Spring Fundamentals

Spring Boot Introduction

SoftUni Team

Technical Trainers

Software University

https://softuni.bg

Table of Contents

1. What's Spring Boot?
2. Spring Data

Questions

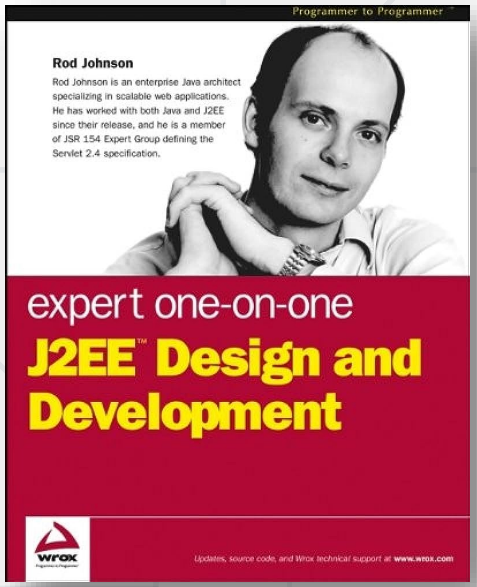
sli.do

#java-web

What is Spring Boot?

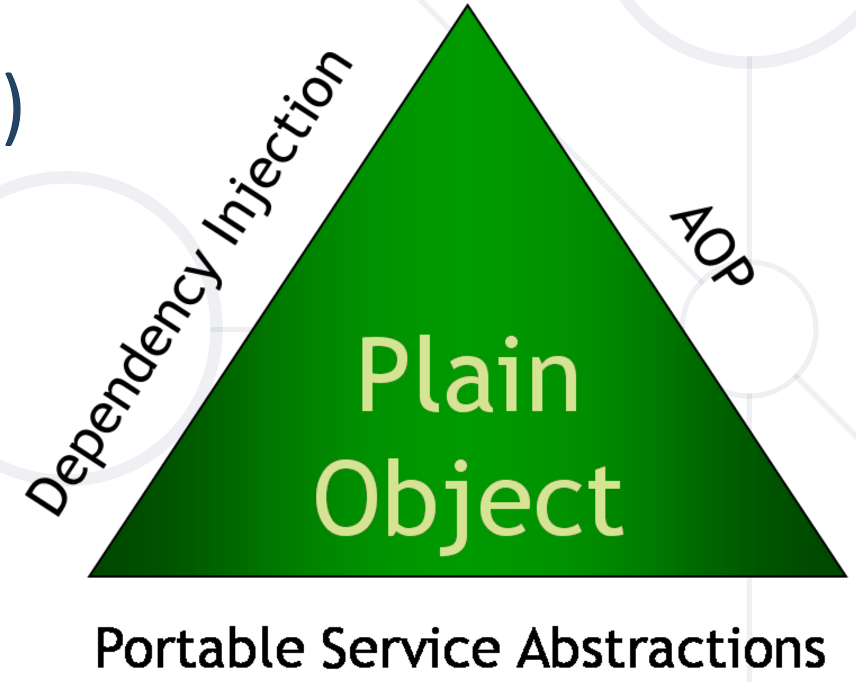
Spring

- In October 2002, Rod Johnson wrote a book titled “Expert One-on-One J2EE Design and Development”
- The book was accompanied by 30,000 lines of framework code also known as Interface21 (Spring 0.9).
- Since java Interfaces were the basic building blocks of dependency injection (DI), he named the root package of the classes as com.interface21.
- Shortly after the release of the book, developers Juergen Hoeller and Yann Caroff persuaded Rod Johnson to create an open source project based on the infrastructure code. In March 2004, spring 1.0 was released.


Programmer to Programmer
Rod Johnson
Rod Johnson is an enterprise Java architect specializing in scalable web applications. He has worked with both Java and J2EE since their release, and he is a member of JSR 154 Expert Group defining the Servlet 2.4 specification.
expert one-on-one
J2EE Design and Development
wrox
Updates, source code, and Wrox technical support at www.wrox.com

Spring Main Concepts

- The four key concepts are:
  - Plain CLR objects
  - Dependency Injection (DI)
  - AOP (Aspect Oriented Programming)
  - Portable Service Abstractions



Inversion of Control

- Spring provides **Inversion of Control** and **Dependency Injection**

UserServiceImpl.java

```java
//Traditional Way
public class UserServiceImpl implements
UserService {

private UserRepository userRepository = new
UserRepositoryImpl();

}
```

UserServiceImpl.java

```java
//Dependency Injection
@Service
public class UserServiceImpl implements
UserService {

@Autowired
private UserRepository userRepository;
}
```

Spring IoC

Meta Data:

1. XML Config
2. Java Config

Configuration

Automatic Beans:

1. @Component
2. @Service
3. @Repository

Explicit Beans

1. @Bean

IoC

Fully Configured System

Beans

- Object that is **instantiated**, **assembled**, and otherwise managed by a **Spring IoC** container

Dog.java

```
public class Dog implements Animal {

    private String name;

    public Dog() {}

    //GETTERS AND SETTERS
}
```

Bean Declaration

MainApplication.java

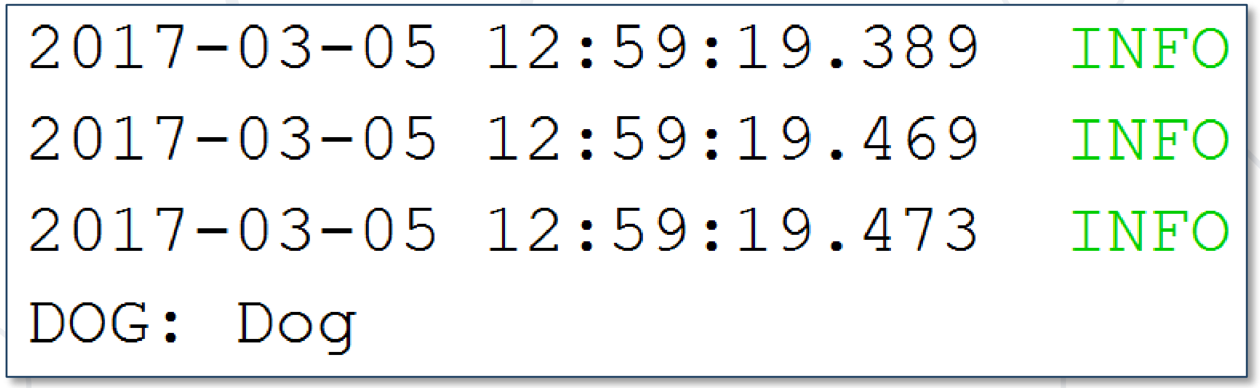
```
@SpringBootApplication
public class MainApplication {

    …

    @Bean
    public Animal getDog(){
        return new Dog();
    }
}
```

Bean Declaration

Get Bean from Application Context

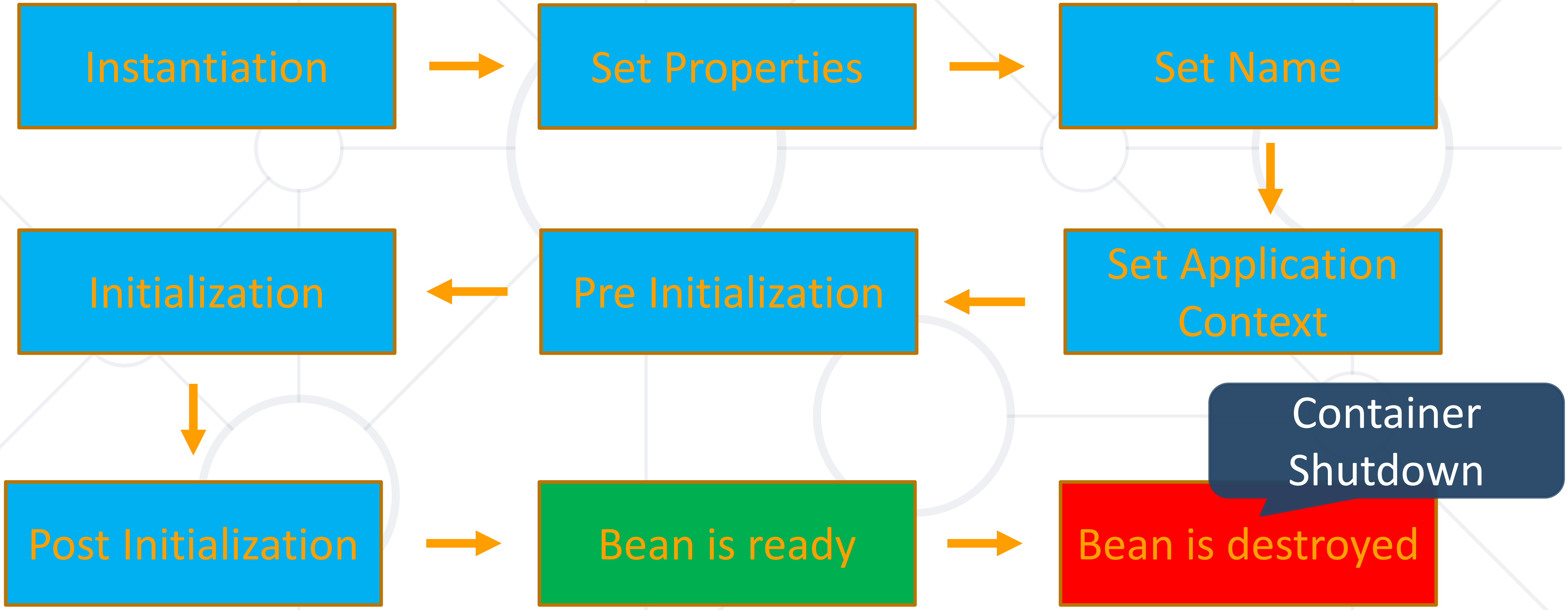
MainApplication.java

```java
@SpringBootApplication
public class MainApplication {
  public static void main(String[] args) {
    ApplicationContext context = SpringApplication.run(MainApplication.class, args);
    Animal dog = context.getBean(Dog.class);
    System.out.println("DOG: " + dog.getClass().getSimpleName());
    }
}
```



```
2017-03-05 12:59:19.389  INFO
2017-03-05 12:59:19.469  INFO
2017-03-05 12:59:19.473  INFO
DOG: Dog
```

Bean Lifecycle



Bean Lifecycle Demo (1)

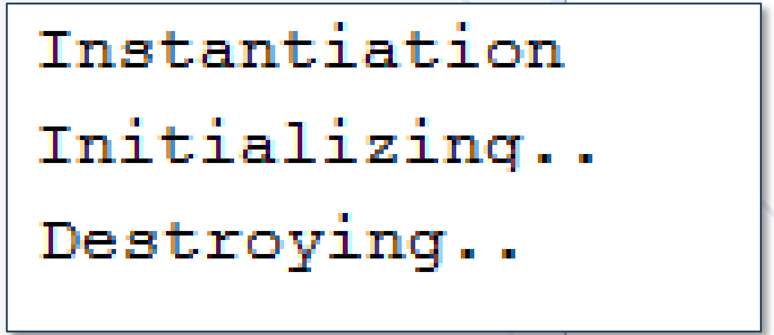
MainApplication.java

```
@SpringBootApplication
public class MainApplication {

 public static void main(String[] args) {
        ApplicationContext context =
       SpringApplication.run(MainApplication.class, args);
        ((AbstractApplicationContext)context).close();
 }
 @Bean(destroyMethod = "destroy", initMethod = "init")
 public Animal getDog(){
     return new Dog();
 }
}
```

Bean Lifecycle Demo (2)

MainApplication.java

```java
public class Dog implements Animal {

    public Dog() {
        System.out.println("Instantiation");
    }

    public void init(){
        System.out.println("Initializing..");
    }

    public void destroy(){
        System.out.println("Destroying..");
    }
}
```



```
Instantiation
Initializing..
Destroying..
```

PostConstruct Annotation

- Spring calls methods annotated with ***@PostConstruct*** only once, just after the initialization of bean

```
@Component
public class DbInit {
    private final UserRepository userRepository;
    public DbUnit(UserRepository userRepository)
       { this. userRepository = userRepository;}

    @PostConstruct
    private void postConstruct() {
        User admin = new User("admin", "admin password");
        User normalUser = new User("user", "user password");
        userRepository.save(admin, normalUser);
    }
}
```

PreDestroy Annotation

- A method annotated with **@PreDestroy** runs only once, just before Spring removes our bean from the application context

```
@Component
public class UserRepository {

    private DbConnection dbConnection;
    @PreDestroy
    public void preDestroy() {
        dbConnection.close();
    }
}
```

BeanNameAware Interface

- BeanNameAware makes the object aware of the bean name defined in the container

```
public class MyBeanName implements BeanNameAware {
    @Override
    public void setBeanName(String beanName) {
        System.out.println(beanName);
    }
}
```

```
@Configuration
public class Config {
    @Bean (name = "myCustomBeanName")
    public MyBeanName getMyBeanName() {
        return new MyBeanName();
    }
}
```

BeanFactoryAware Interface

- BeanFactoryAware is used to **inject** the **BeanFactory object**
- With the **setBeanFactory()** method, we assign the BeanFactory reference from the IoC container to the beanFactory property

```java
public class MyBeanFactory implements BeanFactoryAware {
    private BeanFactory beanFactory;
    @Override
    public void setBeanFactory(BeanFactory beanFactory)throws BeansException {
        this.beanFactory = beanFactory;}
    public void getMyBeanName() {
        MyBeanName myBeanName = beanFactory.getBean(MyBeanName.class);
        System.out.println(beanFactory.isSingleton("myCustomBeanName"));
    }
}
```

InitializingBean Interface

- For bean implemented **InitializingBean**, it will run **afterPropertiesSet()** after all bean properties have been set

```
@Component
public class InitializingBeanExampleBean implements InitializingBean {
    private static final Logger LOG
      = Logger.getLogger(InitializingBeanExampleBean.class);

    @Autowired
    private Environment environment;

    @Override
    public void afterPropertiesSet() throws Exception {
        LOG.info(Arrays.asList(environment.getDefaultProfiles()));
    }
}
```

DisposableBean Interface

- For bean implemented **DisposableBean**, it will run **destroy()** after Spring container is released the bean

```
@Component
public class Bean2 implements DisposableBean {

    @Override
    public void destroy() throws Exception {
        System.out.println(
          "Callback triggered - DisposableBean.");
    }
}
```

Beans Scopes in Spring Framework

- There are part of Beans scopes:
  - Singleton
  - Prototype
  - Request
  - Session

Singletone Scope

- Container creates a **single instance** of that bean, and all requests for that bean name will return the **same object**, which is cached
- This is **default** scope

```
@Bean
@Scope("singleton") <- Can be omitted
public Student student() {
    return new Student();
}
```

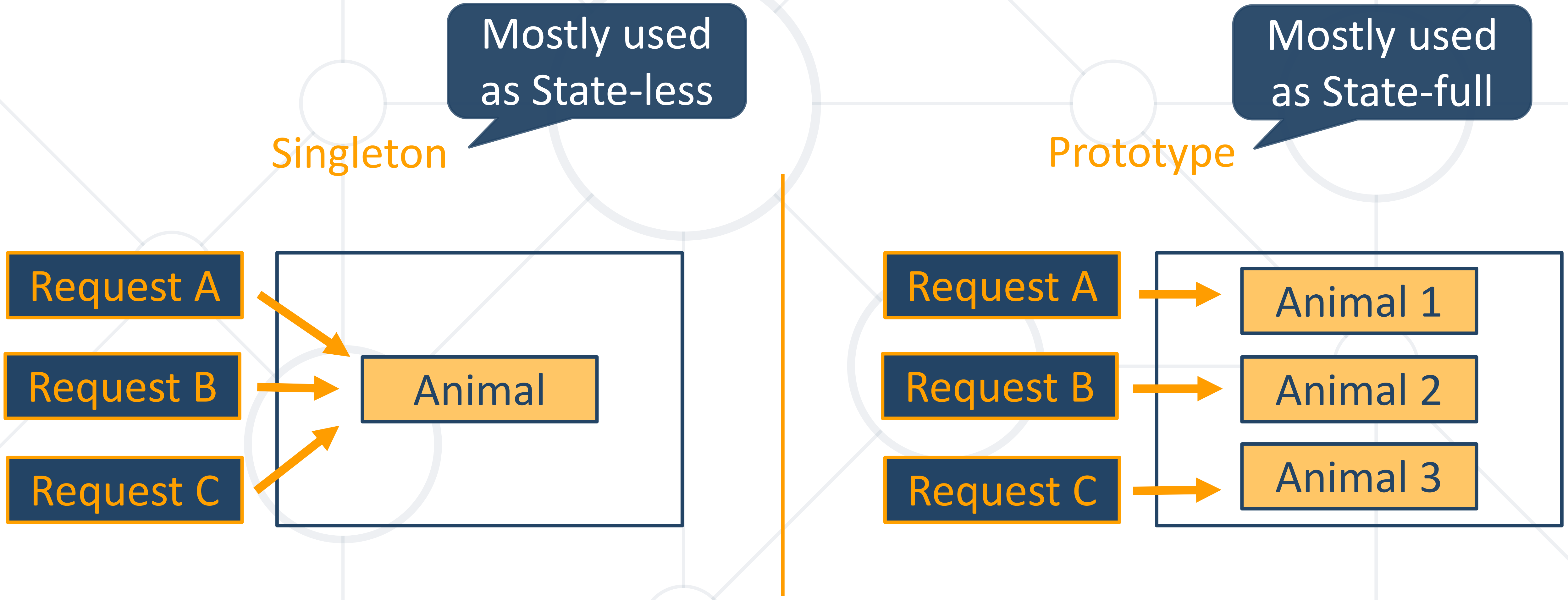
Prototype Scope

- Will return a different instance every time it is requested from the container

```
@Bean
@Scope("prototype")
public Student student() {
    return new Student();
}
```

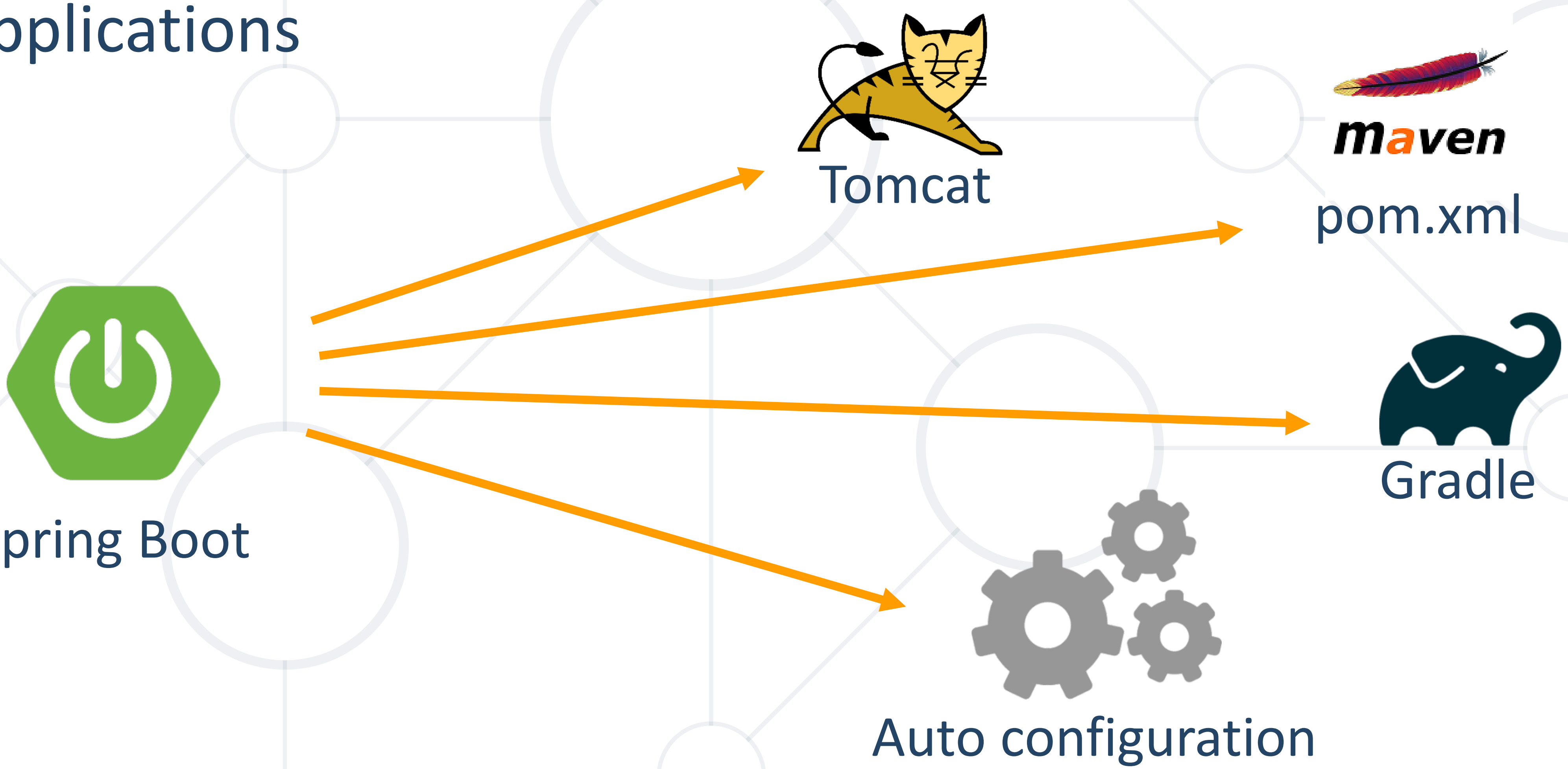
Bean Scope

- The default one is **Singleton**. It is easy to change to **Prototype**



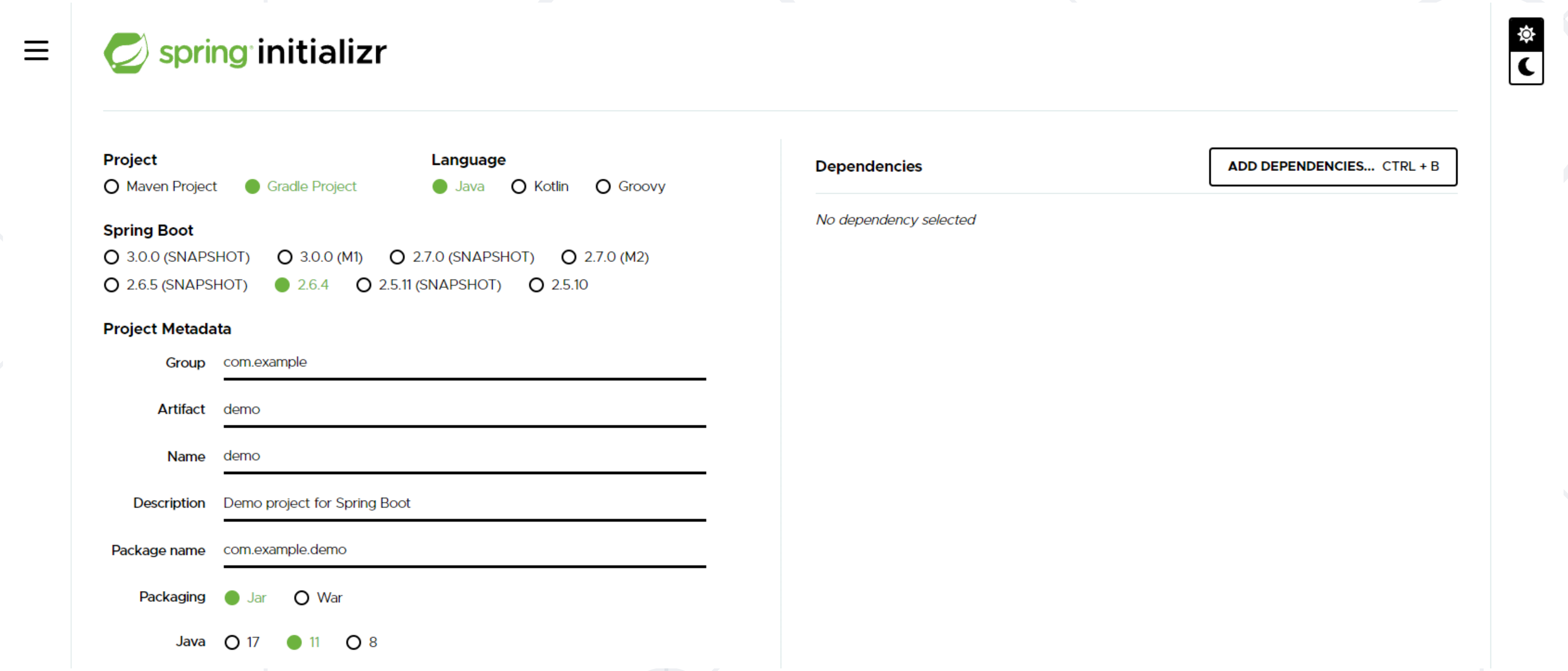
Spring Boot

- **Opinionated view** of building production-ready Spring applications



Creating Spring Boot Project

- Just go to https://start.spring.io/



spring initializr

Project
Maven Project [✓ Gradle Project]

Language
[✓ Java] Kotlin Groovy

Spring Boot
3.0.0 (SNAPSHOT) 3.0.0 (M1) 2.7.0 (SNAPSHOT) 2.7.0 (M2)
2.6.5 (SNAPSHOT) [✓ 2.6.4] 2.5.11 (SNAPSHOT) 2.5.10

Project Metadata

Group	com.example
Artifact	demo
Name	demo
Description	Demo project for Spring Boot
Package name	com.example.demo
Packaging	[✓ Jar] War
Java	17 [✓ 11] 8

Dependencies

No dependency selected

Spring Dev Tools

- Additional set of **tools** that can make the application development **faster** and more **enjoyable**
- In **Maven**:

pom.xml

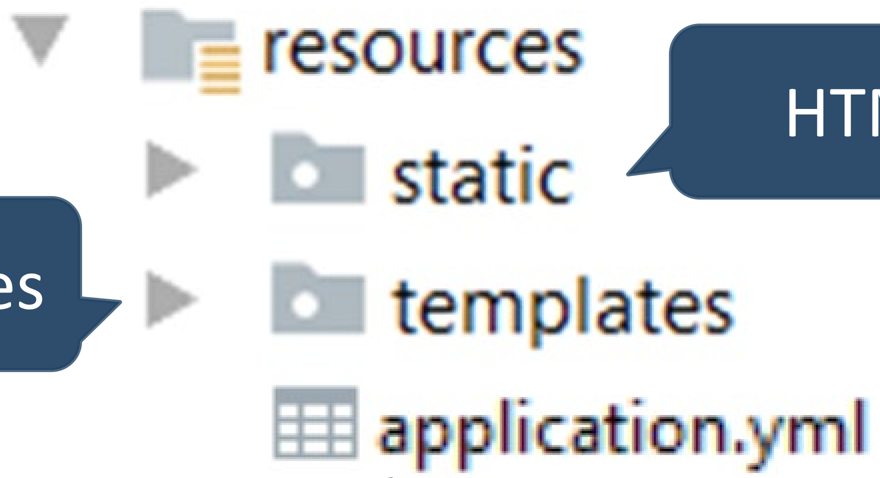
```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-devtools</artifactId>
    <scope>runtime</scope>
</dependency>
```

- In **Gradle**:

build.gradle

```
dependencies {
    compileOnly("org.springframework.boot:spring-boot-devtools")
}
```

Spring Resources



```
resources
    static
    templates
    application.yml
```

HTML, CSS, JS

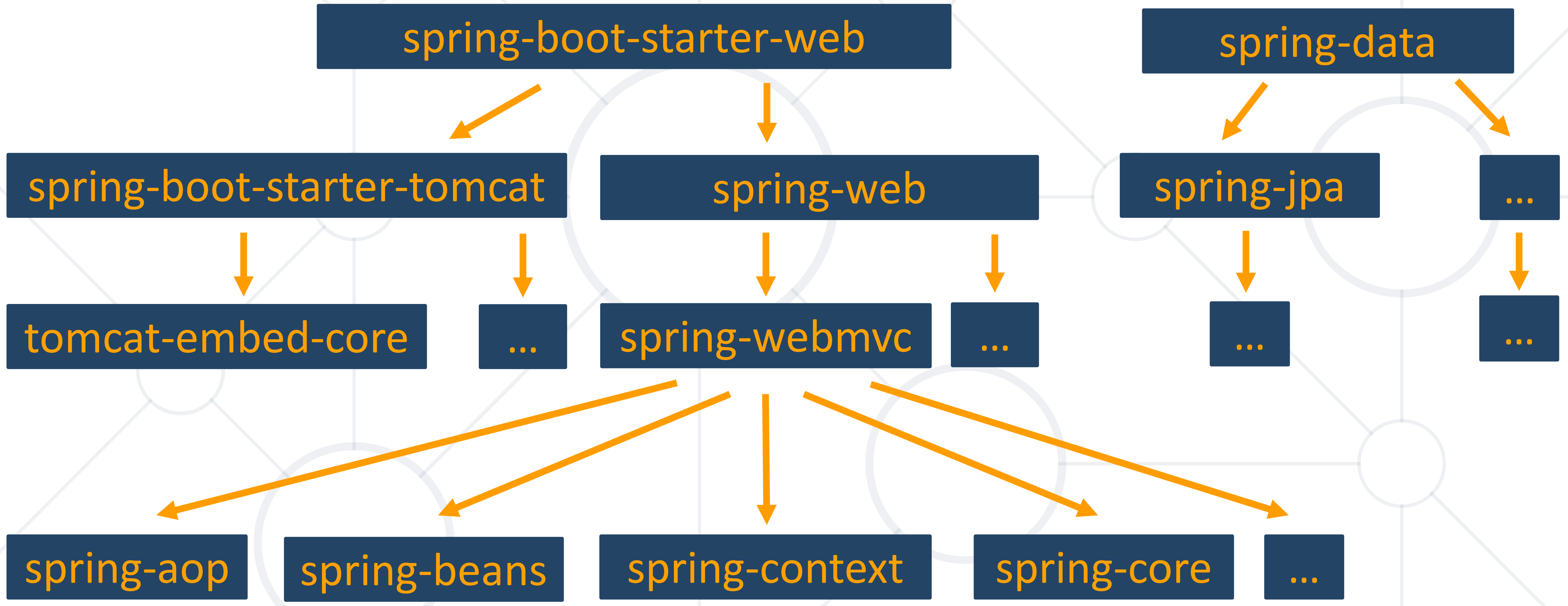
Thymeleaf templates

Yaml configuration

Spring Boot Main Components

- Some main components:
  - **Spring Boot Starters** - combine a group of common or related dependencies into single dependency
  - **Spring Boot Auto-Configuration** - reduce the Spring Configuration
  - **Spring Boot Actuator** – provides EndPoints and Metrics
  - **Spring Data** – unify and ease the access to different kinds of database systems

Spring Boot Starters

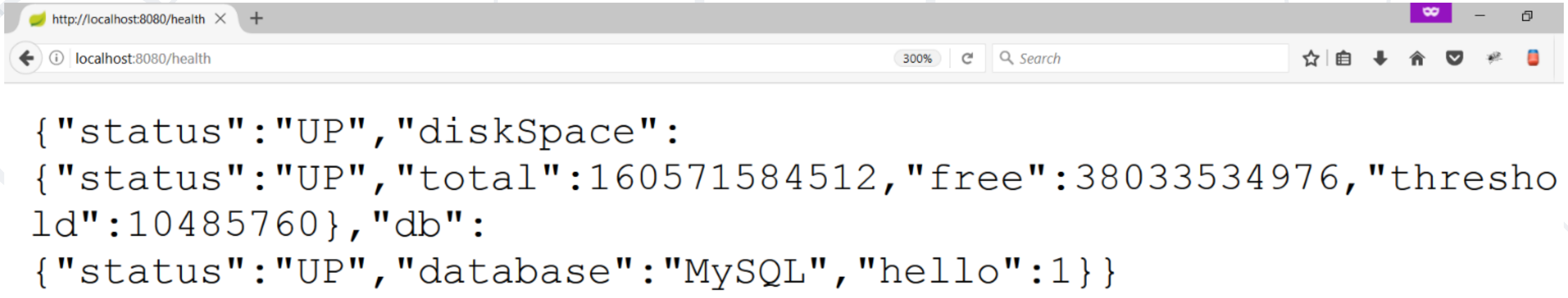


Spring Boot Actuator

- Expose different types of information about the **running application**

build.gradle

```
dependencies {
    compileOnly("org.springframework.boot:spring-boot-starter-actuator")
}
```



```
{"status":"UP","diskSpace":
{"status":"UP","total":160571584512,"free":38033534976,"thresho
ld":10485760},"db":
{"status":"UP","database":"MySQL","hello":1}}
```

Common Application Properties

- Various properties can be specified inside your **application.yaml** file
- Property contributions can come from **additional jar files**
- You can define your **own properties**
- **Link to documentation**

Application Properties Example

application.properties

```
spring.datasource.driverClassName=com.mysql.cj.jdbc.Driver
spring.datasource.url=jdbc:mysql://localhost:3306/thymeleaf_adv_lab_exam_db?c
reateDatabaseIfNotExist=true
spring.datasource.username=root
spring.datasource.password=12345
spring.jpa.properties.hibernate.dialect = org.hibernate.dialect.MySQL8Dialect
spring.jpa.properties.hibernate.format_sql = TRUE
spring.jpa.hibernate.ddl-auto = update
spring.jpa.open-in-view=false
logging.level.org = WARN
logging.level.blog = WARN
logging.level.org.hibernate.SQL = DEBUG
logging.level.org.hibernate.type.descriptor = TRACE
server.port=8000
```

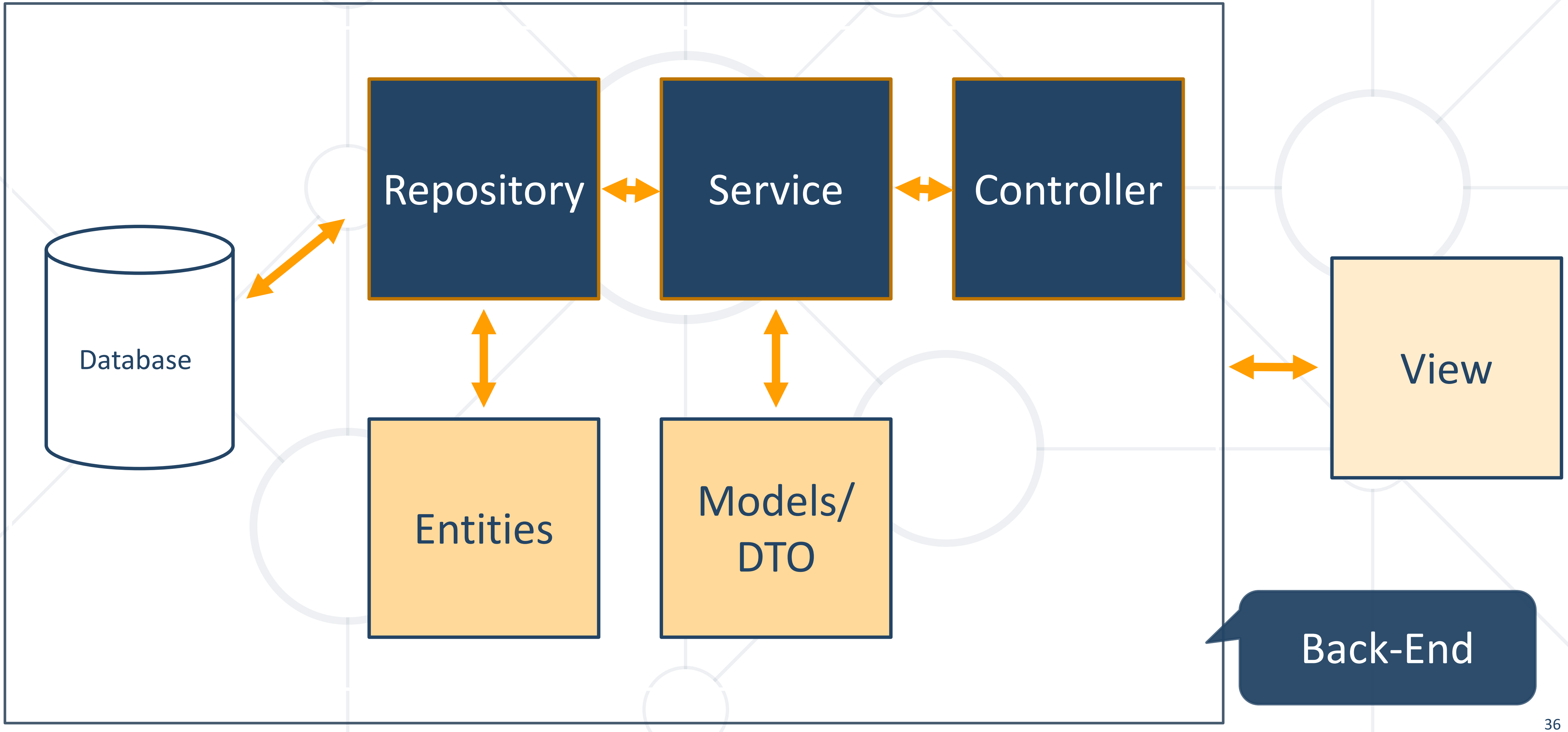
Application Yaml Example

application.yaml

```
spring:
  datasource:
    driverClassName: com.mysql.cj.jdbc.Driver
    password: 12345
    url:
jdbc:mysql://localhost:3306/spring_data_lab_db?allowPublicKeyRetrieval=true&useSSL=fa
lse&createDatabaseIfNotExist=true
    username: root
  jpa:
  database-platform: org.hibernate.dialect.MySQL8Dialect
  hibernate:
      ddl-auto: create-drop
      open-in-view: false
      properties:
      hibernate:
        format_sql: true
```

Spring Data

Overall Architecture



Entities

- Entity is a lightweight persistence domain object

Cat.java

```java
@Entity
@Table(name = "cats")
public class Cat {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private long id;

    private String name;
    //GETTERS AND SETTERS
}
```

Repositories

- **Persistence** layer that works with **entities**

CatRepository.java

```
@Repository
public interface CatRepository extends JpaRepository<Cat, Long> {
}
```

Services

- **Business Layer** - All the business logic is here.

CatService.java

```
@Service
public class CatServiceImpl implements CatService {
    private final CatRepository catRepository;
    @Autowired
    public CatServiceImpl(CatRepository catRepository){
      this.catRepository = catRepository;
    }
    @Override
    public void buyCat(CatModel catModel) { //TODO Implement the method }
}
```

Summary

- **Spring Boot** - **Opinionated view** of building production-ready Spring applications
- **Spring Data** - Responsible for database related operations

Questions?

SoftUni

Software University

SoftUni Creative

SoftUni Digital

SoftUni Foundation

SoftUni Kids

Finance Academy

SoftUni Diamond Partners

Software University

SUPER HOSTING .BG

Coca-Cola HBC Bulgaria

Flutter™ International

INDEAVR Serving the high achievers

AMBITIONED

DRAFT KINGS

SOFTWARE GROUP

BOSCH

Postbank Решения за твоето утре

PHAR VISION

SmartIT

DXC TECHNOLOGY

createX

Trainings @ Software University (SoftUni)

- Software University – High-Quality Education, Profession and Job for Software Developers
  - softuni.bg, about.softuni.bg
- Software University Foundation
  - softuni.foundation
- Software University @ Facebook
  - facebook.com/SoftwareUniversity
- Software University Forums
  - forum.softuni.bg

License