



# 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

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

Help people take action on this Page. X



**[https://github.com/up1/  
course-springboot-2022](https://github.com/up1/course-springboot-2022)**



# 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

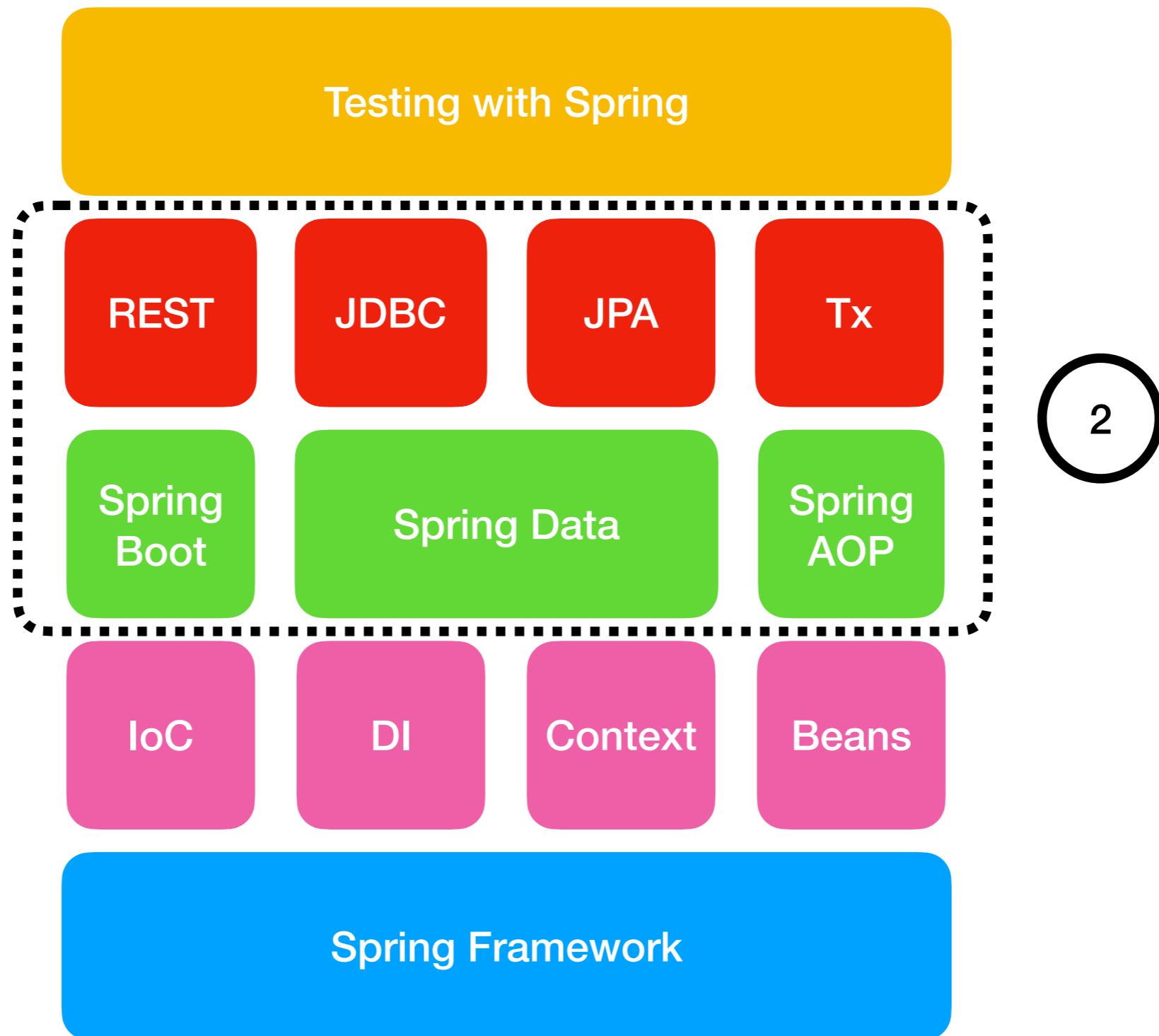
Layer in Spring Boot project

Working with Spring Data (JPA and JDBC)

Manage transaction



# Workshop



# Spring Boot



# Spring Framework



# Spring Boot



# REST



# **REST**

**RE**presentation **S**tate **T**ransfer

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



# **RESTful service**



# 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

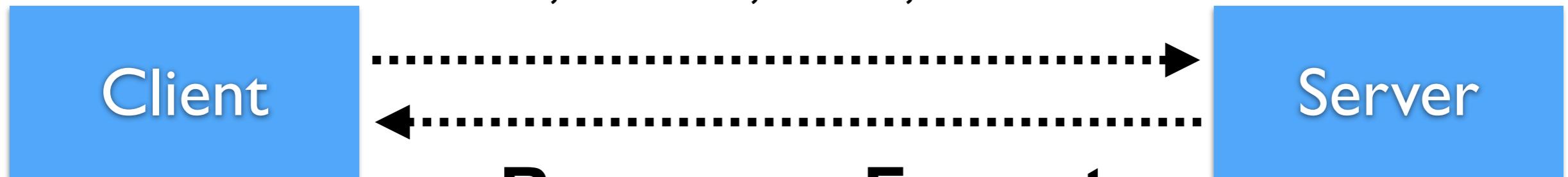
Request can include parameter and data in  
body of request as XML, JSON etc.



# REST Request & Response

**Request Method**

GET, POST, PUT, DELETE



**Response Format**

XML or JSON



# HTTP Methods meaning

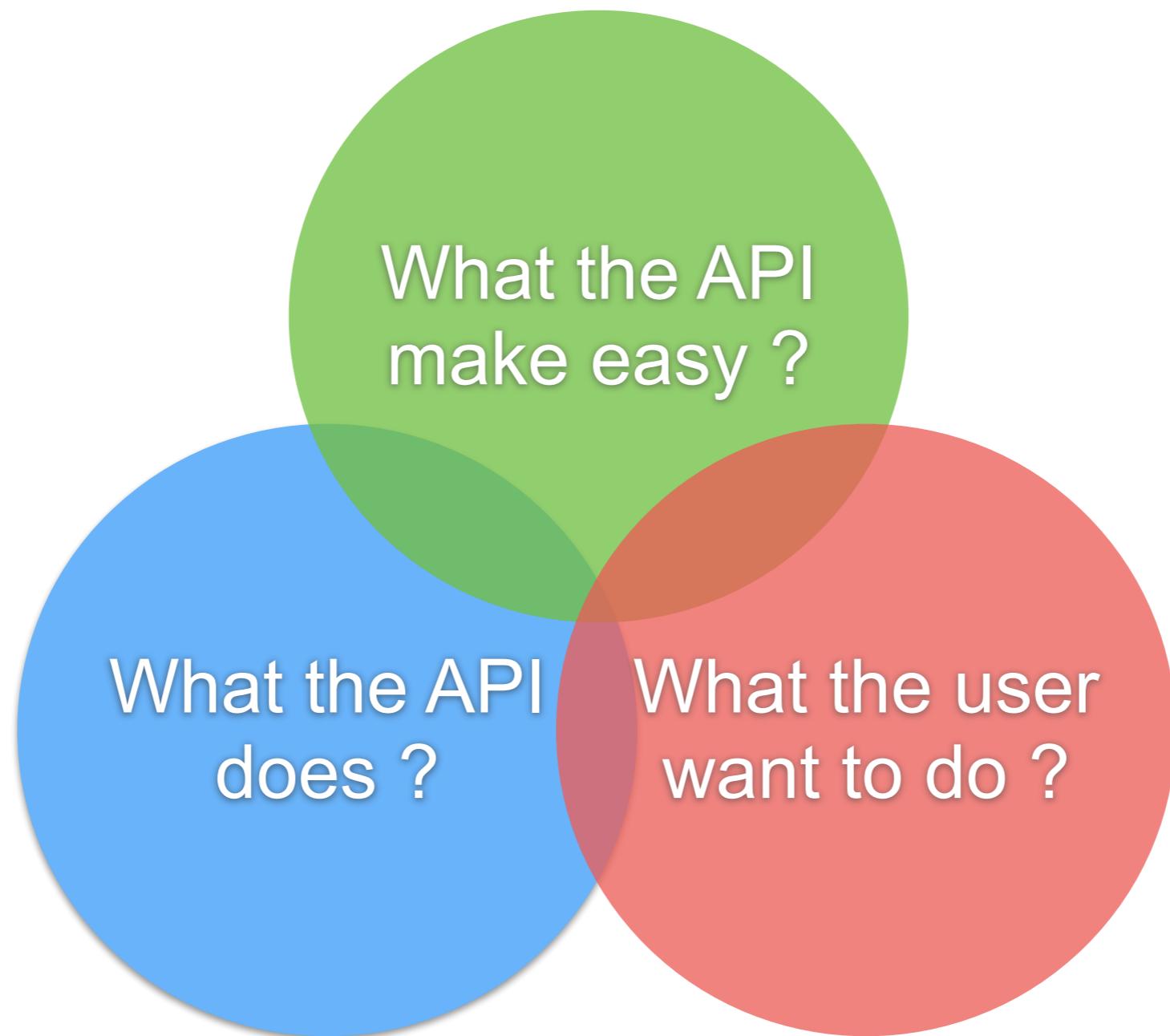
Method	Meaning
GET	Read data
POST	Create/Insert new data
PUT/PATCH	Update data or insert if a new id
DELETE	Delete data



# Response format ?

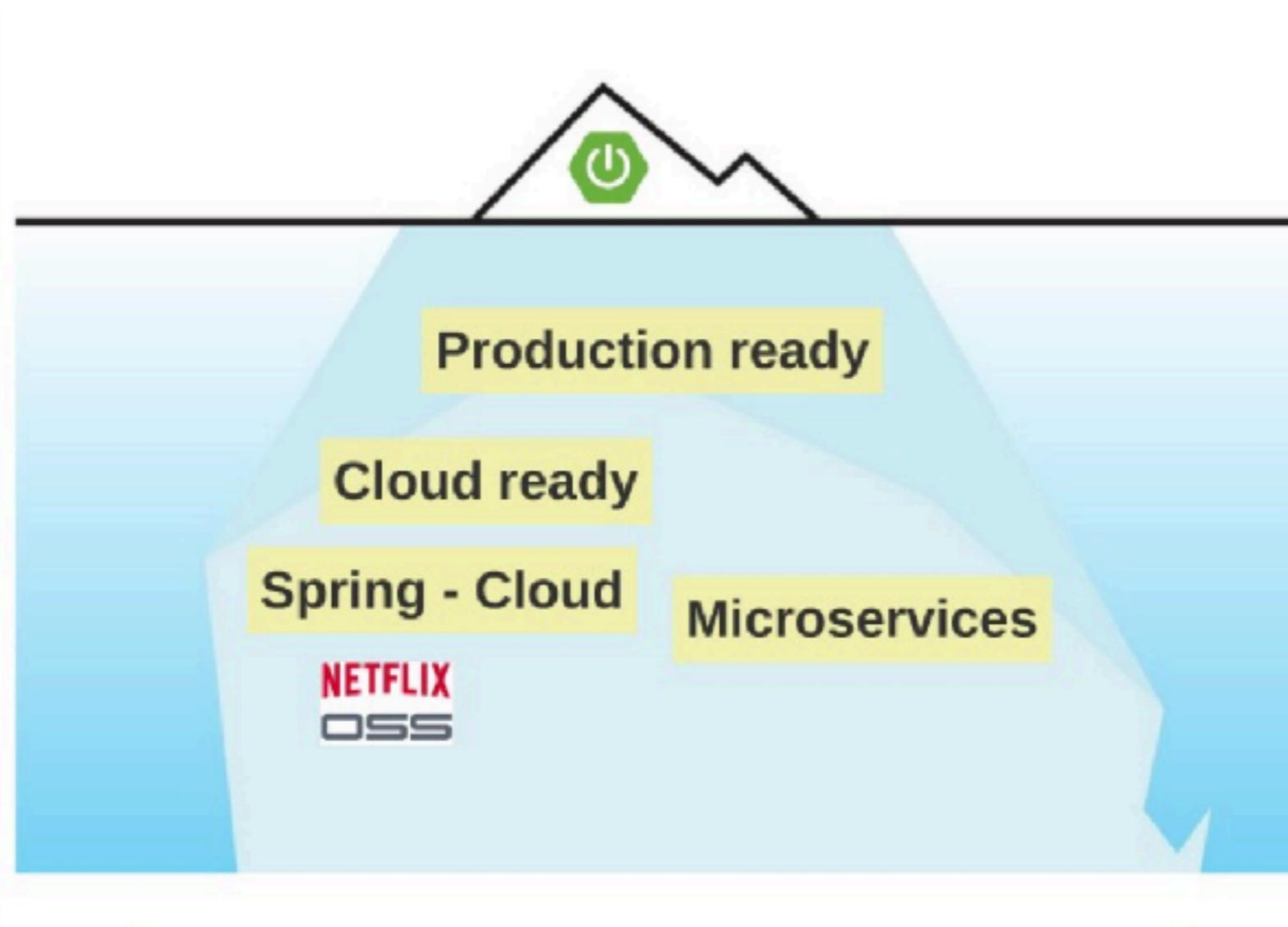


# Good APIs ?



# Why ?

Application skeleton generator  
Reduce effort to add new technologies



# What ?

Embedded application server

Integration with tools/technologies (starter)

Production tools (monitoring, health check)

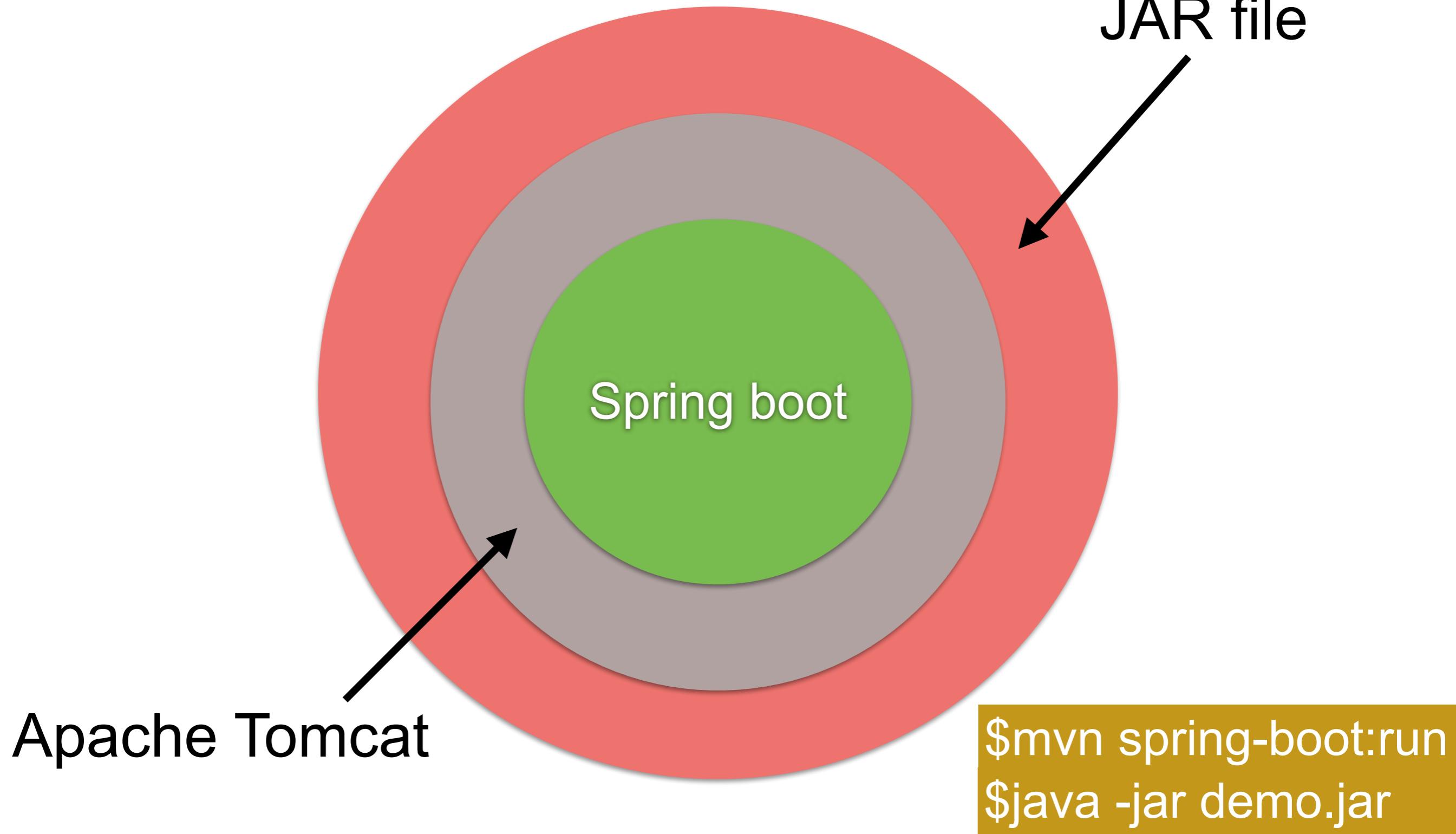
Configuration management

Dev tools

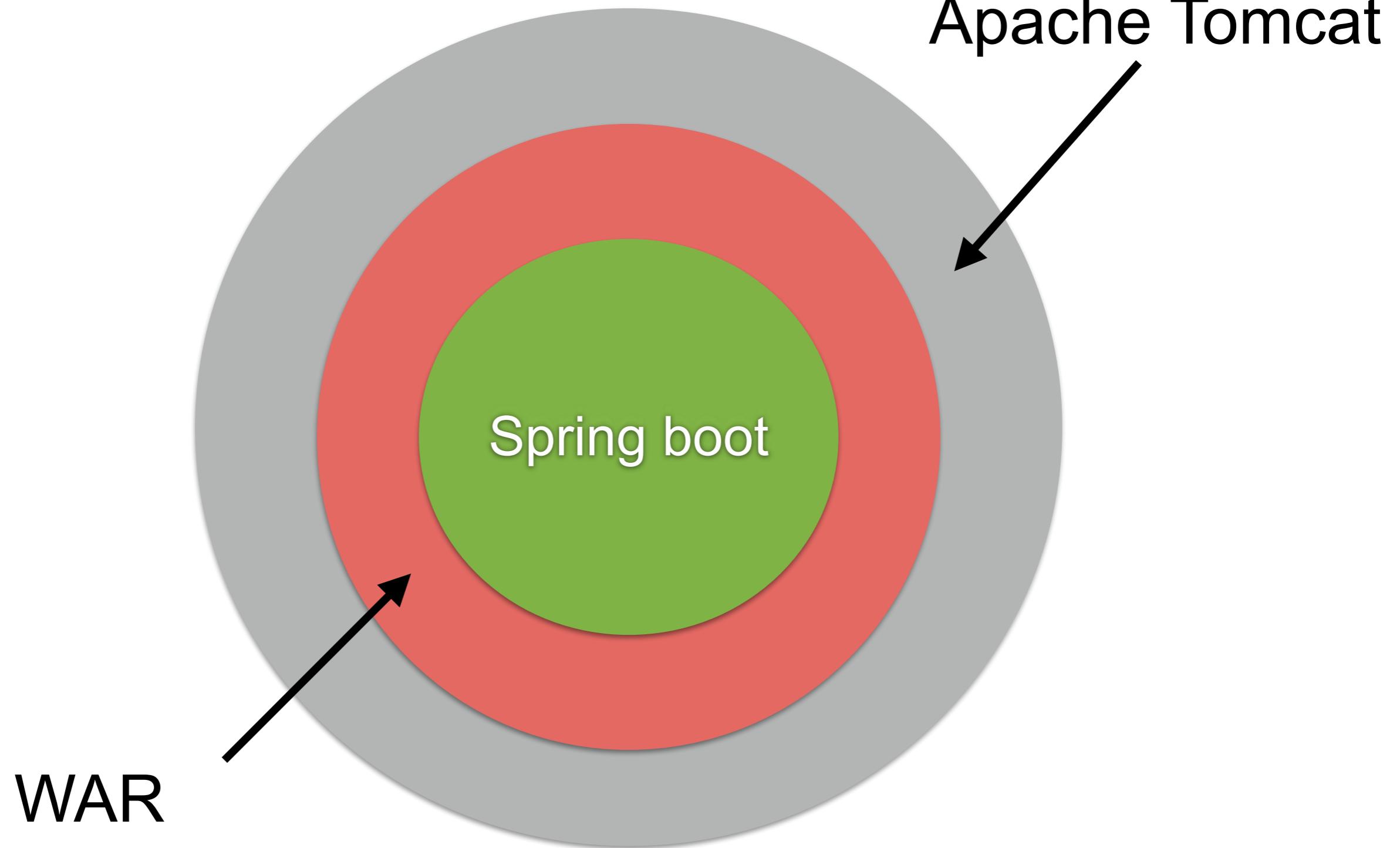
No source code generation, no XML



# How ?



# How ?



# **Building RESTful API with Spring Boot**



# Create new project



**Project**  
 Maven Project  Gradle Project

**Language**  
 Java  Kotlin  Groovy

**Spring Boot**  
 2.5.0 (SNAPSHOT)  2.4.2 (SNAPSHOT)  2.4.1  2.3.8 (SNAPSHOT)  
 2.3.7

**Project Metadata**

Group com.workshop

Artifact day2

Name day2

Description Demo project for Spring Boot

Package name com.workshop.day2

Packaging  Jar  War

Java  15  11  8

## Dependencies

[ADD DEPENDENCIES...](#) ⌘ + B

### Spring Web WEB

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

### Spring Data JPA SQL

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

### H2 Database SQL

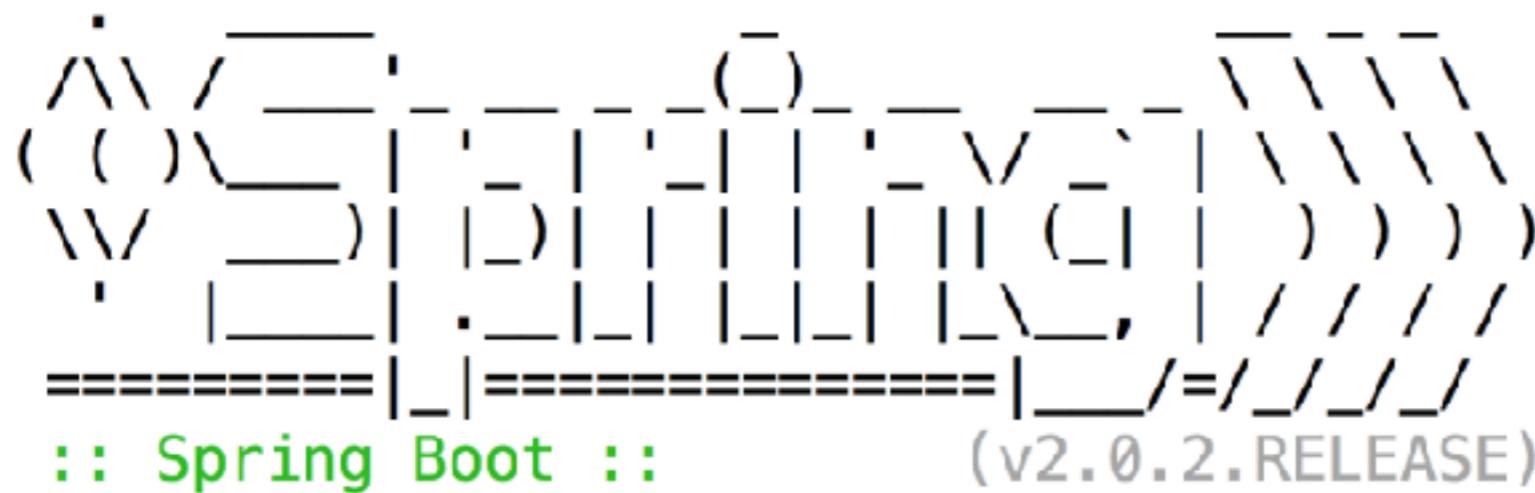
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser-based console application.

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)

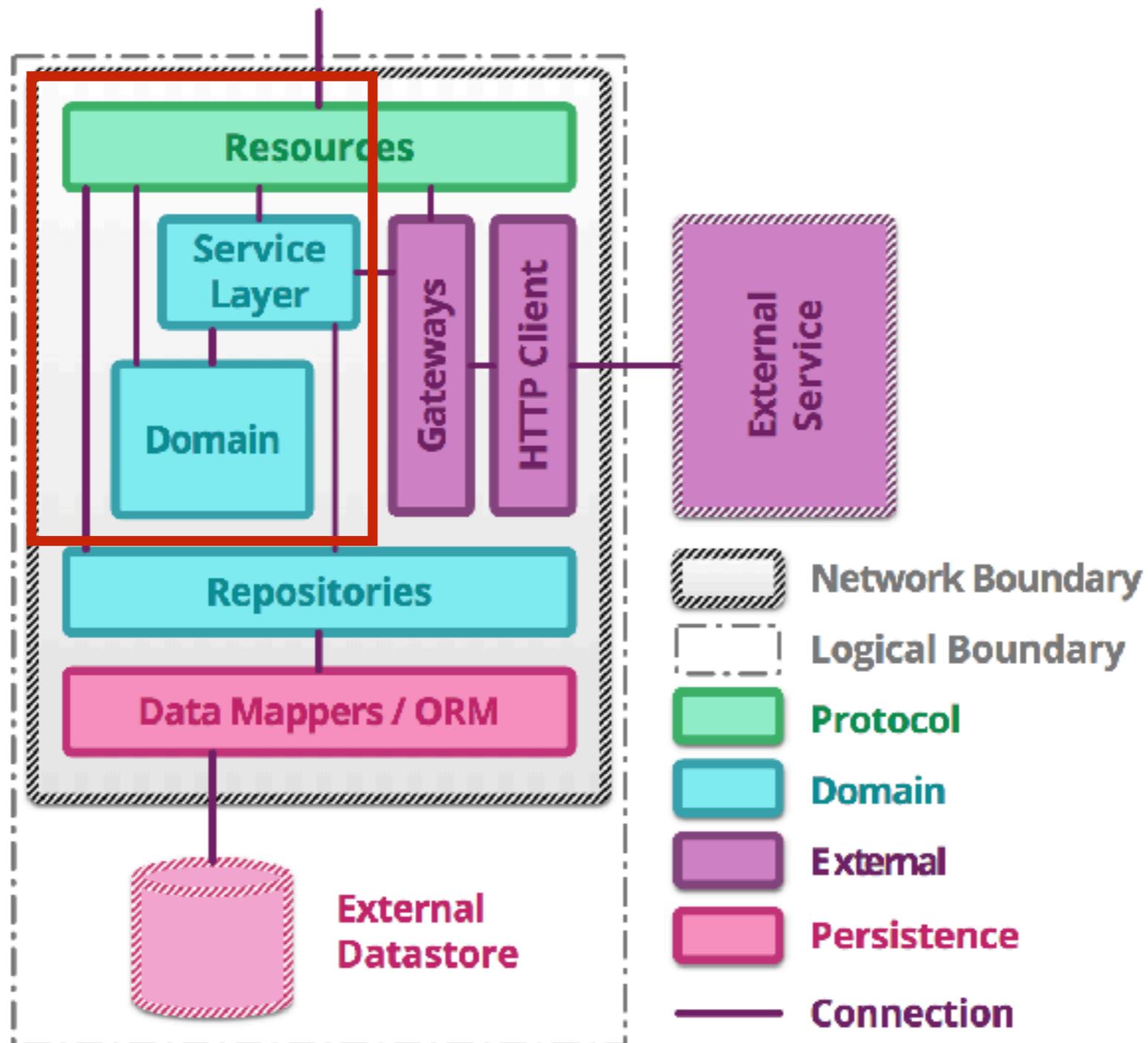
```
$./mvnw package  
$java -jar target/<file name>.jar
```

```
.  
(\ \ )\_) _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _ _  
 \| |\_ ) | | | | | | | | | | | | ) ) ) ) ) )  
' | _ | . _ | | | | | | | | | | | | | | | | | |  
=====|_|=====|_|/_/=/ / / / / / / /  
:: Spring Boot ::          (v2.0.2.RELEASE)
```

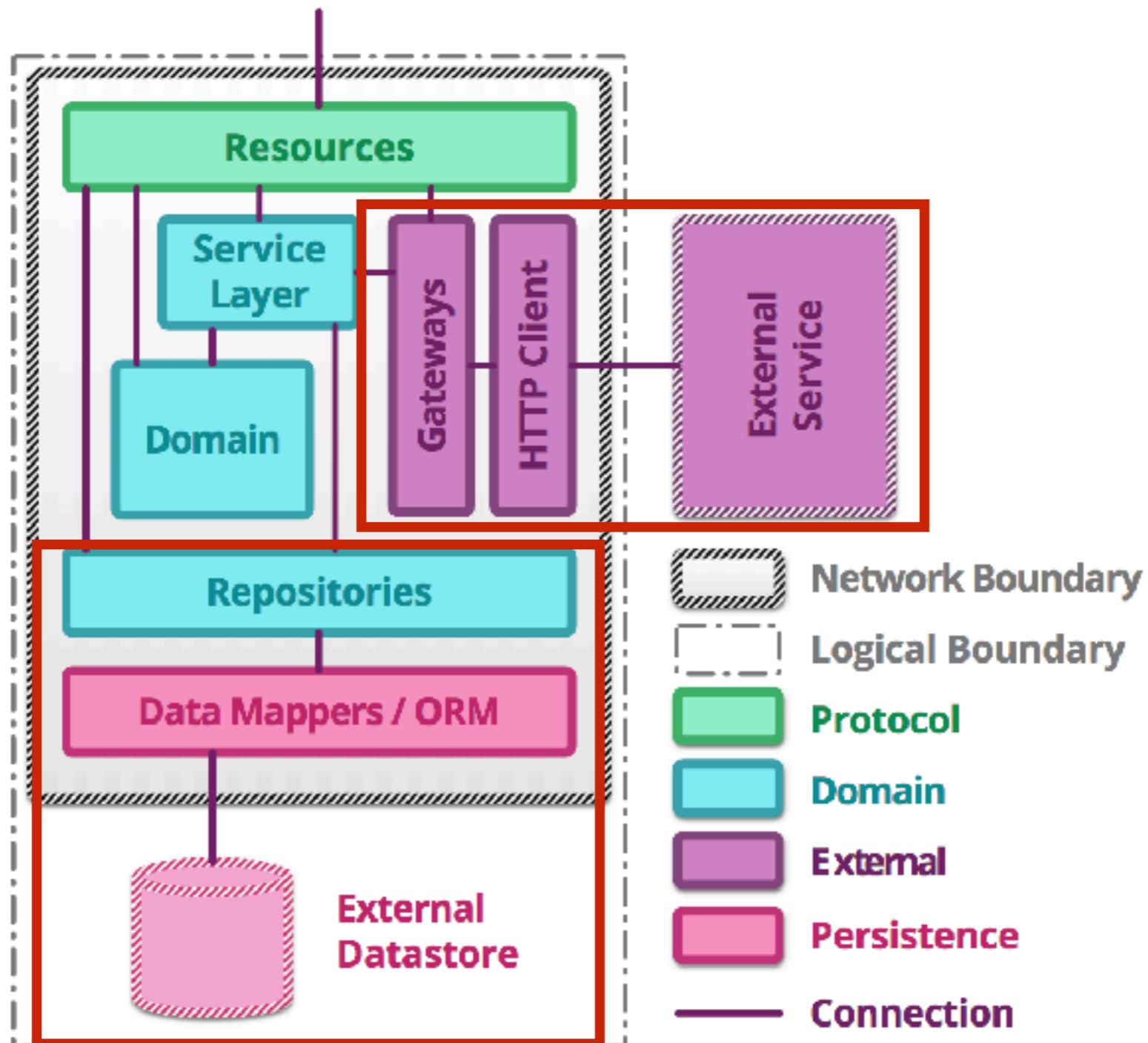
```
2018-06-07 13:03:30.412  INFO 12828 --- [  
  oApplication           : Starting DemoApplication on  
D 12828 (started by somkiat in /Users/somkiat/Down  
2018-06-07 13:03:30.418  INFO 12828 --- [  
  oApplication           : No active profile set, fall
```



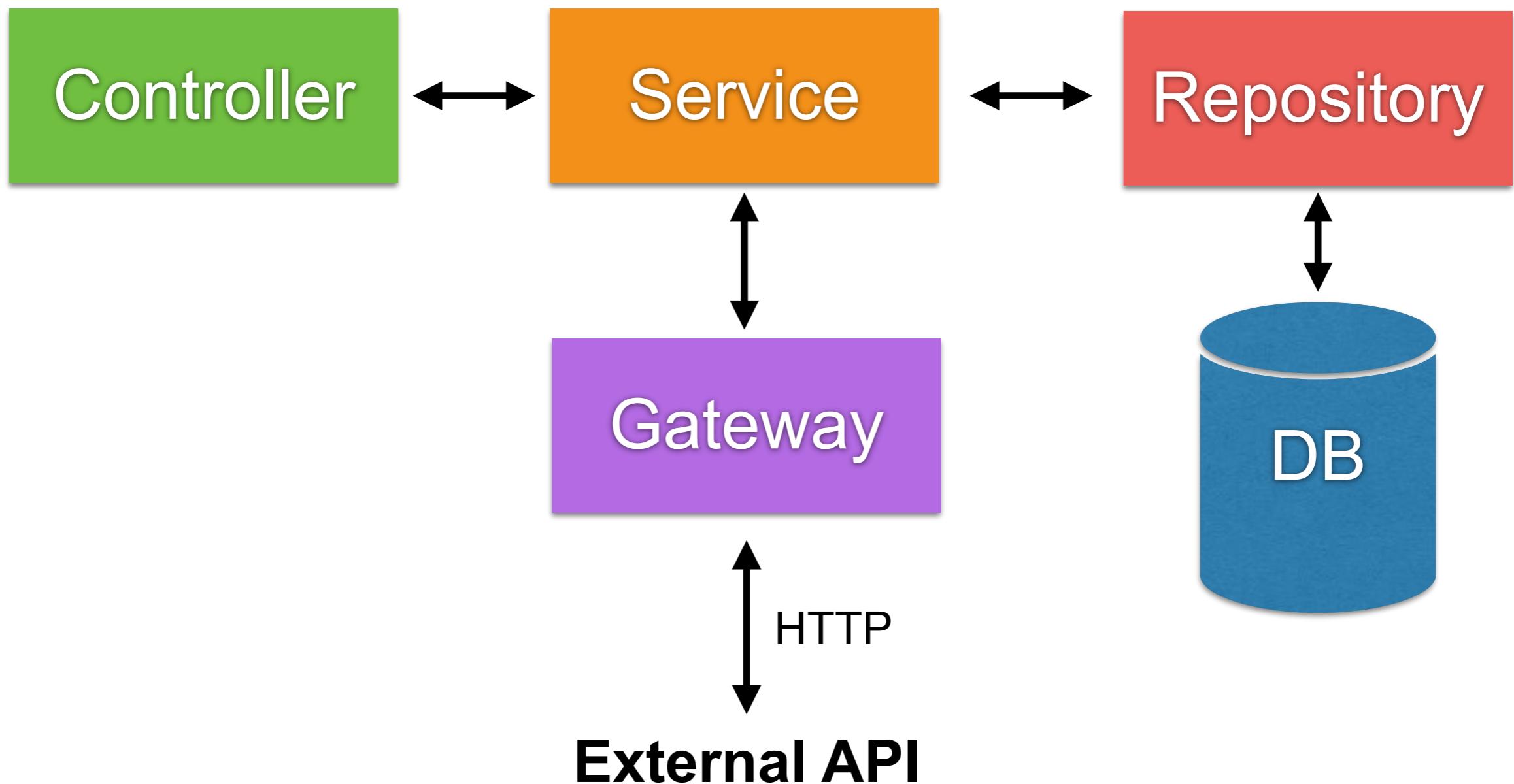
# Service Structure



# 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

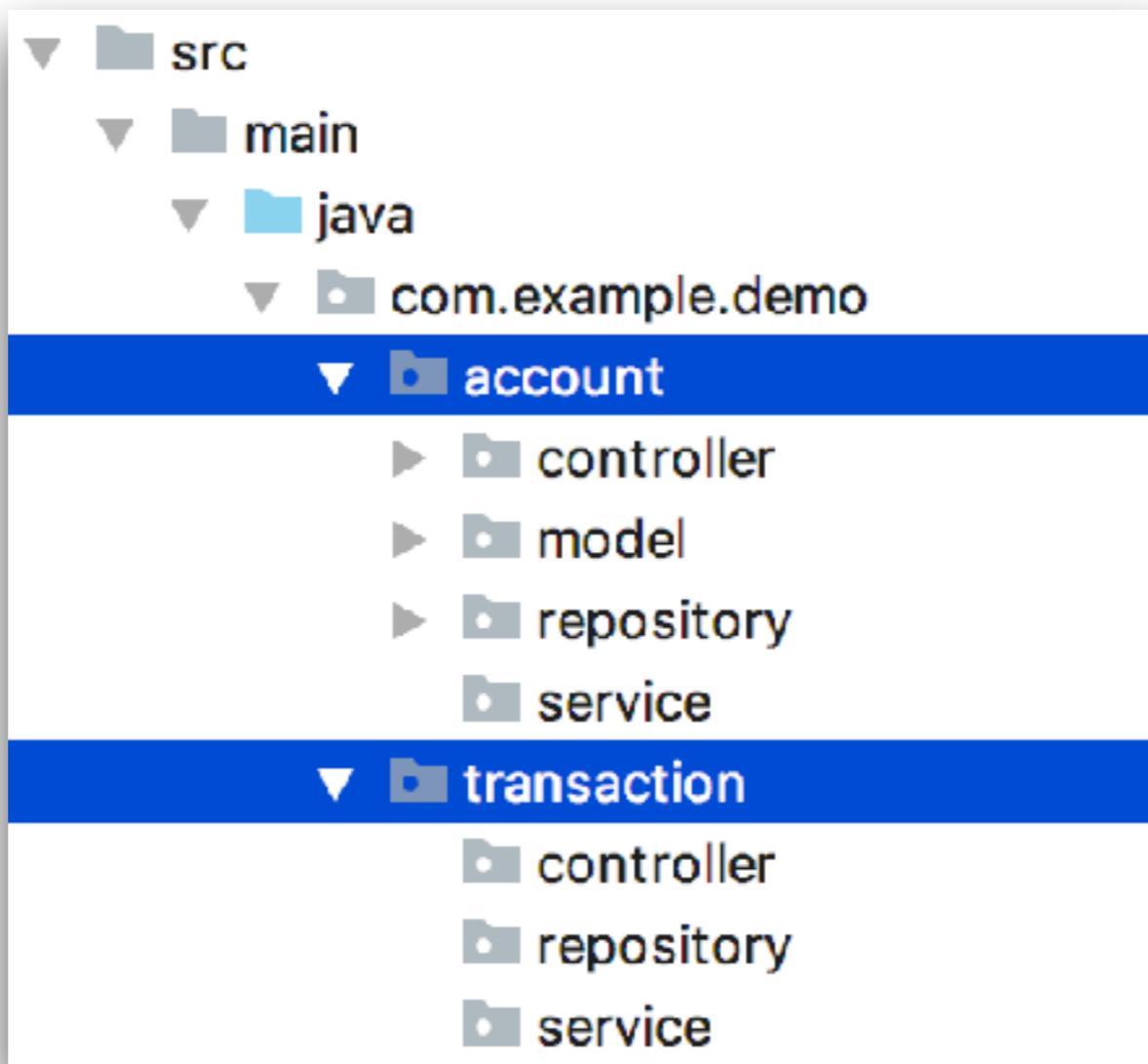
- feature1**
  - controller
  - service
  - repository
- feature2**
  - controller
  - service
  - repository

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



# Spring Boot Structure (2)

Separate by function/domain/feature



# Design RESTful APIs

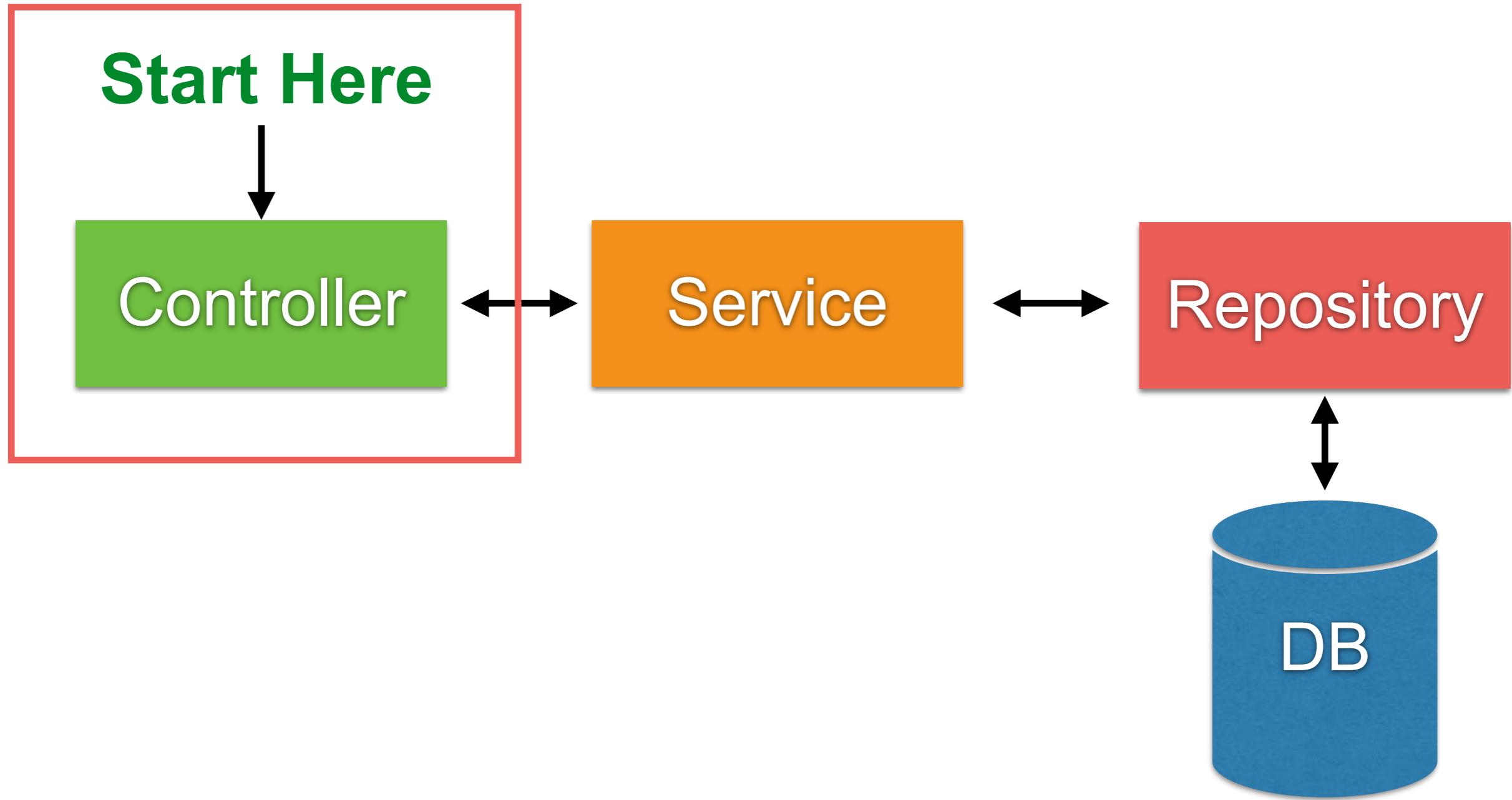


# Users

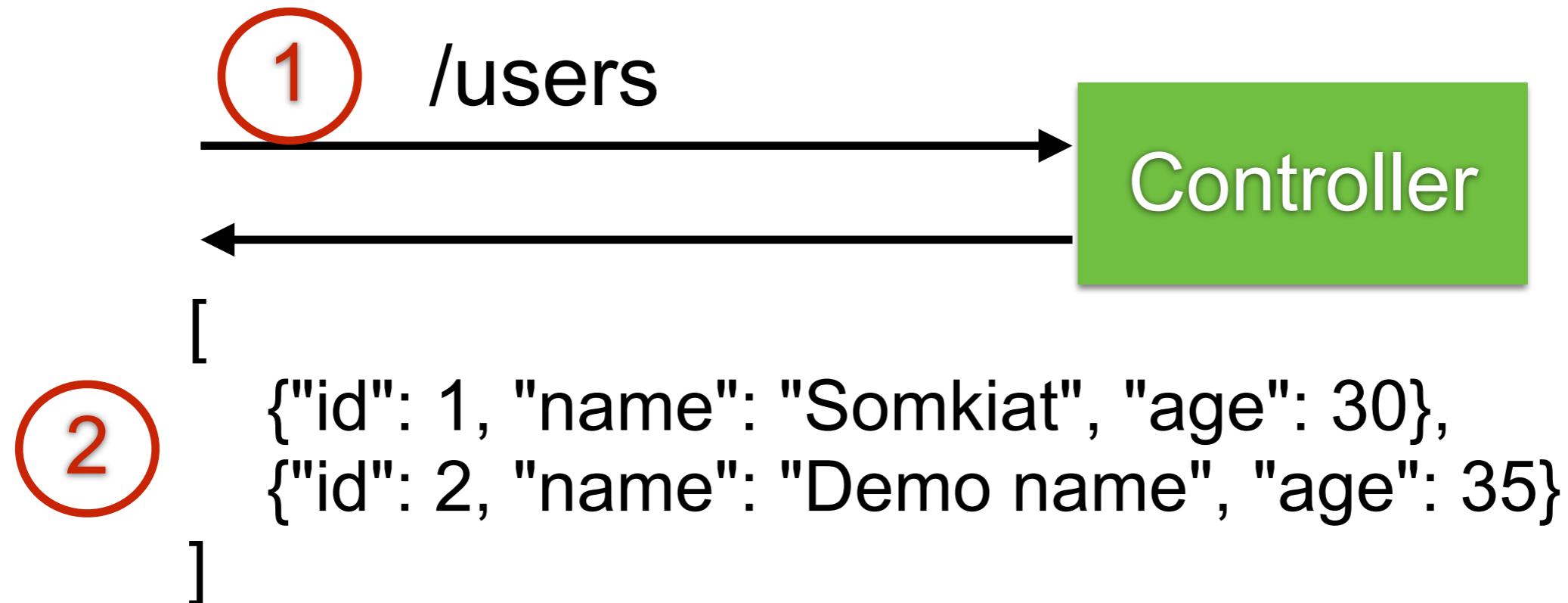
Method	Path	Description
GET	/users	Get all users
GET	/users/{id}	Get user by id
POST	/users	Create new user
PUT	/users/{id}	Update user
DELETE	/users/{id}	Delete user by id



# 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

```
 .--. /==\ |----( )----\ V / \ \ \ \ \ 
( (( )\ \ \ \ |----| |----| |----| |----| |----| ) ) ) ) ) )
 \ \ \ \ \ \ \ \ |----| |----| |----| |----| |----| ) ) ) ) )
   '-----|-----|-----|-----|-----|-----| / / / / /
 :: Spring Boot ::          (v2.0.0.RELEASE)

2018-03-05 23:37:18.018  INFO 30560 --- [           main] : Starting HelloApplication v1.0-SNAPSHOT
th PID 30560 (/Users/somkiat/data/slide/microservice/sl
op/course-microservice/slide/4days-workshop/workshop/he
by somkiat in /Users/somkiat/data/slide/microservice/s
hop/course-microservice/slide/4days-workshop/workshop/he
2018-03-05 23:37:18.023  INFO 30560 --- [           main]
: No active profile set, falling back to default [mair
2018-03-05 23:37:18.138  INFO 30560 --- [           main]
```



# 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 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
}
```



# How to test the User controller ?



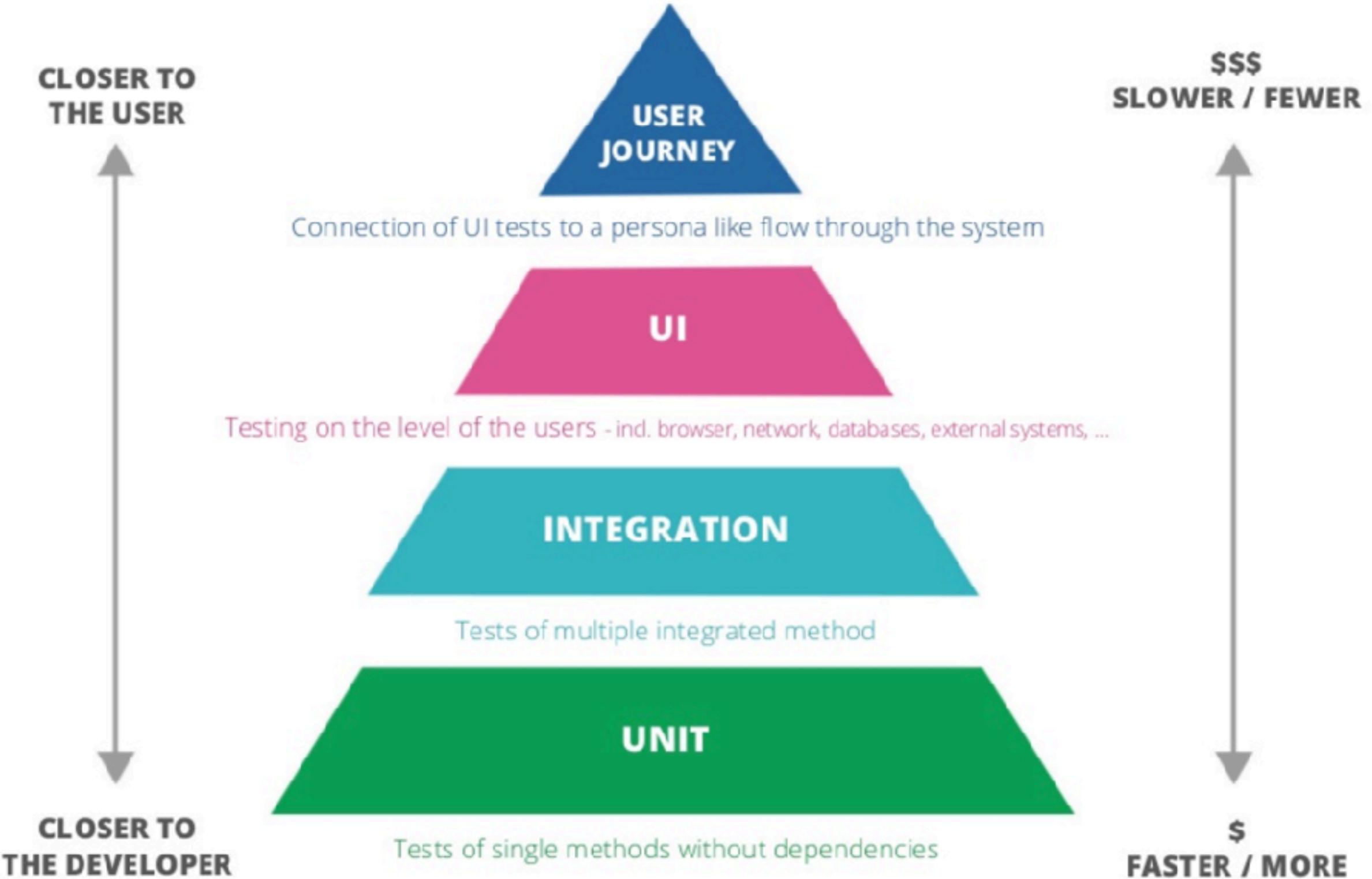
# Postman



POSTMAN

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





# Controller testing

## How to testing with Spring Boot ?



# Testing in Spring Boot

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



# Testing in Spring Boot

@SpringBootTest

@WebMVC Test

@JsonTest

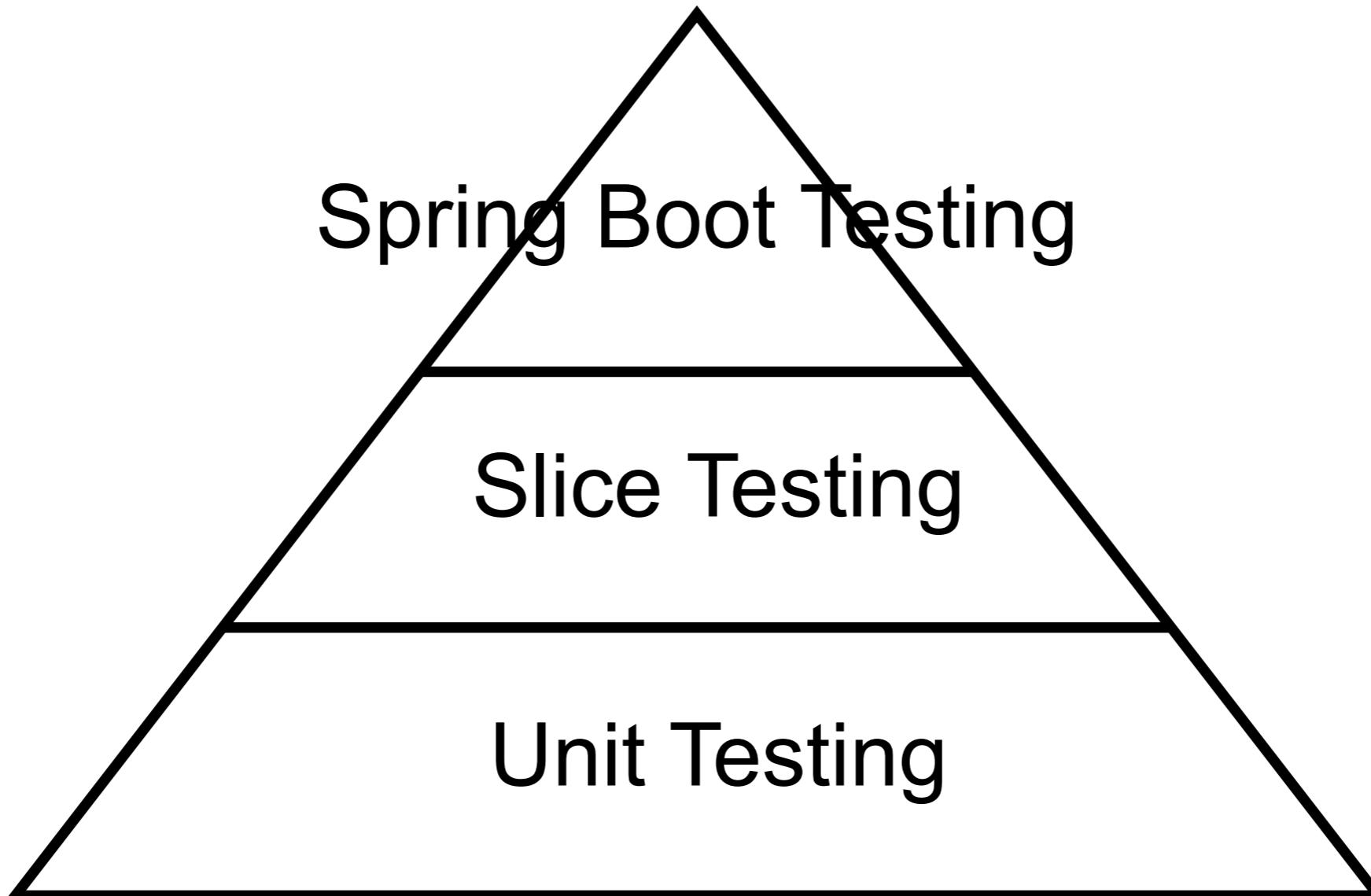
@DataJpaTest

@RestClientTest

**Slice testing**



# Testing in Spring Boot



# Controller testing

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



# 1. SpringBootTest

## Application context

Controller Advice

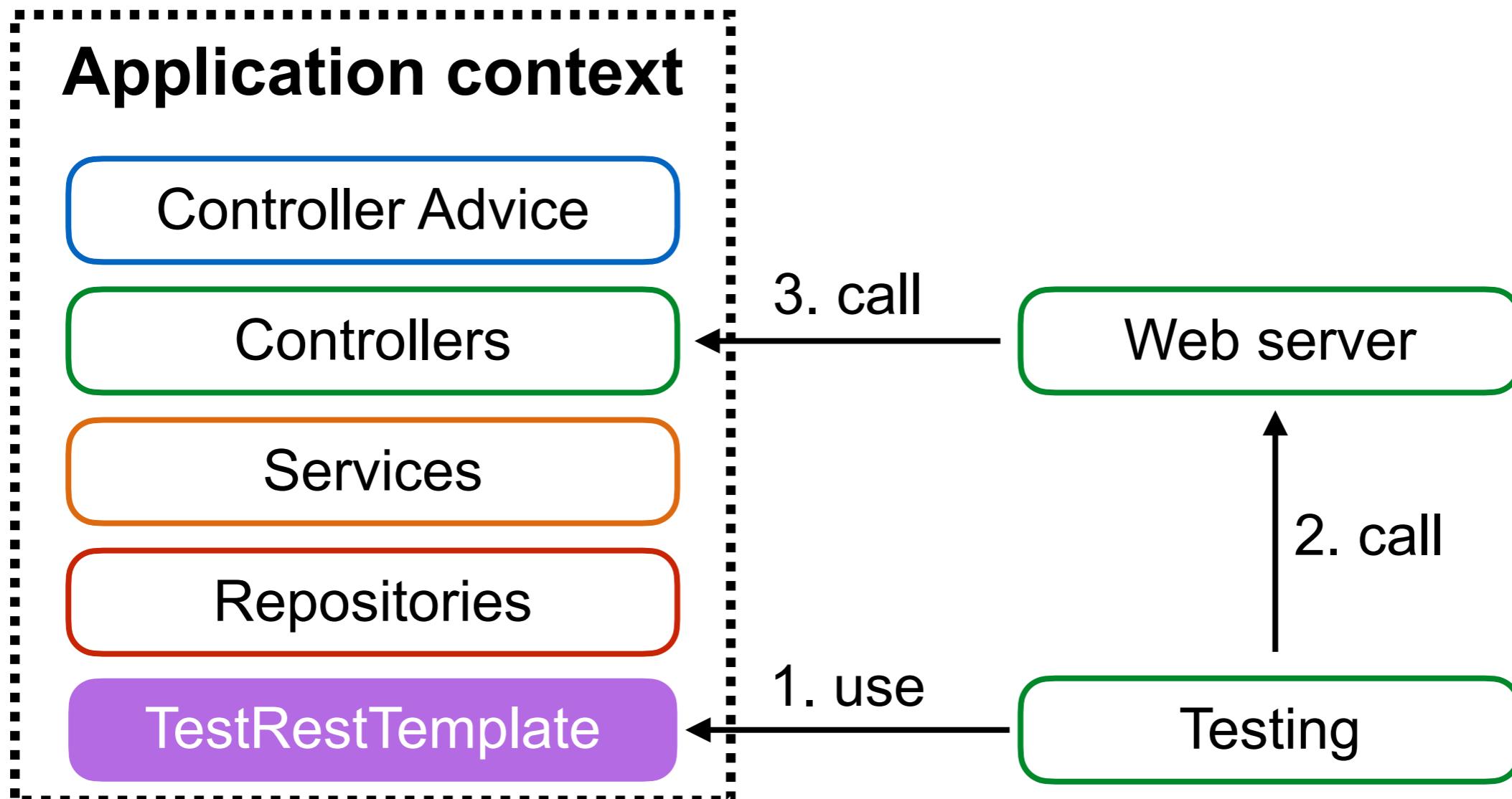
Controllers

Services

Repositories



# 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 {

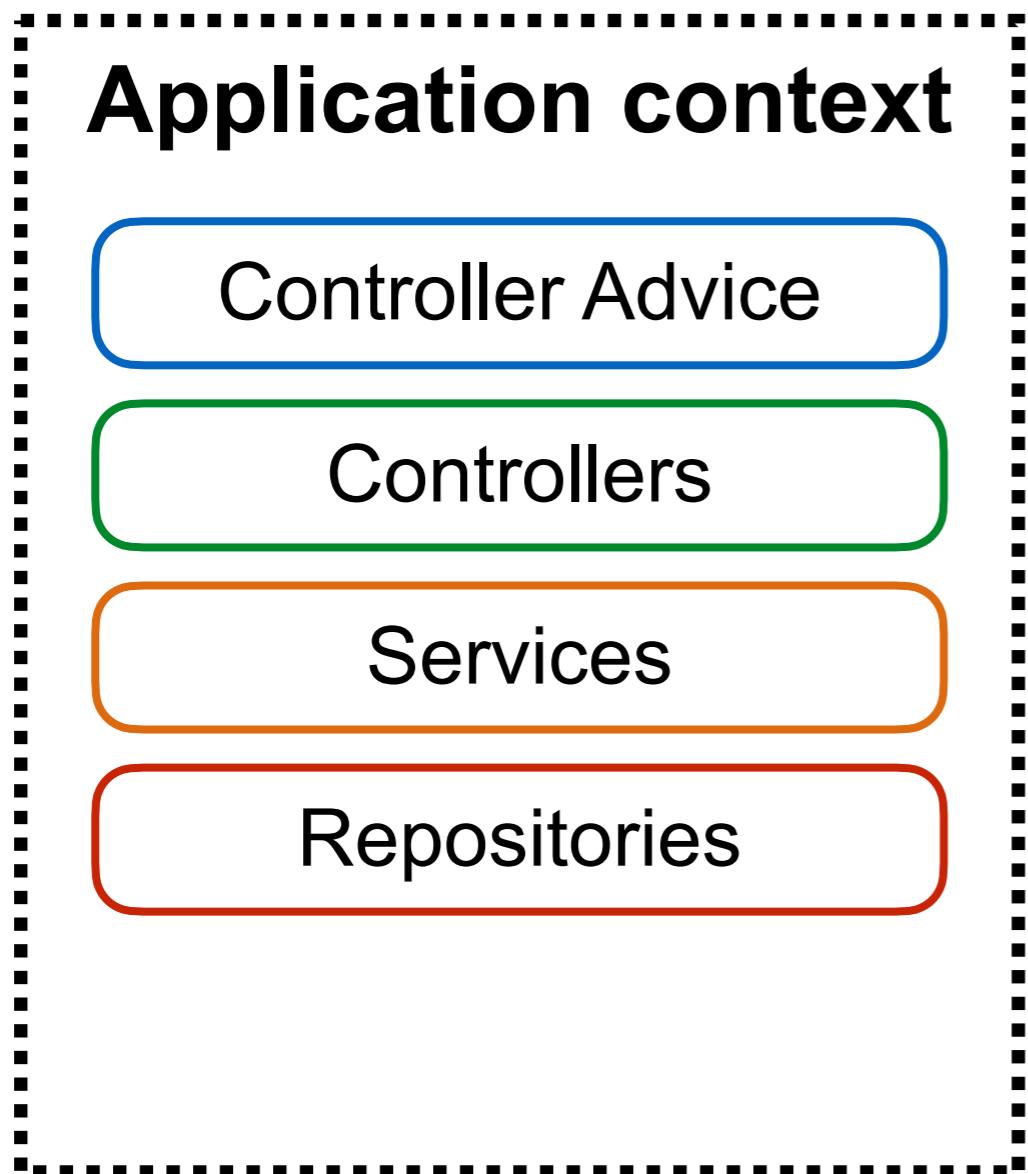
    @Autowired
    private TestRestTemplate testRestTemplate;

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

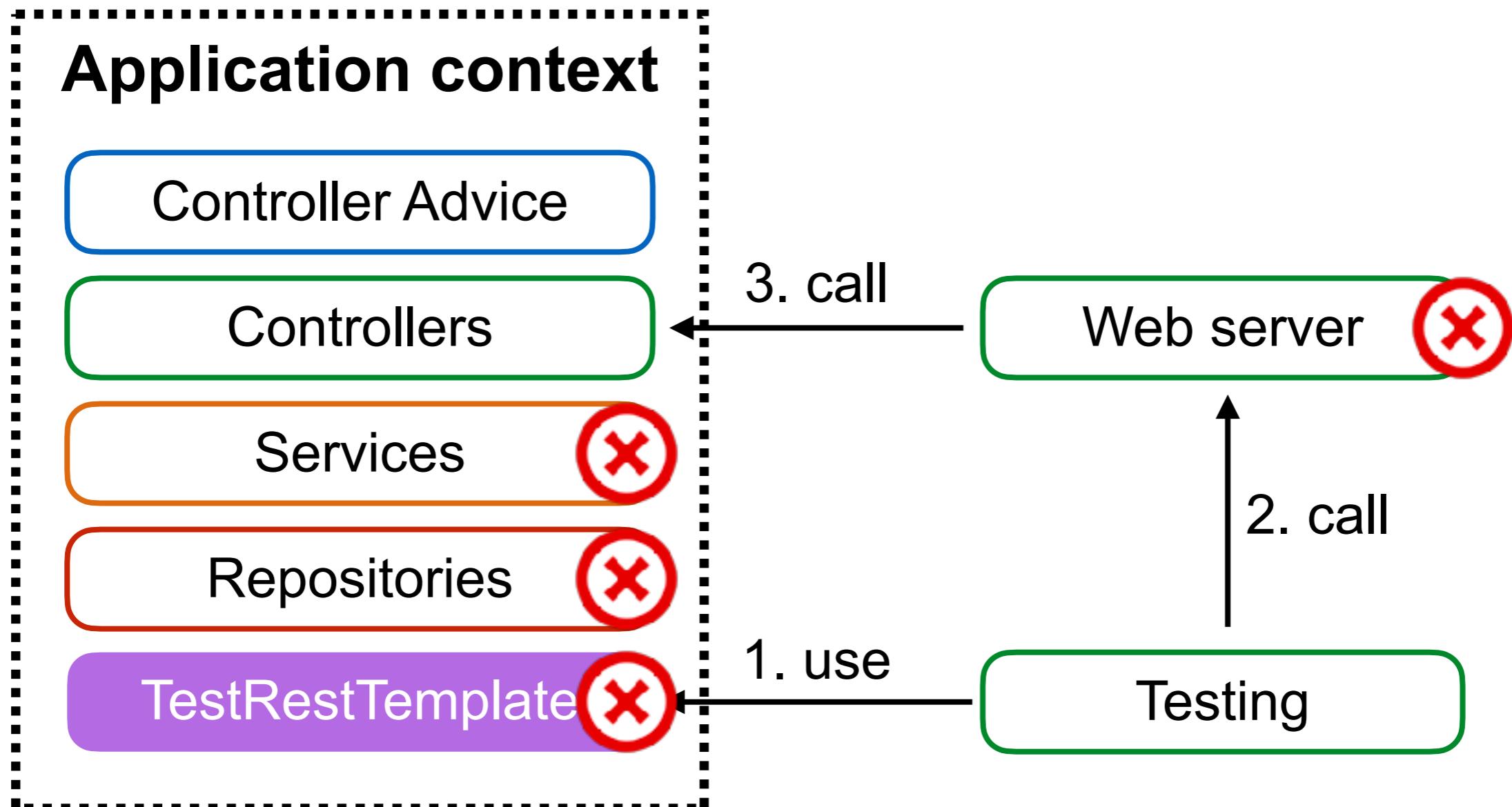
        // Assertion :: Check result with expected result
        assertEquals("Hello, somkiat", actualResult.getMessage());
    }
}
```



# 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



# 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



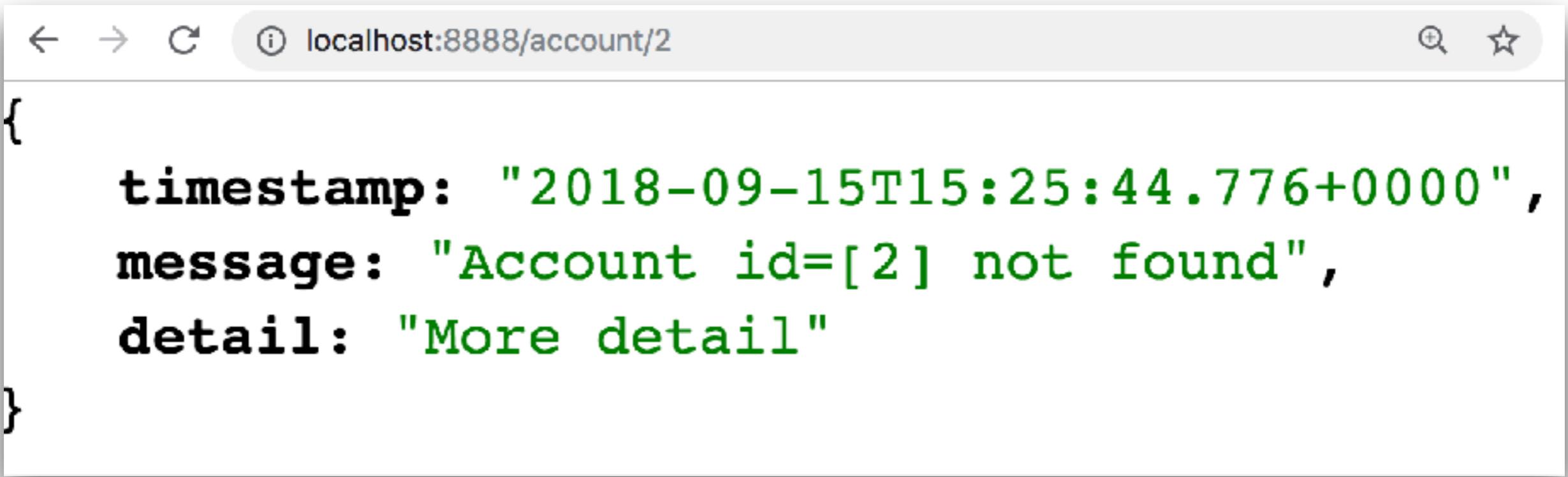
# ExceptionResponse

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



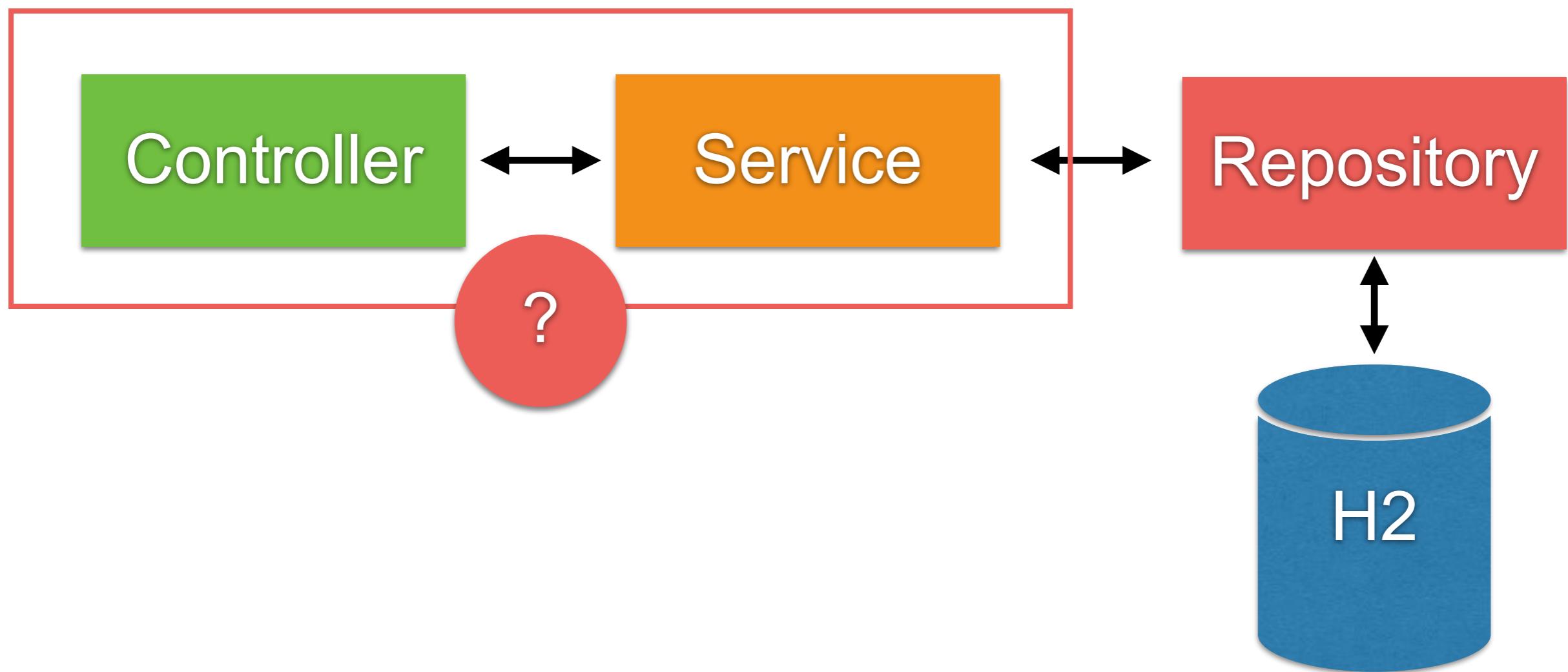
```
← → C ⓘ localhost:8888/account/2 🔍 ⭐  
{  
  timestamp: "2018-09-15T15:25:44.776+0000",  
  message: "Account id=[ 2 ] not found",  
  detail: "More detail"  
}
```



# 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



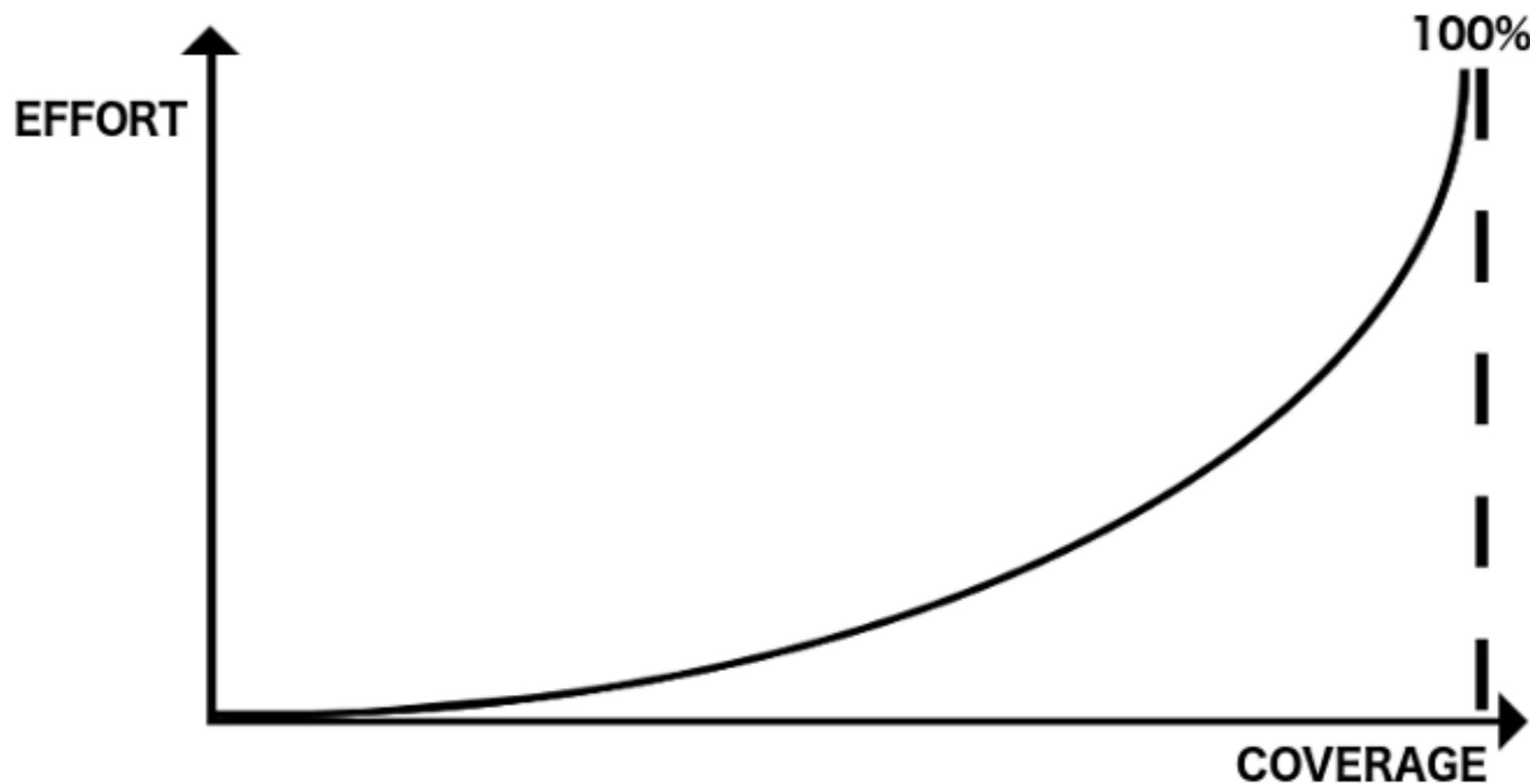
# Code coverage

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.

---

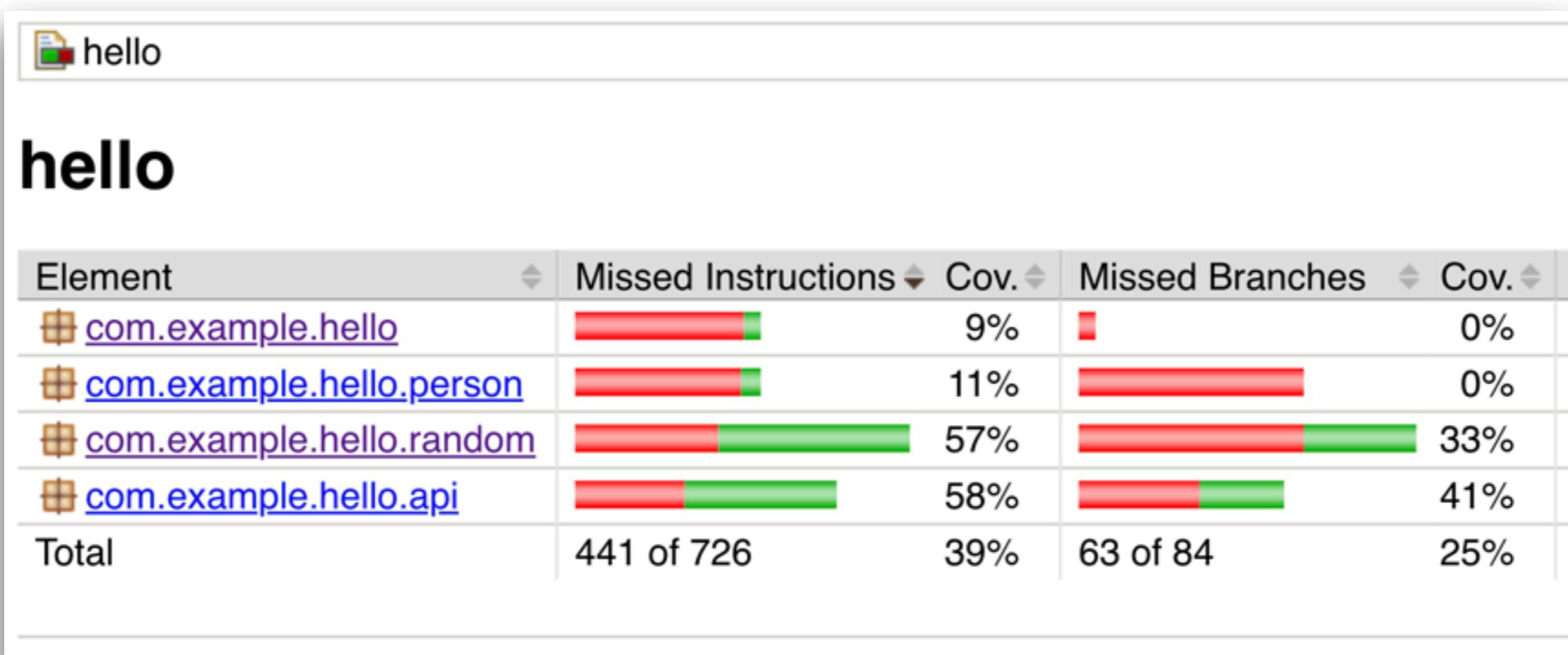
BUILD SUCCESS

---



# Coverage report

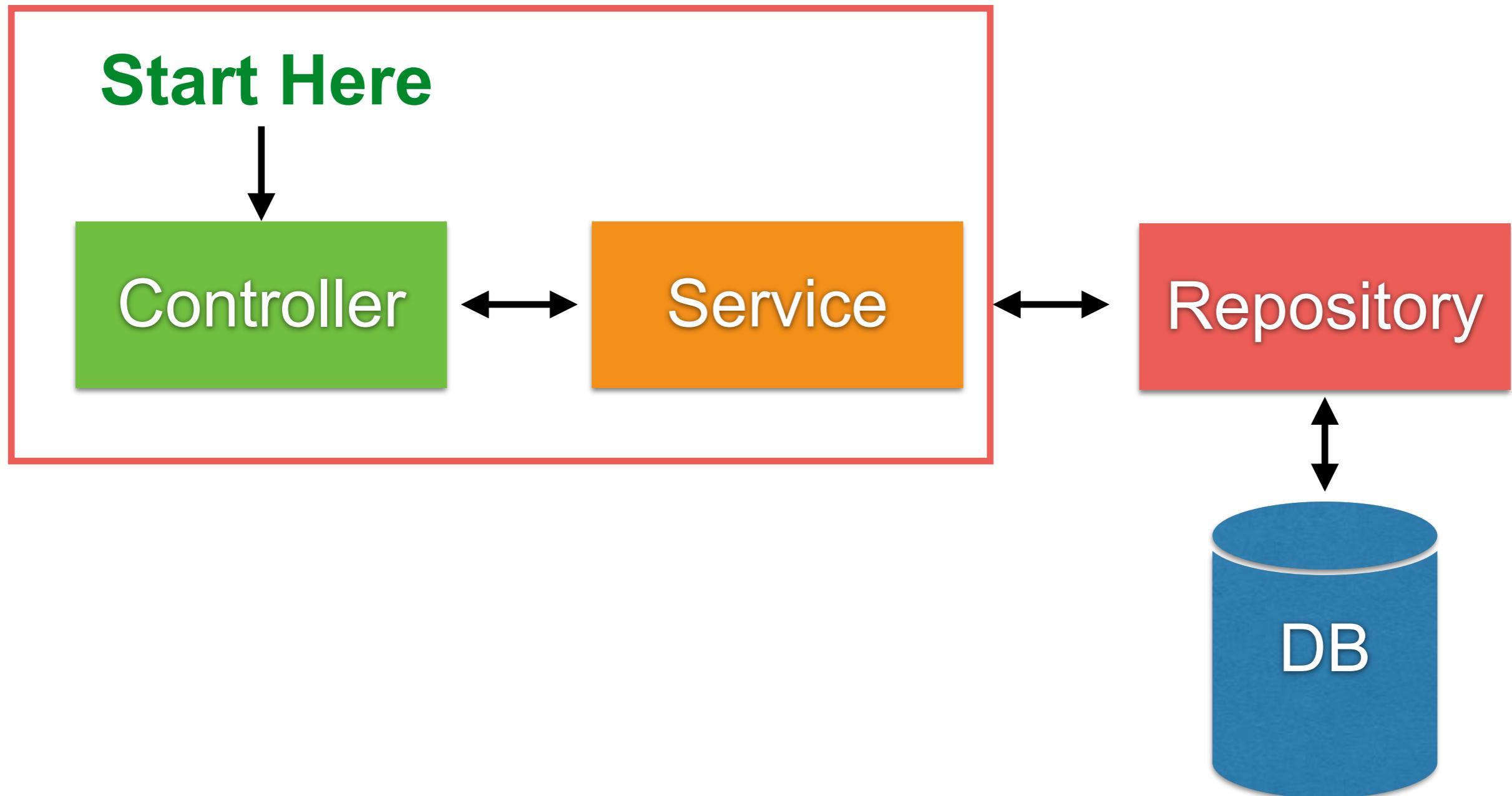
open target/site/jacoco/index.html



# Move business logic to service

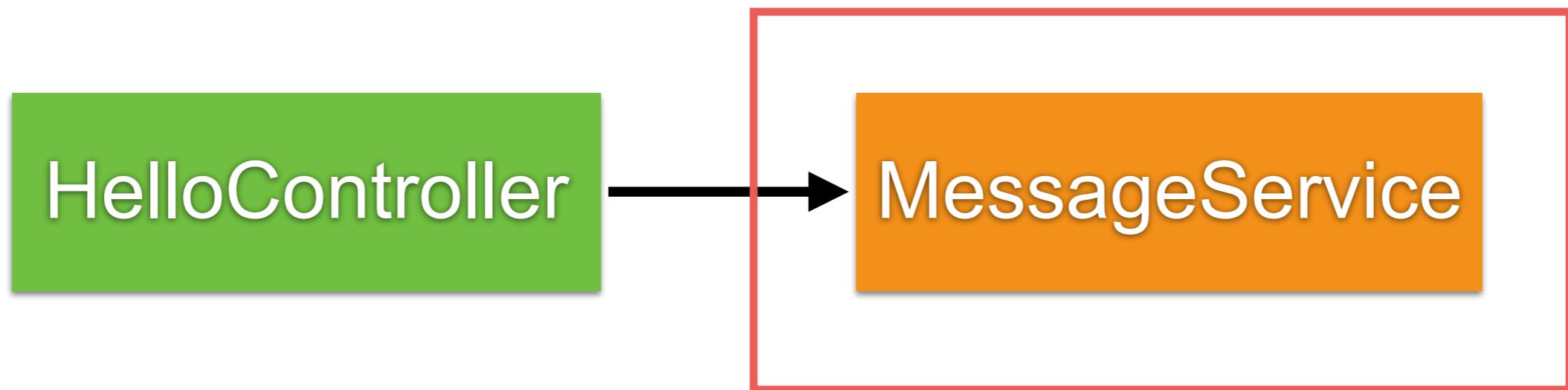


# Working with service



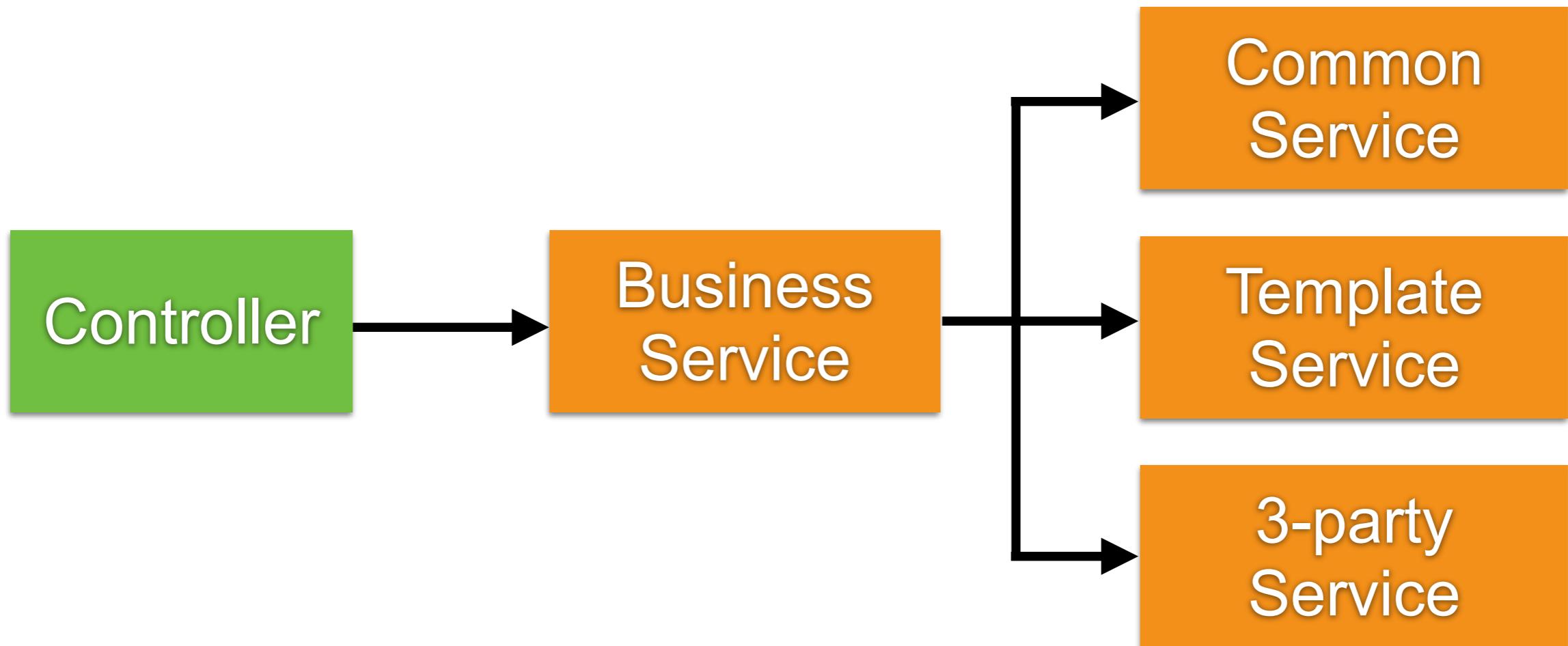
# Move business logic to service

Service class or interface ?

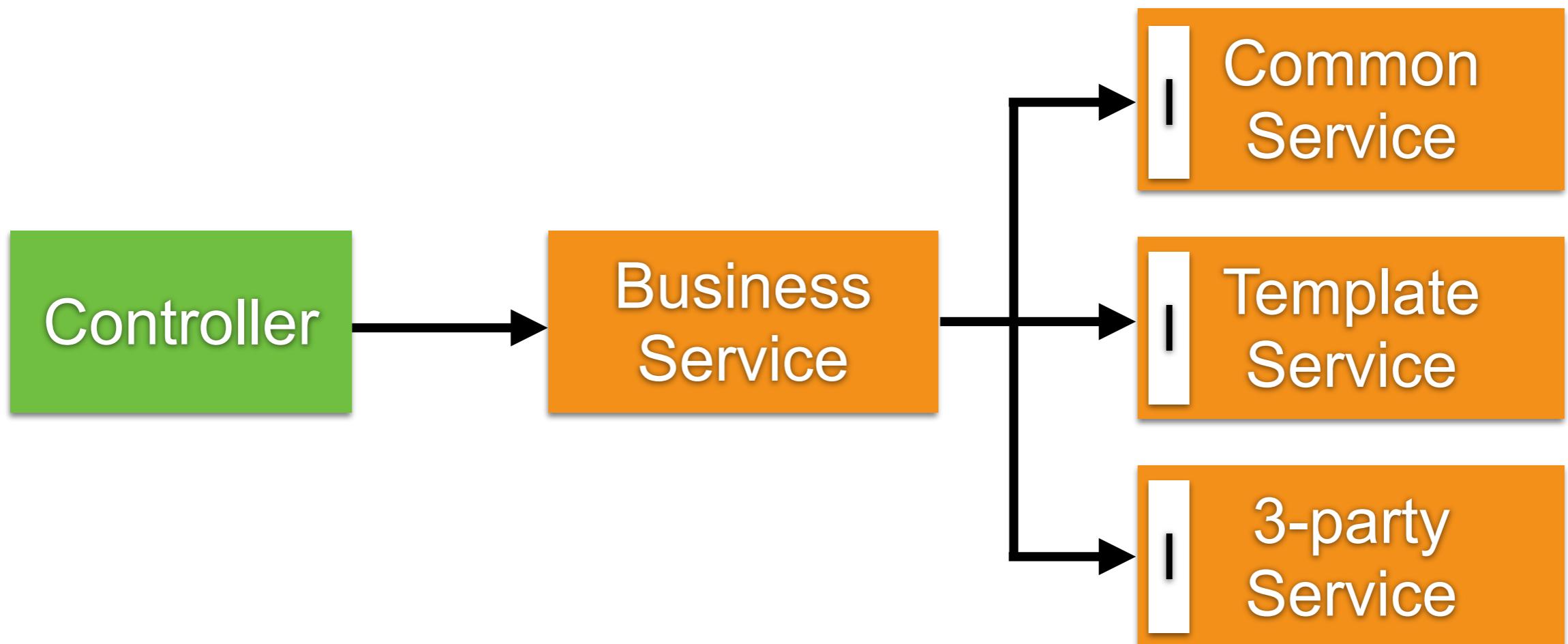


# Types of service

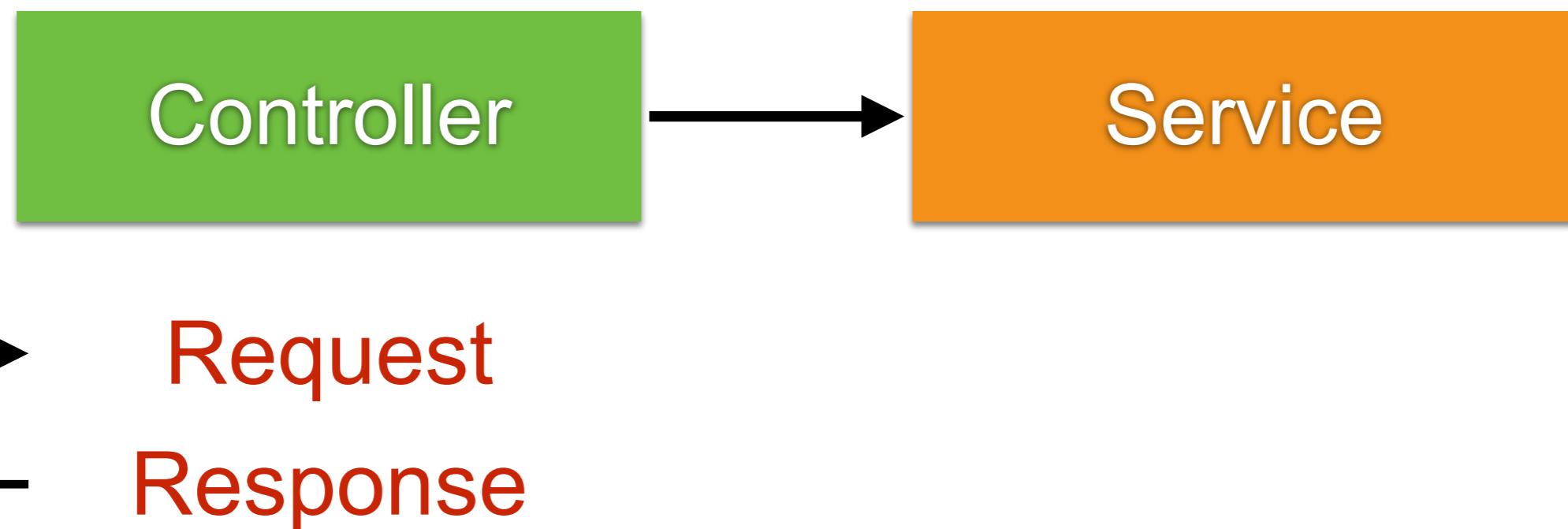
Business services (not reuse)  
Common/Template/3-party services



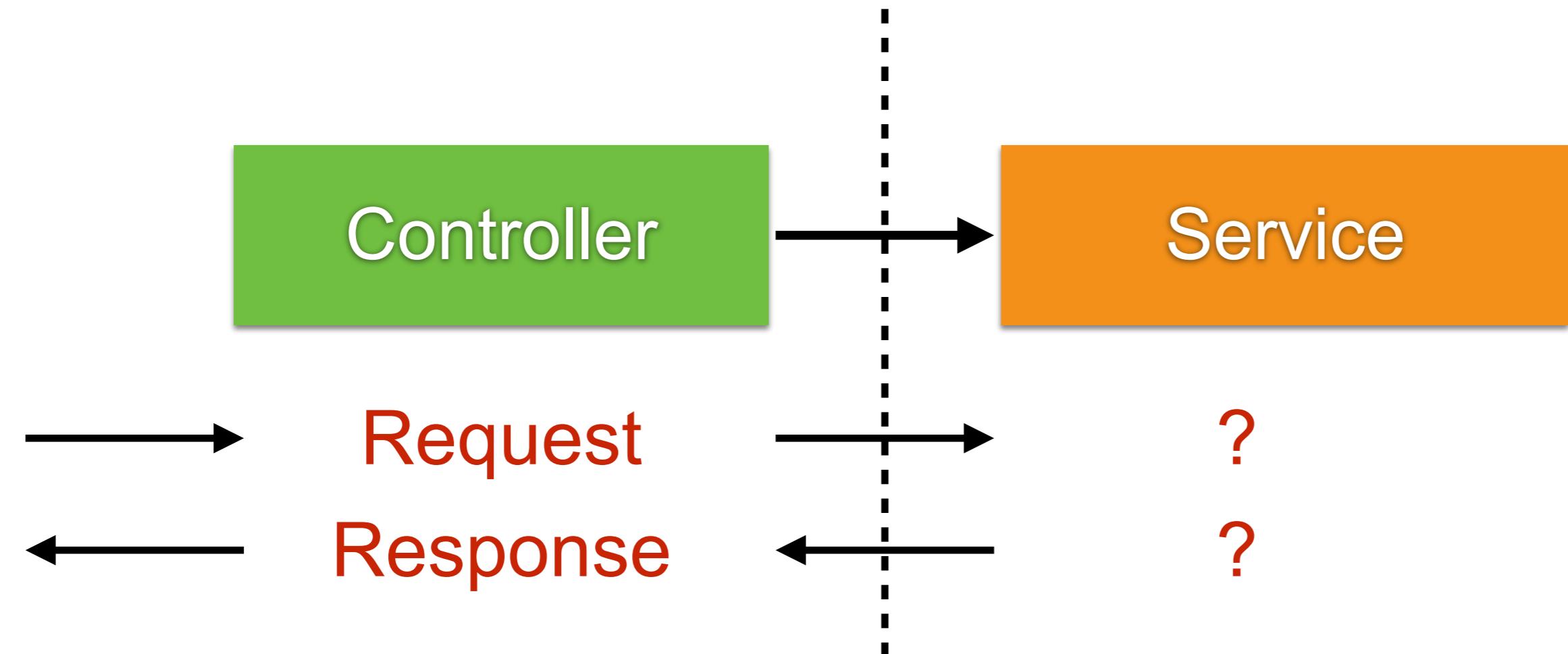
# Interface for common service



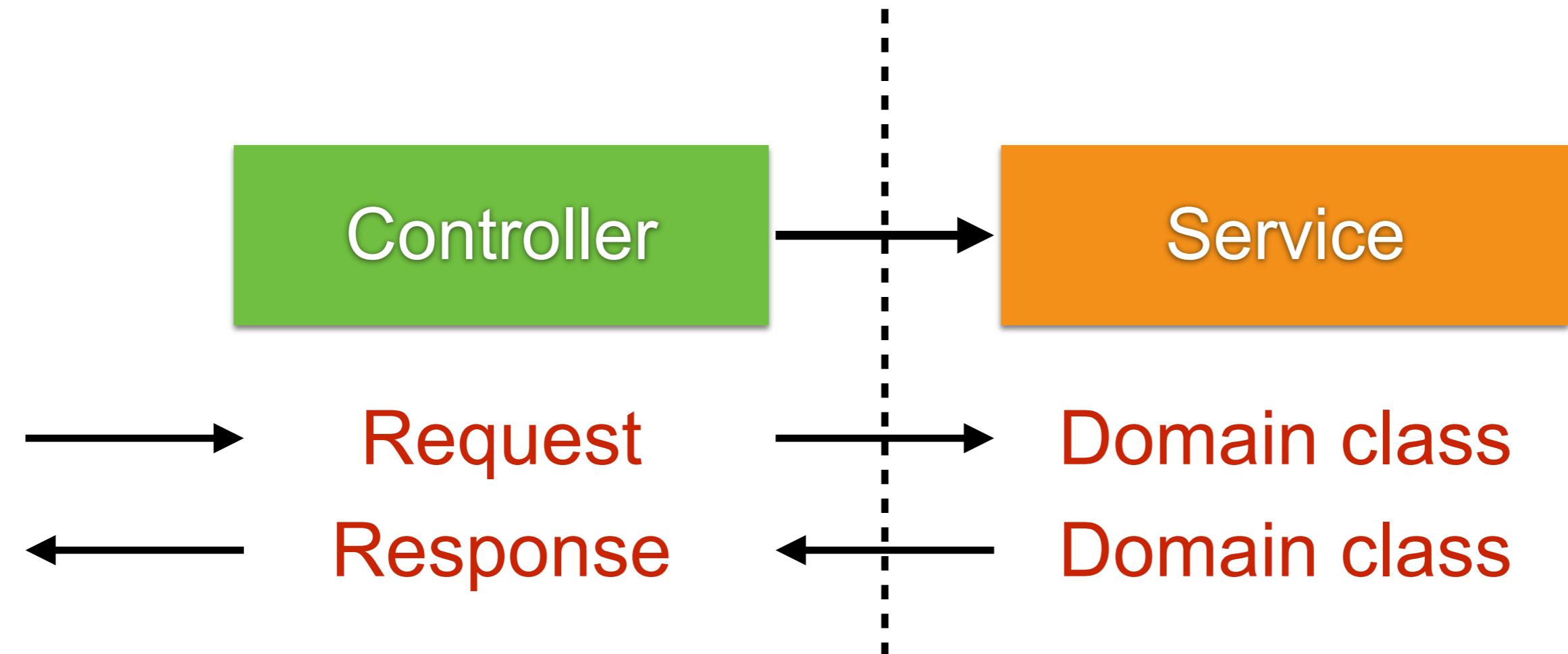
# Data Model for service ?



# Data Model ?

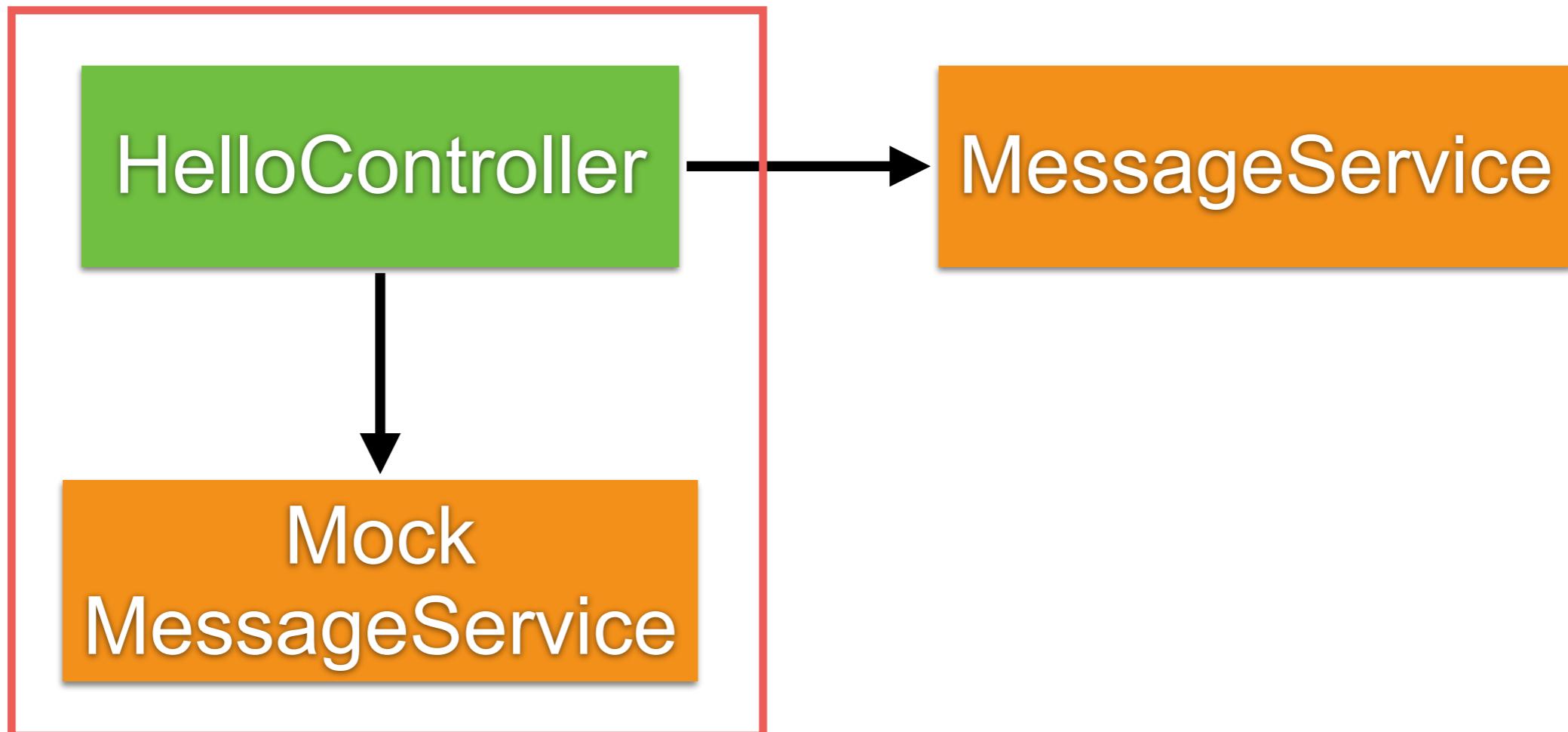


# Data Model ?



# Testing controller with service

Try to mocking service with Mockito



# Run all tests !!

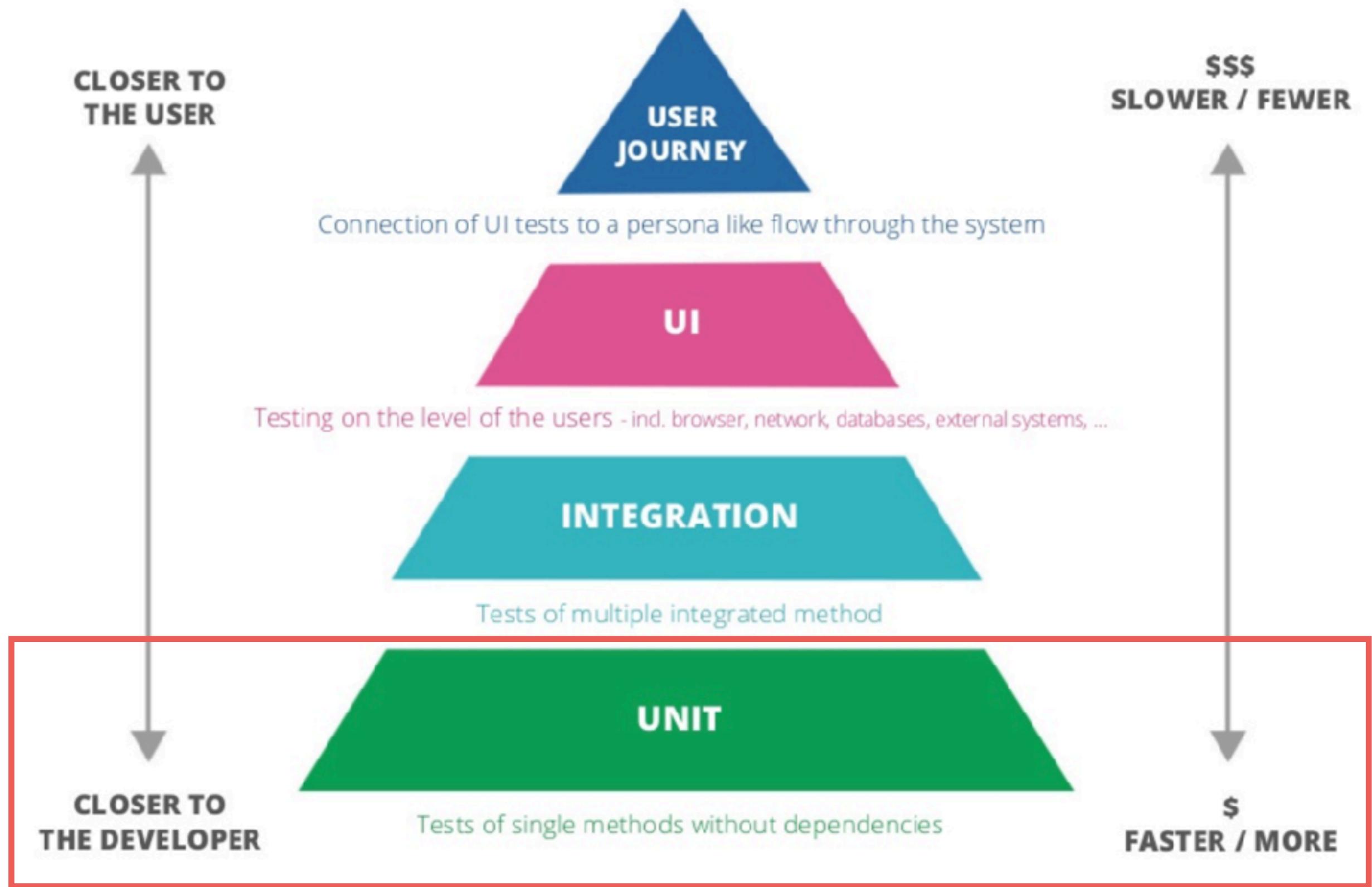
\$mvnw clean test

```
[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] -----
```

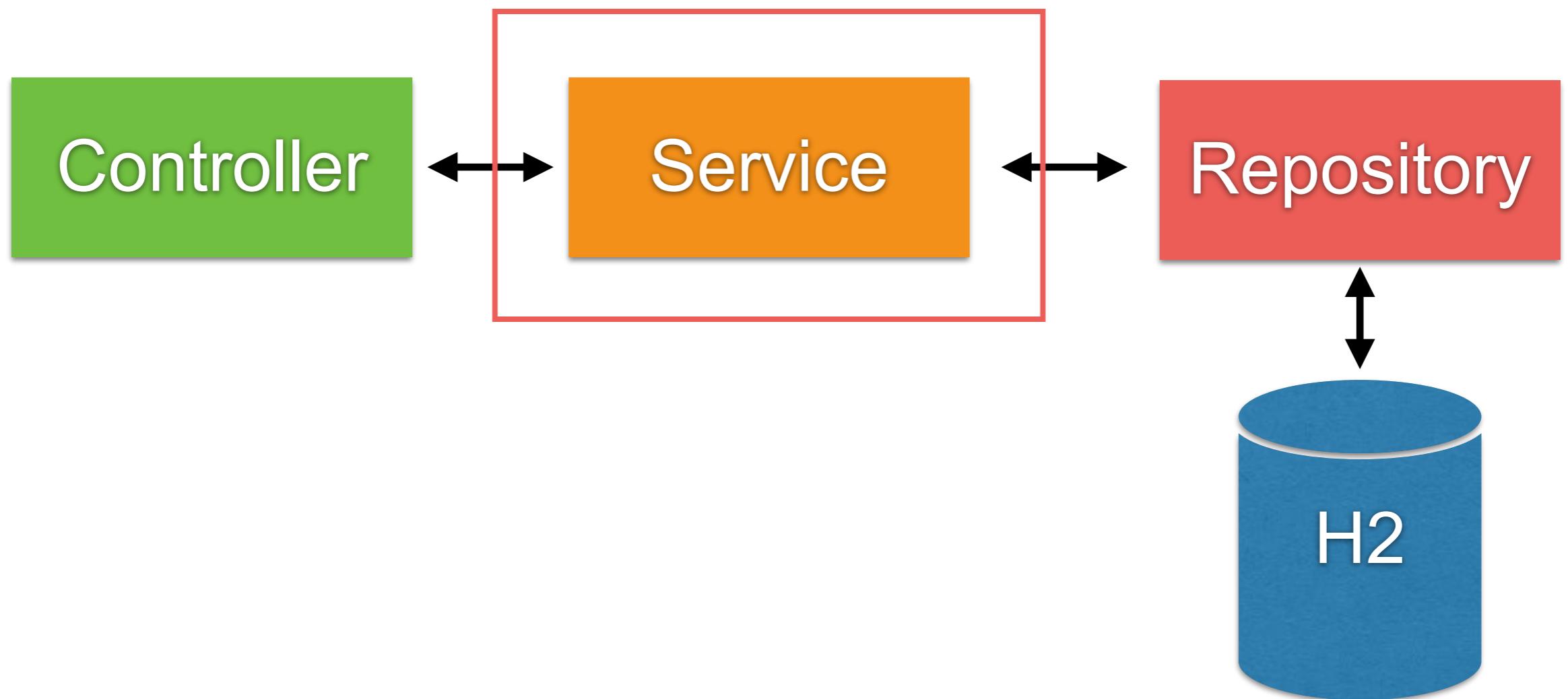


# **How to improve the speed of testing ?**





# 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);
```

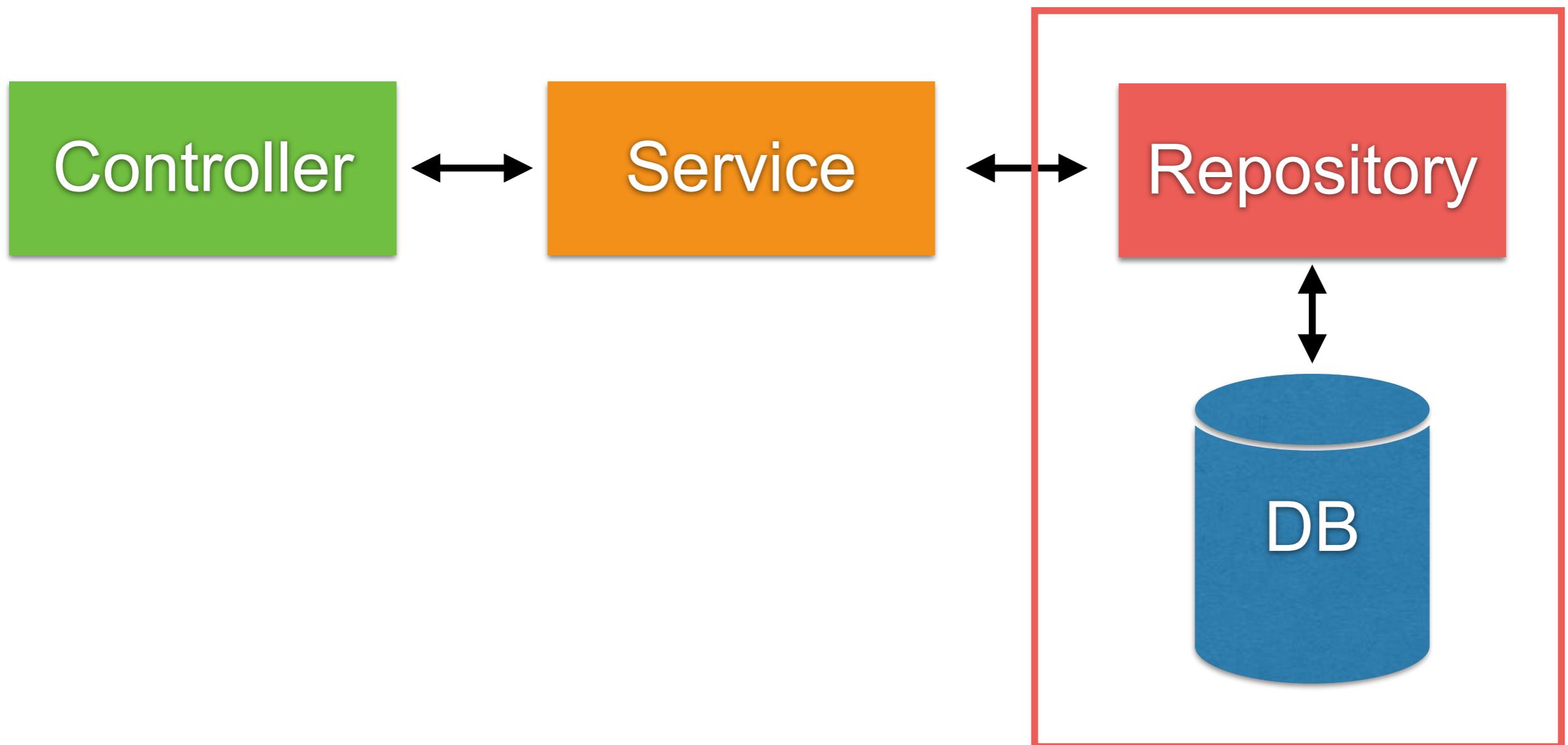
```
}
```



# 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 !!

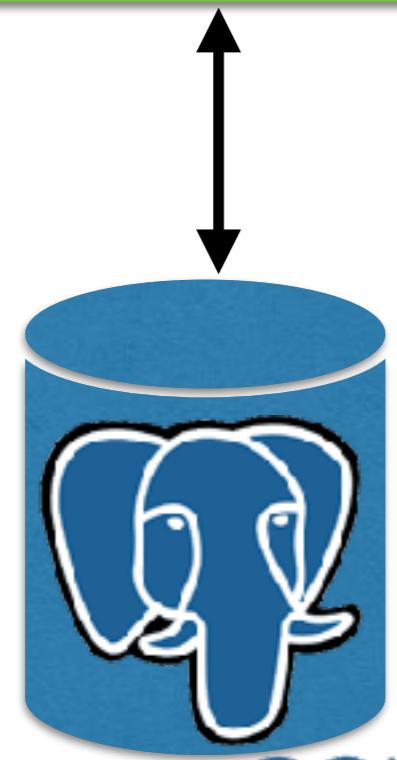
```
// 1. Load jdbc driver  
Class.forName("postgresql");  
// 2. Create connection  
Connection connection = DriverManager.getConnection("", "", "");  
// 3. Prepared Statement  
String sql = "SELECT * FROM TABLE WHERE name=?";  
PreparedStatement pStmt = connection.prepareStatement(sql);  
  
// 4.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

Manage by Framework



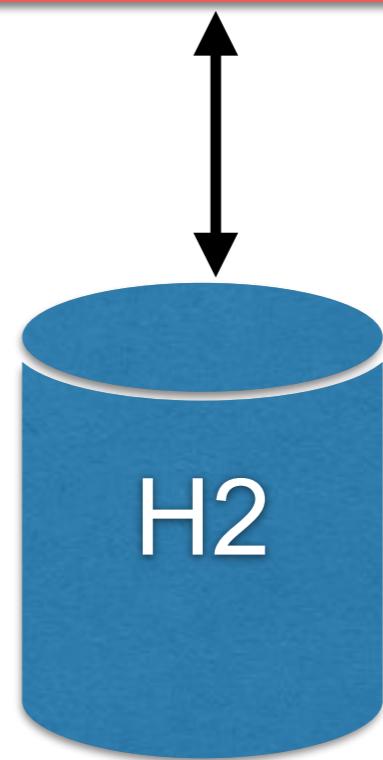
# Working with Database ?

Production



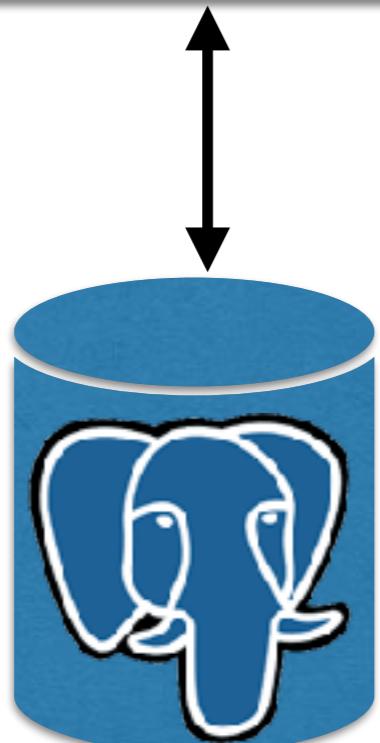
PostgreSQL

Testing



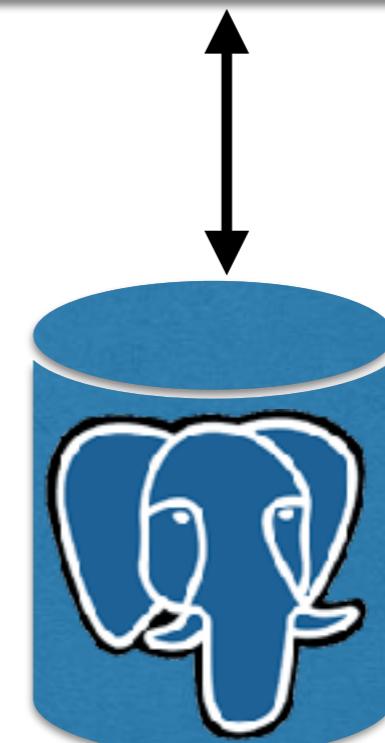
# Working with Database ?

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 ...

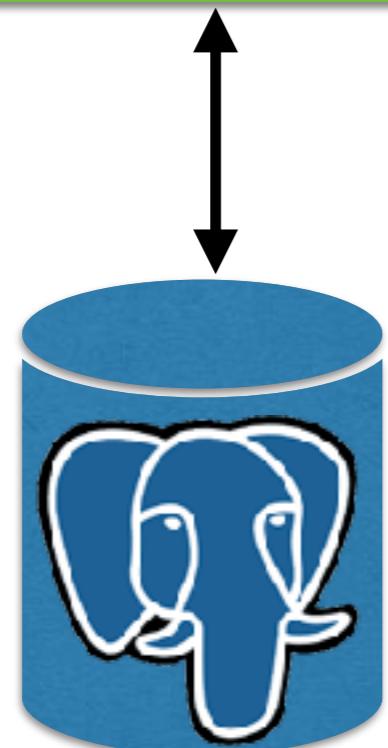


# Working with Spring Data JPA



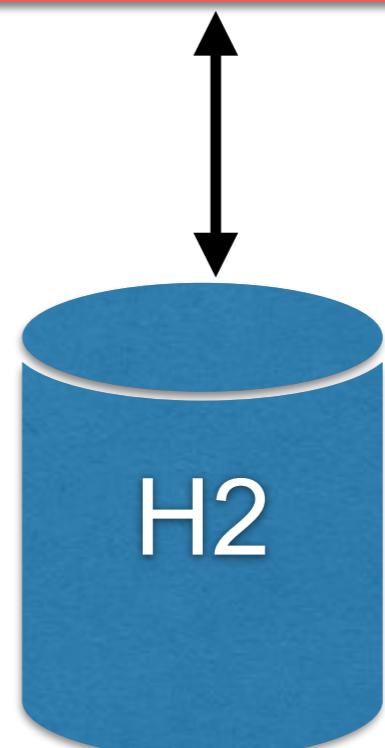
# Working with Database

Production



PostgreSQ<sub>L</sub>

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>

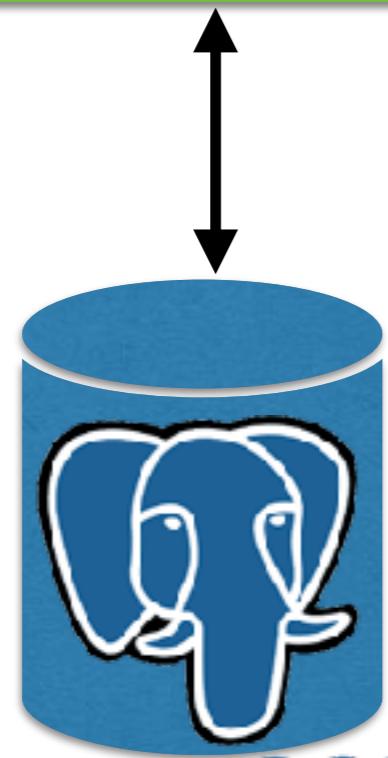
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>test</scope>
</dependency>

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



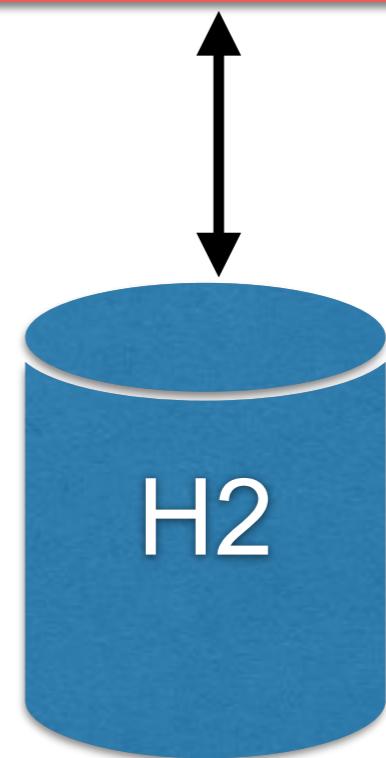
# Start in testing scope

Production

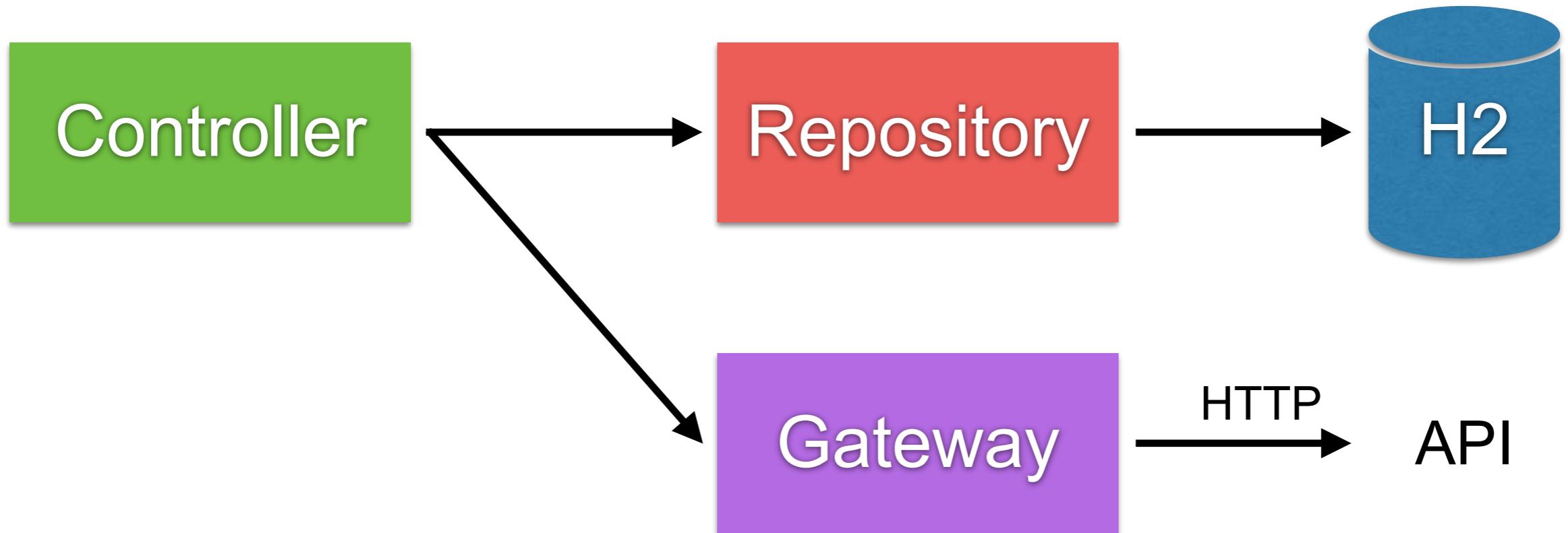


Postgre**SQ**L

Testing

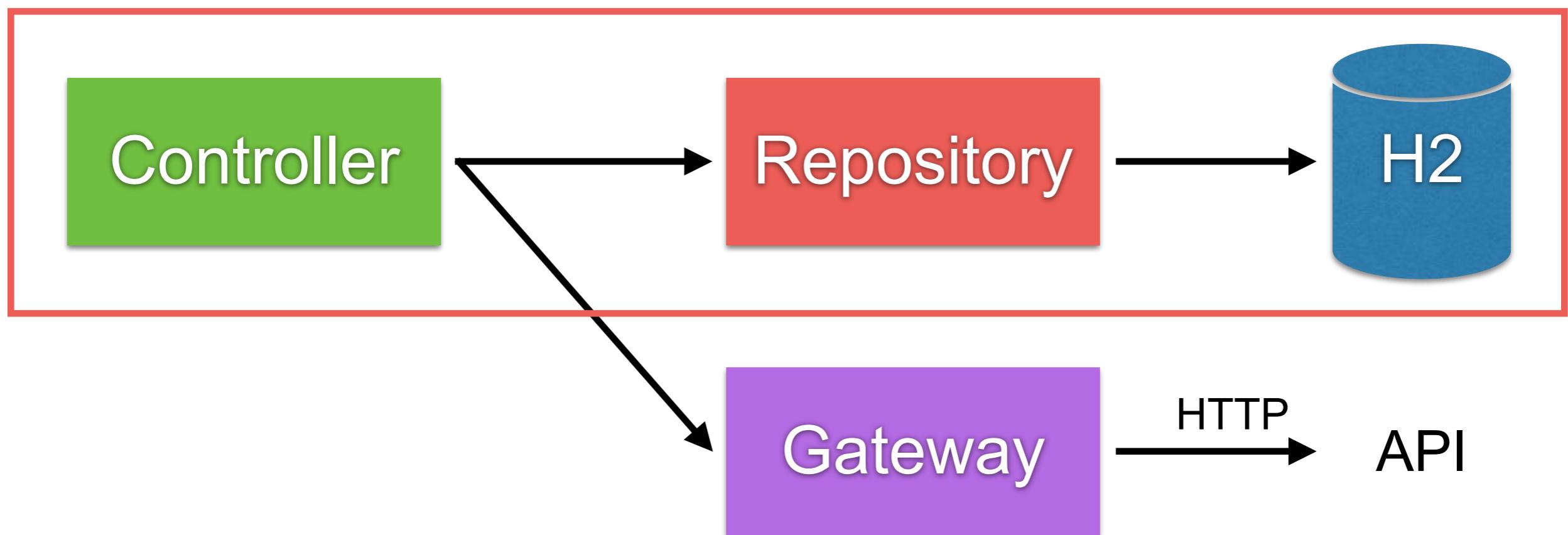


# Use cases



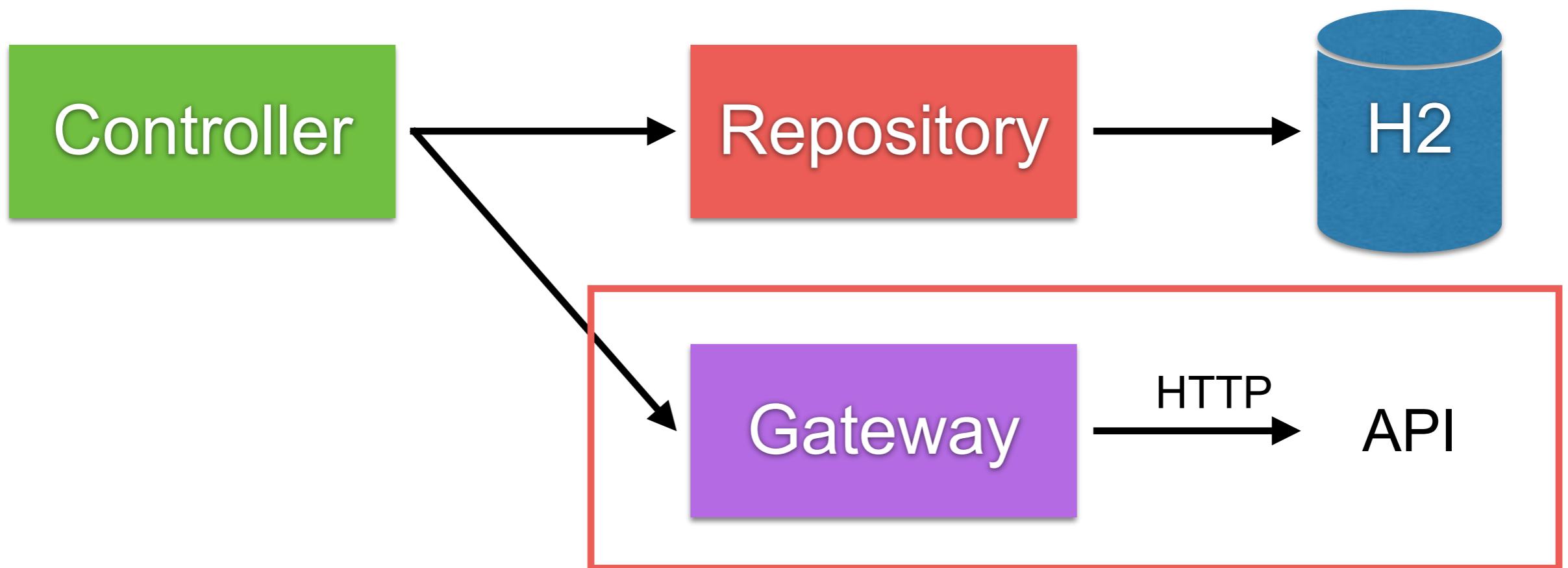
# Use case 1

## Working with repository



# Use case 2

## Working with API



# Use case 1



# Use case 1

Working with repository

3. HelloController



2. PersonRepository



Controller

Repository

H2



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=?*

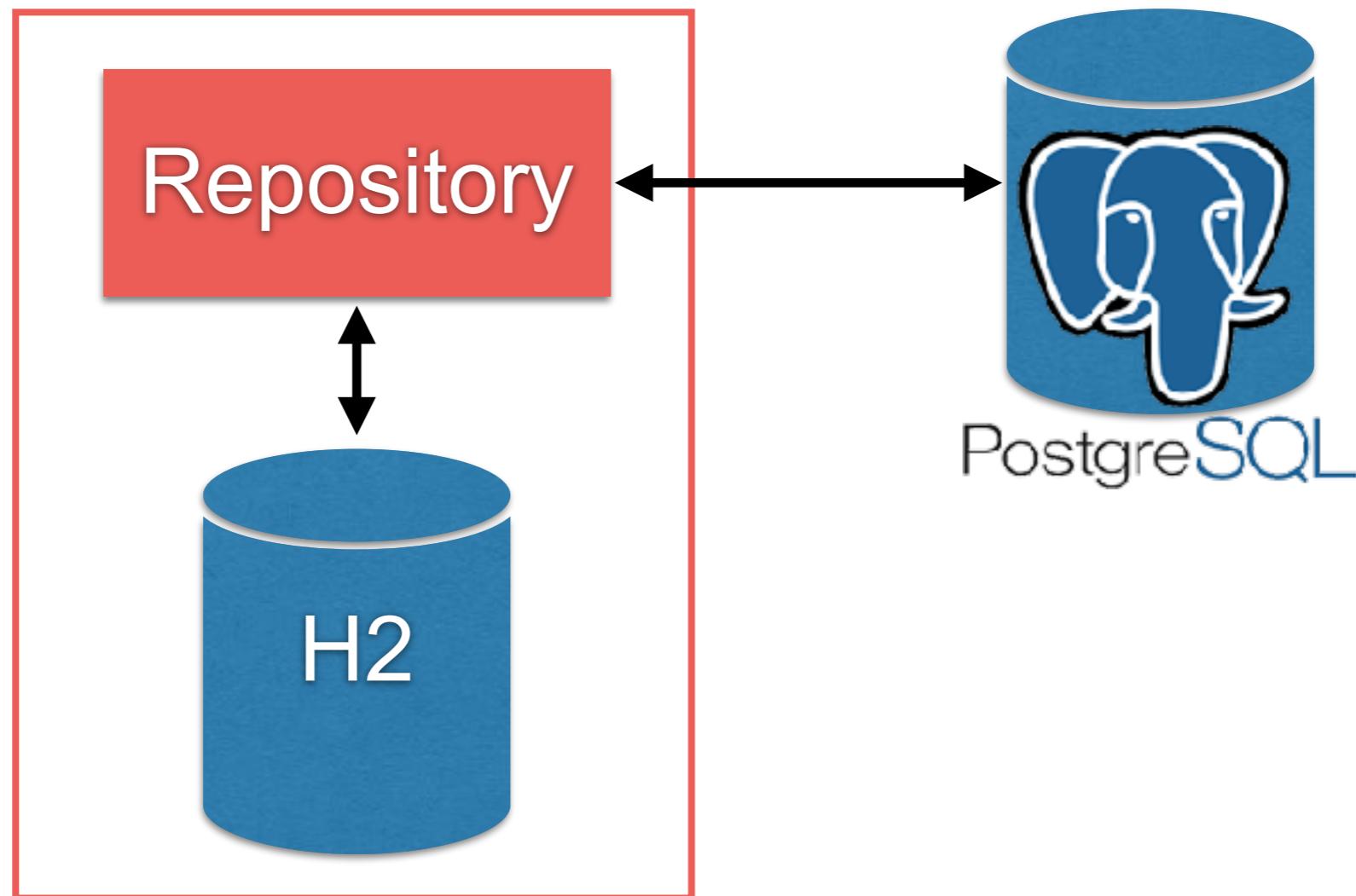


# How to testing repository ?



# 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

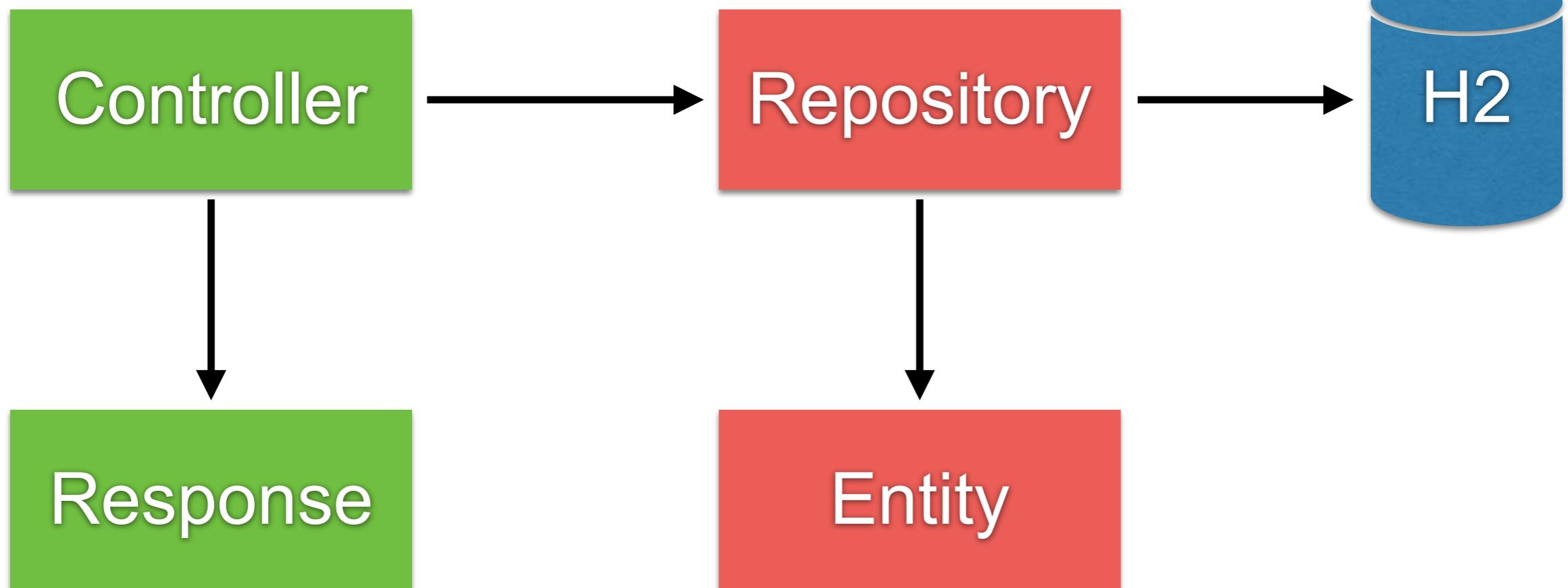
Hibernate: drop sequence if exists hibernate\_sequence



# Use case 1

Integrate repository with controller

3. HelloController      2. PersonRepository



1. Person

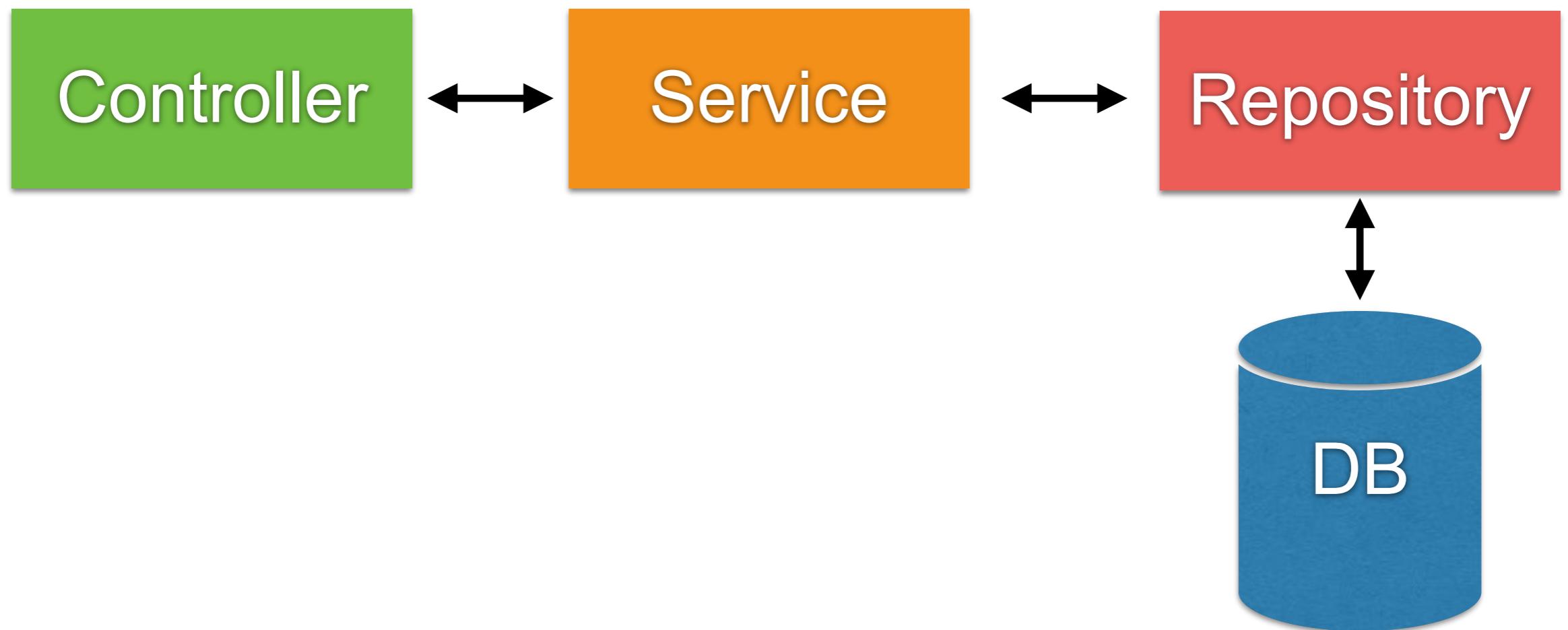
4. PersonResponse



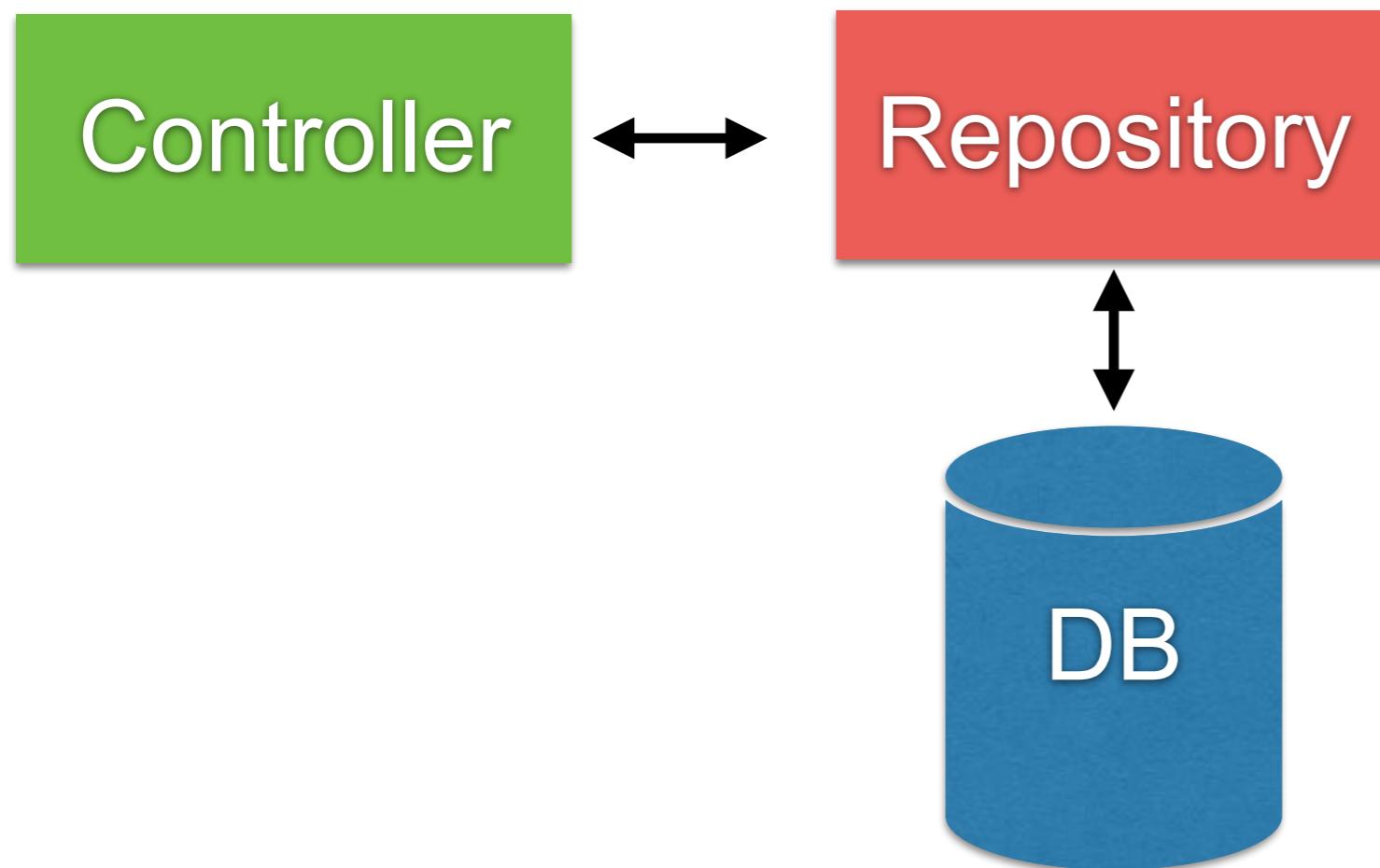
# Integrate repository with service/controller



# 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

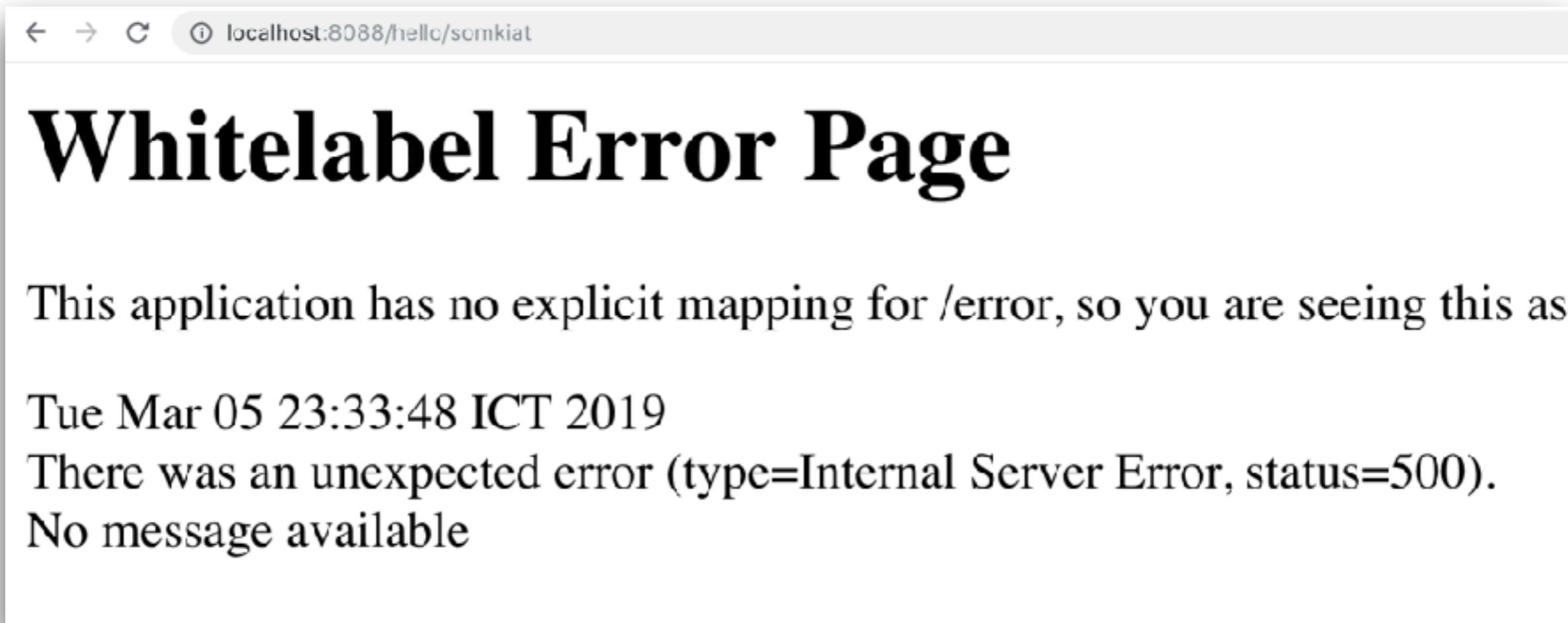
```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <scope>runtime</scope>
</dependency>

<!-- <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**

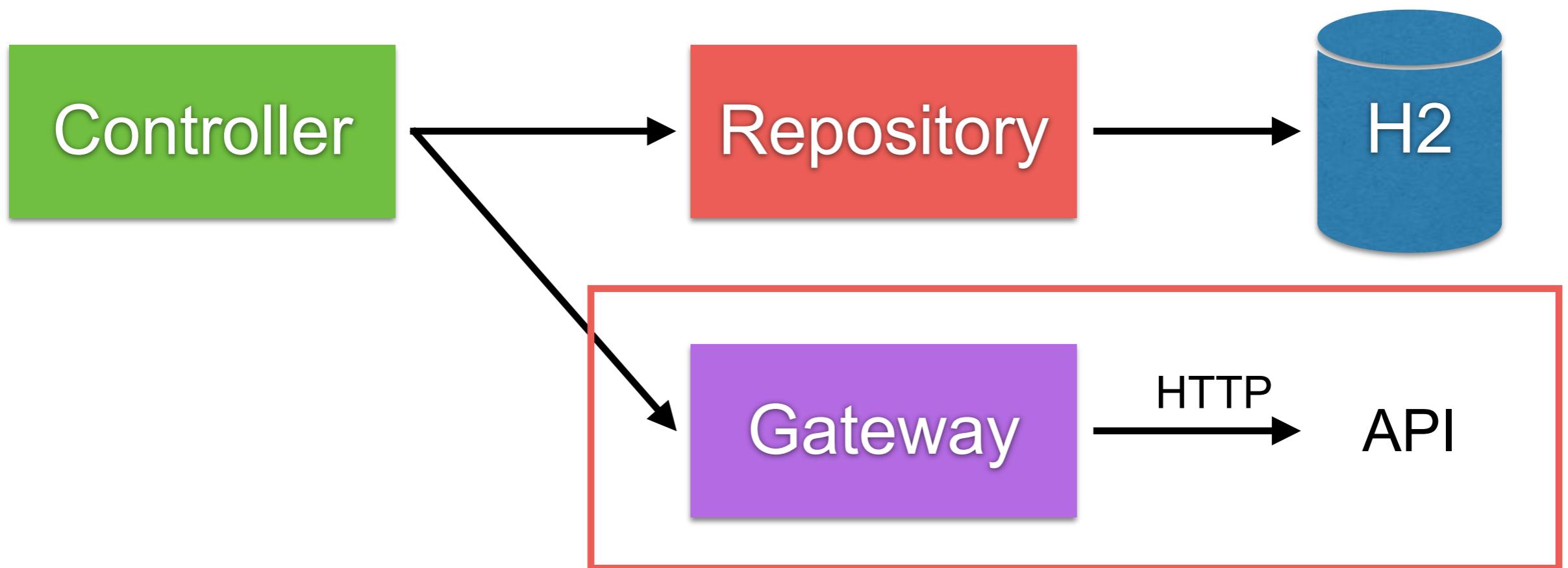


# Use case 2



# Use case 2

## Working with API



# JSON Place Holder

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

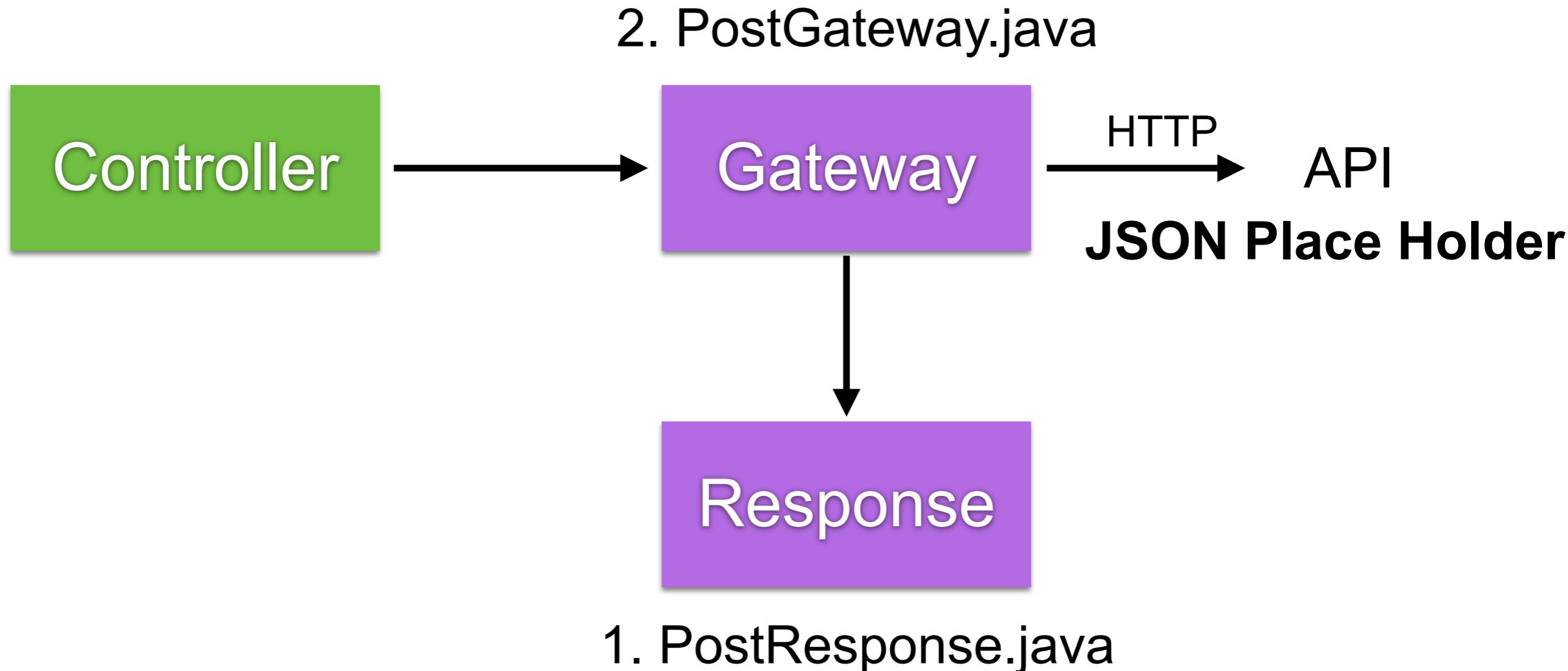
The screenshot shows a web-based API testing interface for JSONPlaceholder. At the top, the title "JSONPlaceholder" is displayed in large, bold letters. Below it, a subtitle reads "Fake Online REST API for Testing and Prototyping" and "Powered by [JSON Server + LowDB](#)". A code snippet in a light gray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



# 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
```



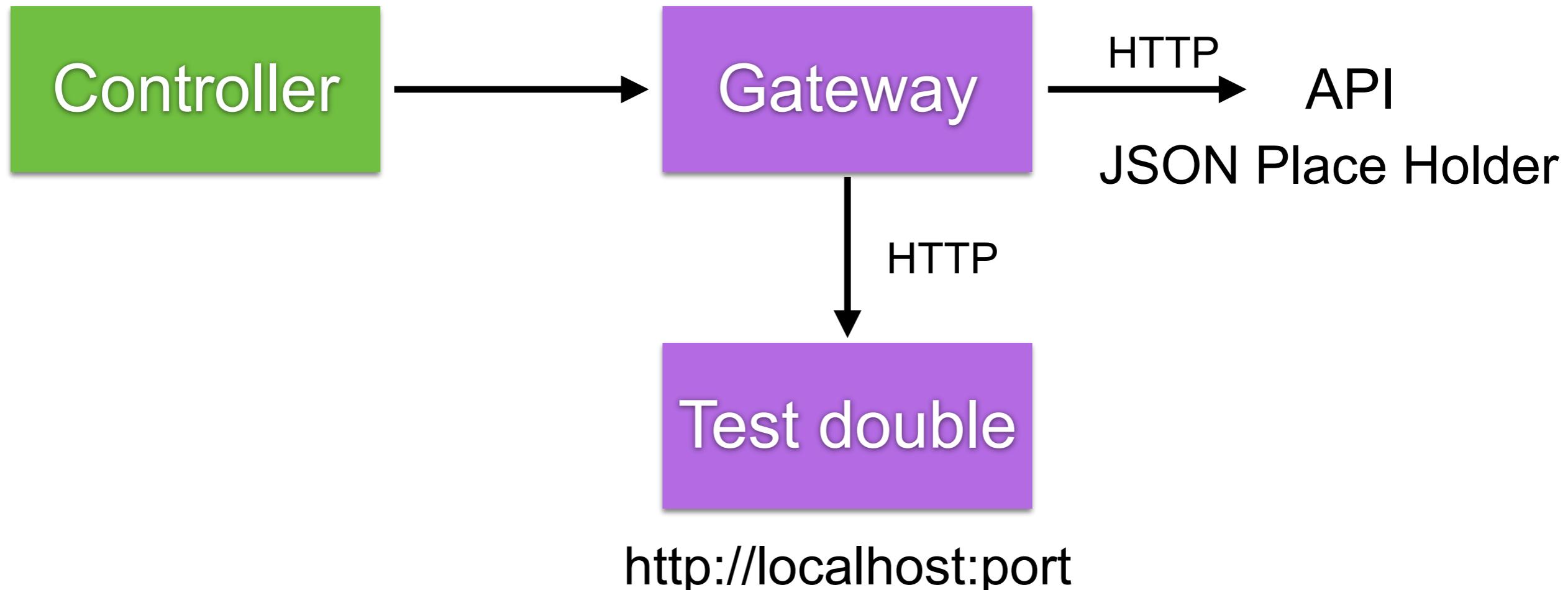
# 2. Create PostGateway class #2

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

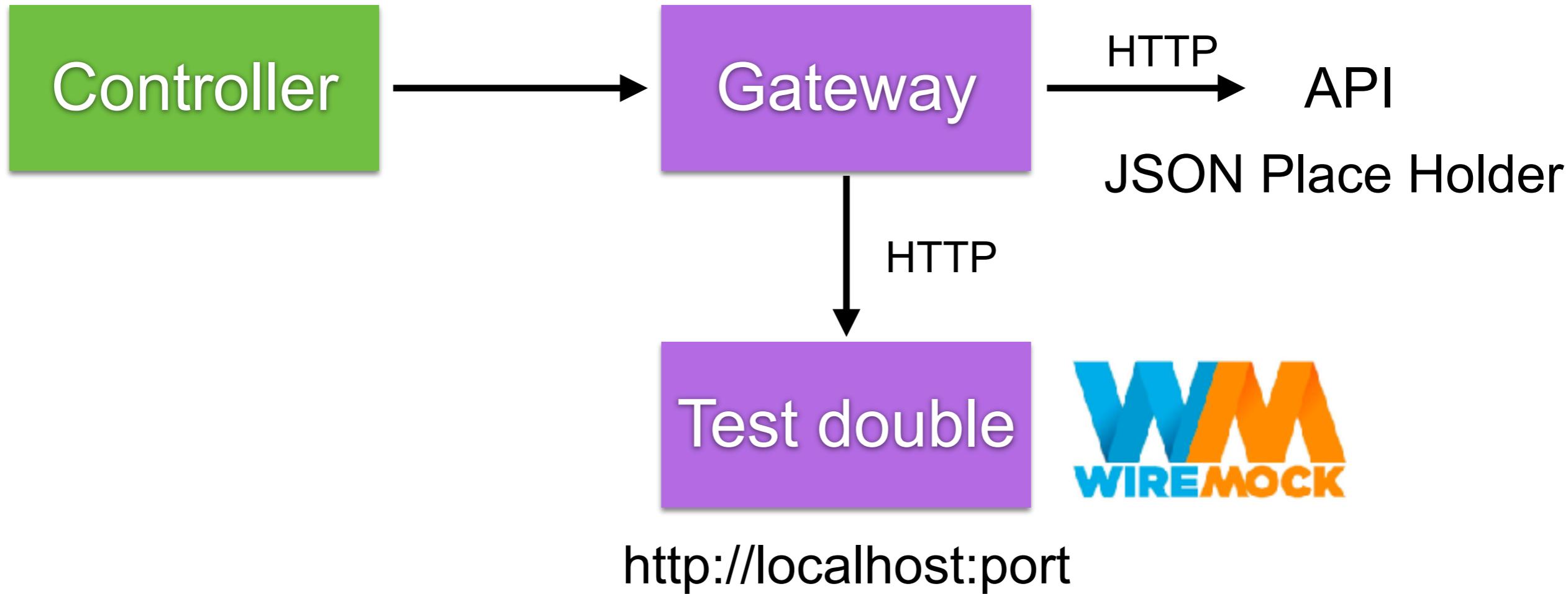
Unit testing with Mockito

**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 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 with category

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



# 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



# Why use Aspects ?

Reduce code duplication

DRY (Don't Repeat Yourself)

Maintain application logic



# Parts of Spring Aspect

Join Point  
Pointcut  
Advice  
Aspect



# Working with transaction



# Working with logging



# Are you too busy to improve?



# Thank you Q/A

