

Spring Framework



Introduction

Introduction

Spring Framework

Spring Boot

Spring Data

Spring Testing

Spring Framework

- Introduction to Spring Framework
- Deep dive into Spring Framework
- Inversion of Control (IoC)
- Dependency Injection
- Beans and BeanFactory

Spring Boot

- RESTful API
- Building RESTful API with Spring Boot
- Spring Boot with Spring Data JPA
- Testing with Spring Boot

Introduction to Spring Framework

What is Spring Framework ?

Application development framework
for Java, Kotlin and Groovy

What is Spring Framework ?

Easy and safe

High productivity

Integrate with new technologies

Testable application

Speed up delivery

Spring projects

Spring Projects

Modular by design

Spring Framework

Spring Boot

Spring Cloud Data Flow

Spring Cloud

Spring Data

Spring Integration

Spring Batch

Spring Security

Spring AMQP

Spring LDAP

Spring WebFlow

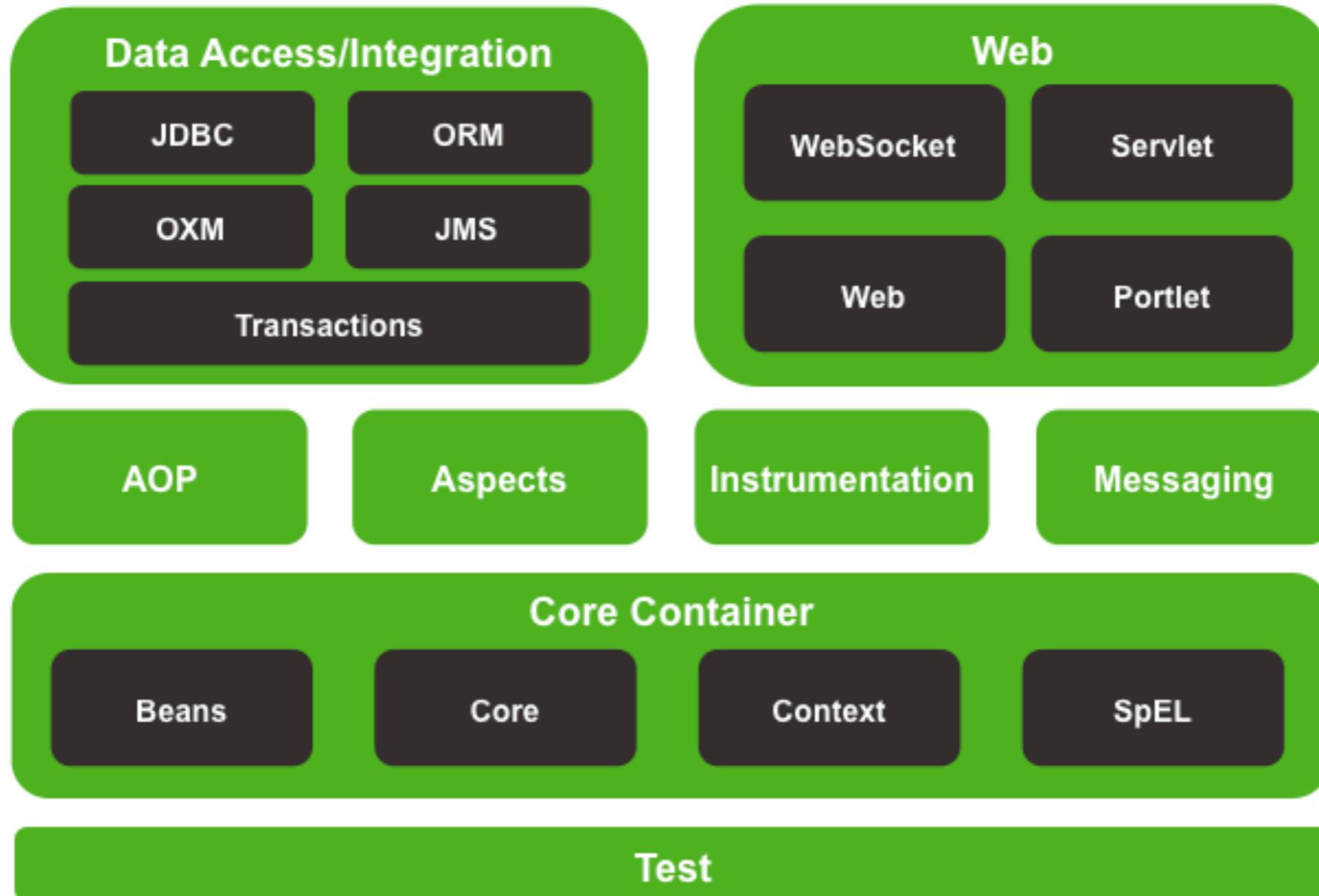
Spring REST Doc

<https://spring.io/projects>

Overview of Spring Framework



Spring Framework Runtime



Deep Dive Spring Framework

Core container

Beans

Core

Container or Spring Container

Beans

Beans

Unit of work

Spring container manages one or more beans

Beans are created with configuration

Bean Definitions

Package-qualified class name

Bean behavioral (scope, lifecycle, callback)

Reference to other beans

Other configuration setting to create new object

Bean Definitions

Property	Section
Class	Instantiating beans
Name	Naming beans
Scope	Beans scopes
Constructor arguments	Dependency Injection
Properties	Dependency Injection
Lazy initialization mode	Lazy-initialized beans

<https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-definition>

Bean Scopes

Scope	Description
singleton	Single instance for each container
prototype	Single bean definition to any number of object instances.
request	HTTP request
session	HTTP session
application	ServletContext

<https://docs.spring.io/spring/docs/current/spring-framework-reference/core.html#beans-factory-scopes>

How to create bean ?

BeanFactory ?

Interface defines basic functionality for Spring container

Factory design pattern

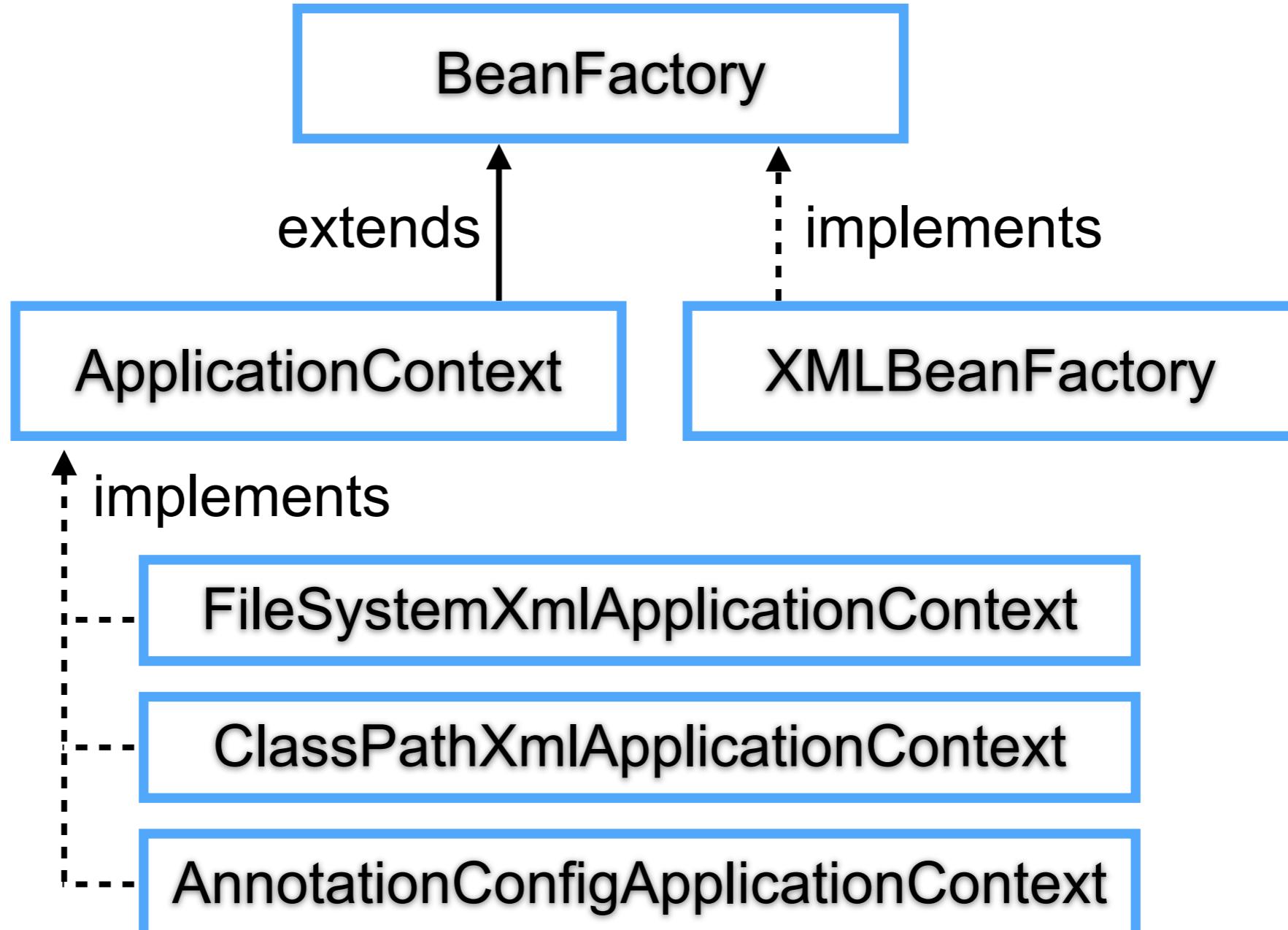
Load beans from configuration source

Instantiated the bean when requested

Wire dependencies and properties for beans

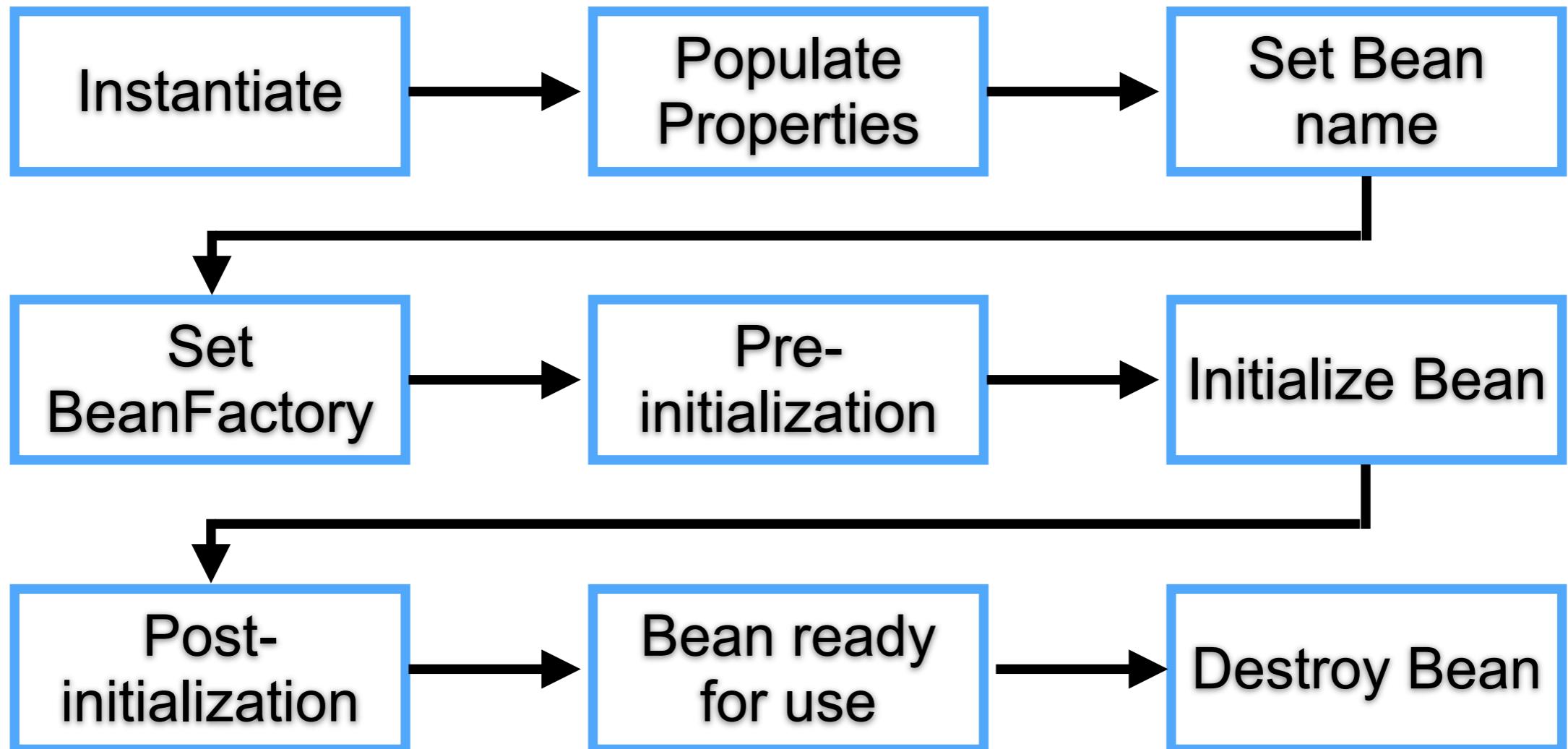
Manage the bean lifecycle

BeanFactory ?



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

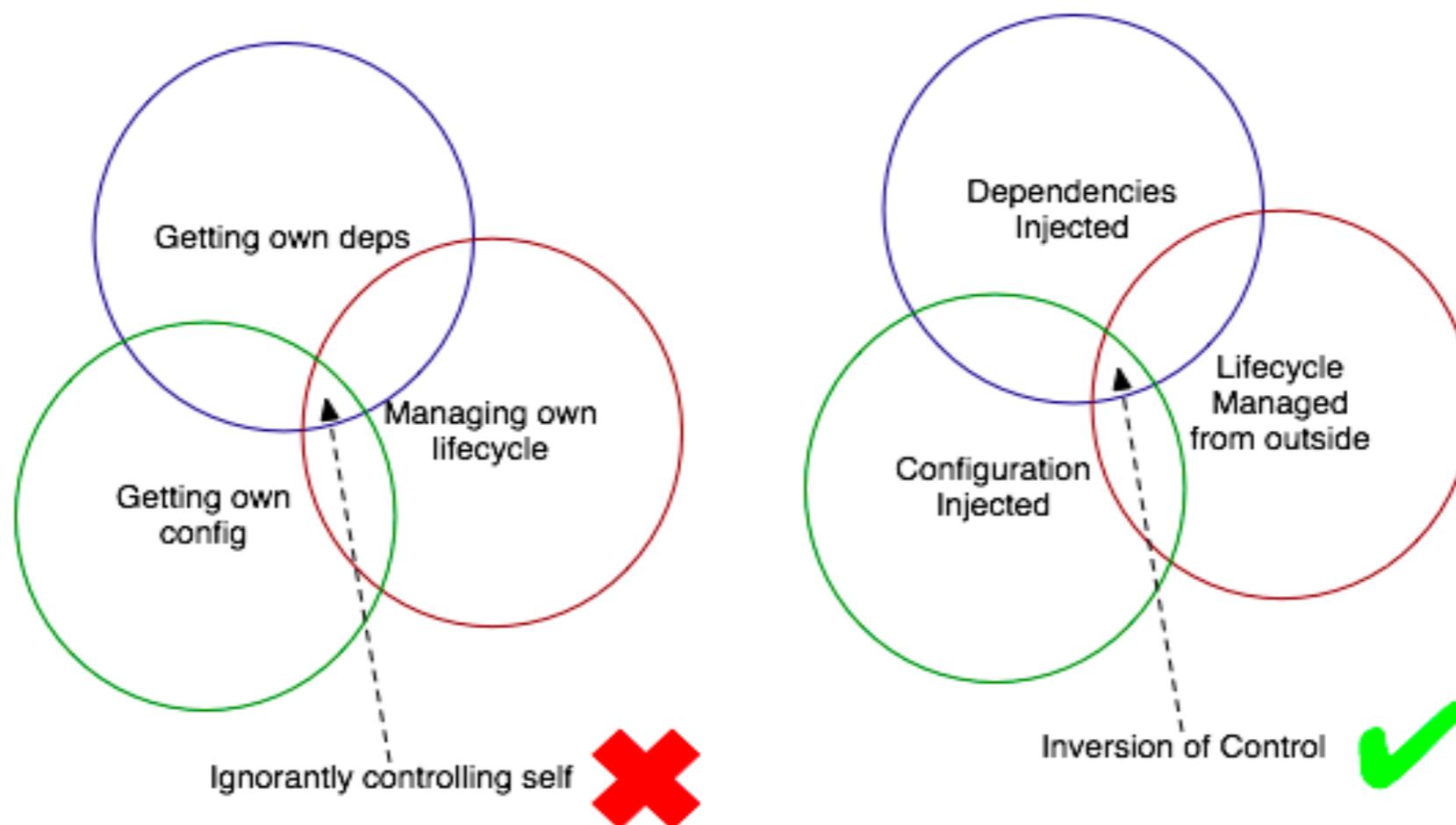
Lifecycle of BeanFactory



Cores

Core

Provide the fundamental parts of framework
Including IoC and Dependency Injection



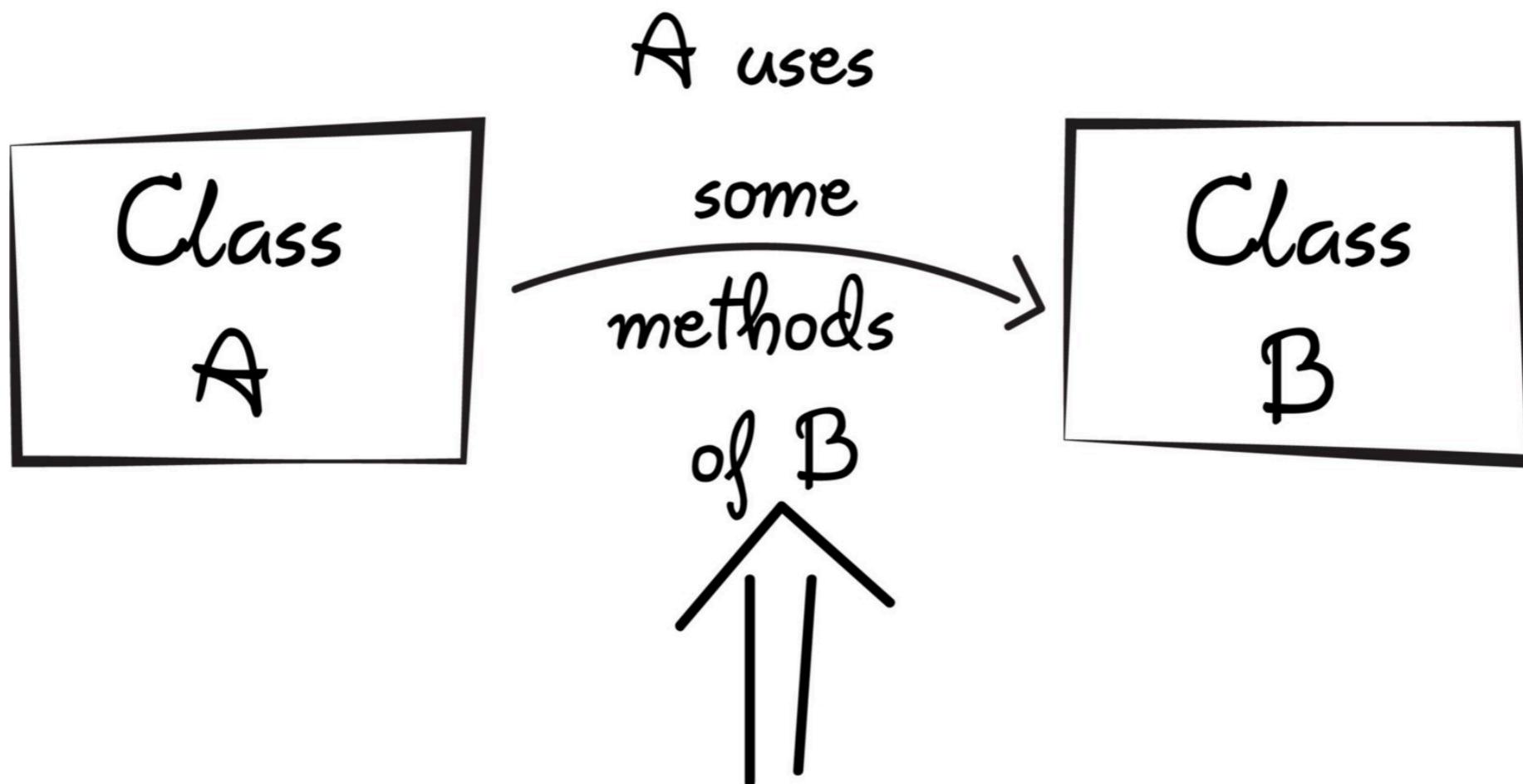
<https://www.martinfowler.com/articles/injection.html>

Inversion of Control (IoC)

Concept in application development

Don't call me, I will call you

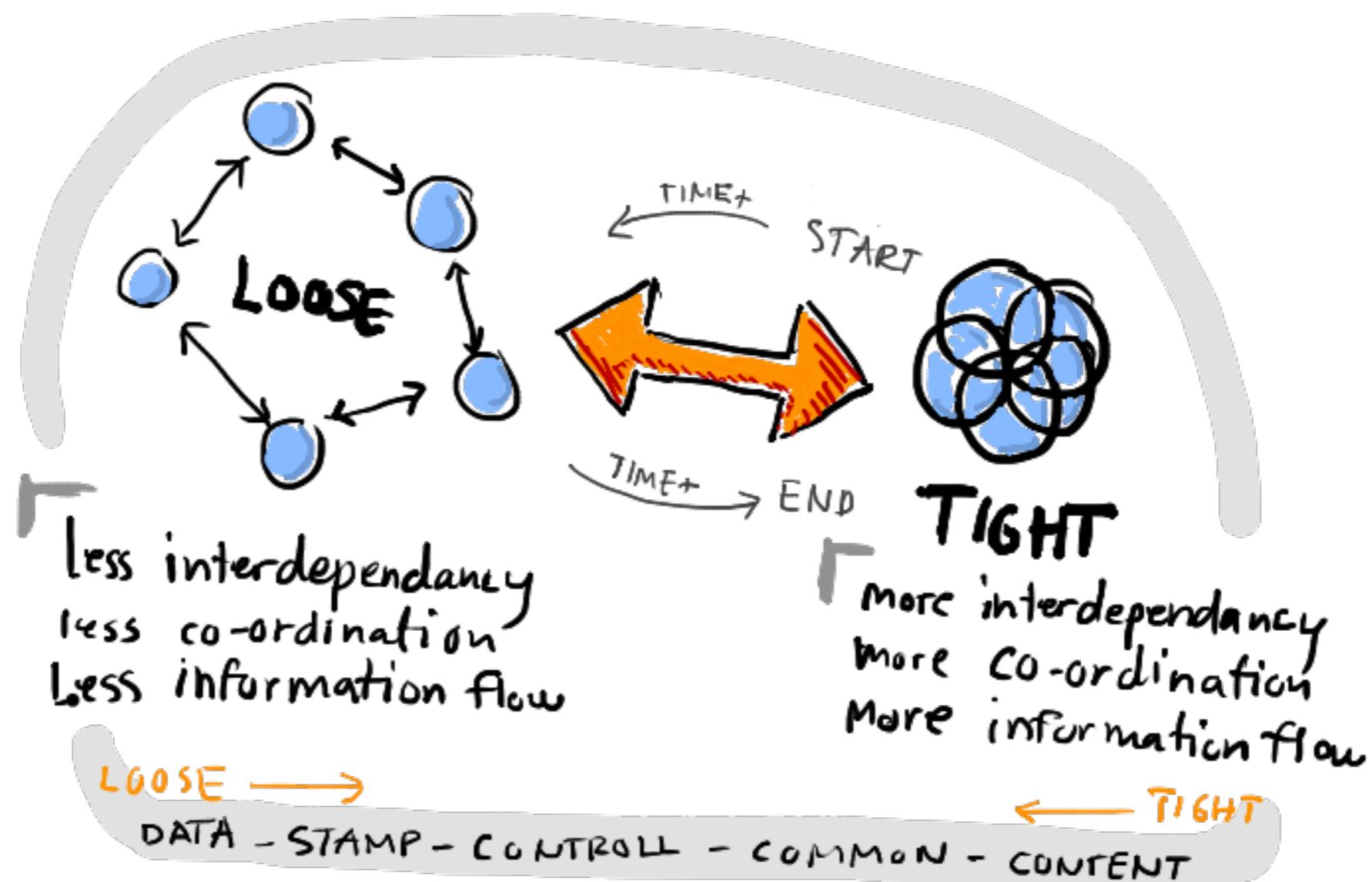
Dependency Injection



Its a dependency

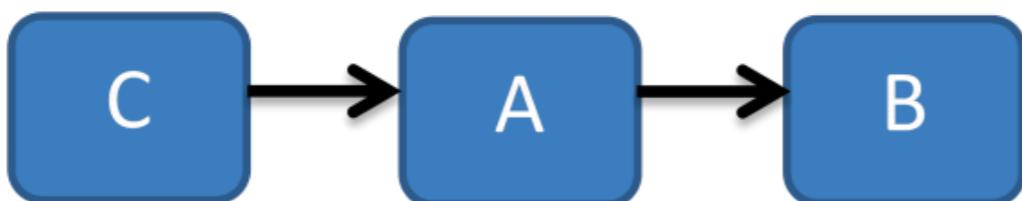
Understanding Dependency Injection

Tight coupling
Loose coupling

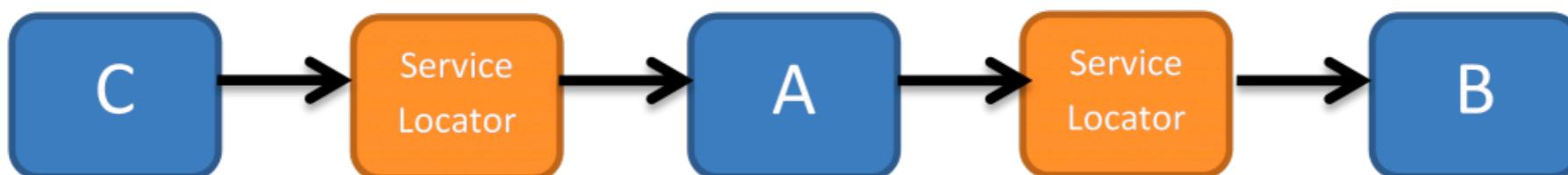


Dependency Injection

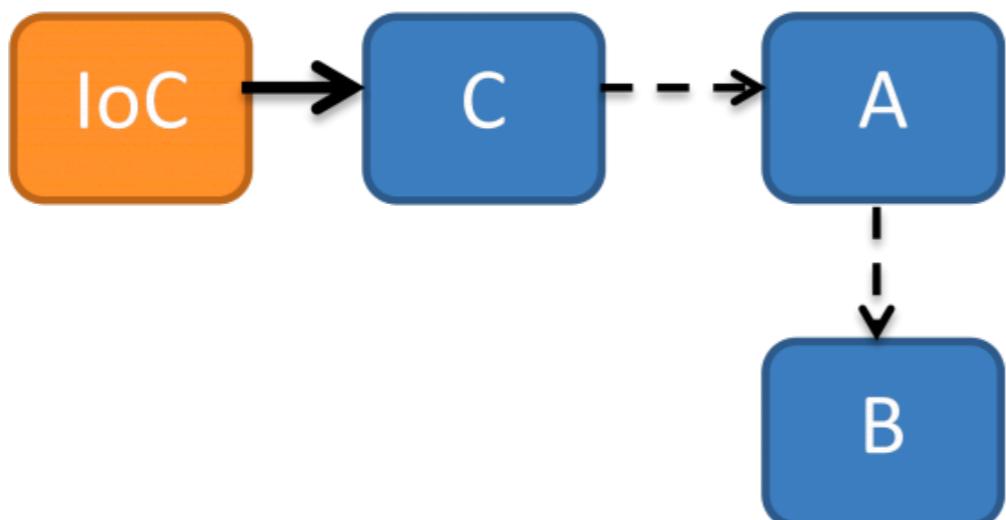
Class Dependencies



Service Location / Active Calling



IoC / DI / Auto-Wiring / Passive Calling



Types of Dependency Injection

Constructor injection

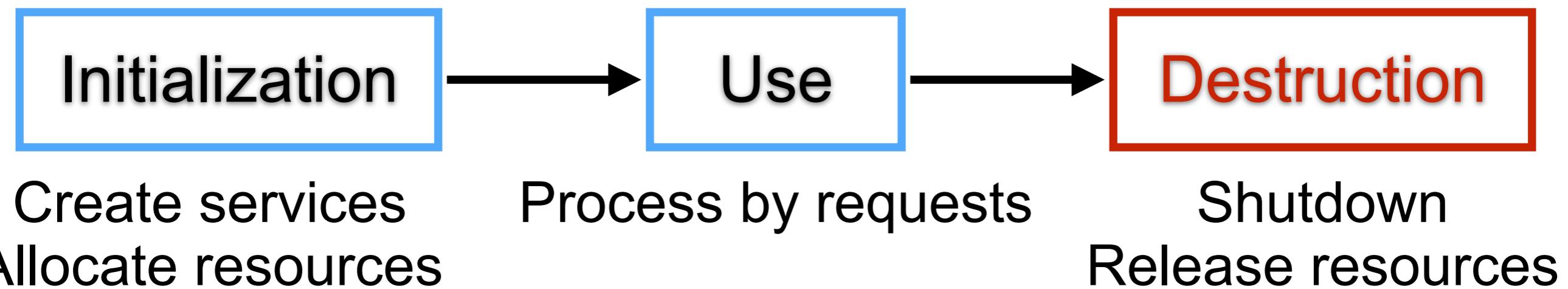
Property/Setter injection

Method injection

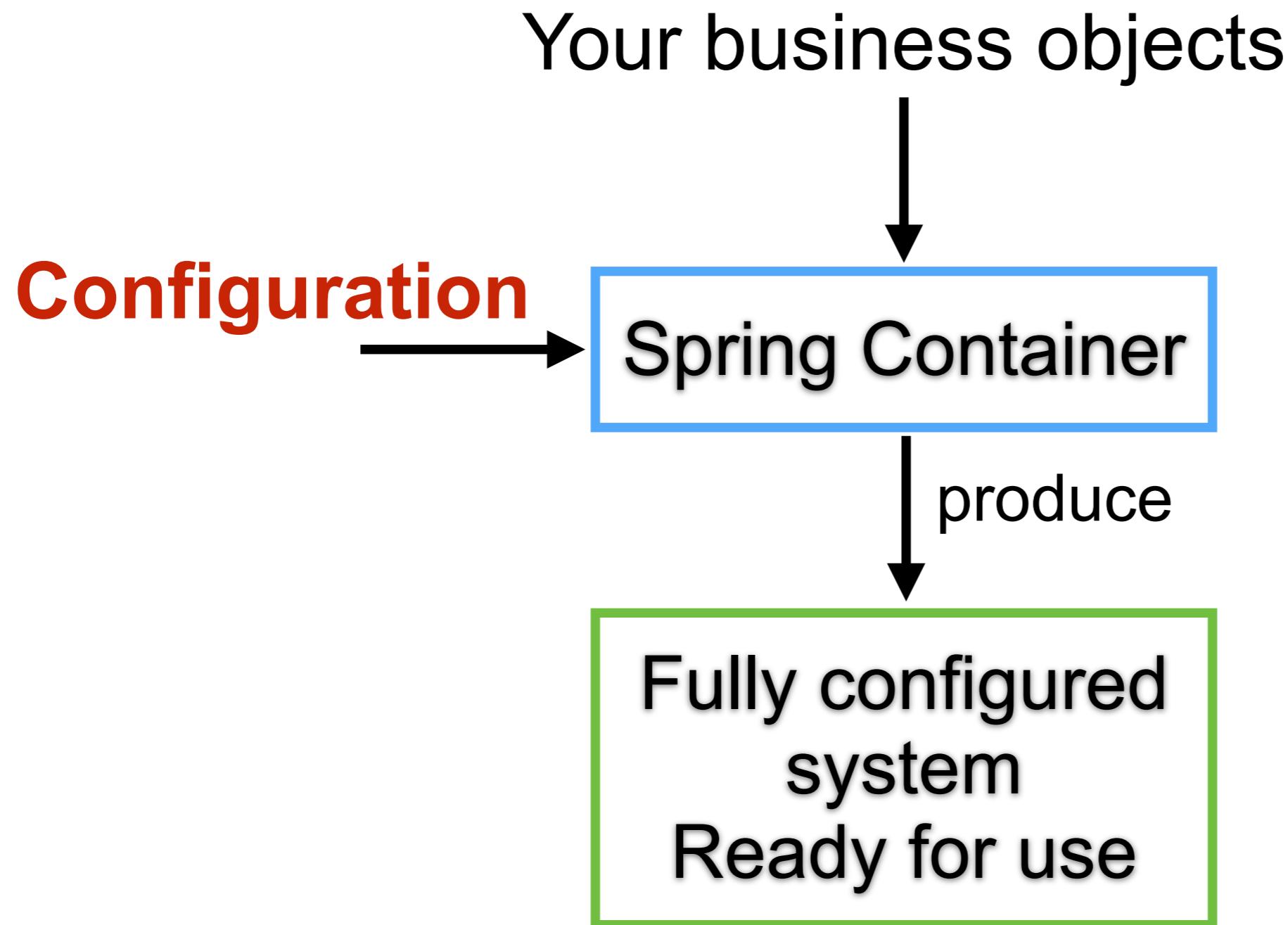
Interface injection

Container

Application Lifecycle



Spring IoC container



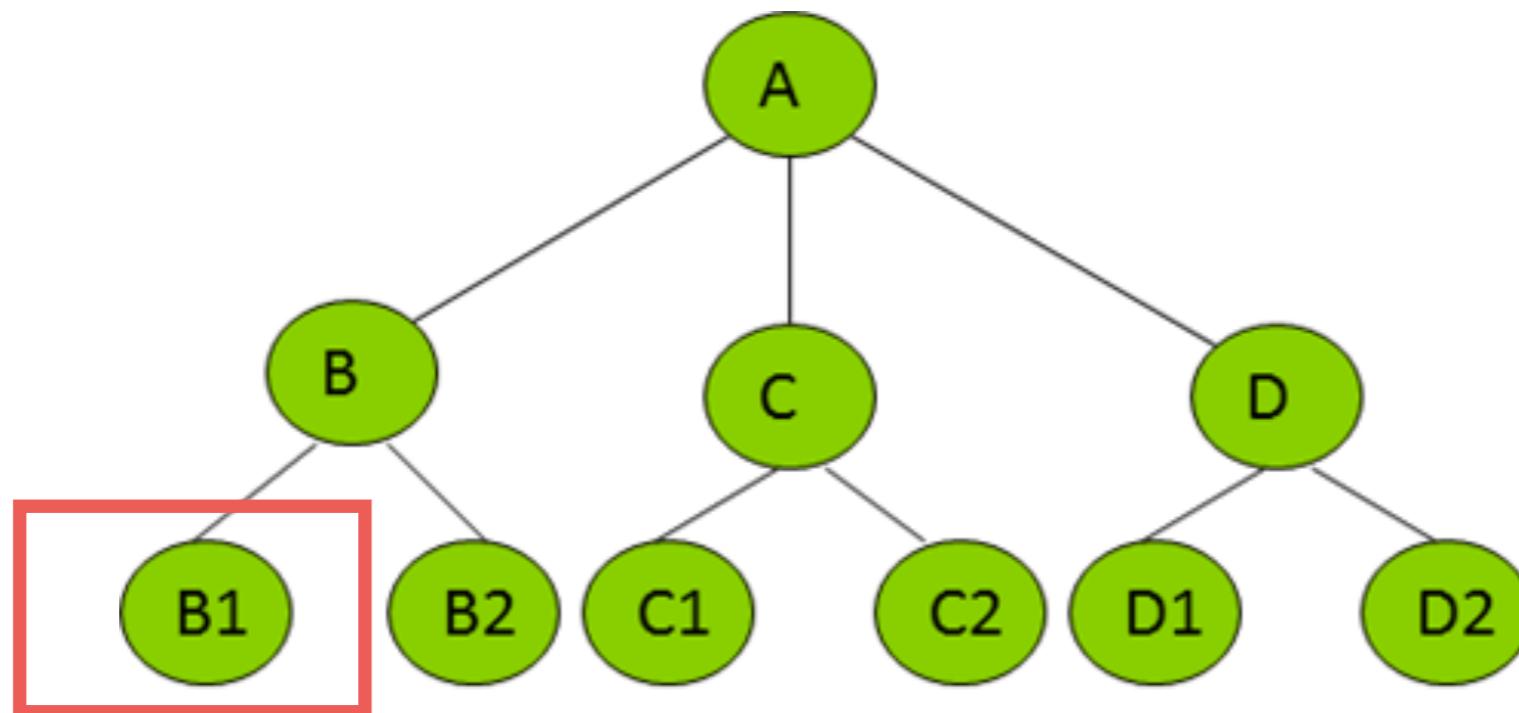
Configuration

From XML file

Annotation-based configuration (2.5)

Java-based configuration (3.0)

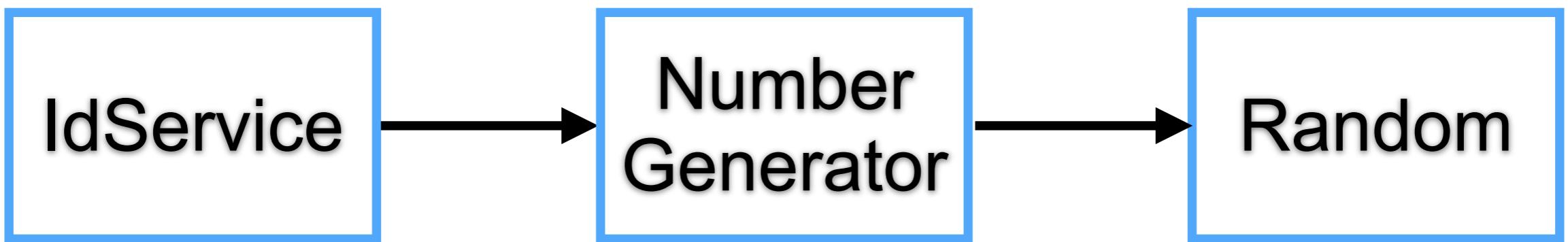
Spring IoC container



Bean B1

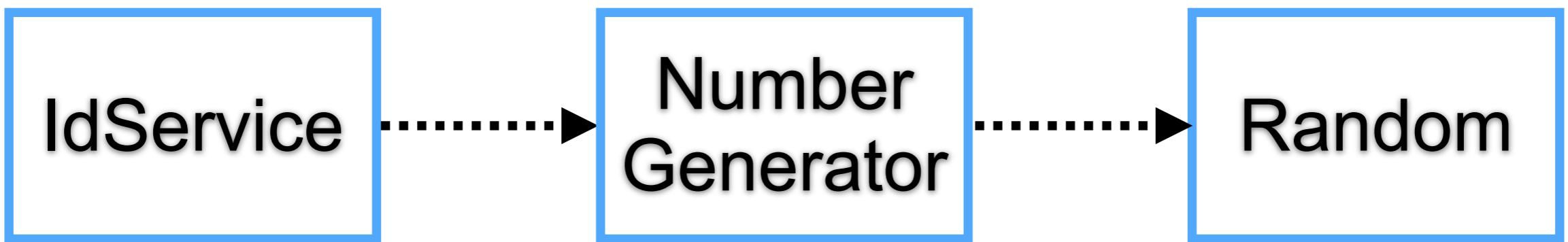
Demo IoC and DI with Java

Structure



Demo IoC and DI with Spring

Structure



Back to Spring Framework

Let's start Spring Framework

Create new Project

Use spring initializr

SPRING INITIALIZR bootstrap your application now

Generate a with and Spring Boot

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

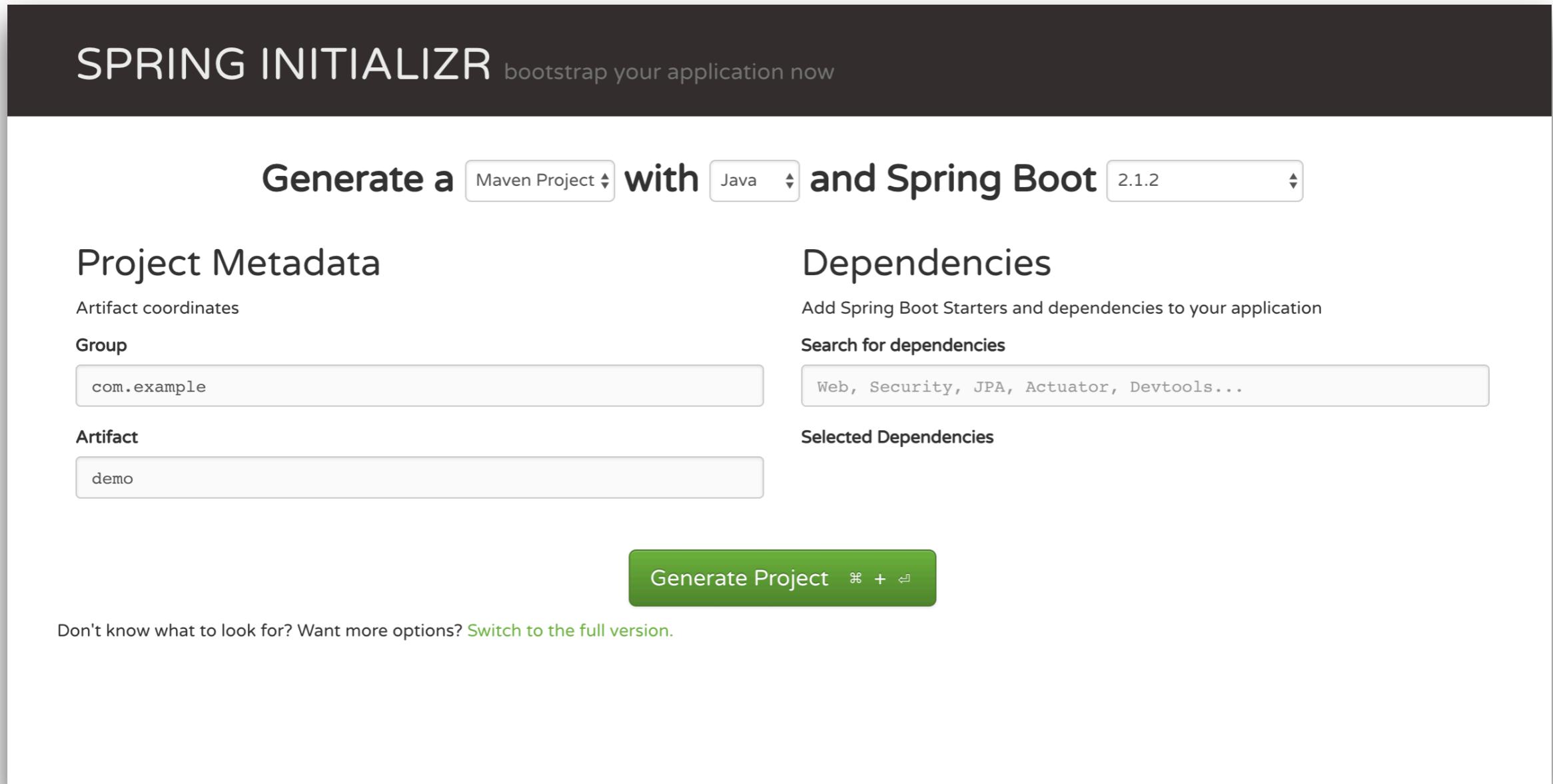
Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

Generate Project ⌘ + ↵

Don't know what to look for? Want more options? [Switch to the full version.](#)



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

Using Spring to manage dependencies

`@Component`

`@Autowired`

Constructor and Setter injection

`@Primary`

`@Qualifier`

Scope of beans

Singleton
Prototype

What are difference of ...

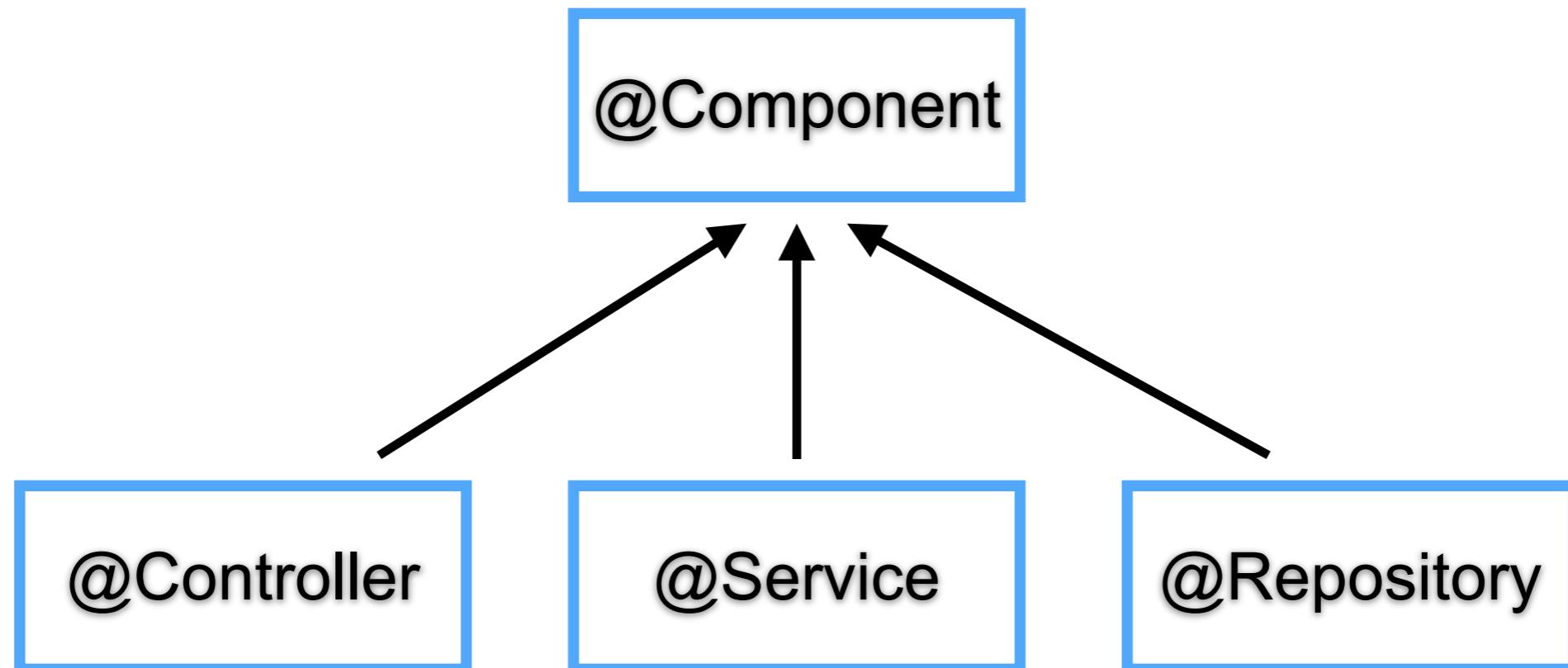
`@Component`

`@Controller`

`@Service`

`@Repository`

What are difference of ...



Layer of application



What are difference of ...

@Component

Generic stereotype for any component or bean

@Controller

Stereotype for the presentation layer (Spring MVC)

@Service

Stereotype for the service layer

@Repository

Stereotype for the persistence layer

Why Spring is popular ?

Enable testable code

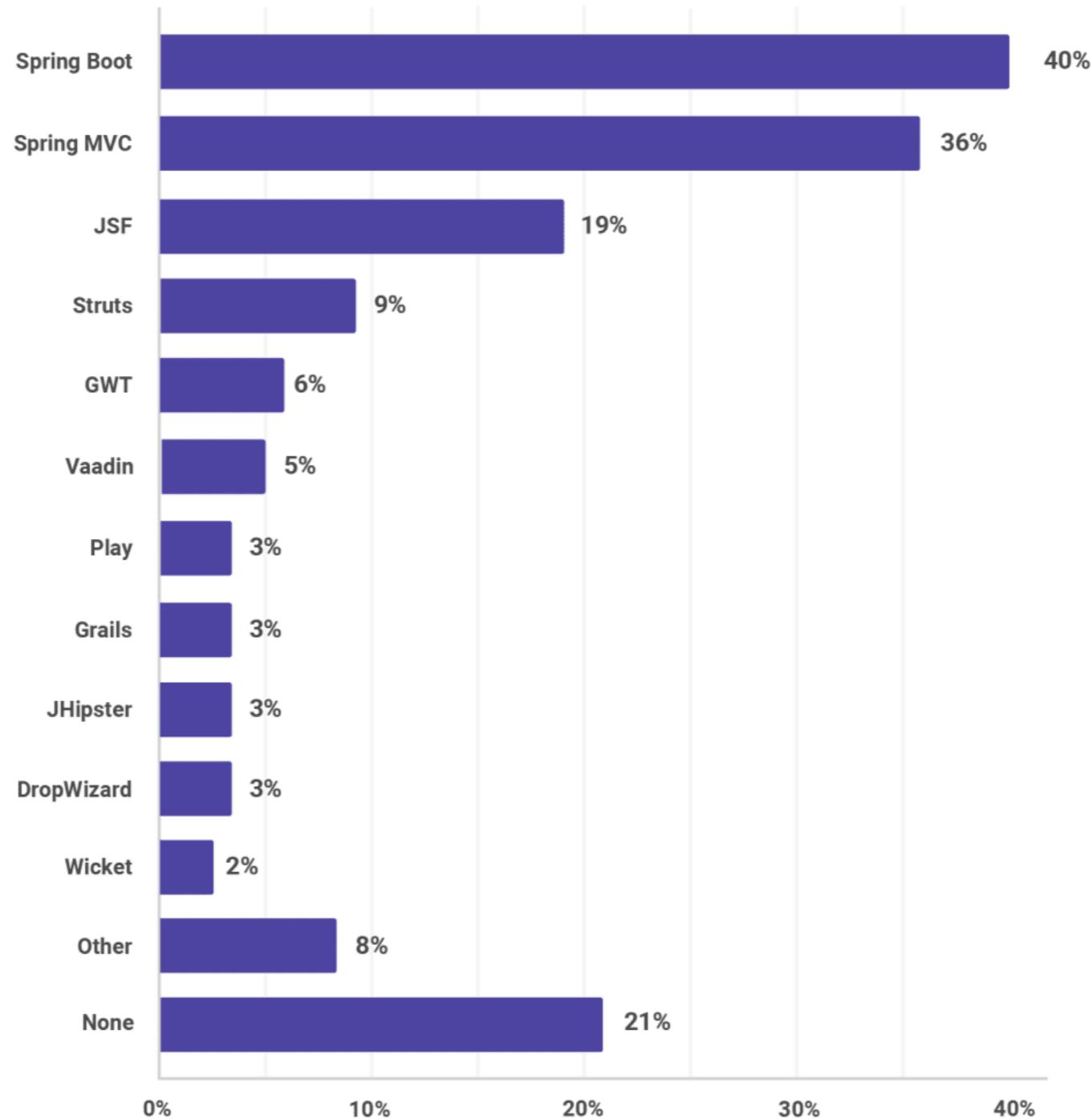
No plumbing code

Flexible architecture

Staying current

Spring Boot

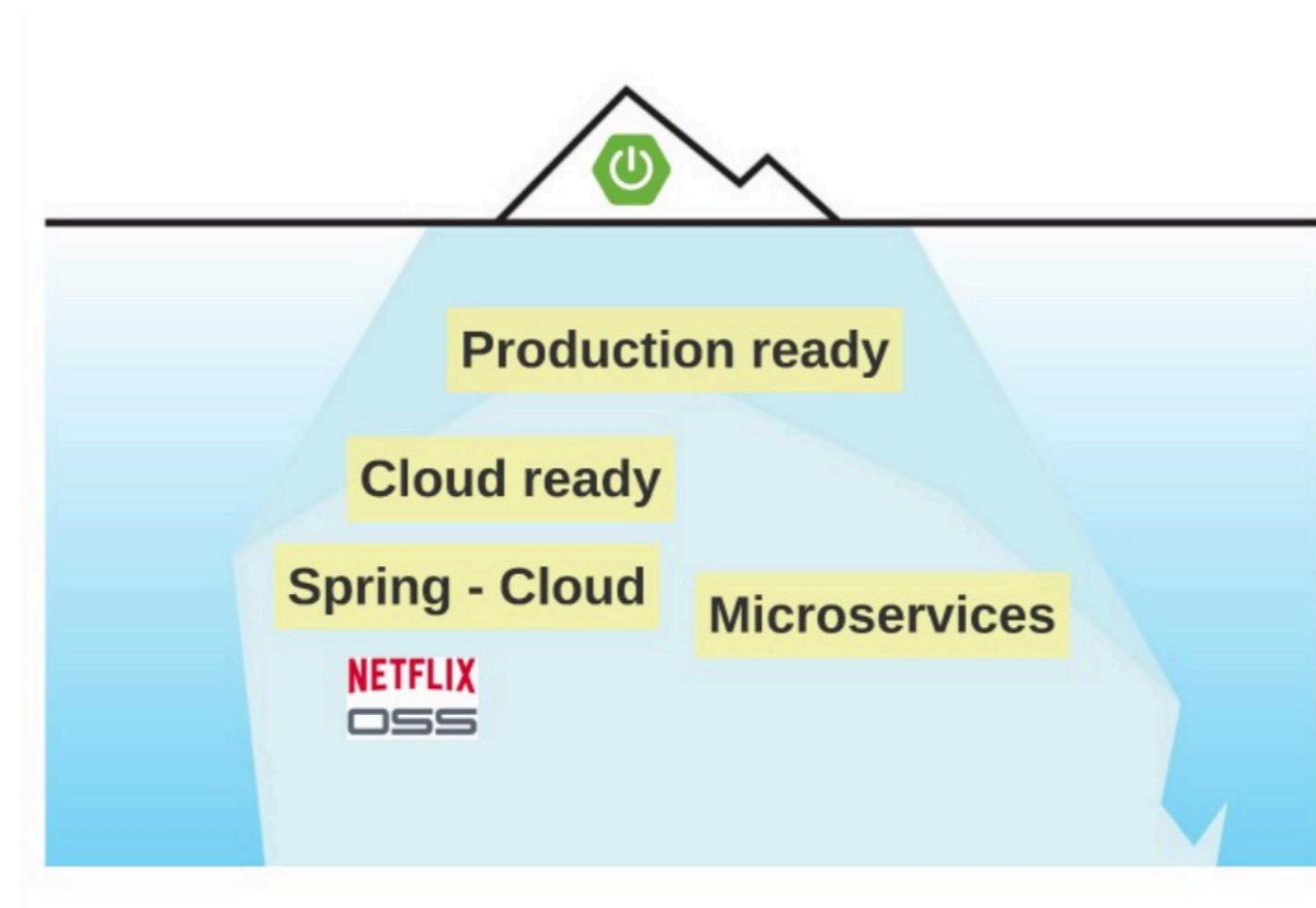
Why ?



<https://snyk.io/blog/jvm-ecosystem-report-2018-platform-application/>

Why Spring Boot ?

Application skeleton generator for java app
Reduce effort to add new technologies



Spring Framework

vs.

Spring Boot

Spring Framework



Spring Boot



Features

Features ?

Embedded application server

Integration with tools/technologies (starter)

Production tools (monitoring, health check)

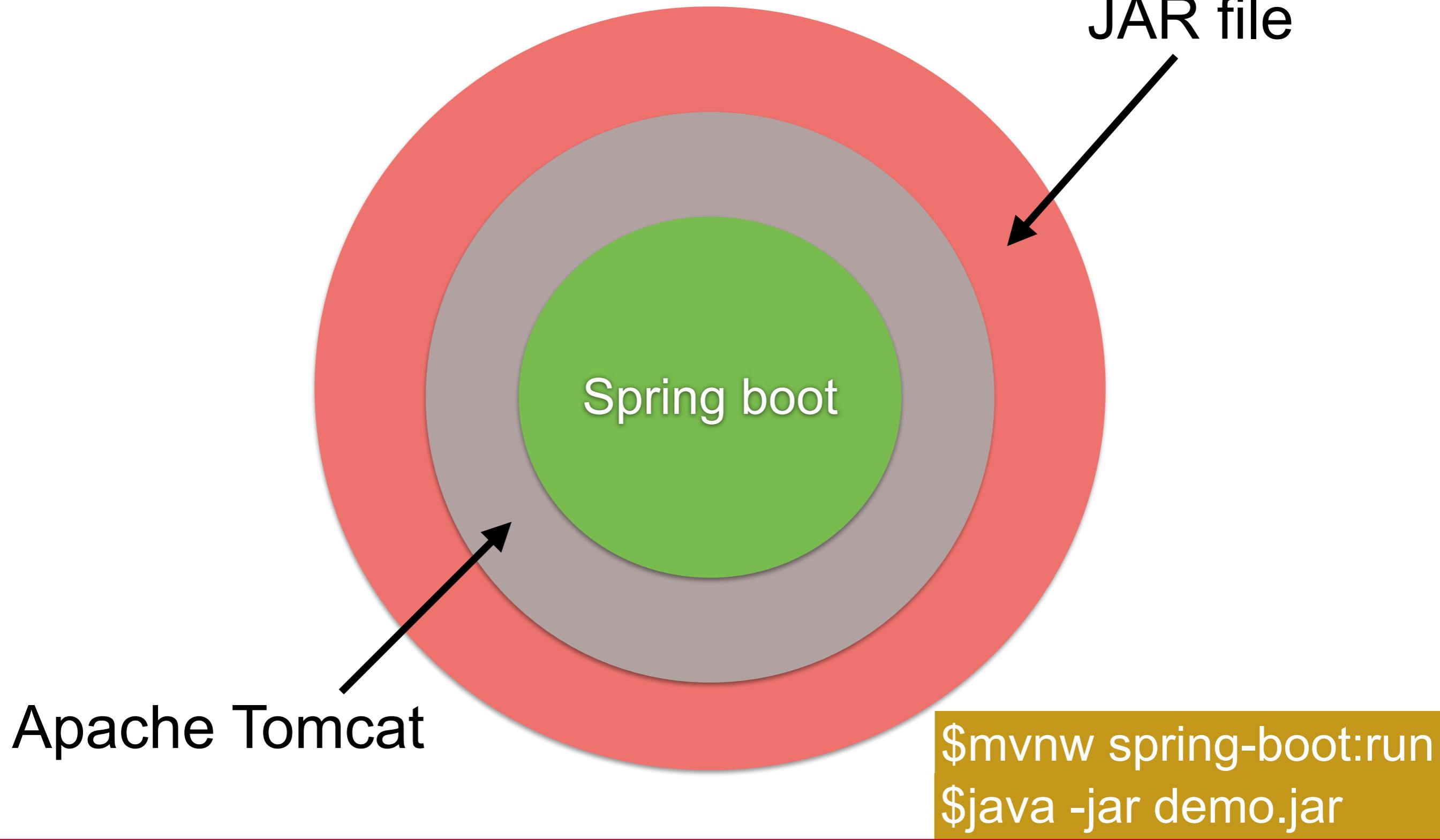
Configuration management

Dev tools

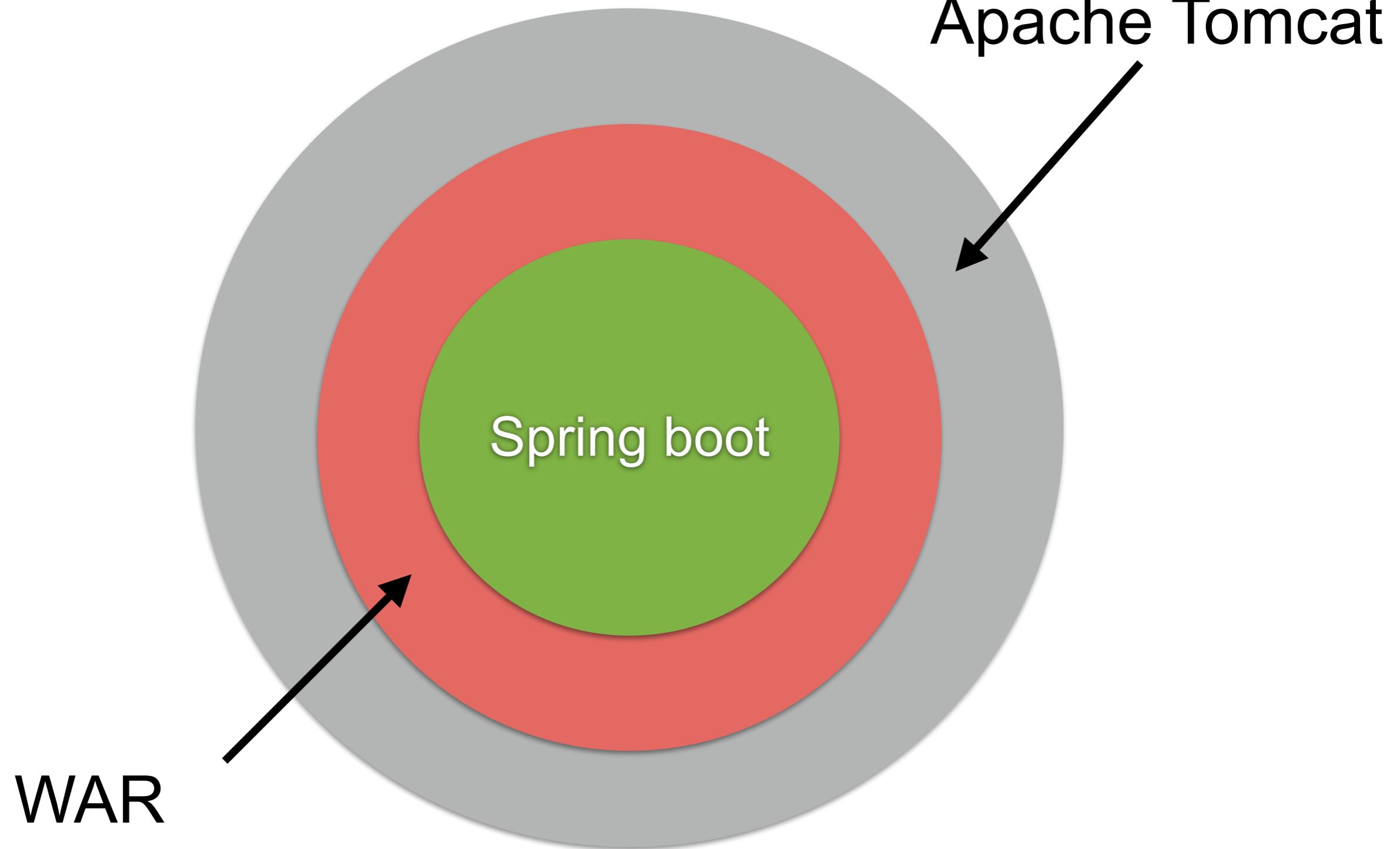
No source code generation, no XML

Package and Deploy

How ?



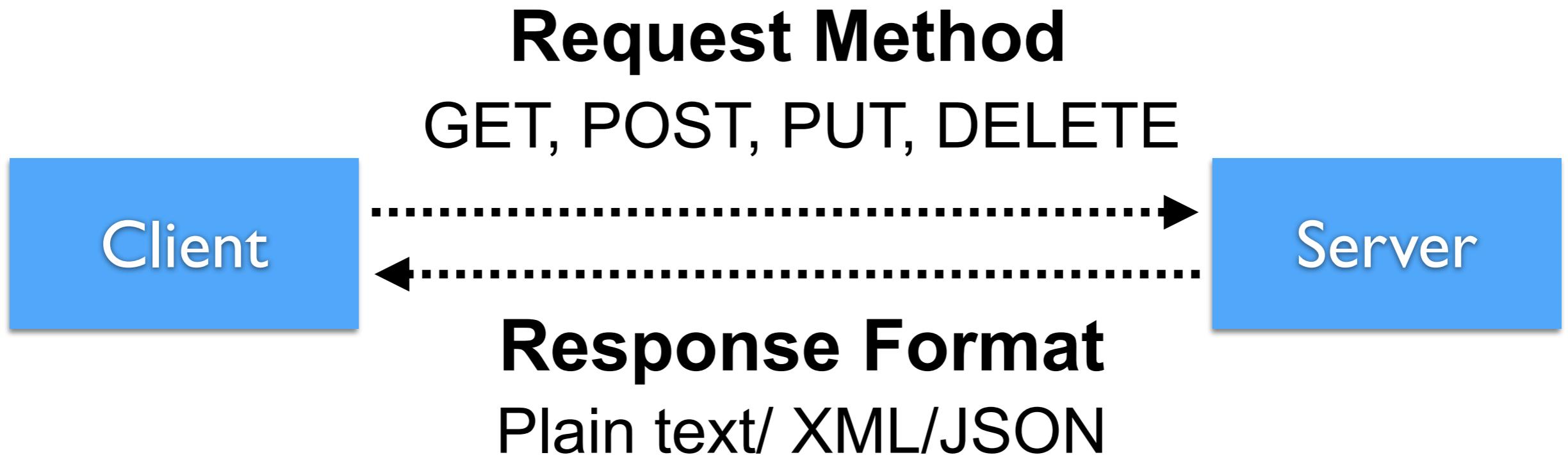
How ?



Building RESTful API with Spring Boot

REST API

REST Request & Response



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

Create project with Spring Initializr

Spring Initializr

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

SPRING INITIALIZR bootstrap your application now

Generate a with and Spring Boot

Project Metadata

Artifact coordinates

Group

Artifact

Dependencies

Add Spring Boot Starters and dependencies to your application

Search for dependencies

Selected Dependencies

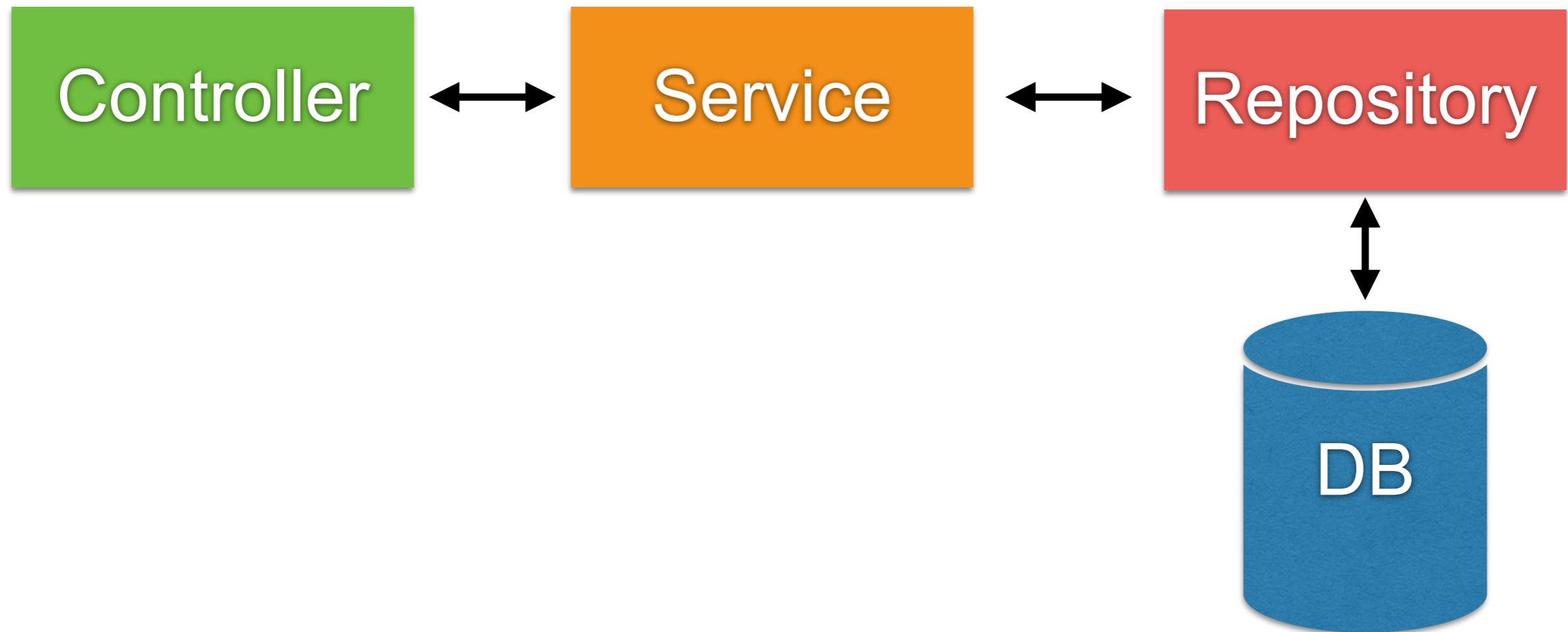
Generate Project

Don't know what to look for? Want more options? [Switch to the full version.](#)

start.spring.io is powered by [Spring Initializr](#) and [Pivotal Web Services](#)

Project structure

Basic structure of Spring Boot



Controller

Request and Response
Validation input

Delegate to others classes such as service and repository

Service

Control the flow of main business logics

Manage database transaction

Don't reuse service

Repository

Manage data in data store such as RDBMS and
NoSQL

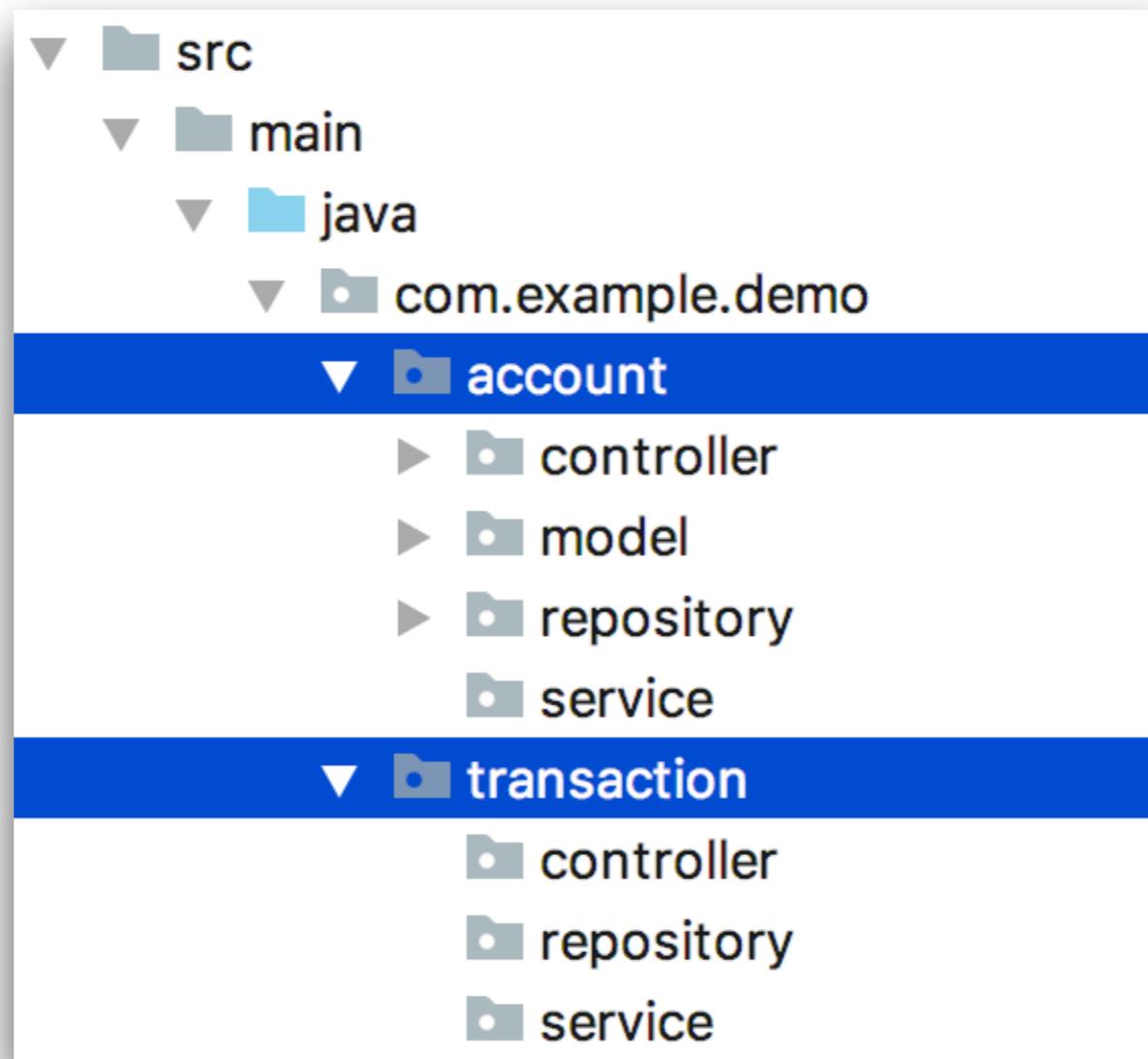
Spring Boot Structure (1)

Separate by function/domain/feature

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

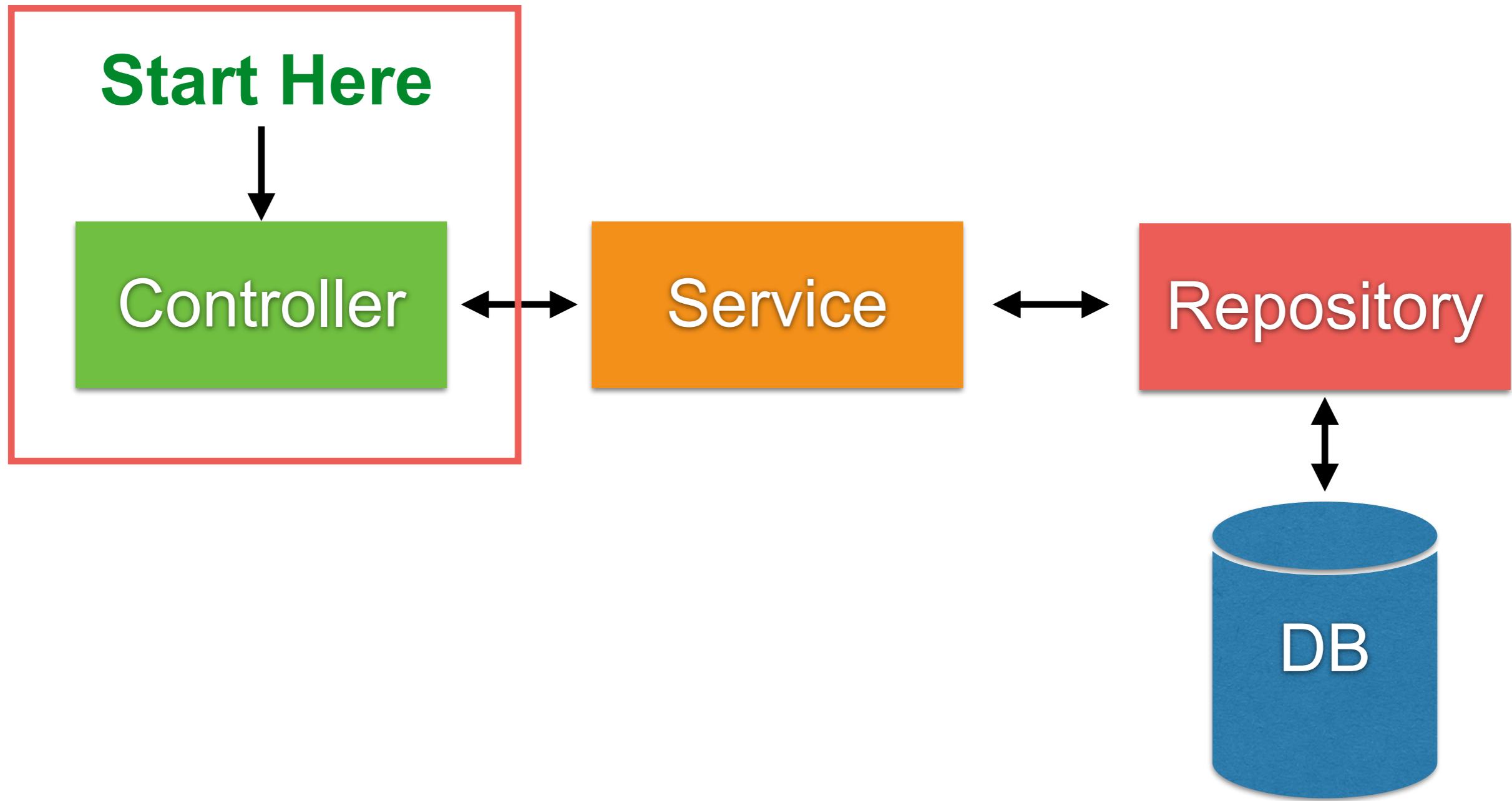
Spring Boot Structure (2)

Separate by function/domain/feature

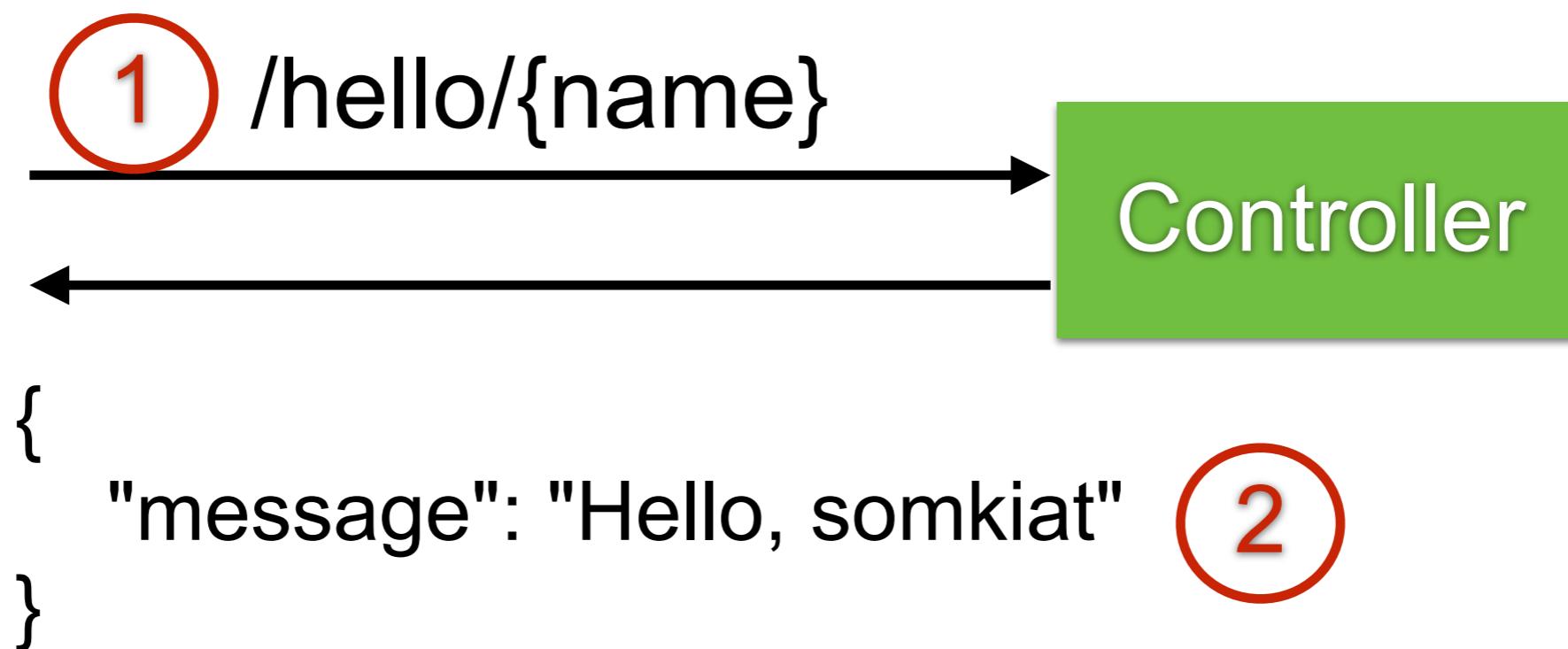


Create first RESTful API

Basic structure of Spring Boot



Hello API



1. Create REST Controller

HelloController.java

```
@RestController
public class HelloController {

    @GetMapping("/hello/{name}")
    public Hello sayHi(@PathVariable String name) {
        return new Hello("Hello, " + name);
    }

}
```

2. Create model class

Hello.java

```
public class Hello {  
  
    private String message;  
  
    Hello(String message) {  
        this.message = message;  
    }  
  
    public String getMessage() {  
        return message;  
    }  
  
    public void setMessage(String message) {  
        this.message = message;  
    }  
}
```

3. Compile and Packaging

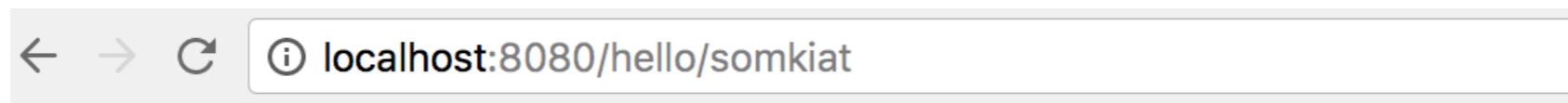
\$mvnw clean package

4. Run

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

5. Open in browser

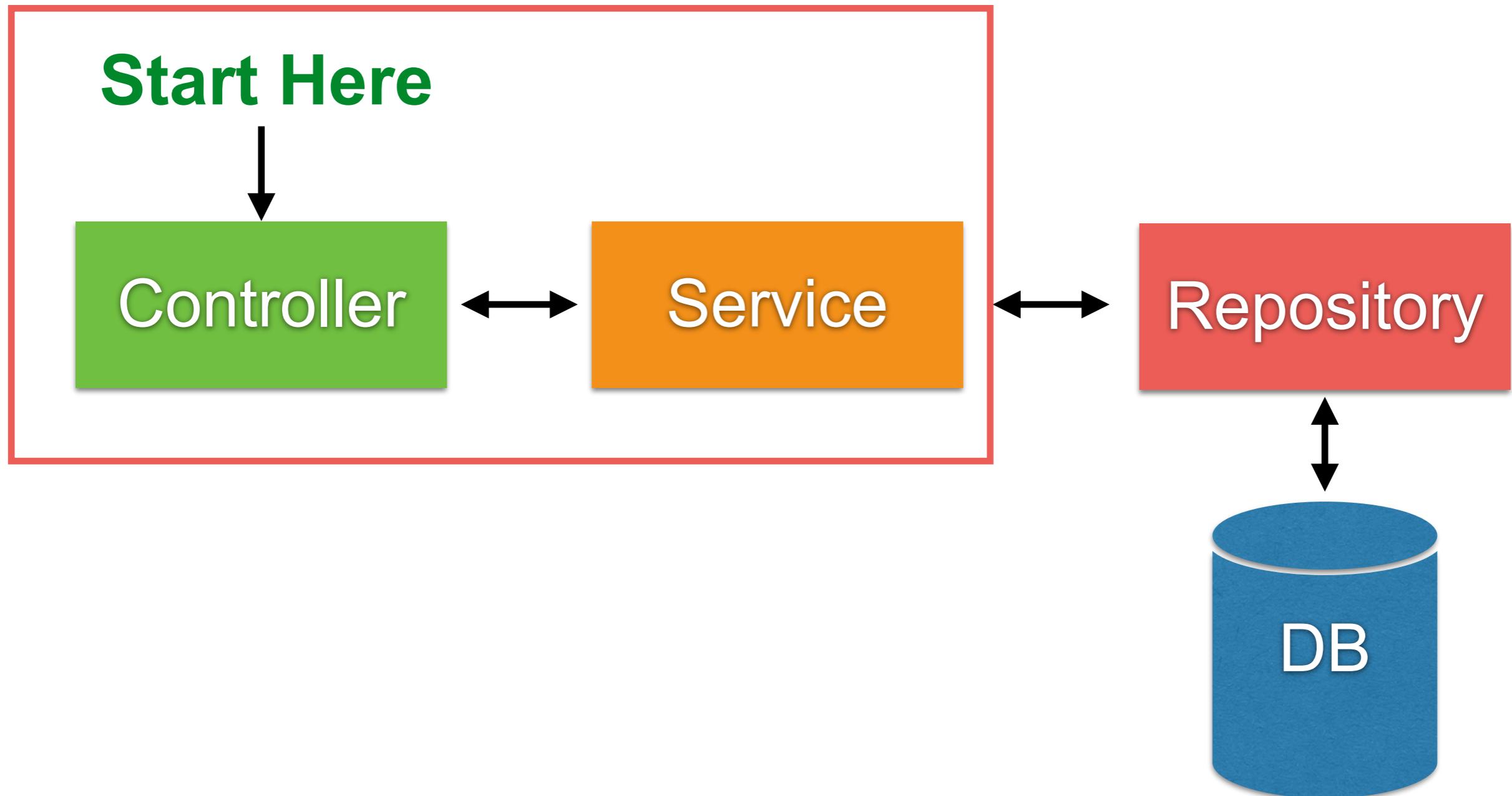
<http://localhost:8080/hello/somkiat>



{ "message": "Hello somkiat" }

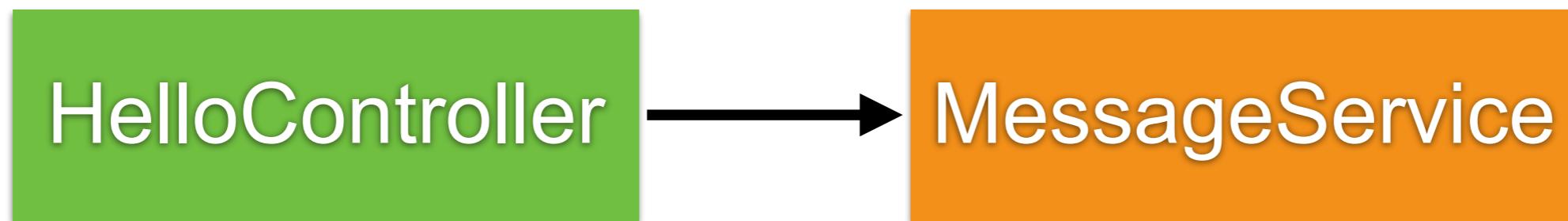
Move business logic to service

Working with service



Move business logic to service

MessageService.java



MessageService.java

```
@Service  
public class MessageService {  
  
    public String concat(String name) {  
        return "Hello, " + name;  
    }  
  
}
```

Edit HelloController.java

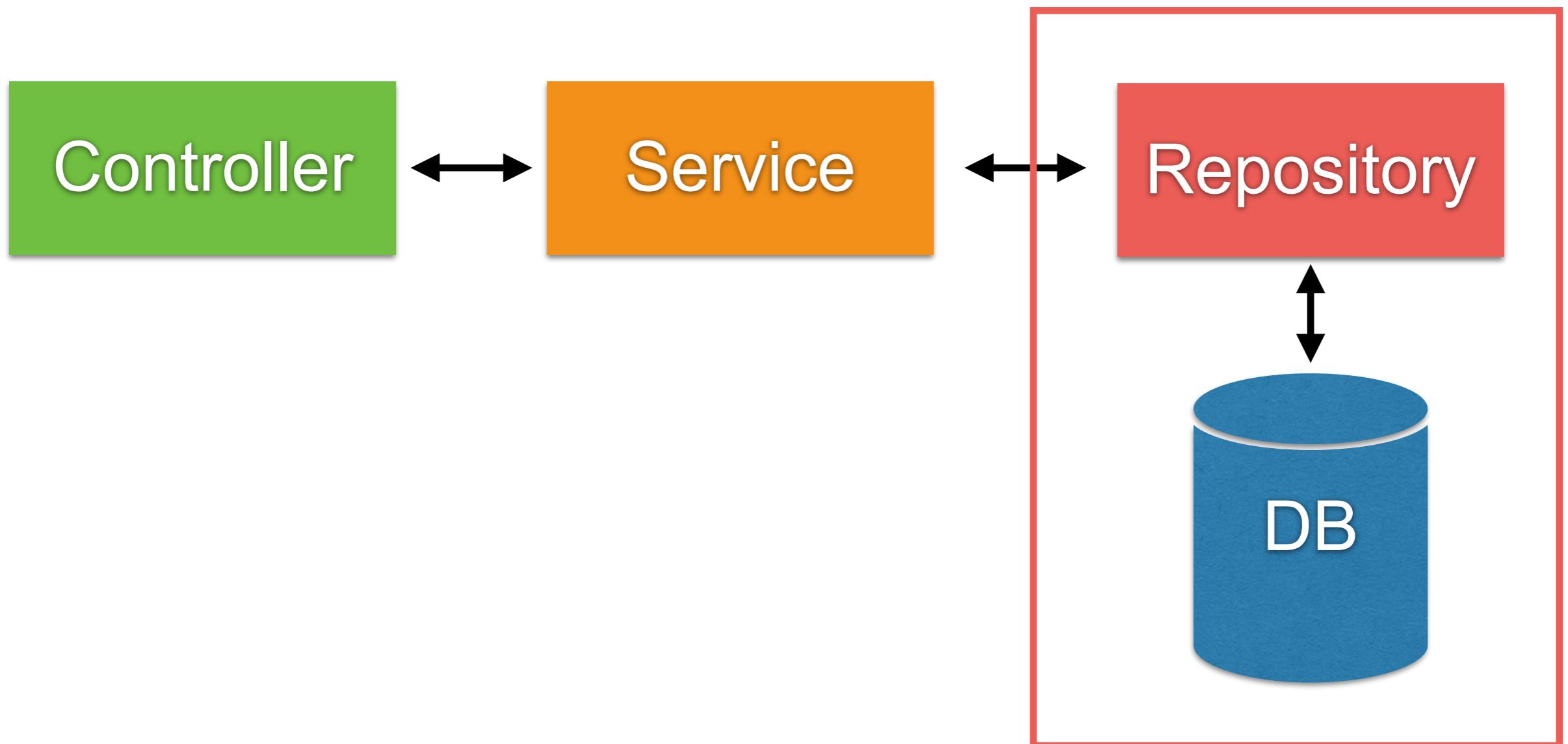
Call method from service

```
@Autowired  
private MessageService messageService;
```

```
@GetMapping("/hello/{name}")  
public HelloResponse  
    sayHi(@PathVariable String name) {  
        String result = messageService.concat(name);  
        return new HelloResponse(result);  
    }
```

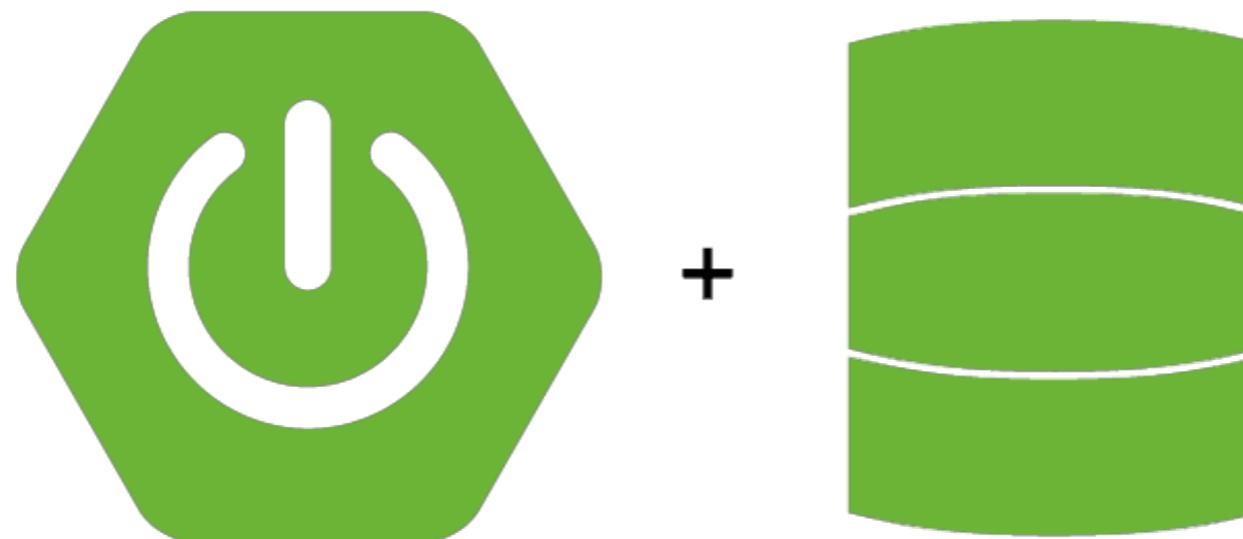
Working with Repository

Working with repository



Working with repository

We're using Spring Data



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

1. Modify pom.xml

Add library of Spring Data such as JPA

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

2. Modify pom.xml (1)

Add library of data store such as MariaDB

```
<dependency>
    <groupId>org.mariadb.jdbc</groupId>
    <artifactId>mariadb-java-client</artifactId>
    <version>2.2.6</version>
</dependency>
```

<https://mvnrepository.com/artifact/org.mariadb.jdbc/mariadb-java-client>

2. Modify pom.xml (2)

Add library of data store such as PostgreSQL

```
<dependency>
    <groupId>org.postgresql</groupId>
    <artifactId>postgresql</artifactId>
    <version>42.2.4</version>
</dependency>
```

<https://mvnrepository.com/artifact/org.postgresql/postgresql>

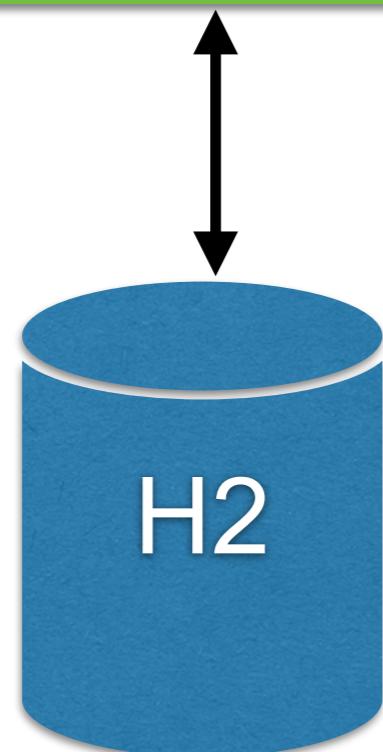
3. Add configuration of database

In src/main/resources/application.properties

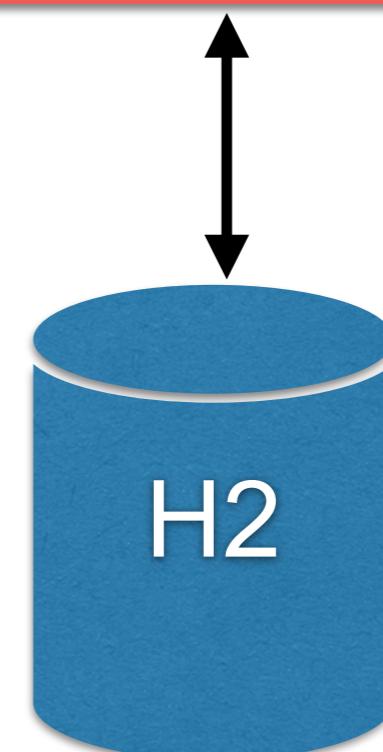
```
spring.jpa.hibernate.ddl-auto=create
spring.datasource.url=<url>
spring.datasource.username=<user>
spring.datasource.password=<password>
```

Working with H2 Database

Production



Testing



Working with H2 database

Add library of data store such as H2

```
<dependency>
    <groupId>com.h2database</groupId>
    <artifactId>h2</artifactId>
    <version>1.4.197</version>
</dependency>
```

<https://mvnrepository.com/artifact/com.h2database/h2/>

Create repository with JPA

MessageRepository.java

```
public interface MessageRepository  
    extends CrudRepository<Message, Integer> {  
  
}
```

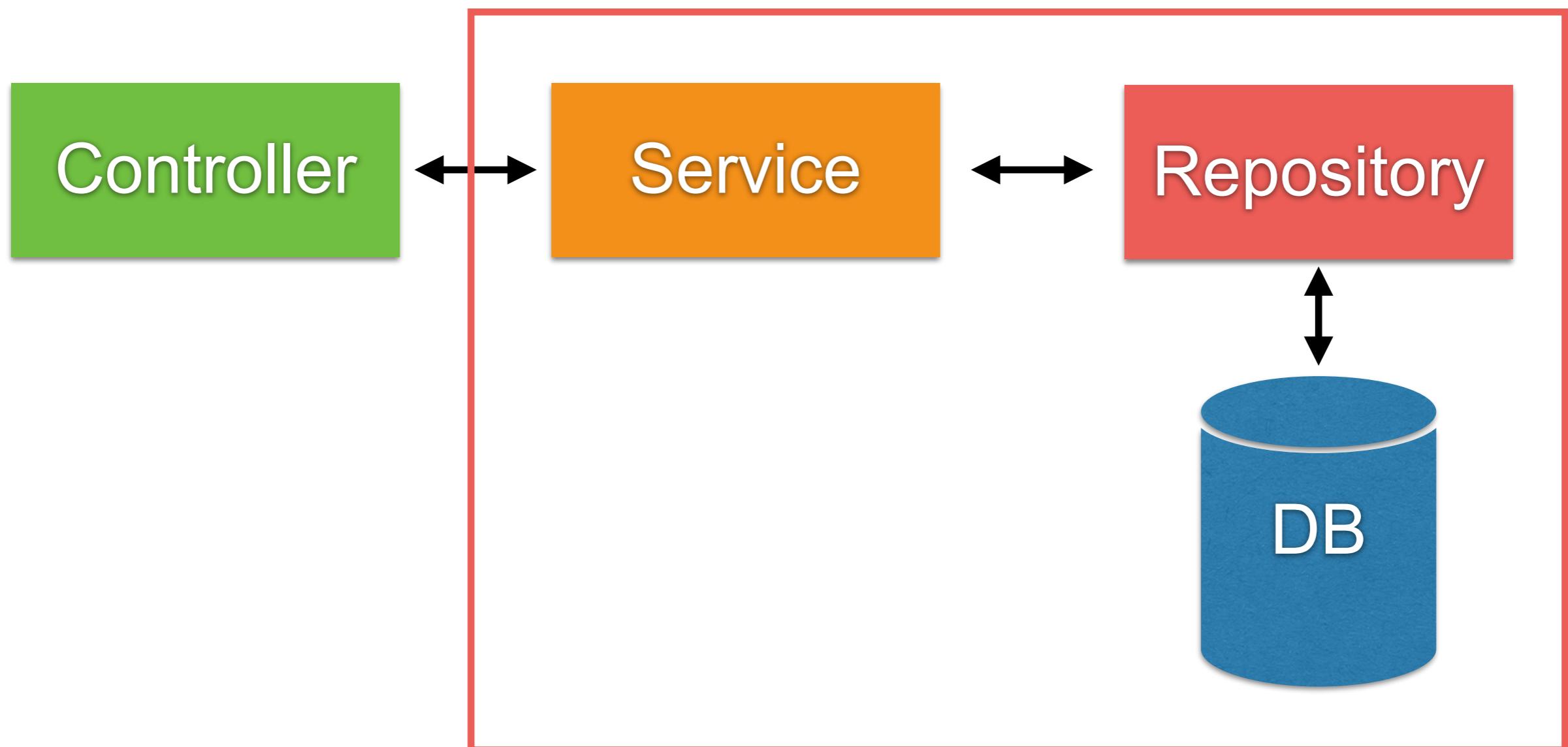
Create Entity class

Message.java

```
@Entity  
public class Message {  
  
    @Id  
    @GeneratedValue(strategy = GenerationType.AUTO)  
    private int id;  
    private String data;
```

Integrate repository with service

Service use repository ?



Service call repository

Edit MessageService.java

```
@Service
public class MessageService {

    @Autowired
    private MessageRepository messageRepository;

    public String concat(String name) {
        Optional<Message> result = messageRepository.findById(1);
        if(result.isPresent()) {
            return "Hello, " + name;
        } else {
            return "Not found";
        }
    }
}
```

Run spring boot

\$mvnw spring-boot:run



Service call repository

```
@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));  
    }  
}
```

1

Service call repository

```
@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));  
    }  
}
```

2

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



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);  
    }  
}
```

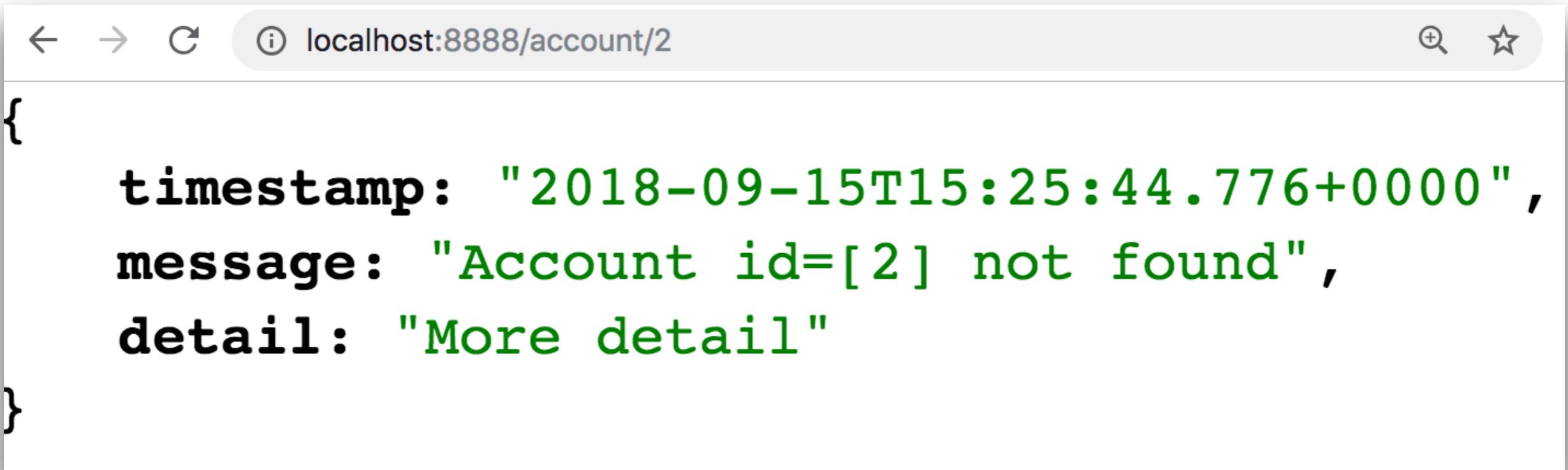
2

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

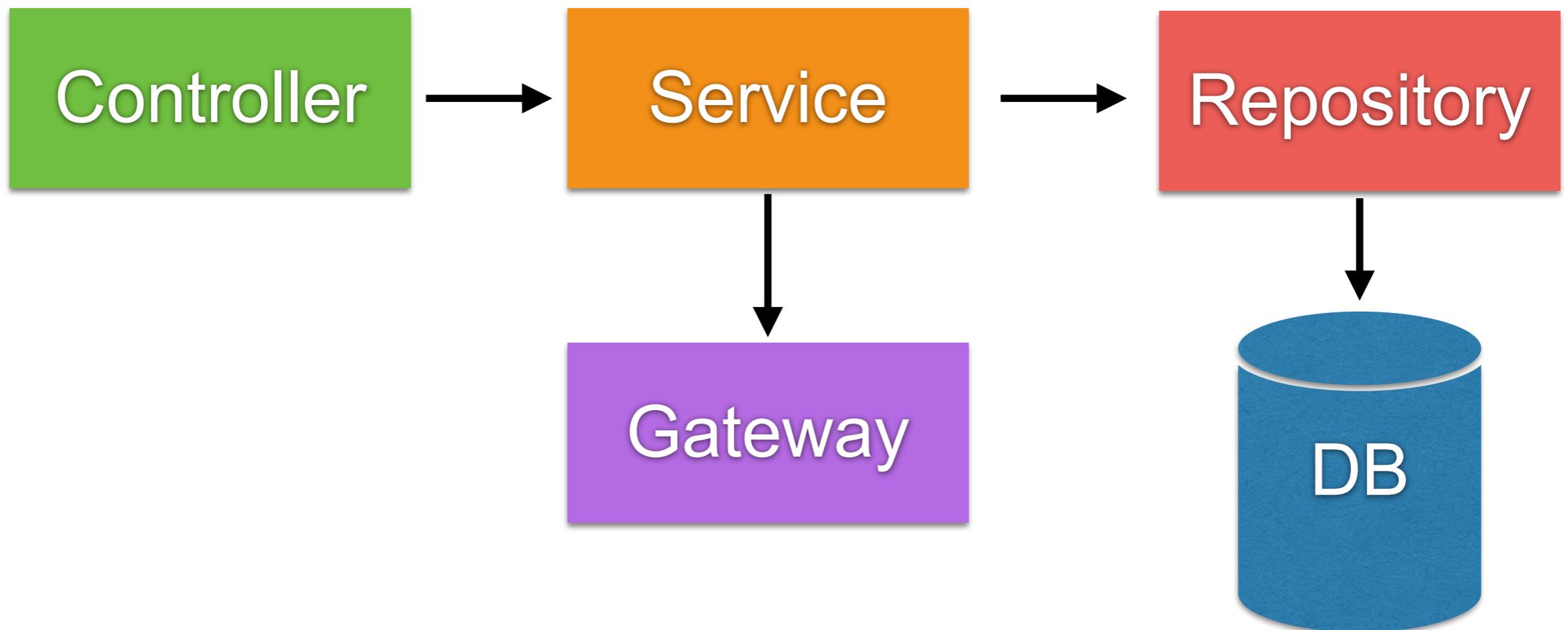


A screenshot of a web browser window. The address bar shows the URL `localhost:8888/account/2`. The main content area displays a JSON response:

```
{  
  timestamp: "2018-09-15T15:25:44.776+0000",  
  message: "Account id=[ 2 ] not found",  
  detail: "More detail"  
}
```

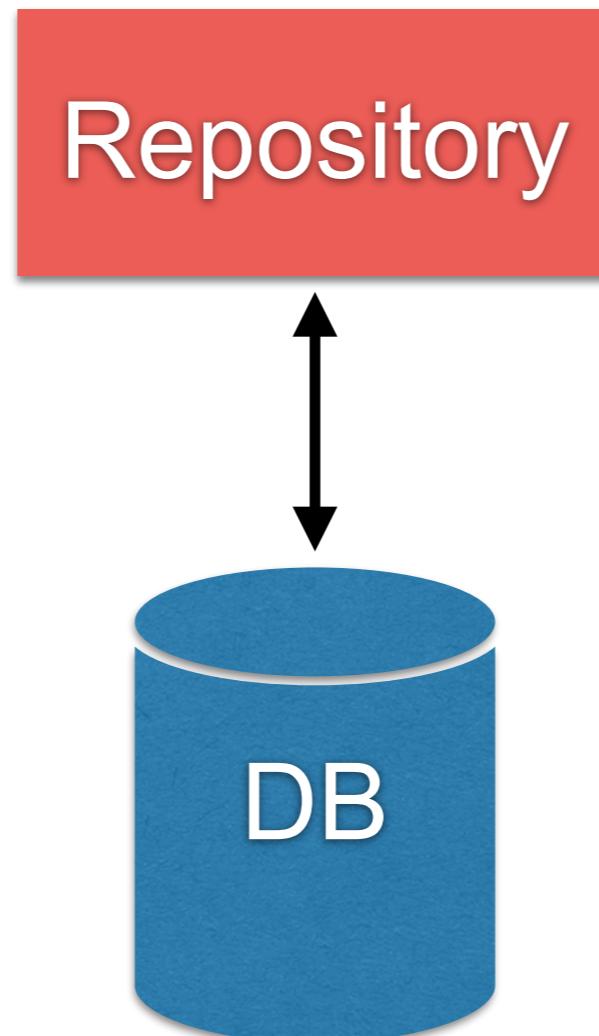
Call External APIs

Call External API



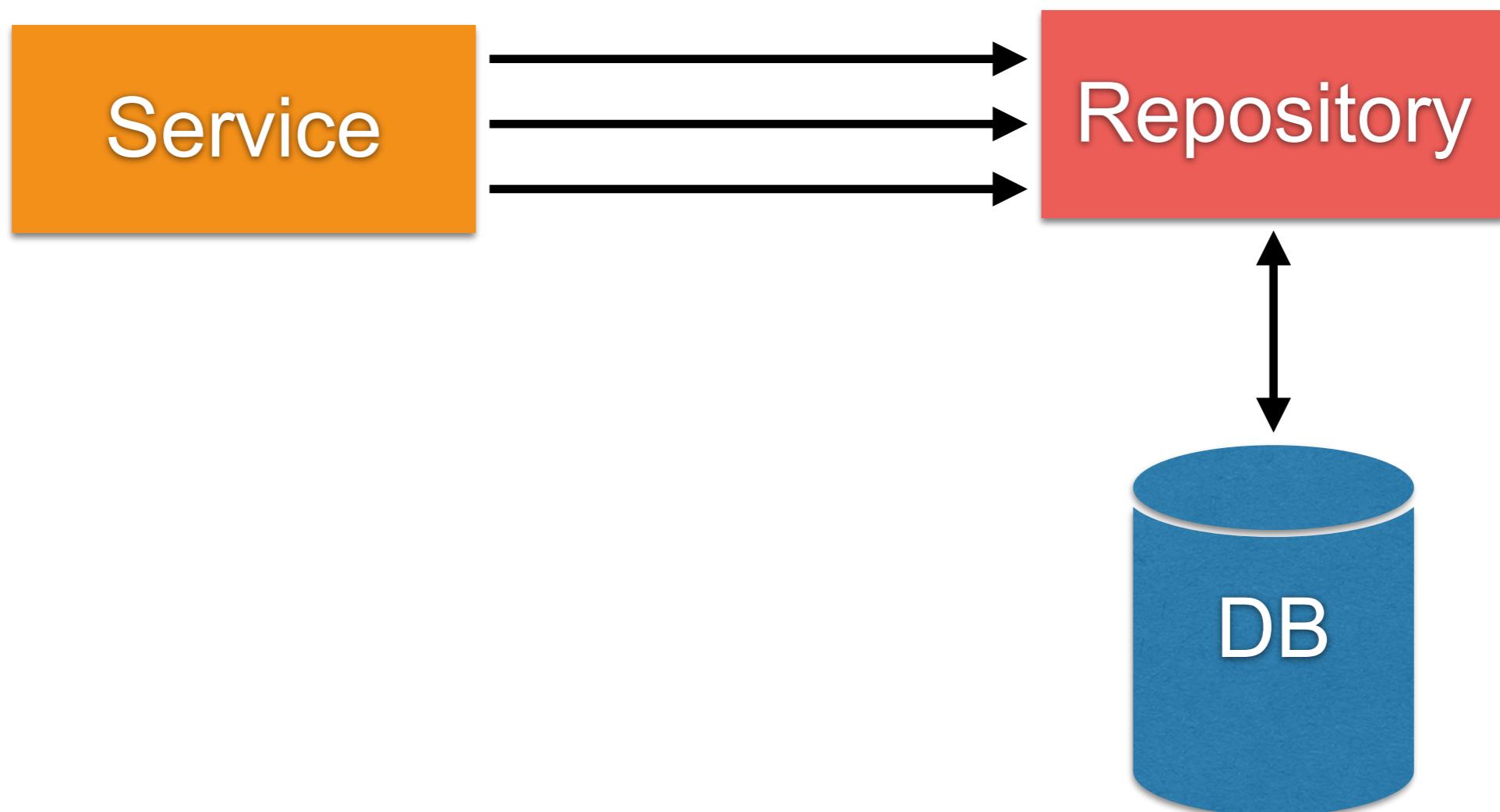
Manage transaction

Enable TransactionManager



Control transaction

@Transactional



Transaction Propagation

REQUIRED

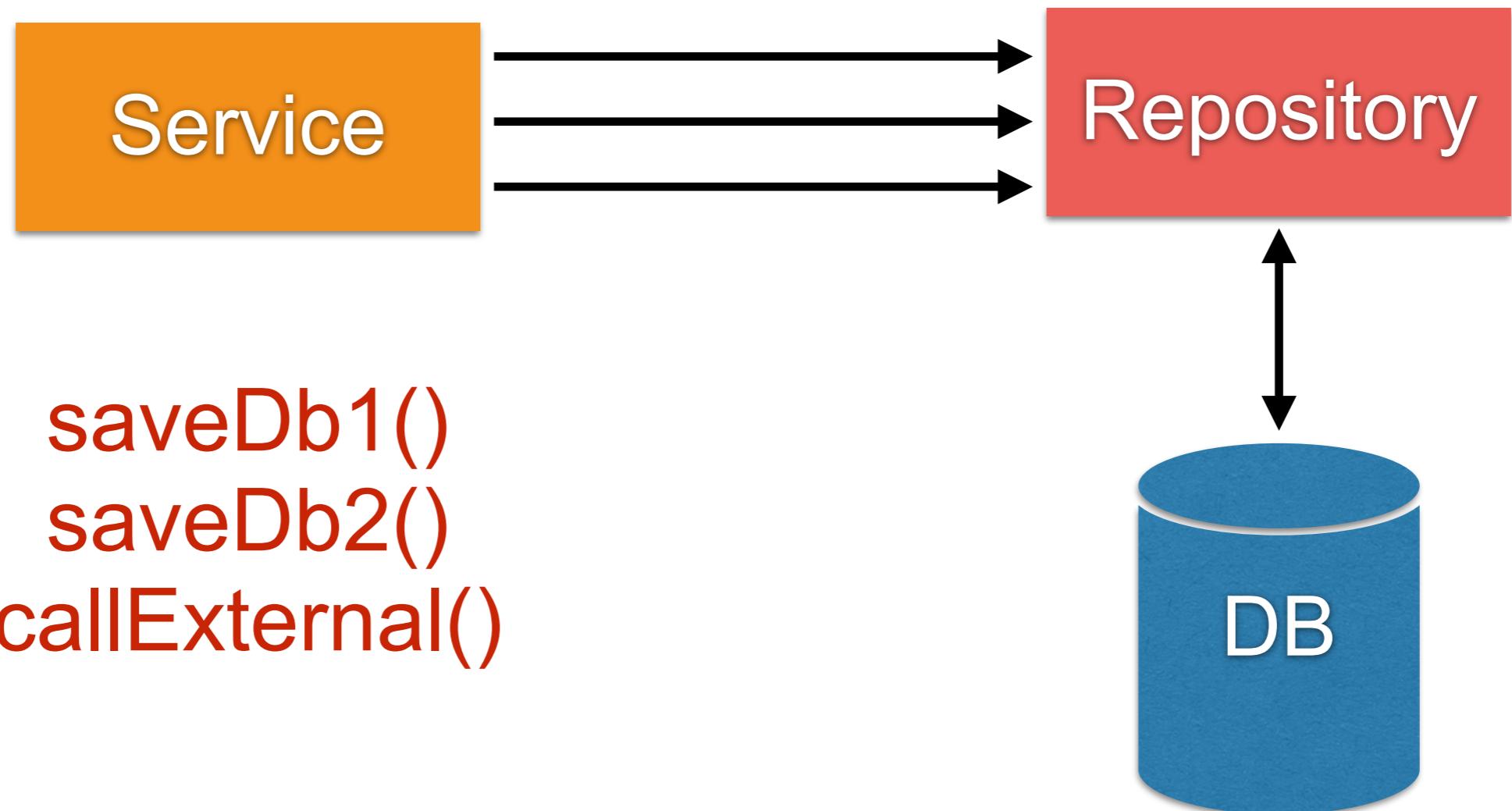
TODO

TODO

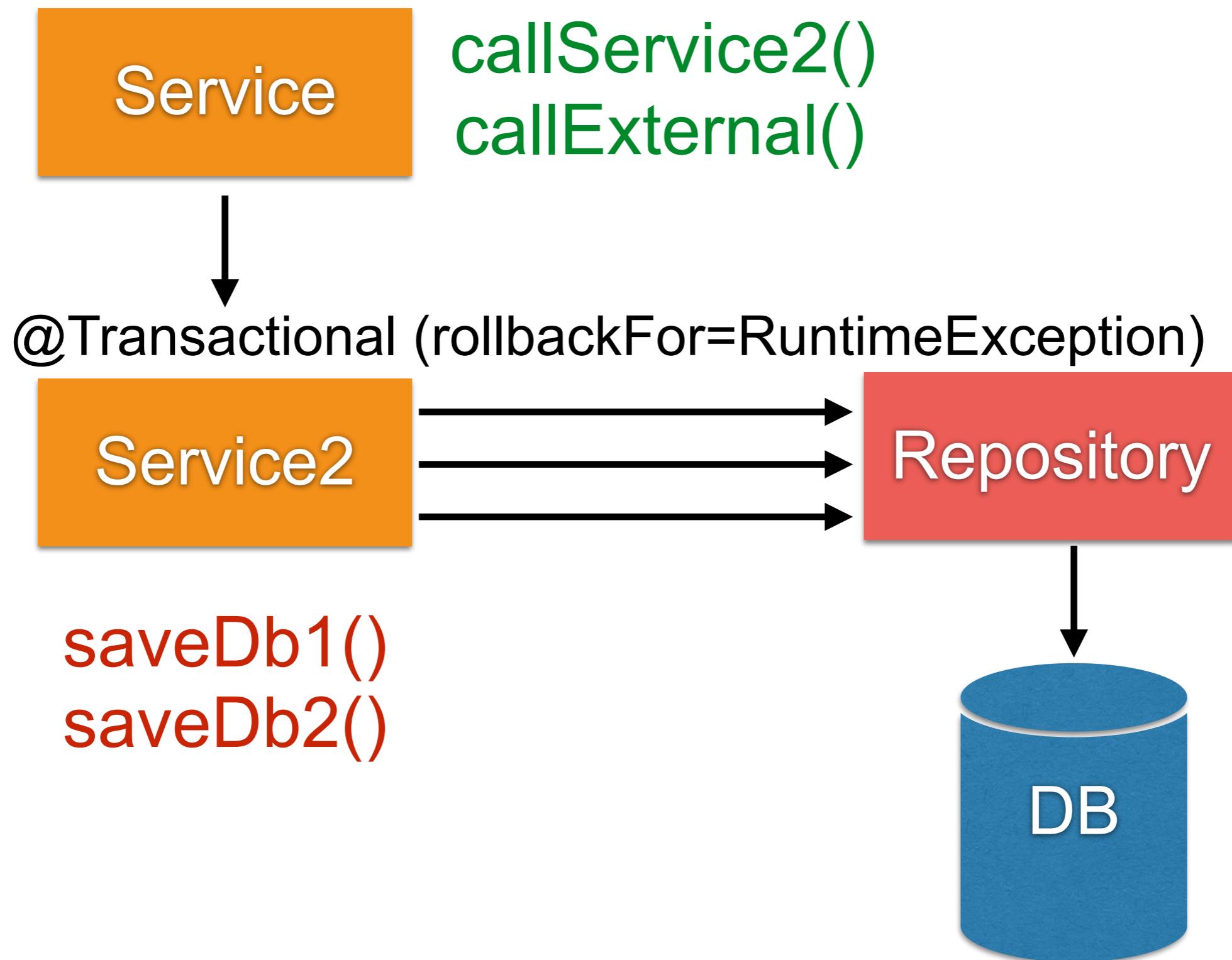
TODO

Pitfall

@Transactional



Pitfall



Transaction propagation !!



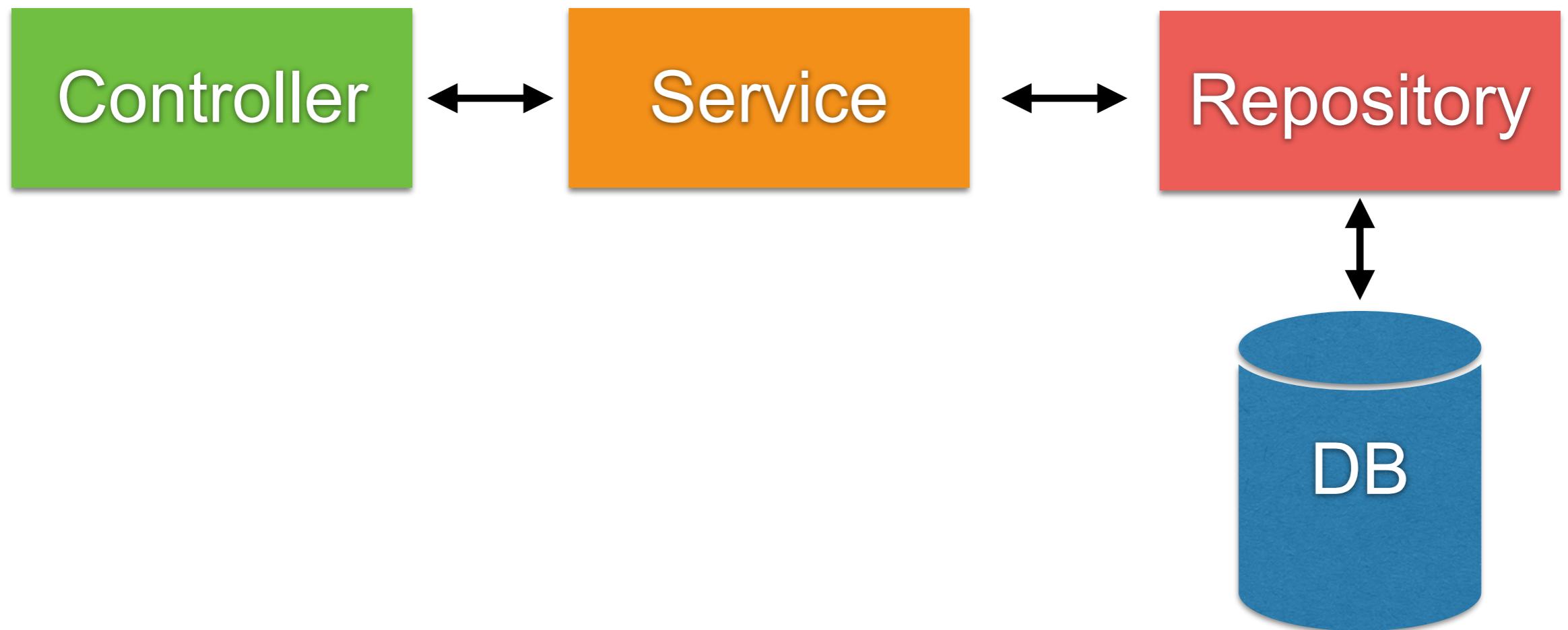
By default :: Use transaction of service 1

Propagation = REQUIRES

Testing

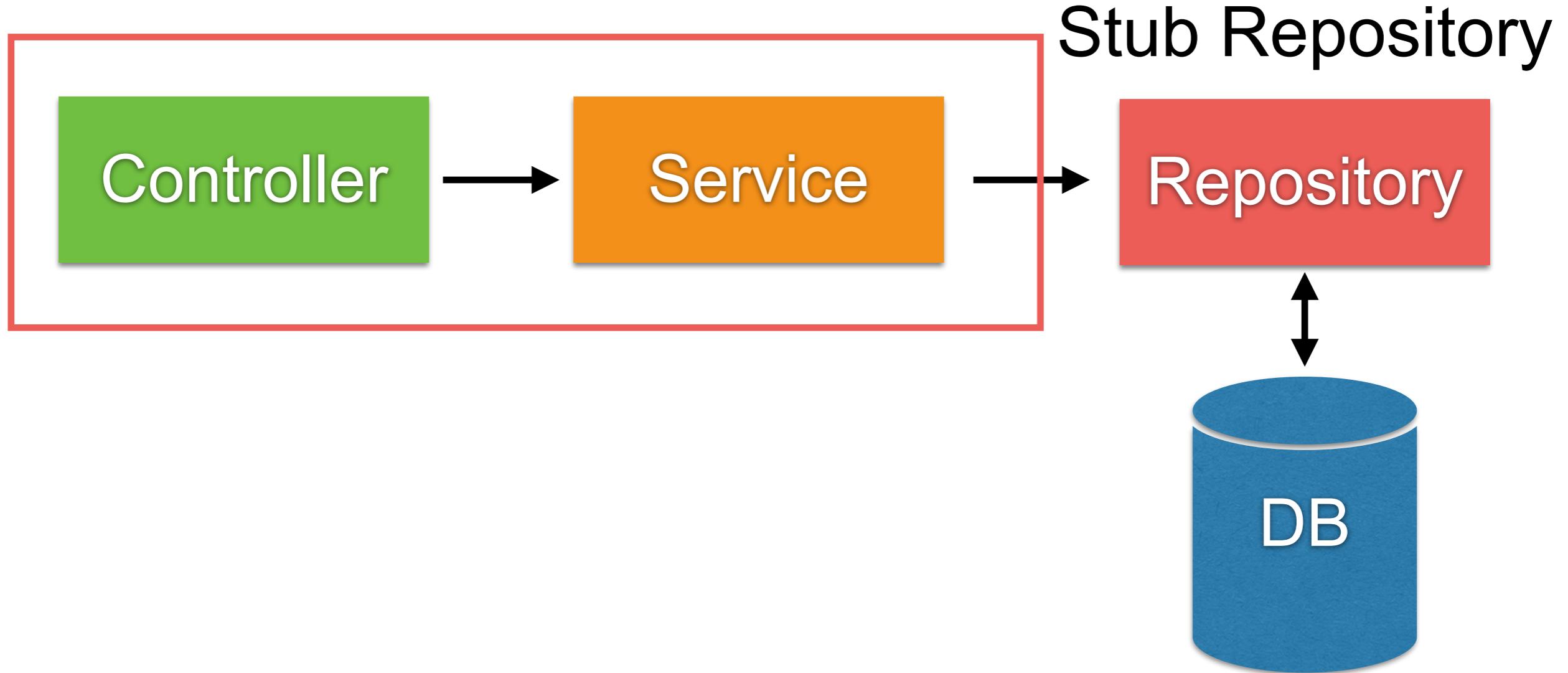
Controller Testing with SpringBootTest

How to test ?

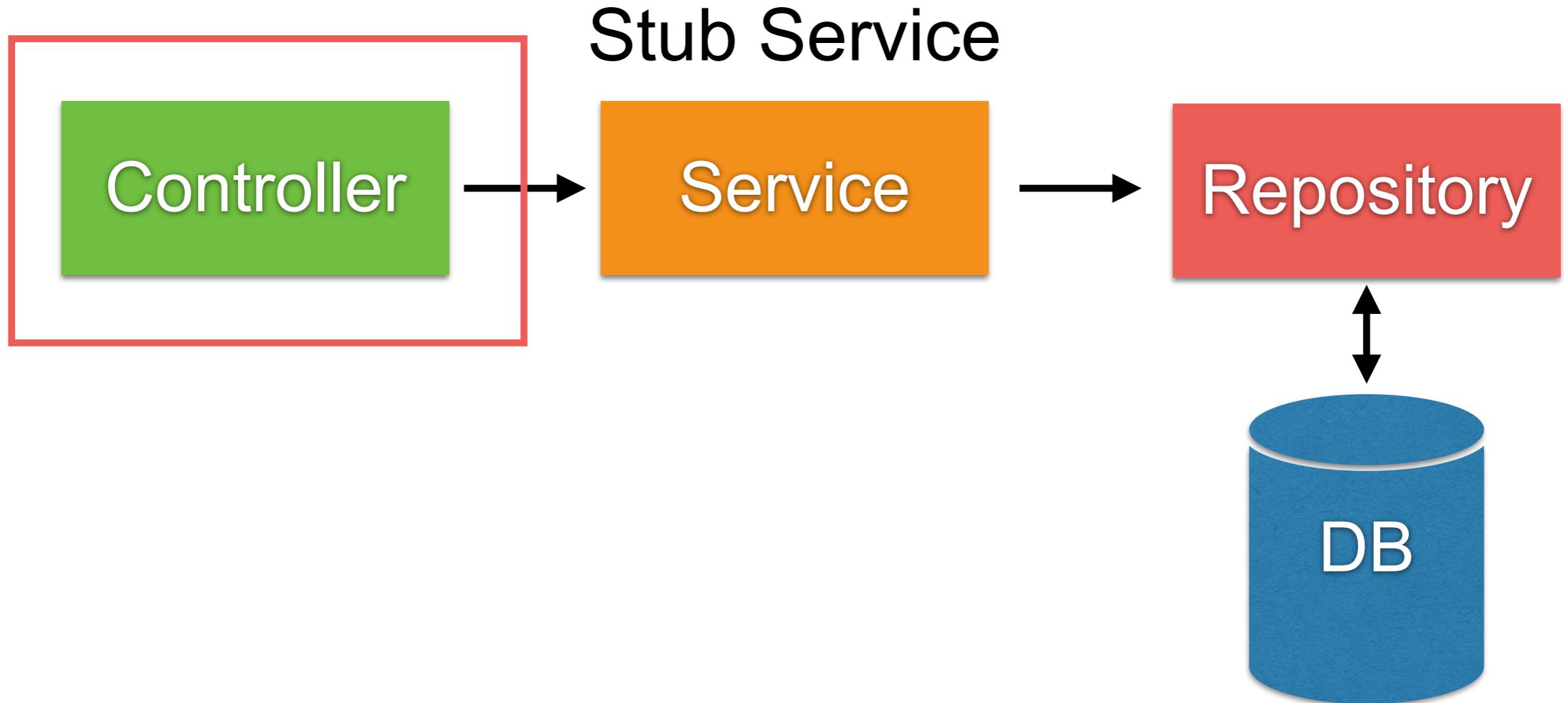


Controller Testing + Stub

How to test ?

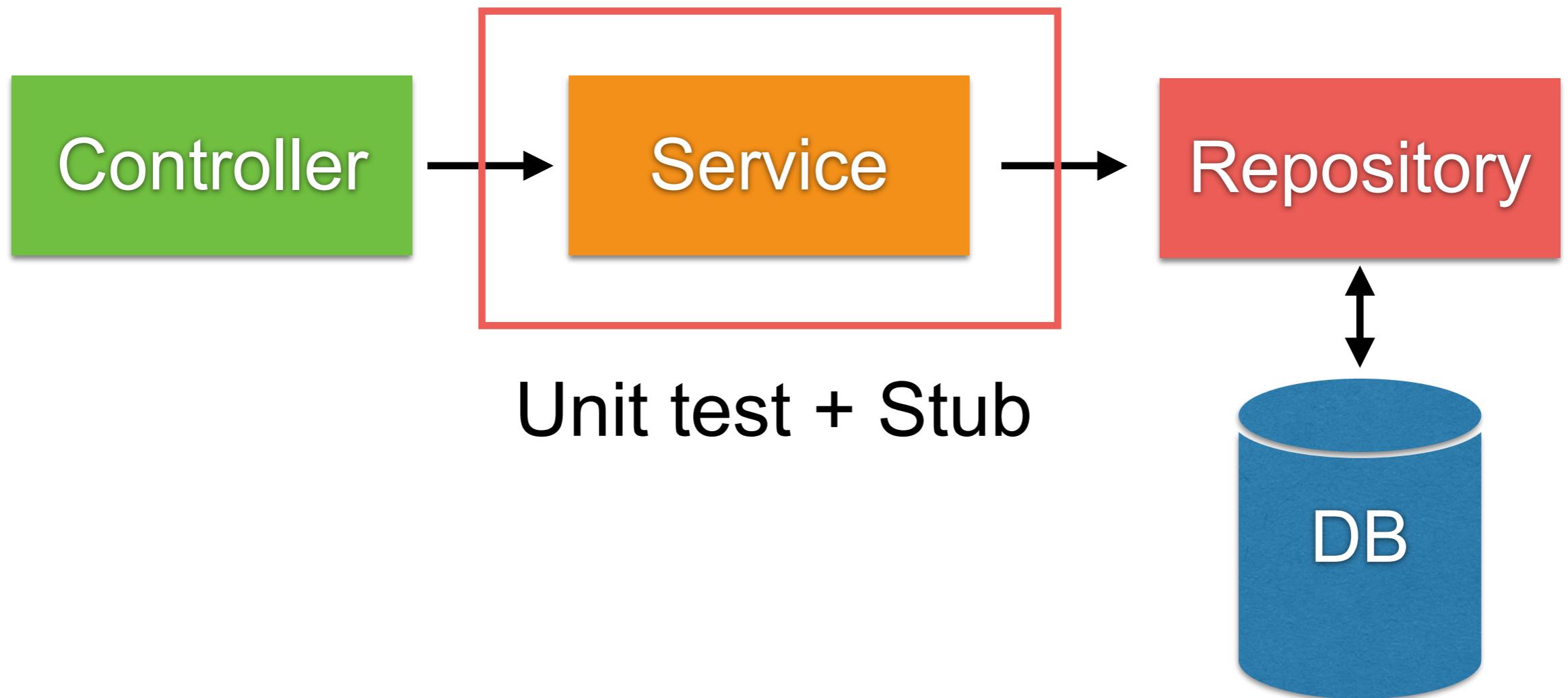


How to test ?



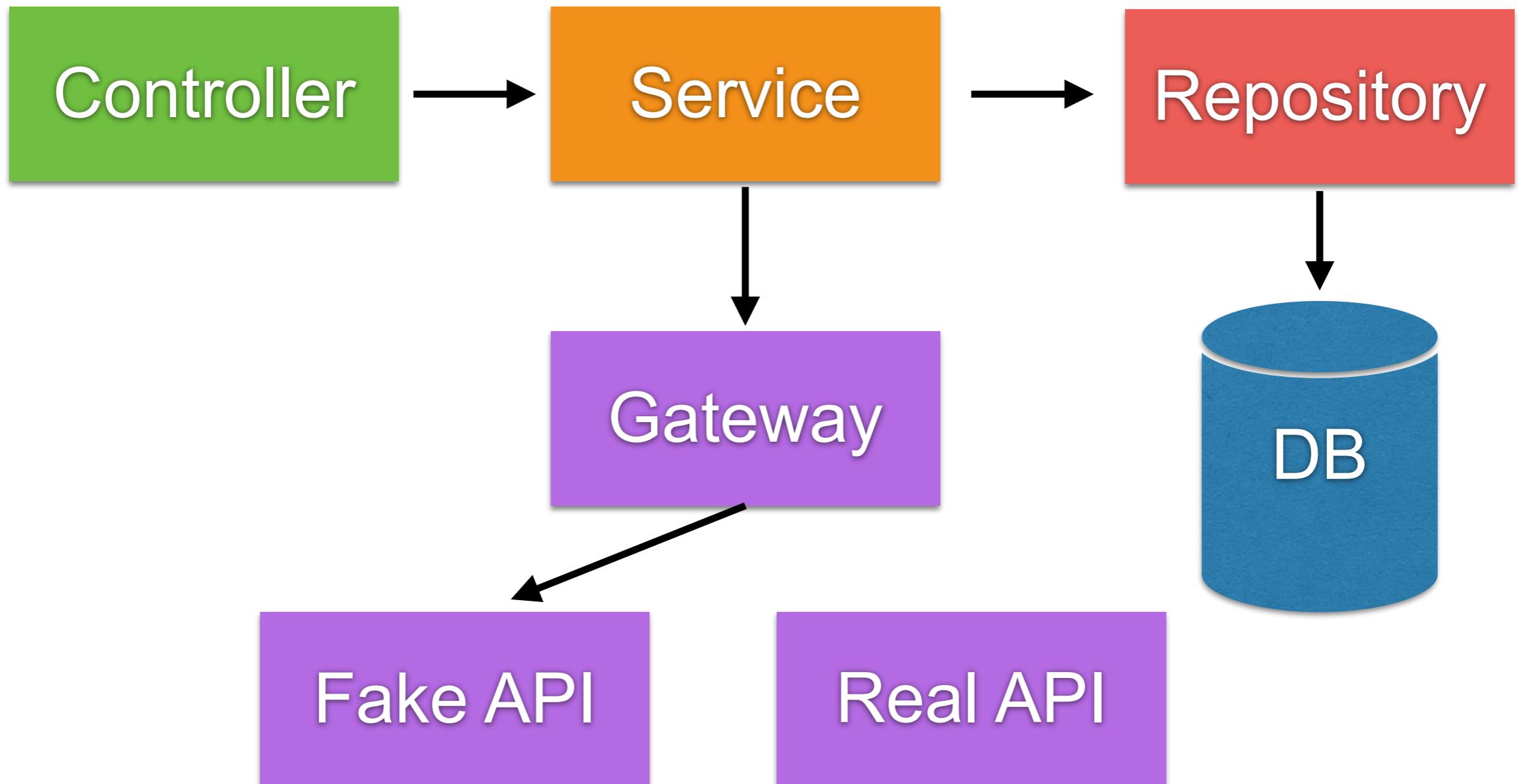
Unit testing for business logic

How to test ?



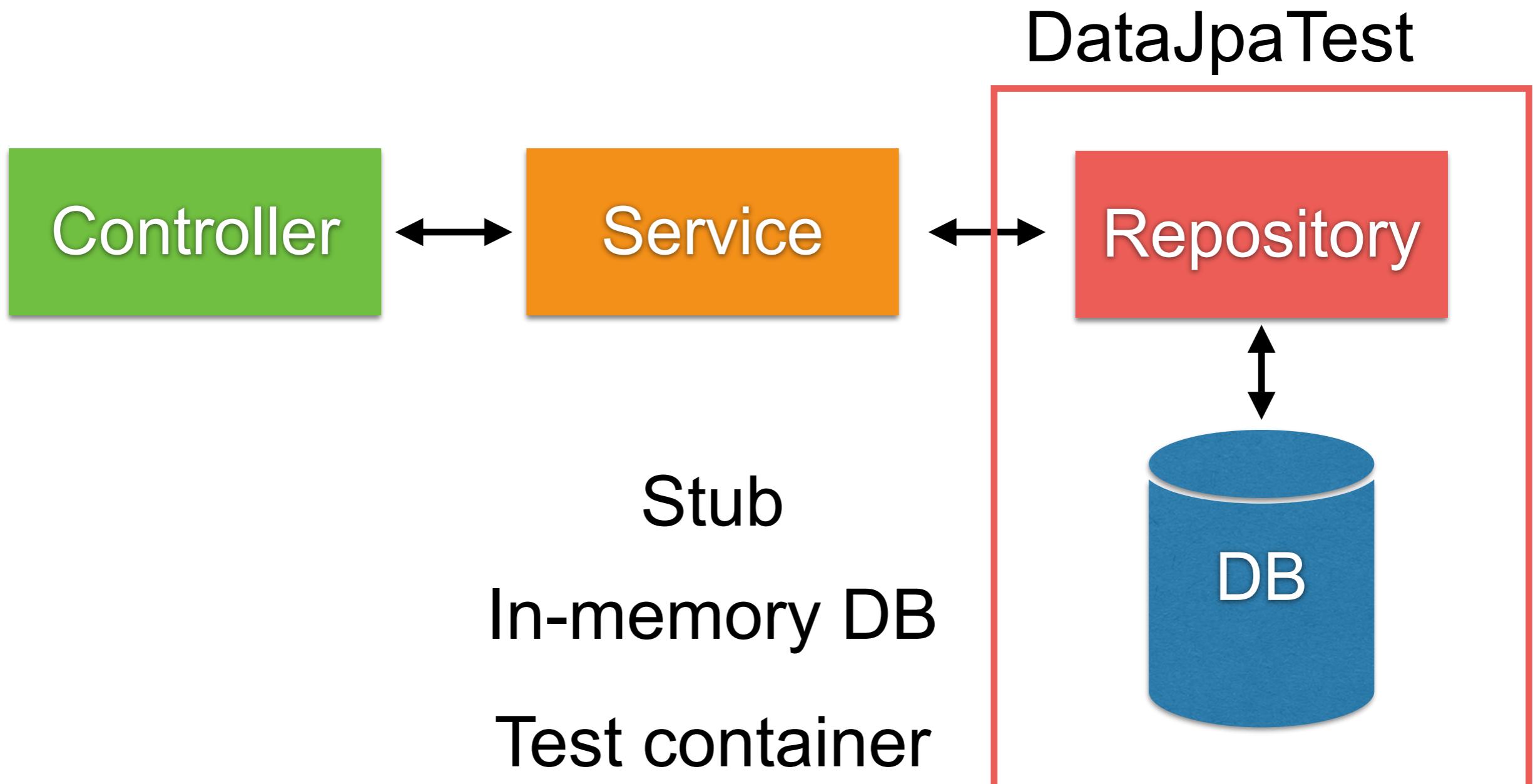
Component testing With External API

Testing External API



Repository testing

How to test ?



Week 01

Week 01

Recap course
Git and Spring Boot
Q/A
Assignment

Recap

Git
Spring Boot

Git commit message

Conventional Commits

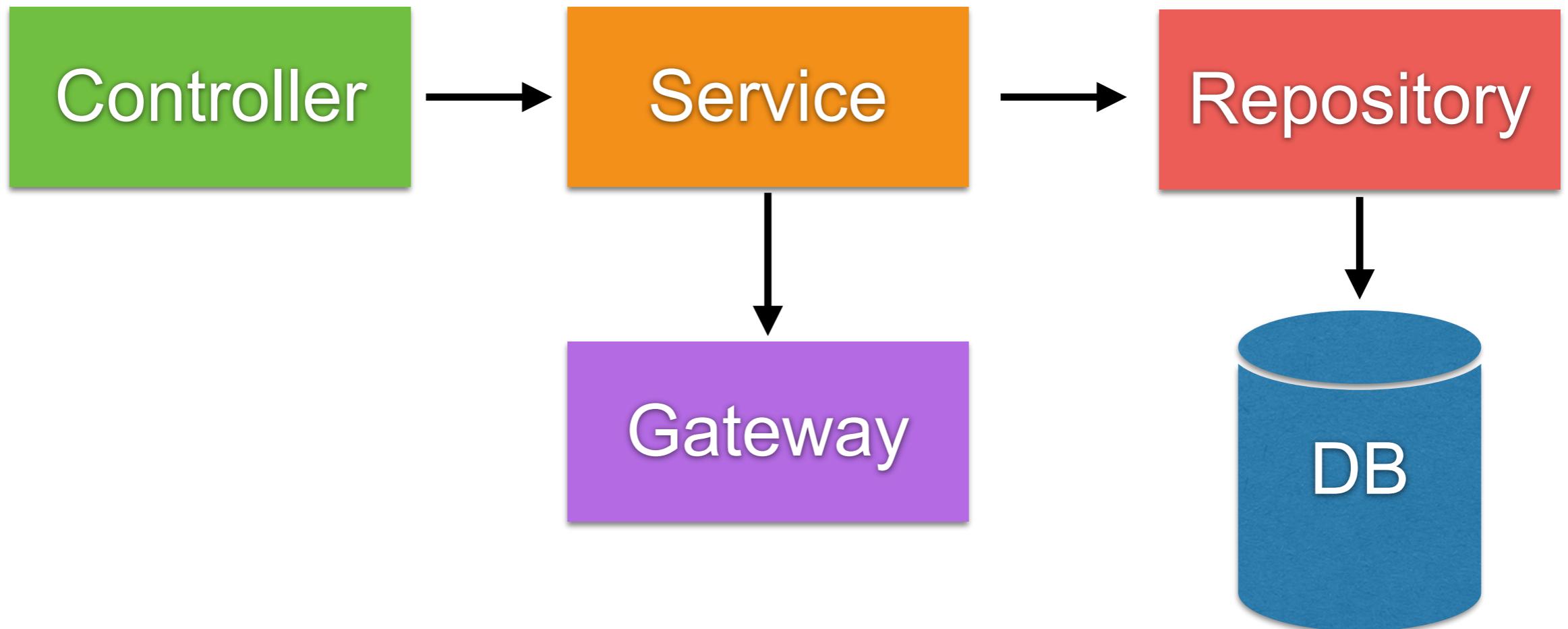
A specification for adding human and machine readable meaning to commit messages

[Quick Summary](#) [Full Specification](#) [Contribute](#)

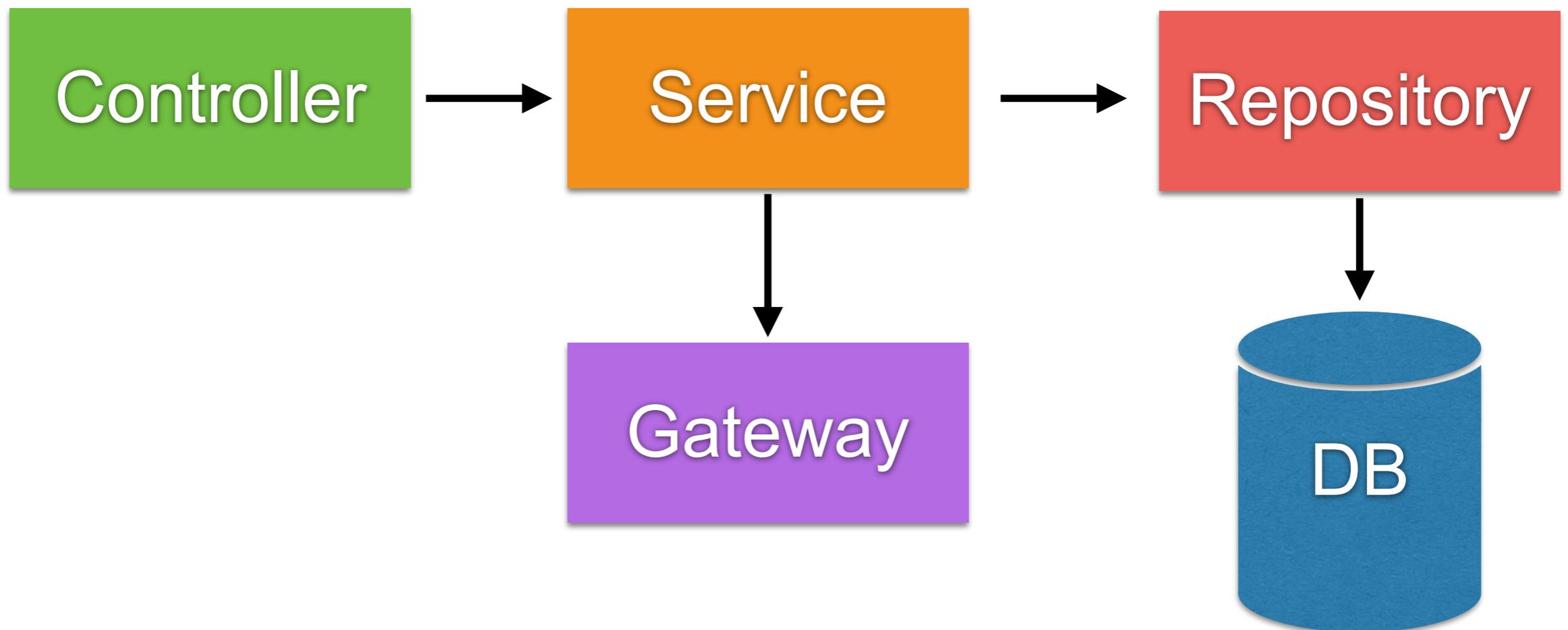


<https://www.conventionalcommits.org/en/v1.0.0/>

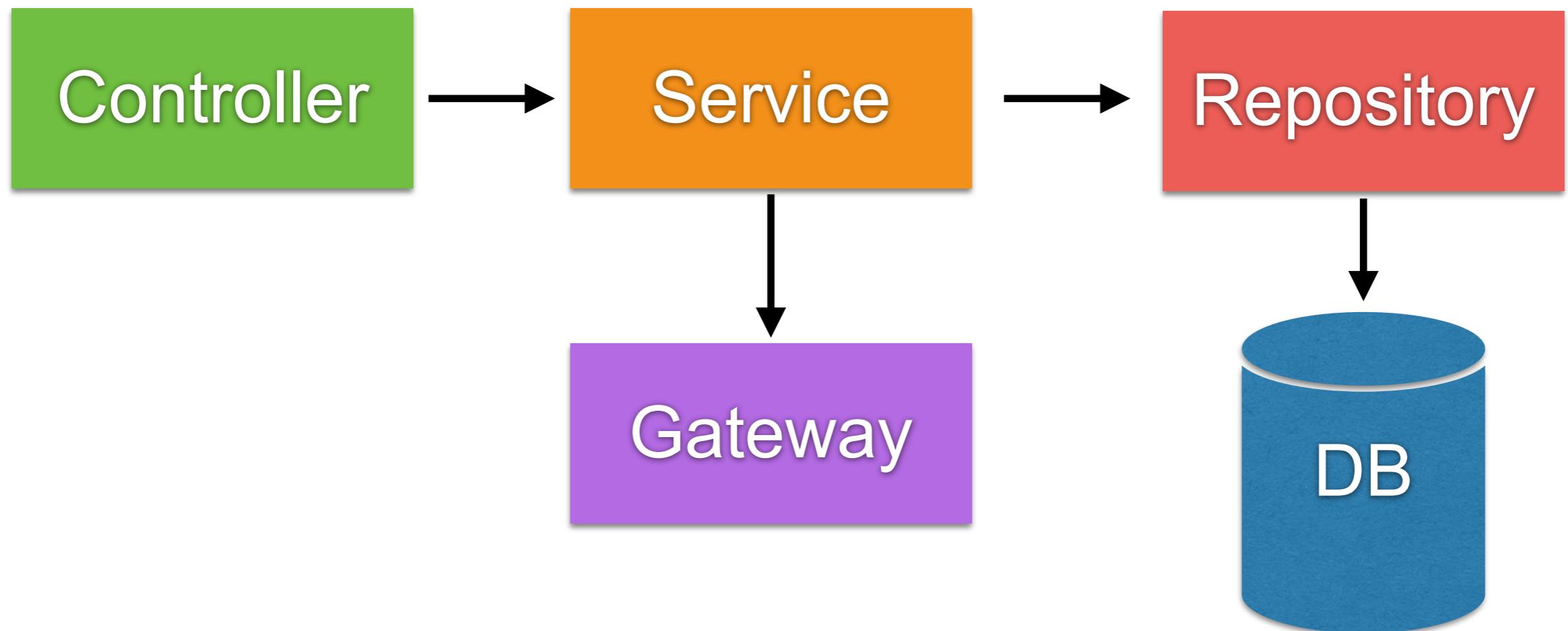
Project structure



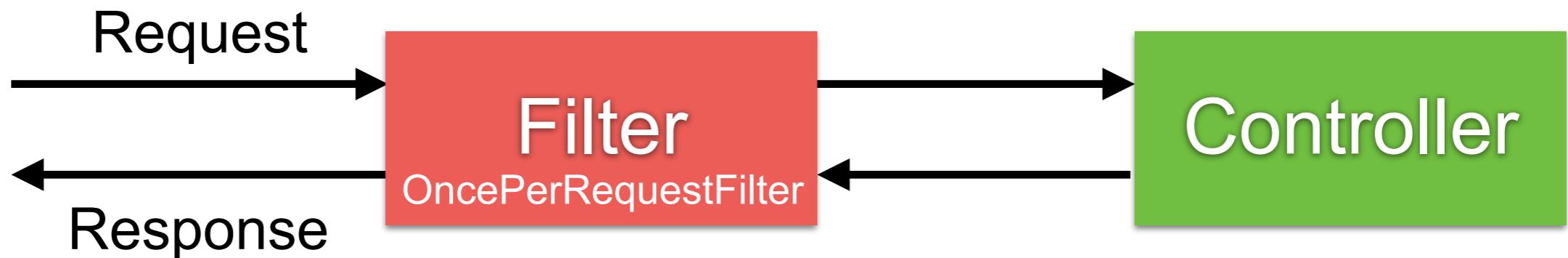
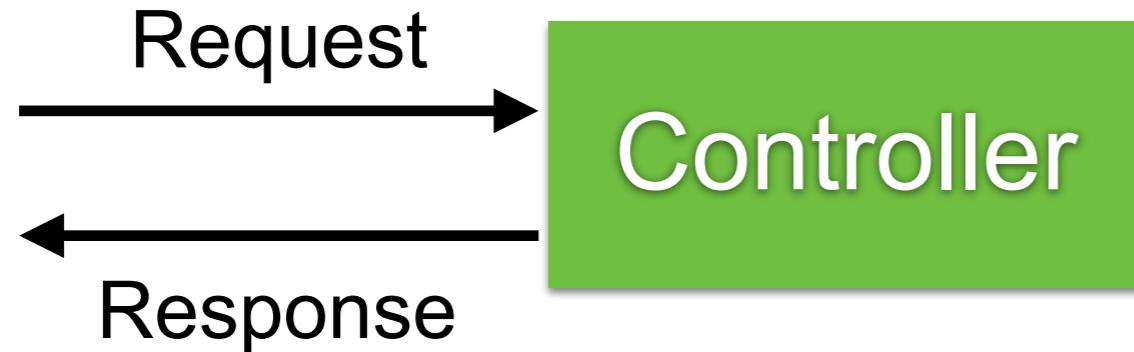
How to test ?



How to run and deploy ?



Request Filter



Assignment

Git commit message

API design + WIKI

Project structure

Testing

Running

GitHub Actions => build, test and package

Next ... Async/Reactive

Next ... security