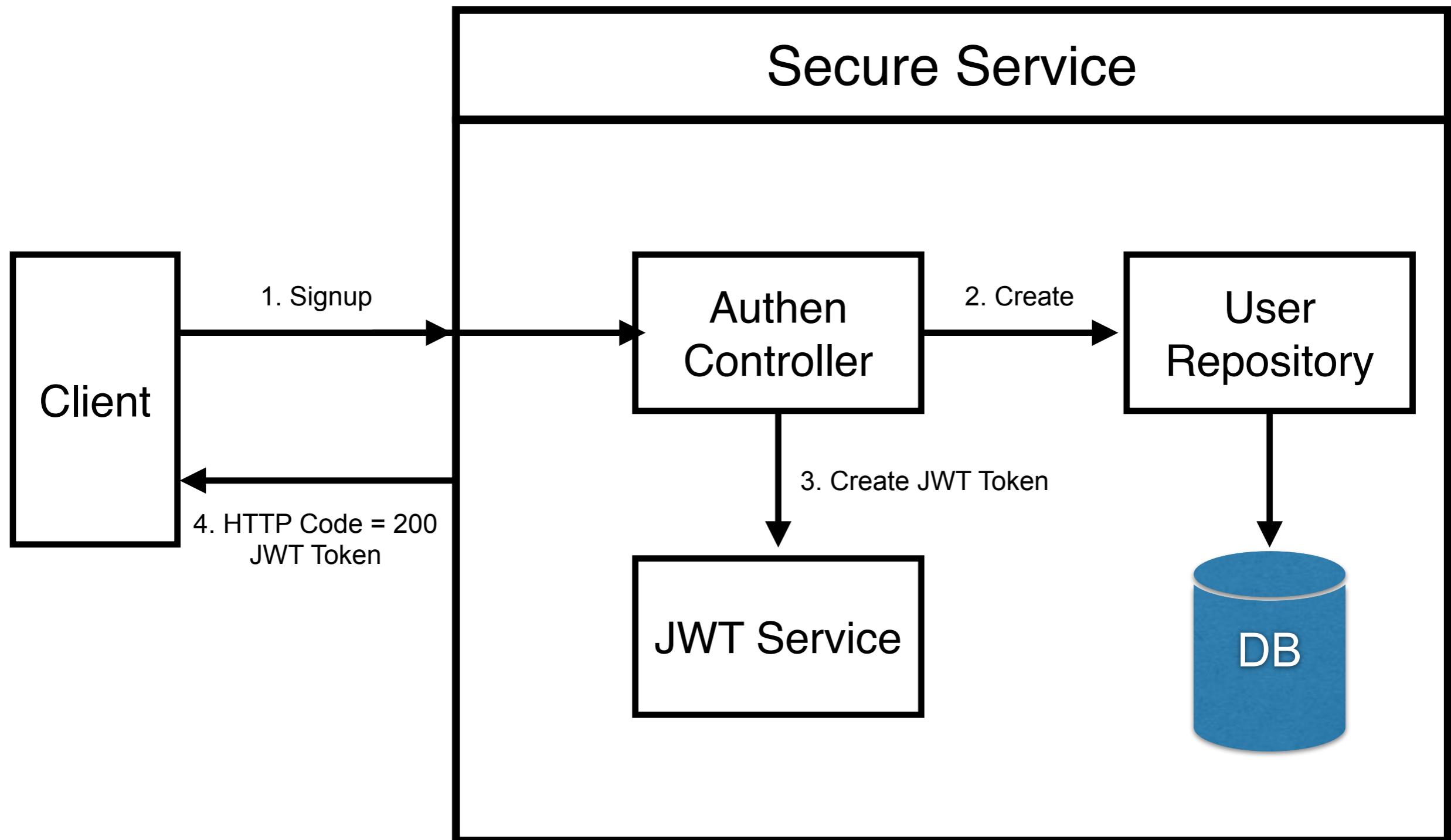


# Simple Flow of Secure Service

## Step 1 :: Signup process

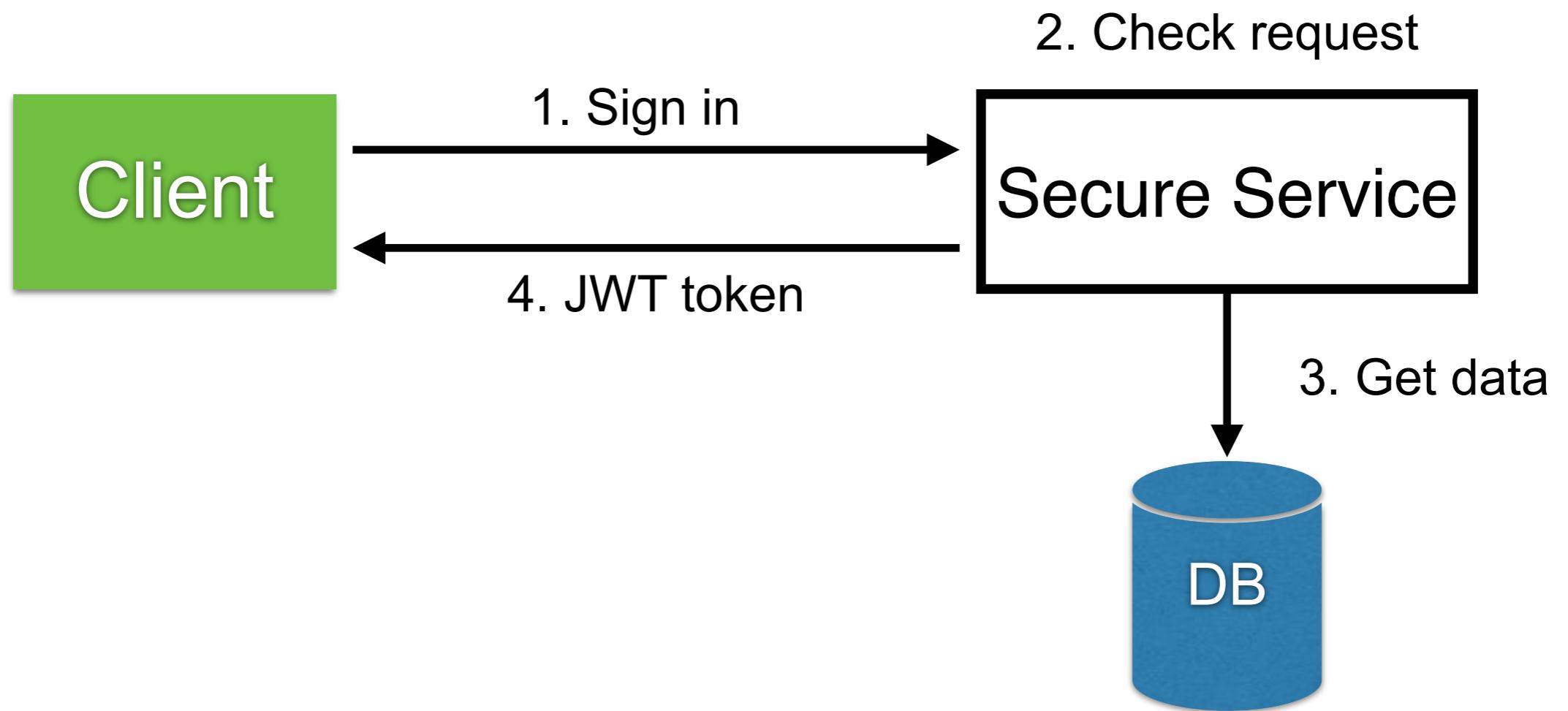


# Flow diagram of Signup

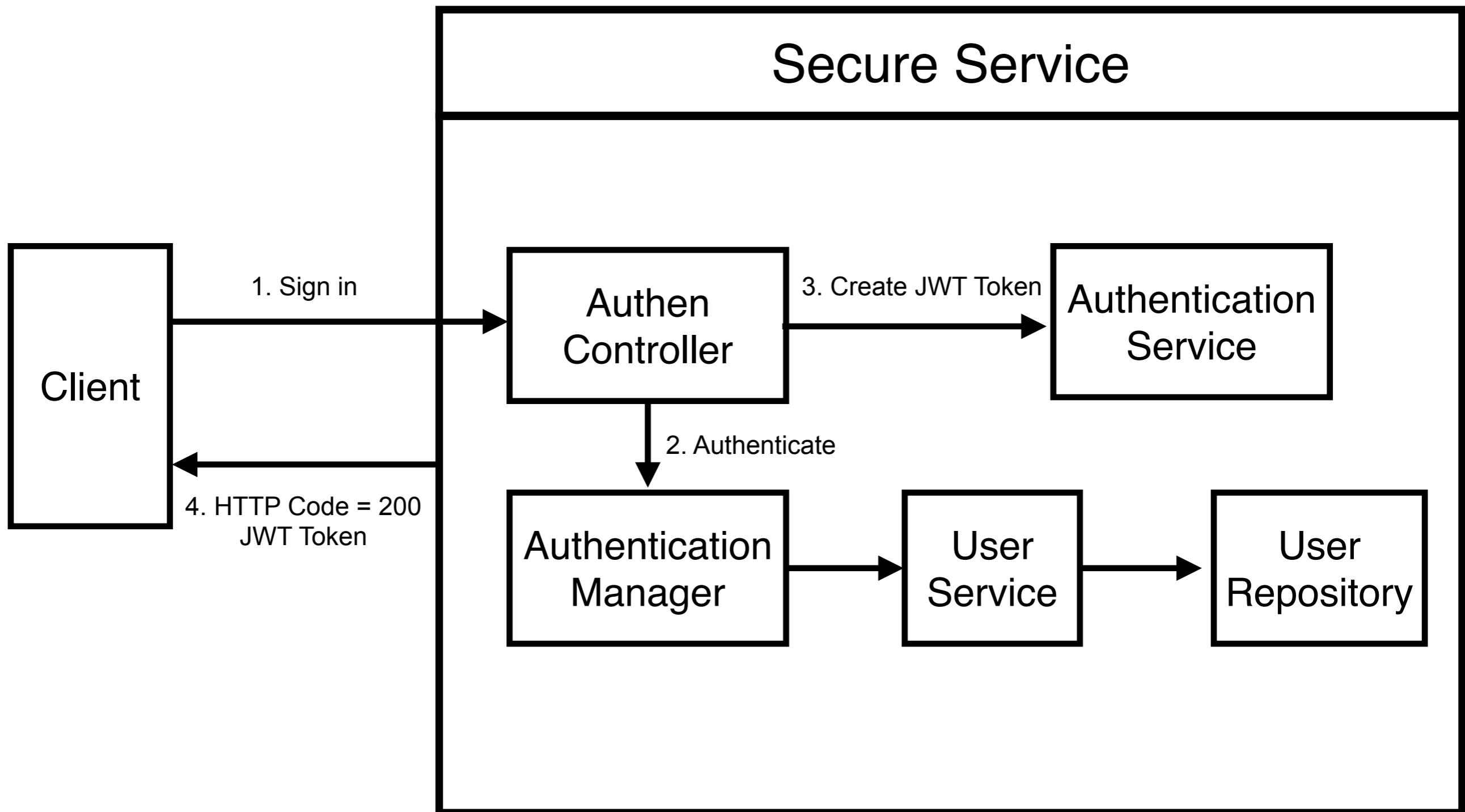


# Simple Flow of Secure Service

## Step 2 :: Sign in process

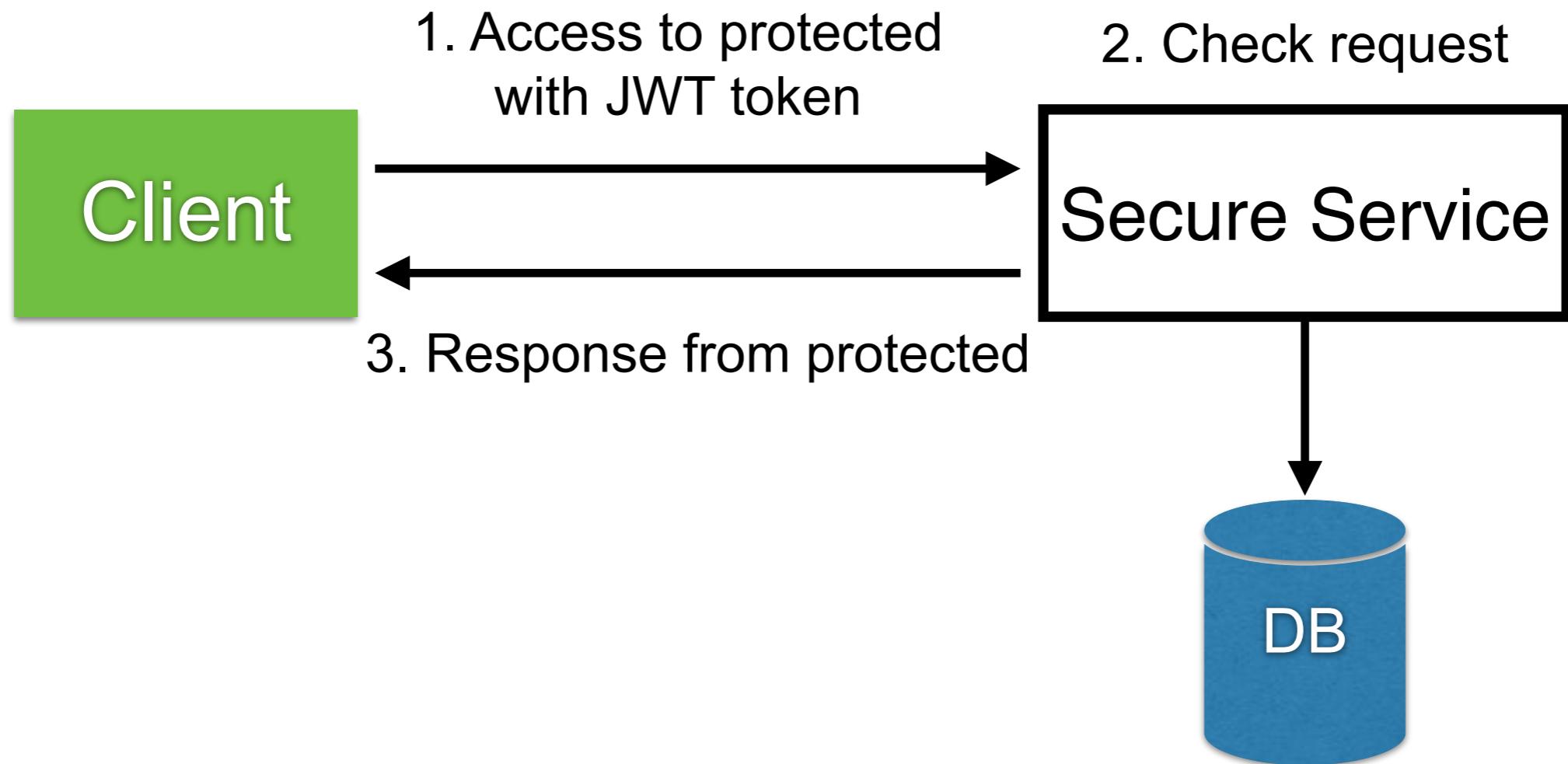


# Flow diagram of Sign in

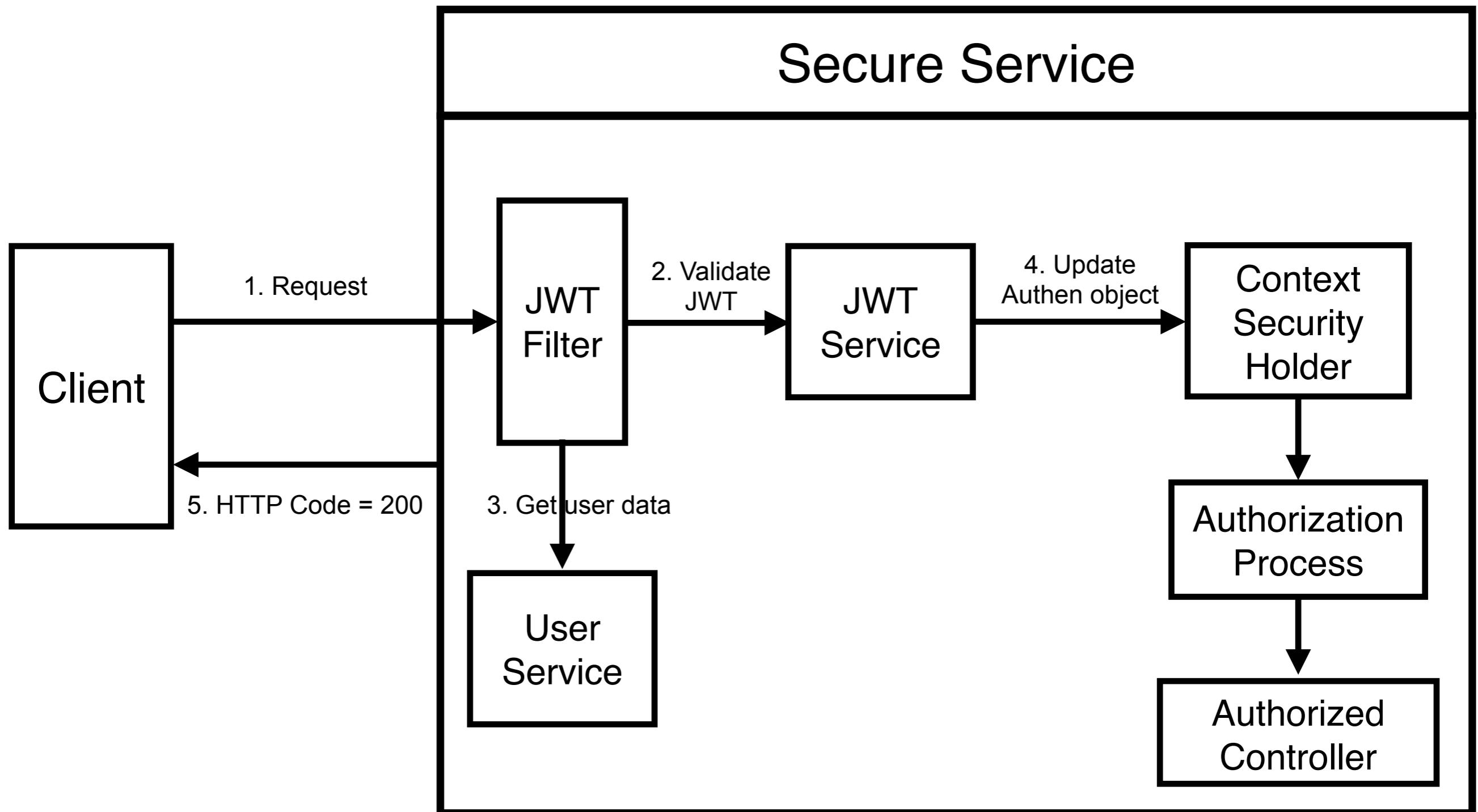


# Simple Flow of Secure Service

## Step 3 :: Access to protected services



# Flow diagram of call service



# JSON Web Tokens (JWT)

**Encoded** PASTE A TOKEN HERE

```
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJzdWIiOiIxMjM0NTY3ODkwIiwibmFtZSI6IkpvG4gRG9lIiwiaWF0IjoxNTE2MjM5MDIyfQ.Sf1KxwRJSMeKKF2QT4fwpNeJf36P0k6yJV_adQssw5c
```

**Decoded** EDIT THE PAYLOAD AND SECRET

**HEADER: ALGORITHM & TOKEN TYPE**

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

**PAYLOAD: DATA**

```
{  
  "sub": "1234567890",  
  "name": "John Doe",  
  "iat": 1516239022  
}
```

**VERIFY SIGNATURE**

```
HMACSHA256(  
  base64UrlEncode(header) + "." +  
  base64UrlEncode(payload),  
  your-256-bit-secret  
)  secret: base64 encoded
```

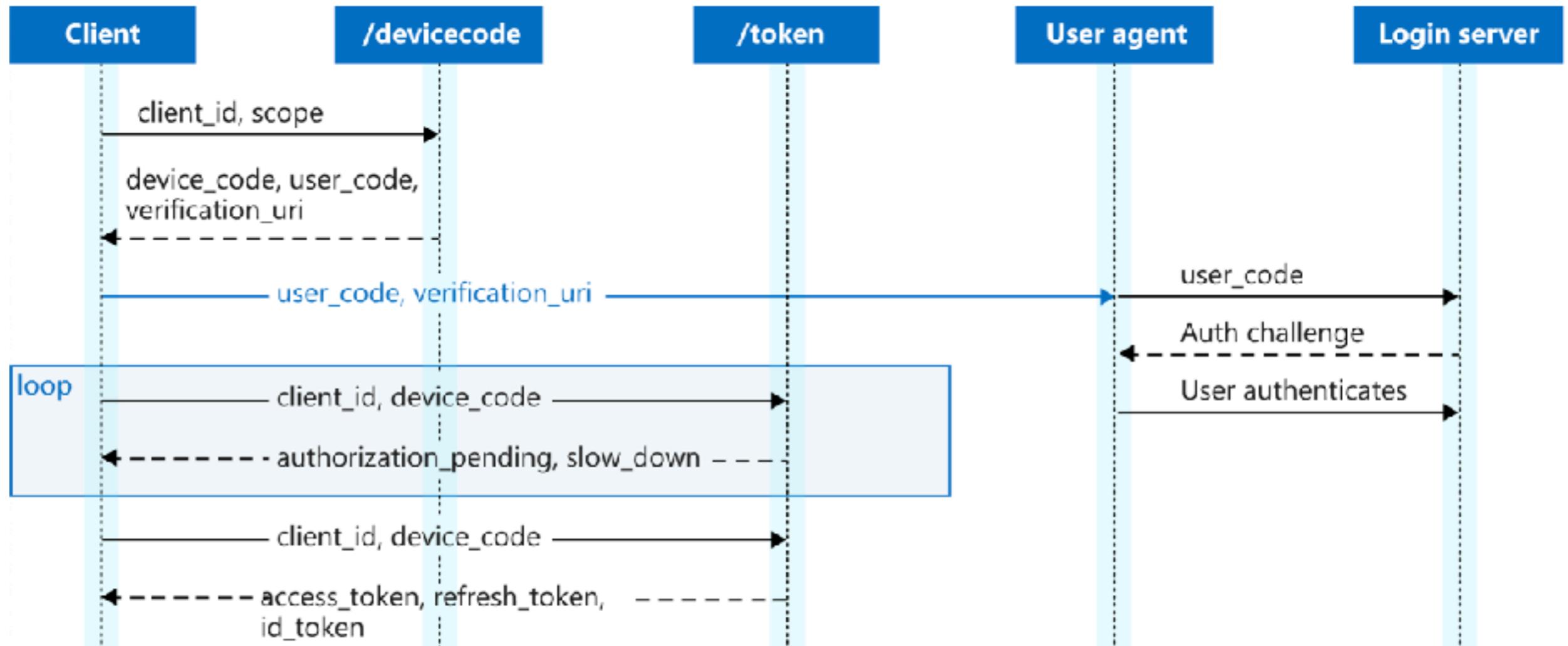
 Signature Verified

[SHARE JWT](#)

<https://jwt.io/>



# Example



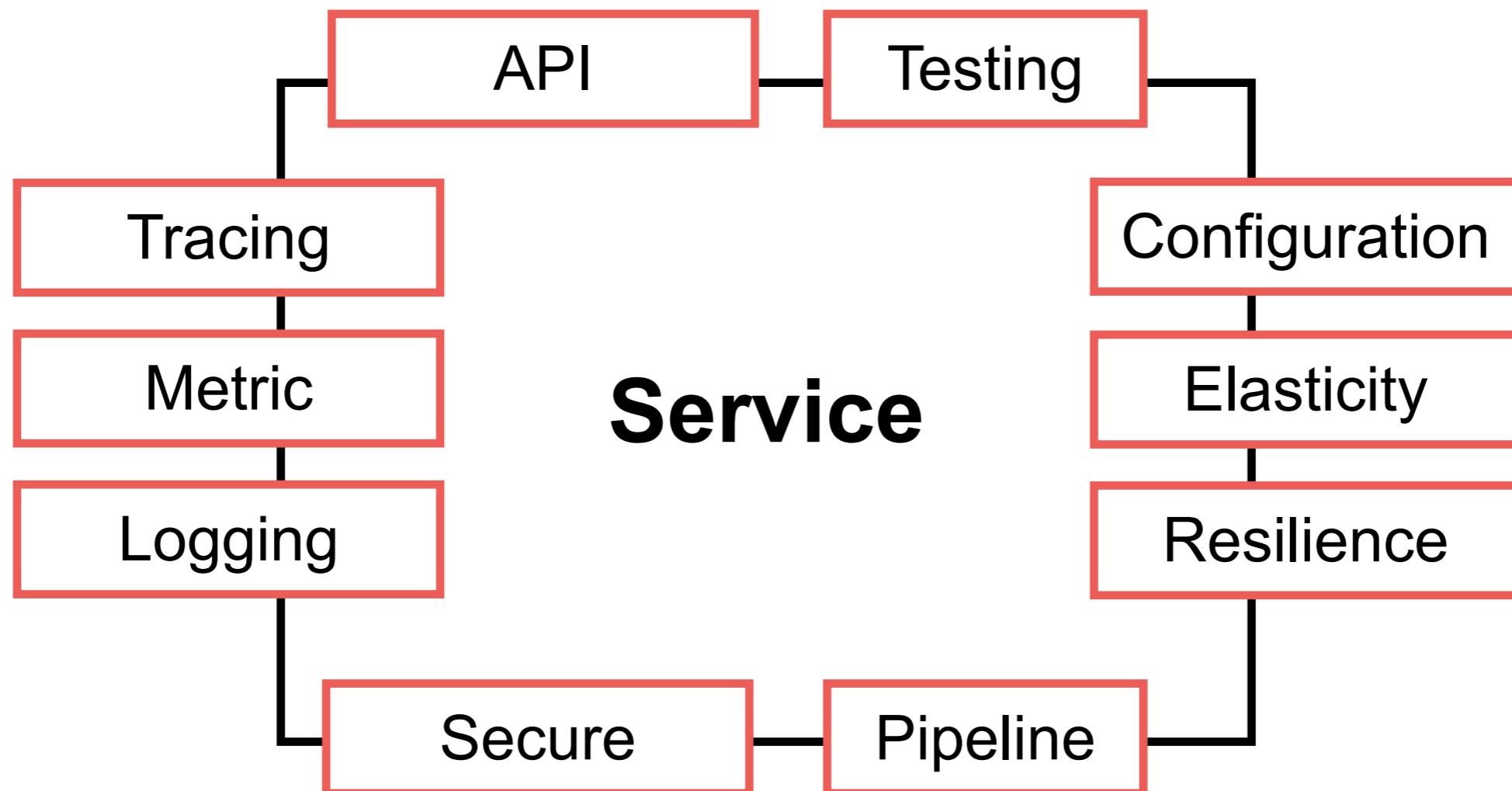
<https://learn.microsoft.com/en-us/azure/active-directory/develop/v2-oauth2-device-code>



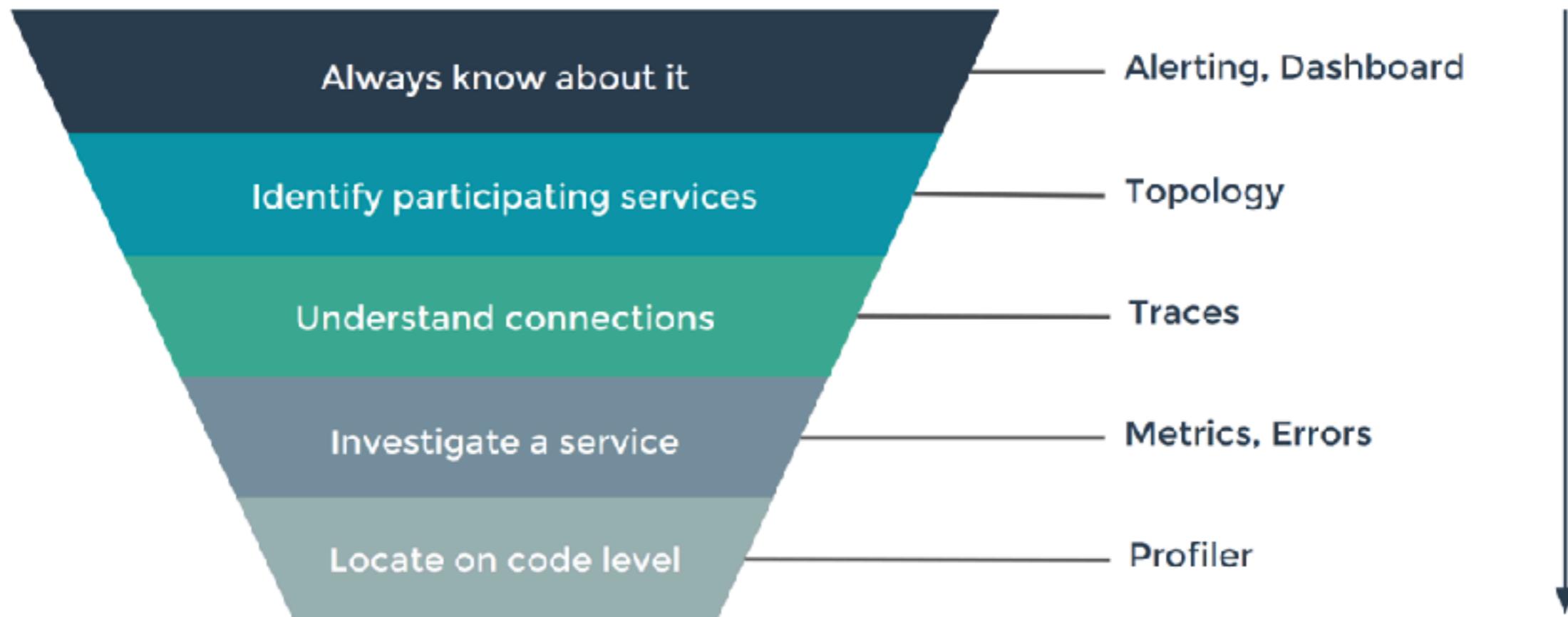
# Observable Service



# Properties of Service



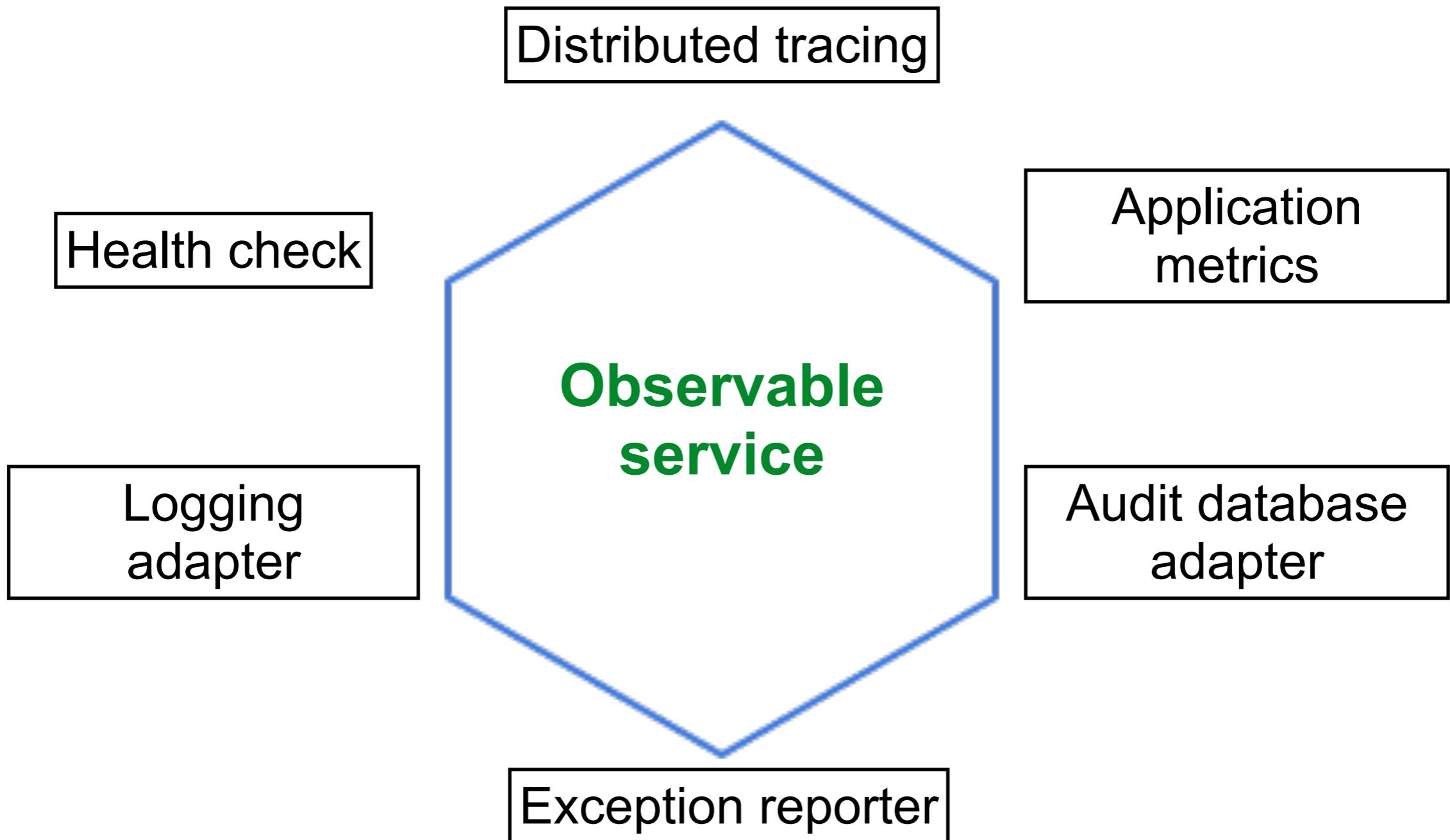
# How to find an issue ?



# Observability vs Monitoring

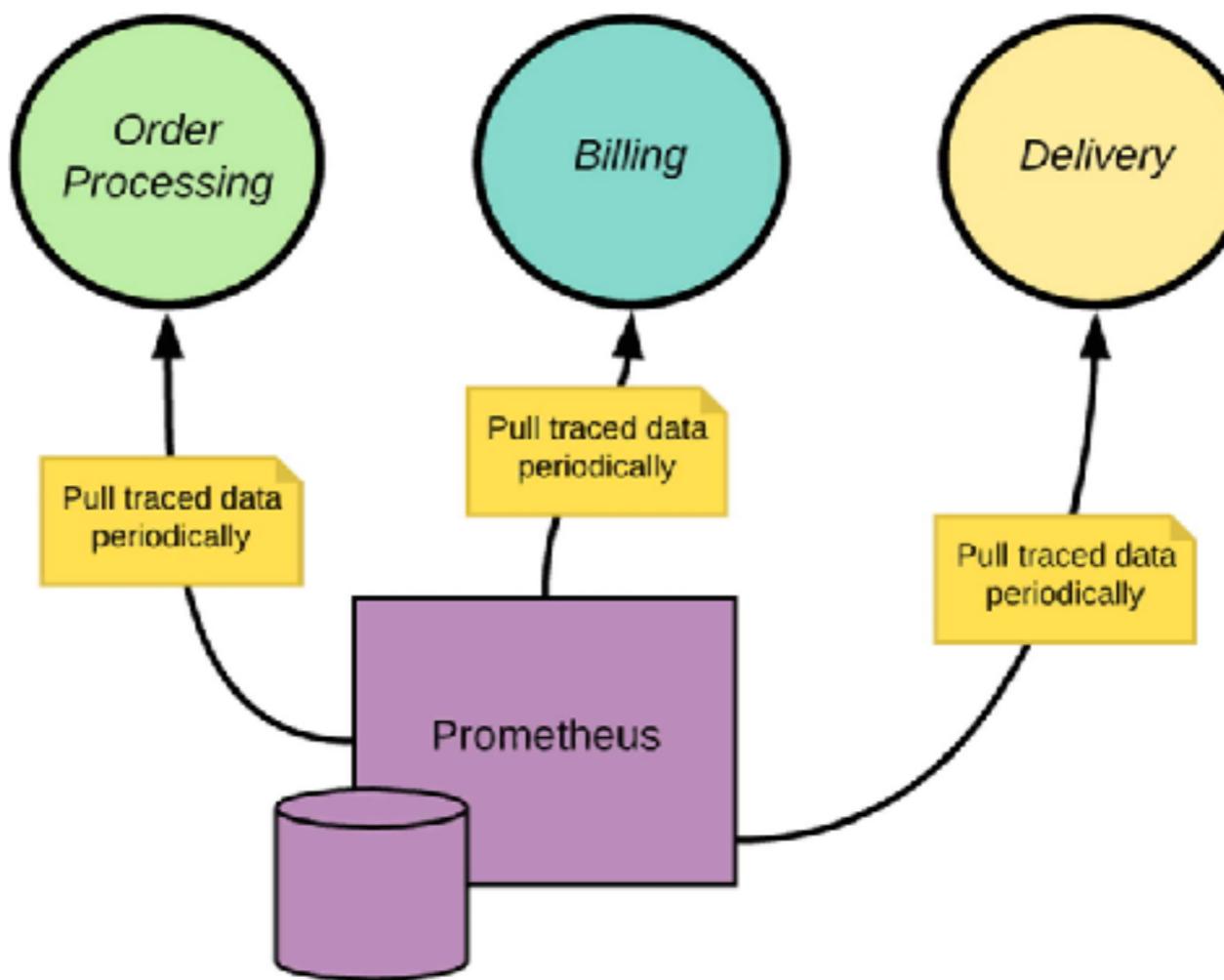


# Observable services



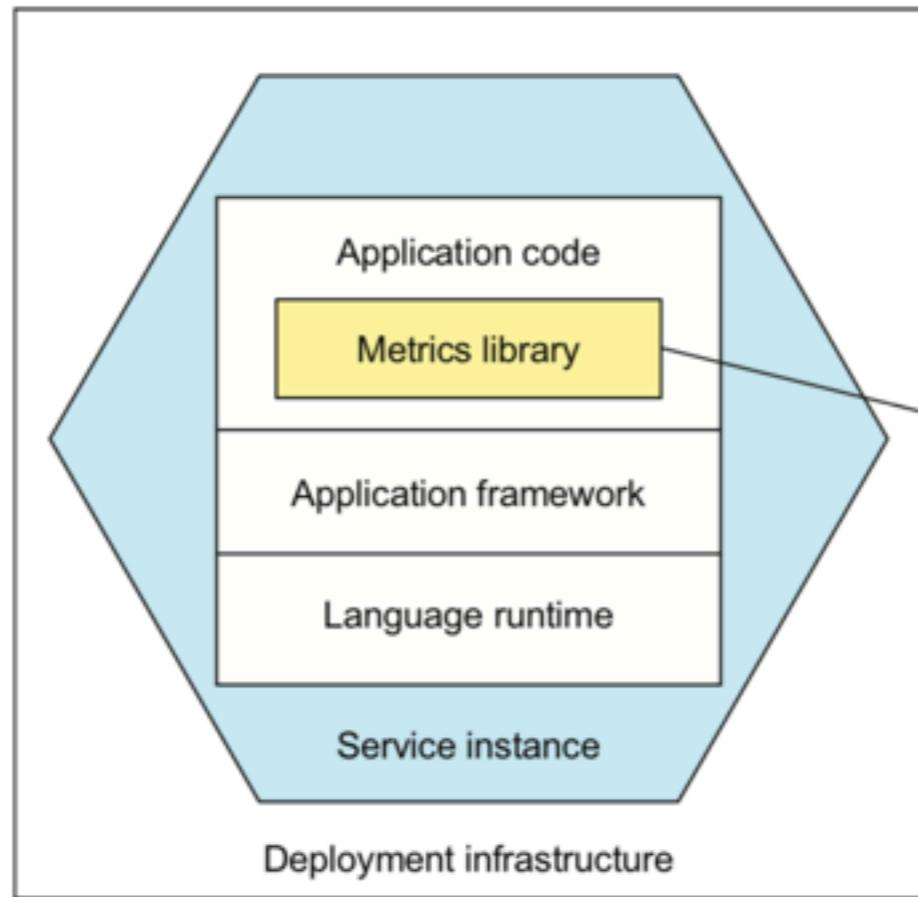
# Application metrics

Services maintain metrics and expose to metric server (counters, gauges)

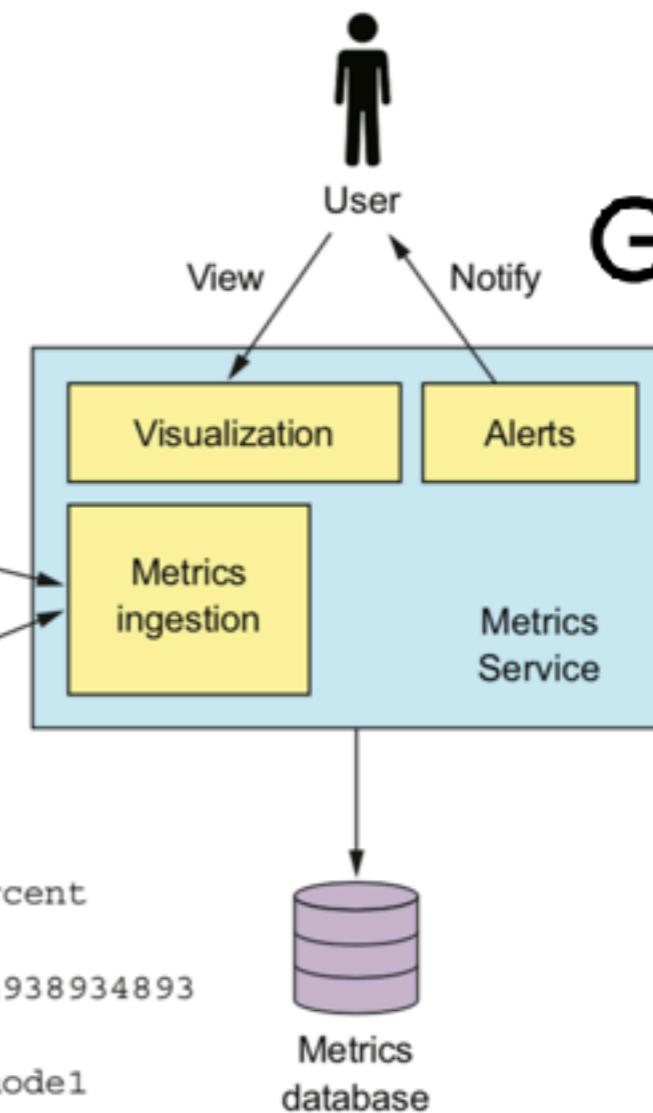


# Application metrics tools

## Spring Boot Actuator



Metrics sample:  
name=cpu\_percent  
value=68  
timestamp=34938934893  
dimensions:  
machine=node1  
...



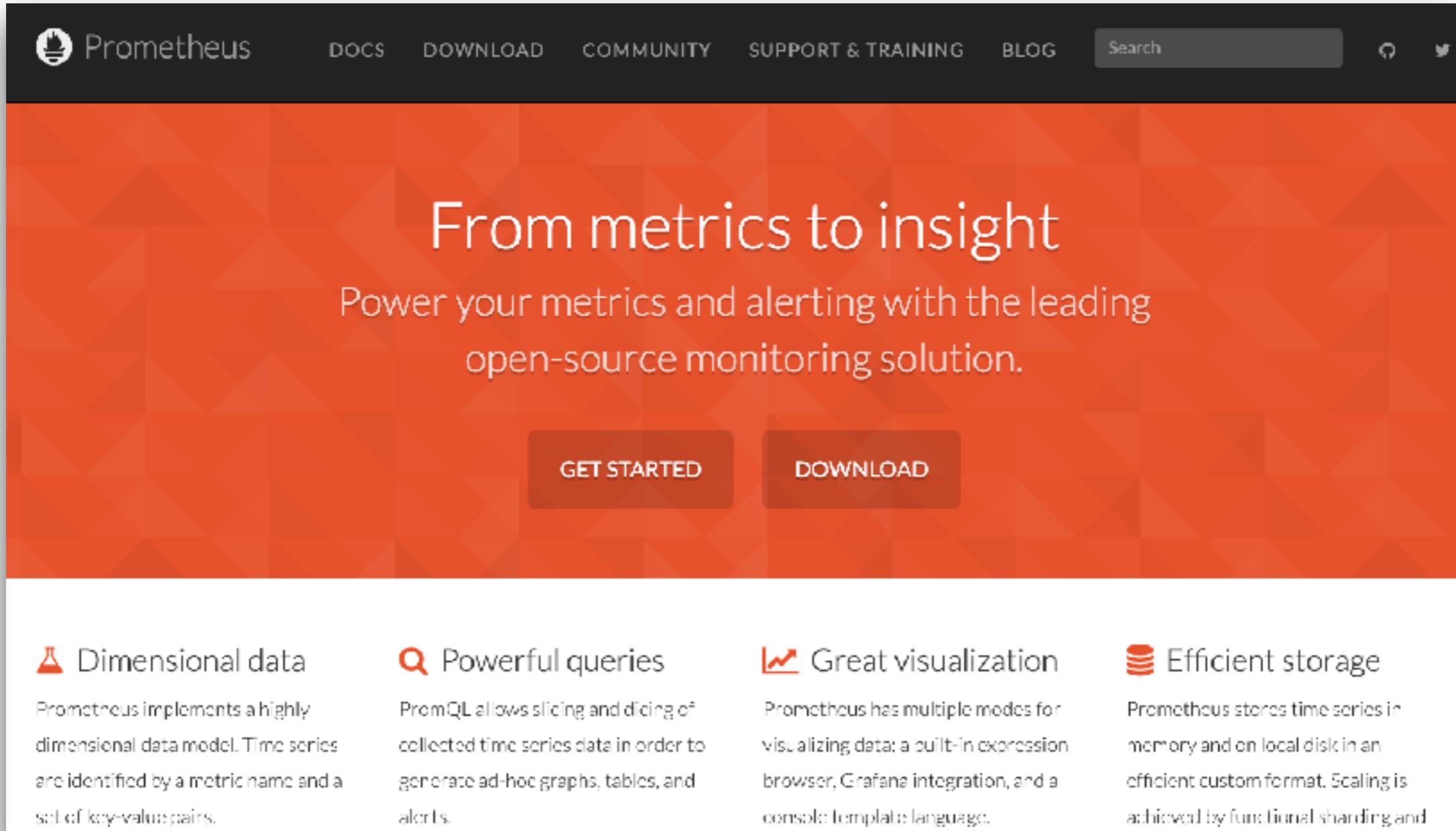
**Grafana**



**Prometheus**



# Prometheus



The screenshot shows the official Prometheus website. At the top, there's a dark navigation bar with the Prometheus logo, a search bar, and social media links for GitHub and Twitter. The main heading "From metrics to insight" is prominently displayed in white on an orange background. Below it, a sub-headline reads "Power your metrics and alerting with the leading open-source monitoring solution." Two large buttons, "GET STARTED" and "DOWNLOAD", are visible. The bottom section features four cards with icons and descriptions: "Dimensional data" (Prometheus implements a highly dimensional data model), "Powerful queries" (PromQL allows slicing and dicing of collected time series data), "Great visualization" (Prometheus has multiple modes for visualizing data), and "Efficient storage" (Prometheus stores time series in memory and on local disk in an efficient custom format). Each card also includes a short explanatory paragraph.

Dimensional data

Prometheus implements a highly dimensional data model. Time series are identified by a metric name and a set of key-value pairs.

Powerful queries

PromQL allows slicing and dicing of collected time series data in order to generate ad-hoc graphs, tables, and alerts.

Great visualization

Prometheus has multiple modes for visualizing data: a built-in expression browser, Grafana integration, and a console template language.

Efficient storage

Prometheus stores time series in memory and on local disk in an efficient custom format. Scaling is achieved by functional sharding and

<https://prometheus.io/>



# Grafana

Grafana Labs Products Open source Solutions Learn Company Downloads Contact us Sign in

Compose and scale observability with one or all pieces of the stack

Your observability wherever you need it

Cloud

Self-managed

Grafana

API

Plugins

Dashboards

Alerts

Usage insights

Reports

Governance

Metrics

Logs

Traces

Applications

Infrastructure

Cloud

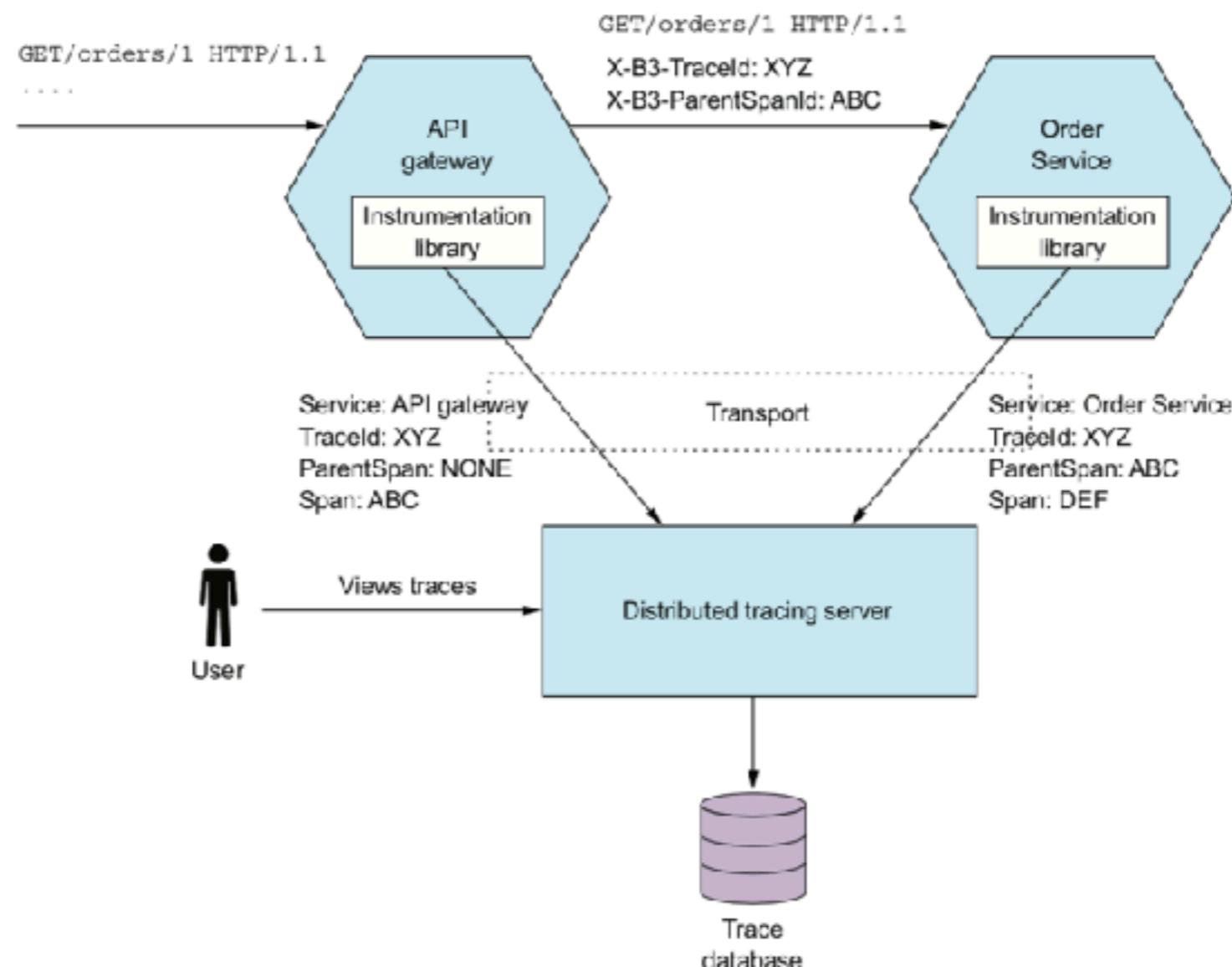
Self-managed

<https://grafana.com/>



# Distributed tracing

Assign each external request a unique ID and trace requests as flow between services



# Distributed tracing tools

Format standard with OpenTelemetry  
Zipkin, Jaeger, Tempo, AWS X-Ray, Elastic APM



JAEGER



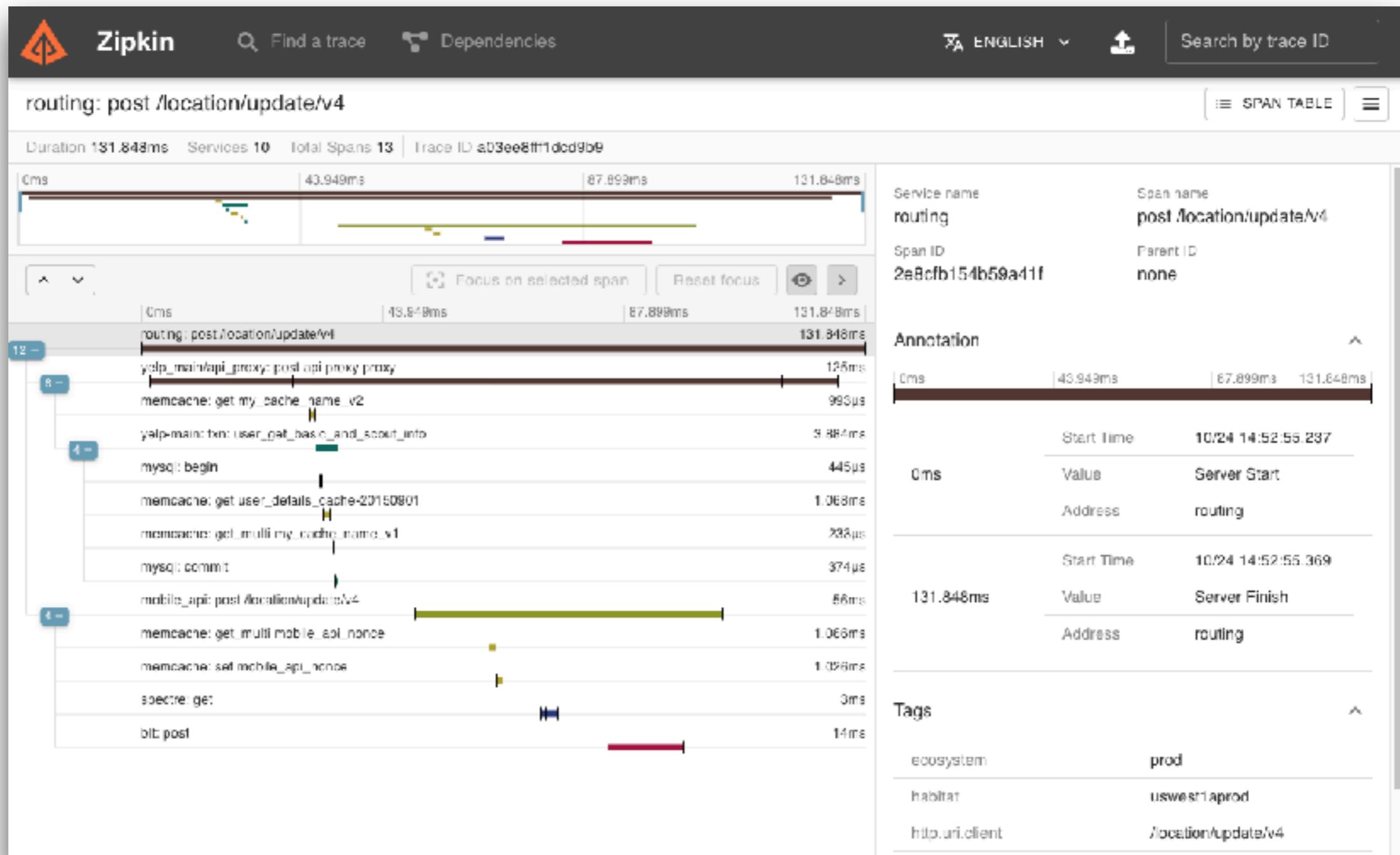
Grafana Tempo



OpenTelemetry



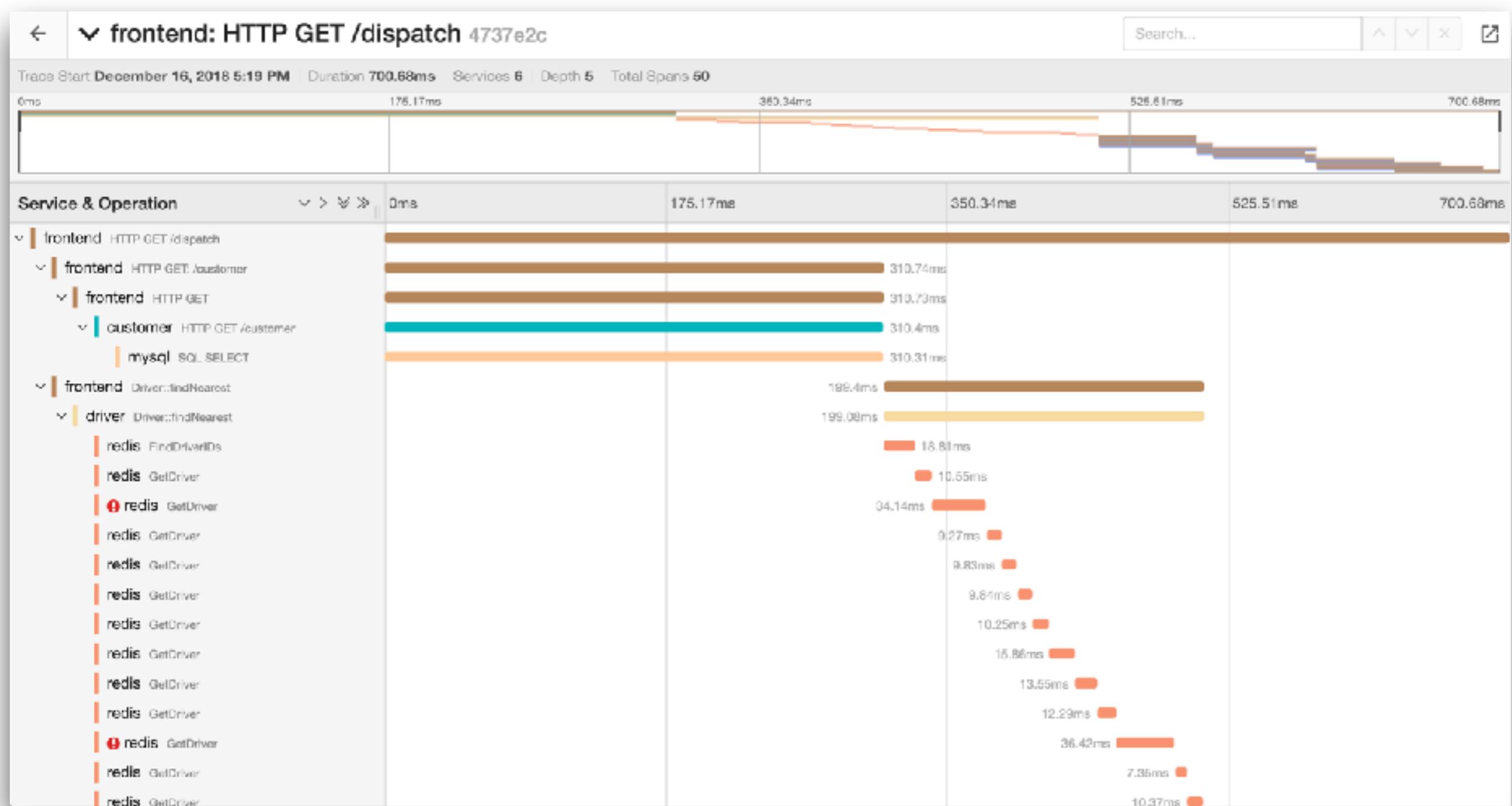
# Zipkin



<https://zipkin.io/>



# Jaeger

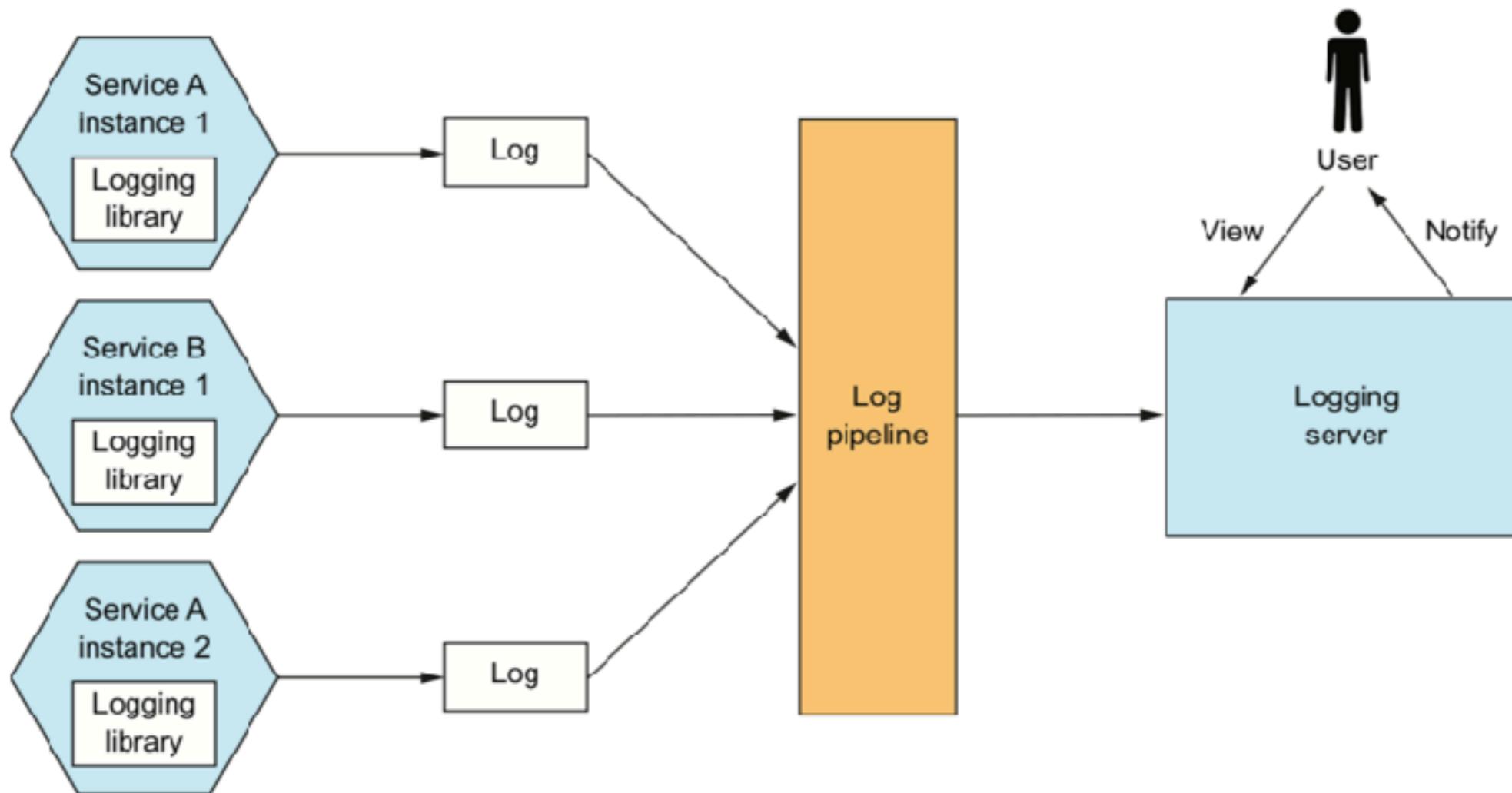


<https://www.jaegertracing.io>



# Centralized logging

Log service activity and write logs into a centralized logging server. (searching, alerting)



# Effective Logging

Define event to log

Use structured logging

Exclude sensitive information

Log at the correct level

Be specific in your message

Don't log large message

Make sure you keep trace Id in the log

[https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Logging\\_Cheat\\_Sheet.md](https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/Logging_Cheat_Sheet.md)



# Consistent Structure across all logs

Property	Description	Example
<b>Timestamp</b>	Date and time of the log	2023-07-01
<b>Log level</b>	DEBUG, INFO, ERROR	
<b>Trace Id or Correlation Id</b>	Unique identifier that refer to other logs from all services	
<b>Event/Action Name</b>	Identify to event or action of log	Authentication fail
<b>Service ID/ Name</b>	Identify to service	
<b>Request path</b>	Path for the request	/api/products

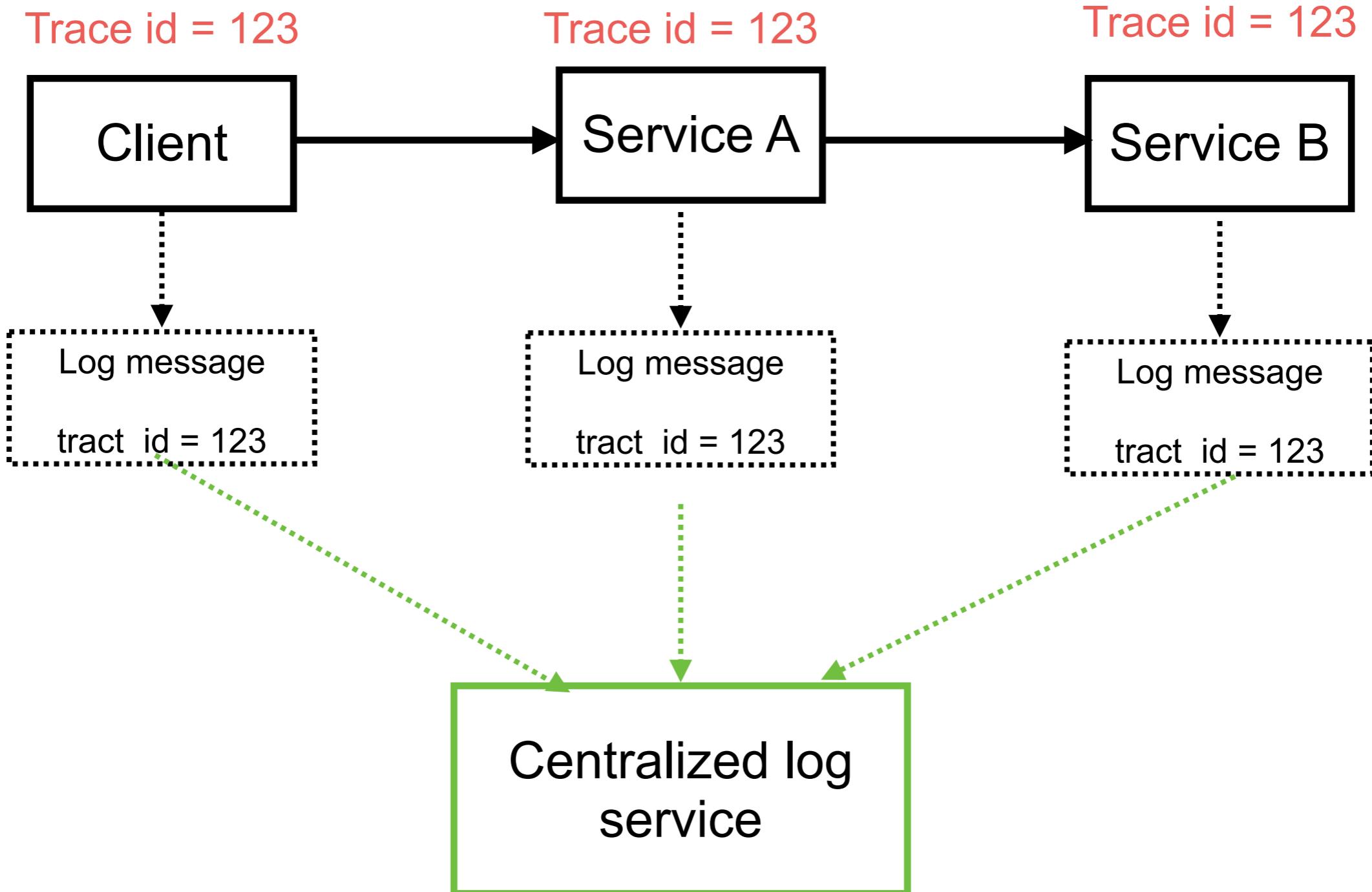


# Don't keep !!

```
com.framework.FrameworkException: Error in web request
  at com.framework.ApplicationStarter.lambda$start$0(ApplicationStarter.java:15)
  at spark.RouteImpl$1.handle(RouteImpl.java:72)
  at spark.http.matching.Routes.execute(Routes.java:61)
  at spark.http.matching.MatcherFilter.doFilter(MatcherFilter.java:134)
  at spark.embeddedserver.jetty.JettyHandler.doHandle(JettyHandler.java:50)
  at org.eclipse.jetty.server.session.SessionHandler.doScope(SessionHandler.java:1568)
  at org.eclipse.jetty.server.handler.ScopedHandler.handle(ScopedHandler.java:144)
  at org.eclipse.jetty.server.handler.HandlerWrapper.handle(HandlerWrapper.java:132)
  at org.eclipse.jetty.server.Server.handle(Server.java:503)
  at org.eclipse.jetty.server.HttpChannel.handle(HttpChannel.java:364)
  at org.eclipse.jetty.server.HttpConnection.onFillable(HttpConnection.java:260)
  at org.eclipse.jetty.io.AbstractConnection$ReadCallback.succeeded(AbstractConnection.java:305)
  at org.eclipse.jetty.io.FillInterest.fillable(FillInterest.java:103)
  at org.eclipse.jetty.io.ChannelEndPoint$2.run(ChannelEndPoint.java:118)
  at org.eclipse.jetty.util.thread.QueuedThreadPool.runJob(QueuedThreadPool.java:765)
  at org.eclipse.jetty.util.thread.QueuedThreadPool$2.run(QueuedThreadPool.java:683)
  at java.base/java.lang.Thread.run(Thread.java:834)
Caused by: com.project.module.MyProjectFooBarException: The number of FooBars cannot be zero
  at com.project.module.MyProject.anotherMethod(MyProject.java:20)
  at com.project.module.MyProject.someMethod(MyProject.java:12)
  at com.framework.ApplicationStarter.lambda$start$0(ApplicationStarter.java:13)
  ... 16 more
Caused by: java.lang.ArithmaticException: The denominator must not be zero
  at org.apache.commons.lang3.math.Fraction.getFraction(Fraction.java:143)
  at com.project.module.MyProject.anotherMethod(MyProject.java:18)
  ... 18 more
```



# Centralized logging



# Observable Service Workshop

Metric

Tracing

Logging

<https://github.com/up1/demo-spring-observability>



# Distributed Tracing



<https://zipkin.io/>



# Create project with Tracing

org.springframework.boot:spring-boot-starter-actuator  
io.micrometer:micrometer-tracing-bridge-otel  
io.opentelemetry:opentelemetry-exporter-zipkin

Actuator

Micrometer  
Tracing Otel

Opentelemetry  
zipkin exporter

<https://docs.spring.io/spring-boot/docs/3.2.0/reference/html//actuator.html#actuator.micrometer-tracing>



# Spring Boot 3.2.0

## Virtual Threads

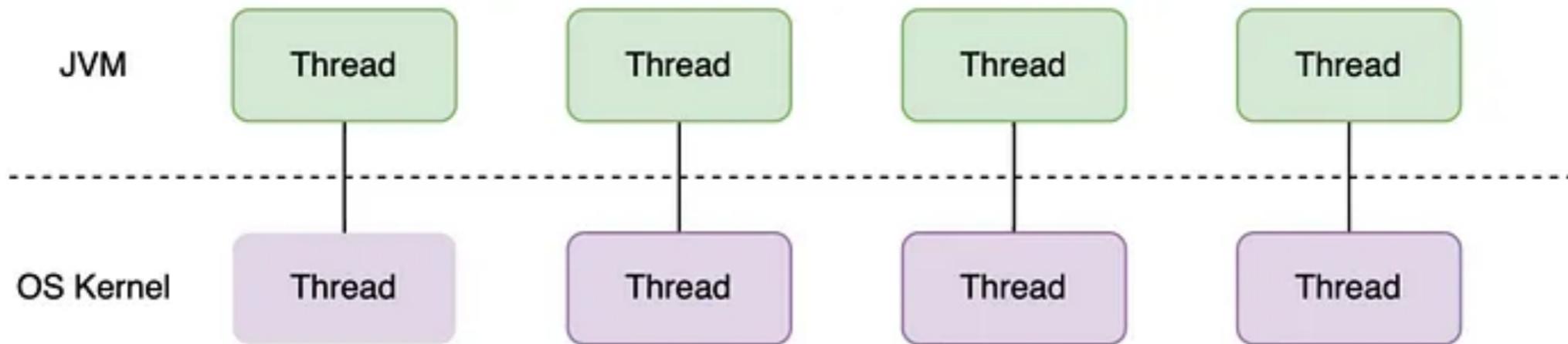
<https://openjdk.org/jeps/425>

<https://spring.io/blog/2023/09/09/all-together-now-spring-boot-3-2-graalvm-native-images-java-21-and-virtual>



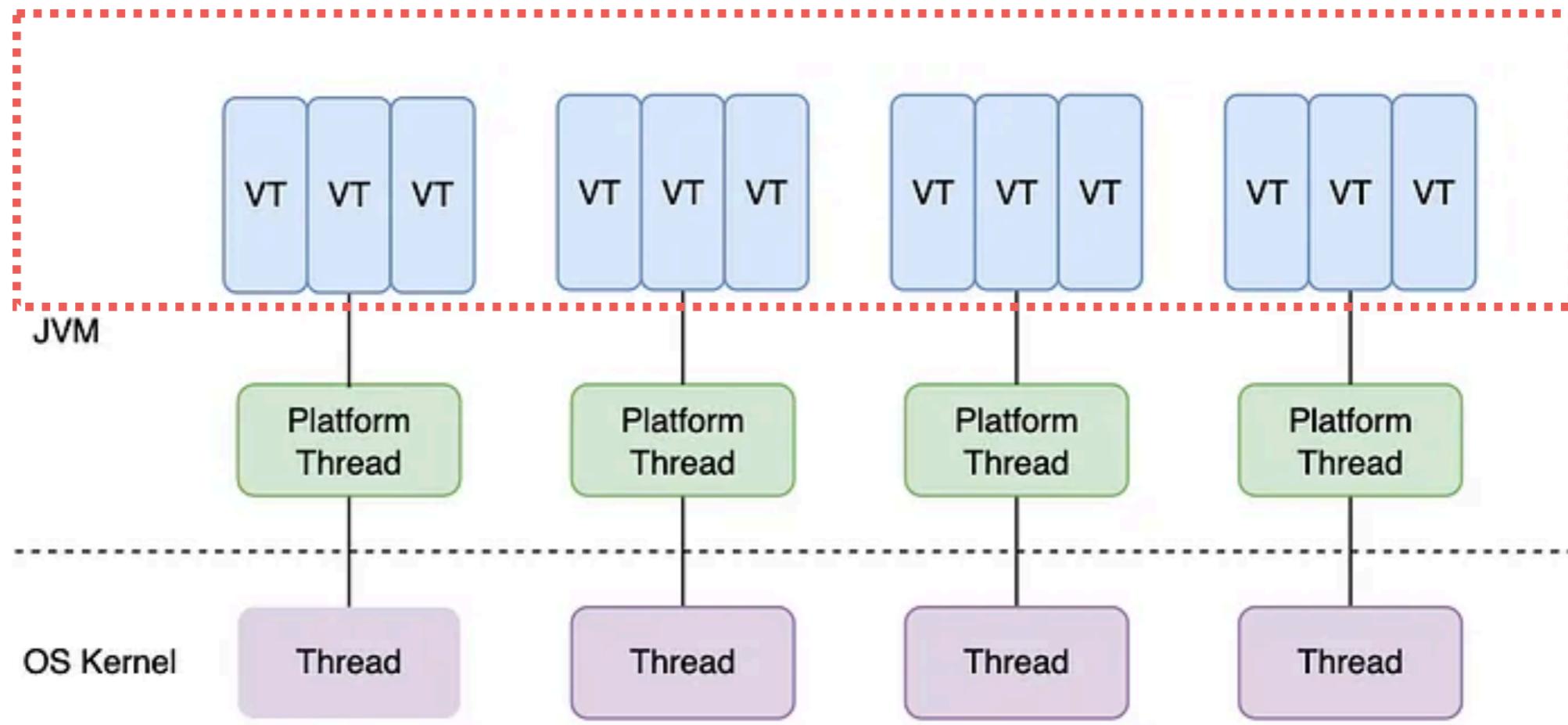
# Java Threads

## Thread pools



# Support Virtual Threads

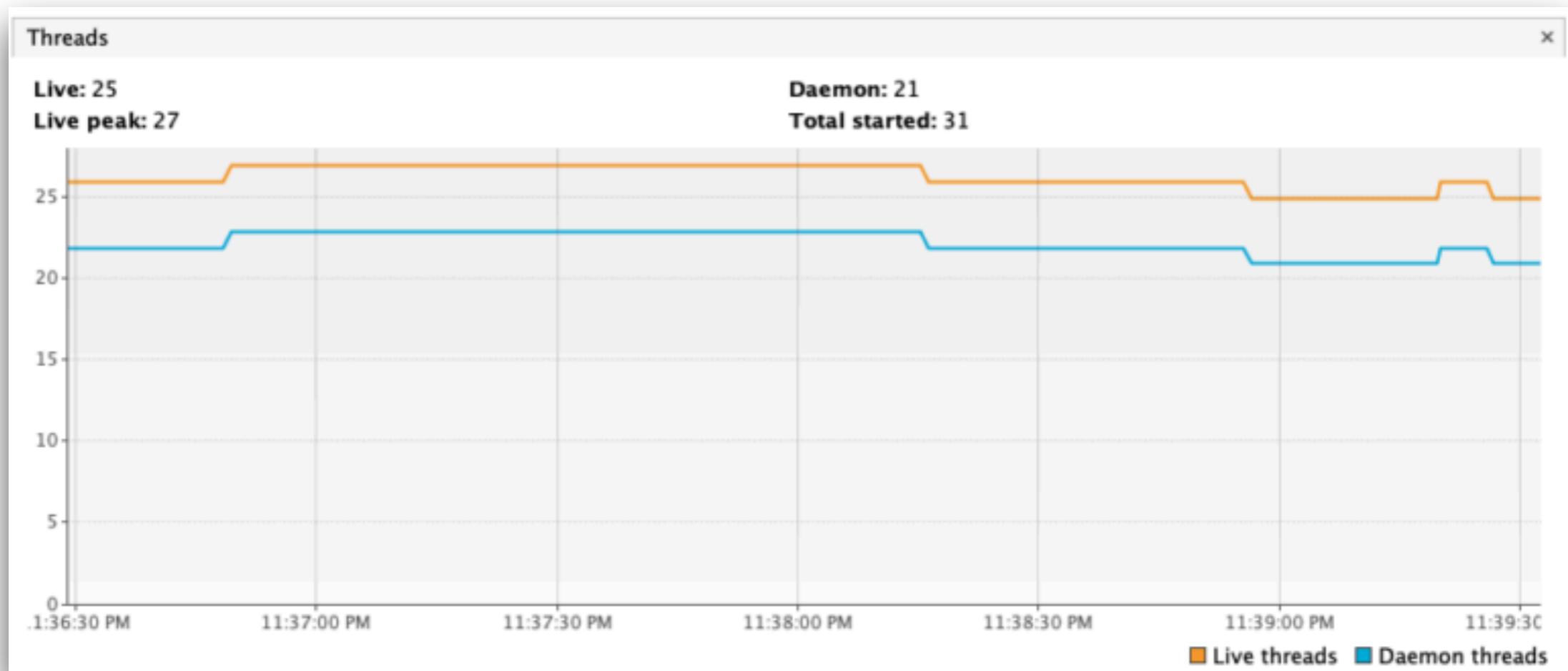
## Thread pools



# Enable Virtual Thread in Spring Boot

## application.properties

```
spring.threads.virtual.enabled=true
```



# **Spring Boot 3.2.0**

# **GraalVM native image support**

<https://www.graalvm.org/>

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





**Project**

Gradle - Groovy    Gradle - Kotlin    Maven

**Language**

Java    Kotlin    Groovy

**Spring Boot**

3.2.1 (SNAPSHOT)    3.2.0    3.1.7 (SNAPSHOT)    3.1.6

**Project Metadata**

Group: com.example

Artifact: demo

Name: demo

Description: Demo project for Spring Boot

Package name: com.example.demo

Packaging:  Jar    War

Java:  21    17

## Dependencies

[ADD DEPENDENCIES...](#) ⌘ + ⌘

### Spring Web WEB

Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

### GraalVM Native Support DEVELOPER TOOLS

Support for compiling Spring applications to native executables using the GraalVM native-image compiler.

<https://start.spring.io/>



# **Spring Boot 3.2.0**

## **Docker compose**

<https://docs.spring.io/spring-boot/docs/current/reference/htmlsingle/#features.docker-compose>





**Project**

Gradle - Groovy  Gradle - Kotlin  
 Maven

**Language**

Java  Kotlin  Groovy

**Spring Boot**

3.2.1 (SNAPSHOT)  3.2.0  3.1.7 (SNAPSHOT)  3.1.6

**Project Metadata**

Group: com.example

Artifact: demo

Name: demo

Description: Demo project for Spring Boot

Package name: com.example.demo

Packaging:  Jar  War

Java:  21  17

## Dependencies

[ADD DEPENDENCIES... ⌘ + B](#)

## Docker Compose Support DEVELOPER TOOLS

Provides docker compose support for enhanced development experience.

## Testcontainers TESTING

Provide lightweight, throwaway instances of common databases, Selenium web browsers, or anything else that can run in a Docker container.

<https://github.com/up1/demo-spring-docker-compose>



# **Spring Boot 3.3.0**

# **SBOM (Software Bill of Materials)**

<https://spring.io/blog/2024/05/24/sbom-support-in-spring-boot-3-3>



# Enable SBOM in project

**Dependencies**

**ADD DEPENDENCIES... ⌘ + B**

**CycloneDX SBOM support** OPS  
Creates a Software Bill of Materials in CycloneDX format.

**Spring Boot Actuator** OPS  
Supports built in (or custom) endpoints that let you monitor and manage your application - such as application health, metrics, sessions, etc.

## application.properties

```
management.endpoints.web.exposure.include=health,sbom
```



# Build project

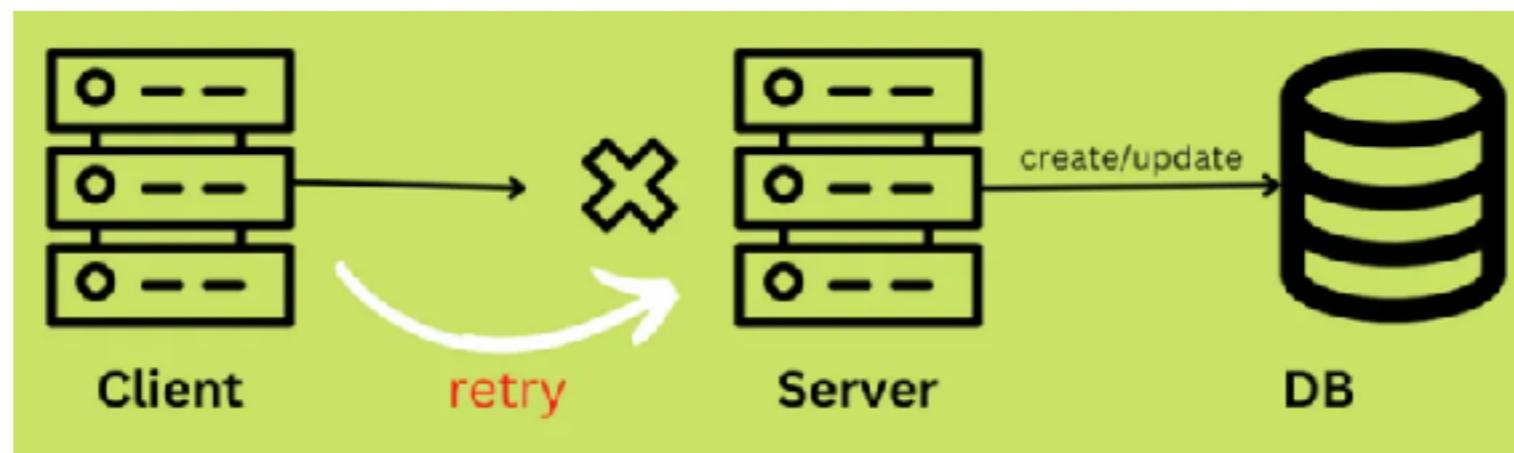
\$mvnw clean package

```
localhost:8080/actuator/sbom/application

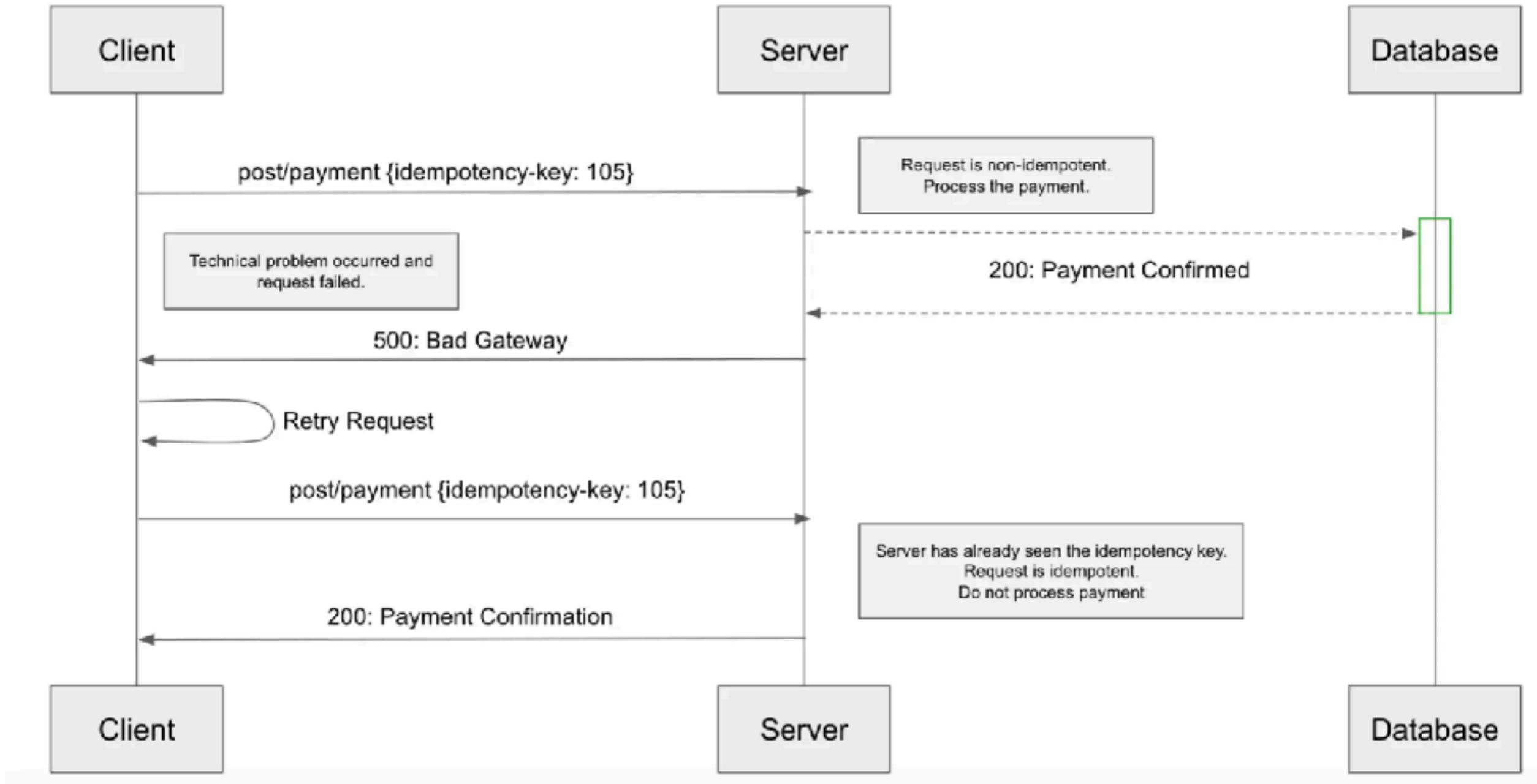
{
  "bomFormat": "CycloneDX",
  "specVersion": "1.5",
  "serialNumber": "urn:uuid:95933afb-5c32-3d63-b28f-cff74821ede9",
  "version": 1,
  "metadata": {
    "timestamp": "2024-08-14T15:46:41Z",
    "lifecycles": [
      {
        "phase": "build"
      }
    ],
    "tools": [
      {
        "vendor": "OWASP Foundation",
        "name": "CycloneDX Maven plugin",
        "version": "2.8.0",
        "hashes": []
      }
    ]
  }
}
```



# Idempotent for APIs



# Flow of Payment API



# Q/A



# **Spring Boot 3.4.0**

**Next ...**

