

Spring Boot and Angular

Dr. Ernesto Lee

Introduction to Spring Boot

- Open source micro framework from Pivotal.
- Enterprise-level framework for standalone applications on JVMs.
- Primary focus: Shorten code for easier application runs.
- Extends the Spring Framework:
 - Provides an opinionated way of configuring applications.
- Built-in autoconfiguration capabilities:
 - Configures Spring Framework and third-party packages based on user settings.
- Aims to reduce boilerplate code during setup.
- Helps avoid configuration errors.

Advantages of SpringBoot

- Autoconfiguration
- Opinionated
- Spring Starters
- Create Standalone Apps



[This Photo](#) by Unknown Author is licensed under [CC BY](#)

Differences Between Spring and Spring Boot

- **Configuration:**
 - Spring: Manual dependency setup.
 - Spring Boot: Automatic with Spring Starters.
- **Usage:**
 - Spring: Builds Java EE applications.
 - Spring Boot: Common for REST APIs.
- **Features:**
 - Spring: Emphasizes DI and IOC.
 - Spring Boot: Introduces Spring Boot Actuator.
- **Framework:**
 - Spring Boot is an extension of Spring.
- **Learning Curve:**
- No need to master Spring to use Spring Boot.

Minimum Dependency Needed for Spring

- `<dependency>`
- `<groupId>org.springframework</groupId>`
- `<artifactId>spring-web</artifactId>`
- `<version>5.3.5</version>`
- `</dependency>`
- `<dependency>`
- `<groupId>org.springframework</groupId>`
- `<artifactId>spring-webmvc</artifactId>`
- `<version>5.3.5</version>`
- `</dependency>`

```
<dependency>  
<groupId>org.springframework.boot</groupId>  
<artifactId>spring-boot-starter-web</artifactId>  
<version>2.4.4</version>  
</dependency>
```

Spring Configurations

```
public class SpringInitializer implements WebApplicationInitializer {  
    @Override  
    public void onStartup(ServletContext container) {  
        AnnotationConfigWebApplicationContext context =  
            new AnnotationConfigWebApplicationContext();  
        context.setConfigLocation("com.springexample");  
        container.addListener(new ContextLoaderListener(context));  
        ServletRegistration.Dynamic dispatcher =  
            container.addServlet("dispatcher",  
                new DispatcherServlet(context));  
        dispatcher.setLoadOnStartup(1);  
        dispatcher.addMapping("/");  
    }  
}
```

More Spring Configurations...

@EnableWebMvc

@Configuration

public class SpringWebConfig implements WebMvcConfigurer {

 @Bean

 public ViewResolver viewResolver() {

 InternalResourceViewResolver bean =

 new InternalResourceViewResolver();

 bean.setViewClass(JstlView.class);

 bean.setPrefix("/WEB-INF/view/");

 bean.setSuffix(".jsp");

 return bean;

 }

}

Spring Boot

- In Spring Boot, all these configurations will be omitted because this code is already included in the Spring Starters.
- We only need to define some properties for our application to run using the web starter:

```
spring.mvc.view.prefix=/WEB-INF/jsp/
```

```
spring.mvc.view.suffix=.jsp
```

Now for Angular

- **Overview:**

- Free and open source JavaScript framework maintained by Google.
- Primarily for web applications; also supports mobile and desktop apps using plugins.
- Component-based architecture.

- **Features:**

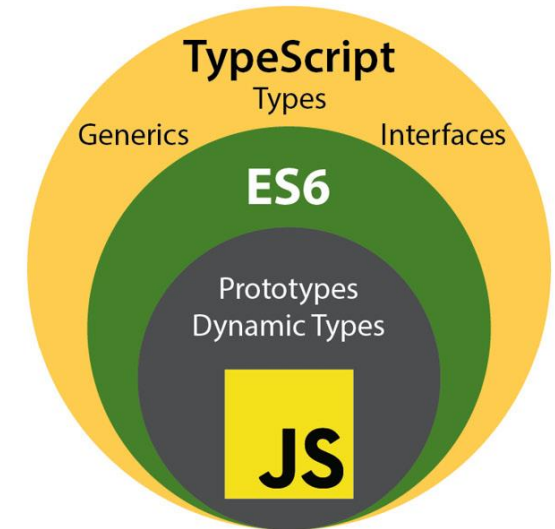
- Progressive framework with extensive libraries and extensions.
- Reduces development time for large-scale applications.

- **Popularity & Usage:**

- Preferred framework for giants like Samsung, Upwork, PayPal, and Google.
- 76,000 stars on GitHub with ~1,500 contributors.
- Thousands of functional NPM libraries for enhanced development.

Why Angular?

- 1.TypeScript-Based:**
- 2.Object-Oriented Programming (OOP):**
- 3.Built-in Features:**
- 4.Progressive Web Apps (PWAs):**
- 5.Angular CLI:**
- 6.Modular and Component-Based:**
- 7.Cross-Platform Capability:**
- 8.Web Components (Angular Elements):**
- 9.Lazy Loading:**



Summary

- **Spring Boot:**

- An open source extension of the Spring Framework.
- Aims to eliminate boilerplate configuration code.
- Offers Spring Starters for automatic dependency configuration.

- **Angular:**

- A component-based framework utilizing TypeScript.
- Embraces Object-Oriented Programming (OOP) principles.
- Provides cross-platform support: web, mobile, and desktop applications.
- Backed by Google and widely adopted by prominent companies, supported by a vast community.

SpringBoot

Recap of Spring Boot

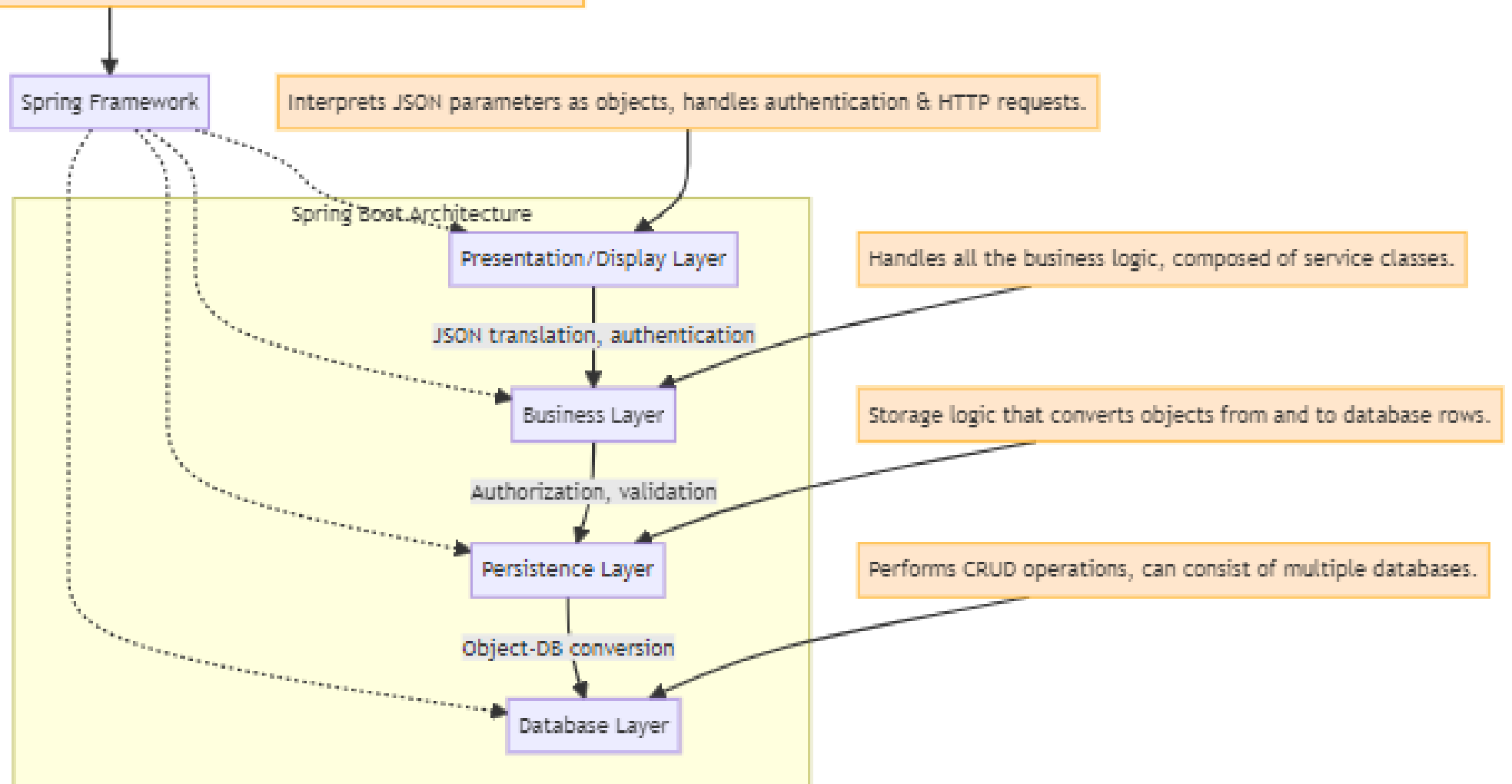
- **What is Spring Boot?**

- An open source micro-framework by Pivotal.
- Designed for standalone applications on JVMs.
- Aims to reduce code length for easier application runs.
- An extension of the Spring Framework, offering an opinionated configuration approach.
- Comes with autoconfiguration capabilities for both Spring Framework and third-party packages.

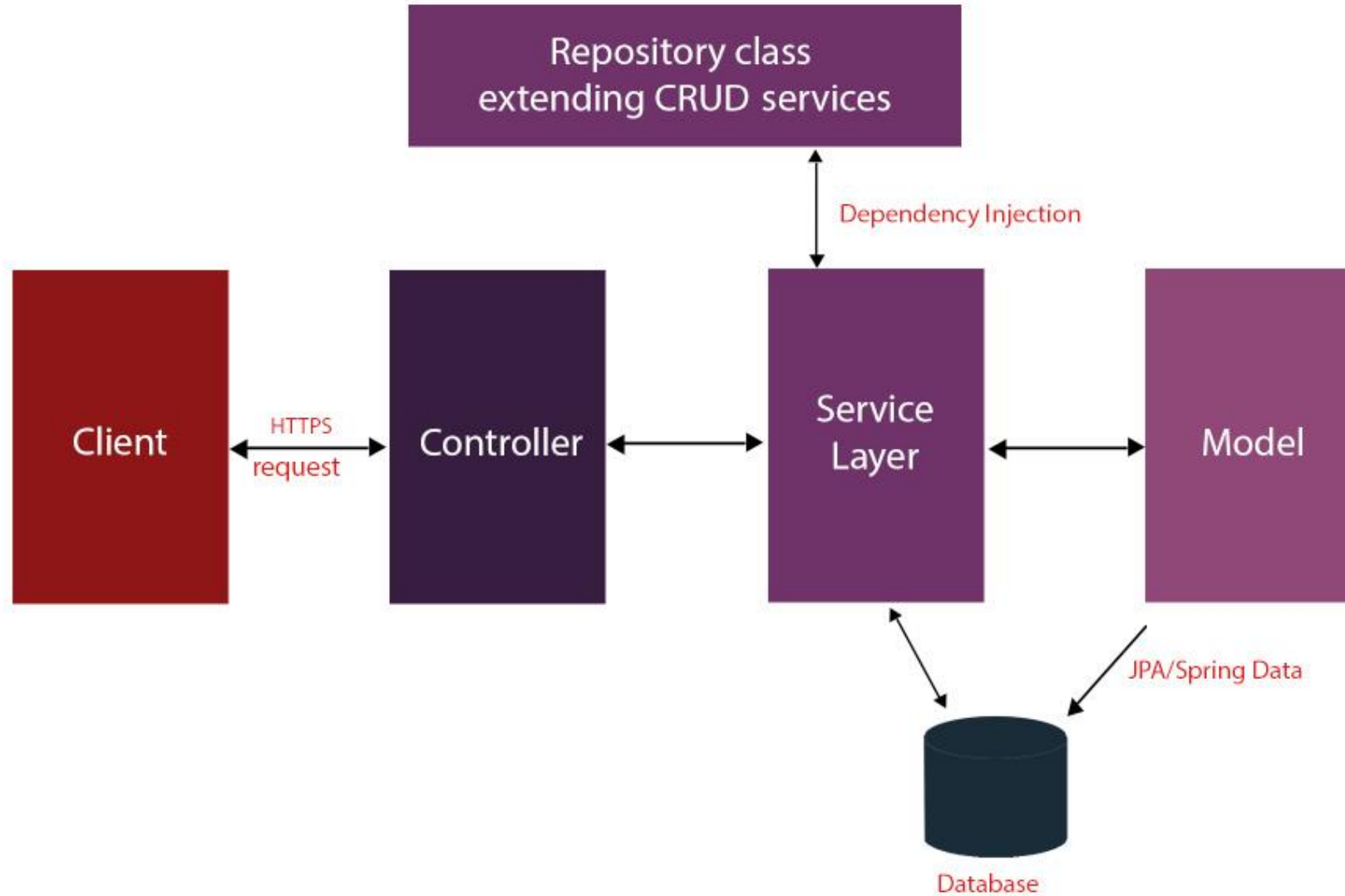
- **Key Advantages of Spring Boot:**

- **Auto-configuration:** Automatically downloads essential dependencies.
- **Opinionated Approach:** Streamlines dependency installation, minimizing manual configuration.
- **Spring Starters:** Allows selection of starter dependencies for application needs, e.g., Spring Web for web applications.

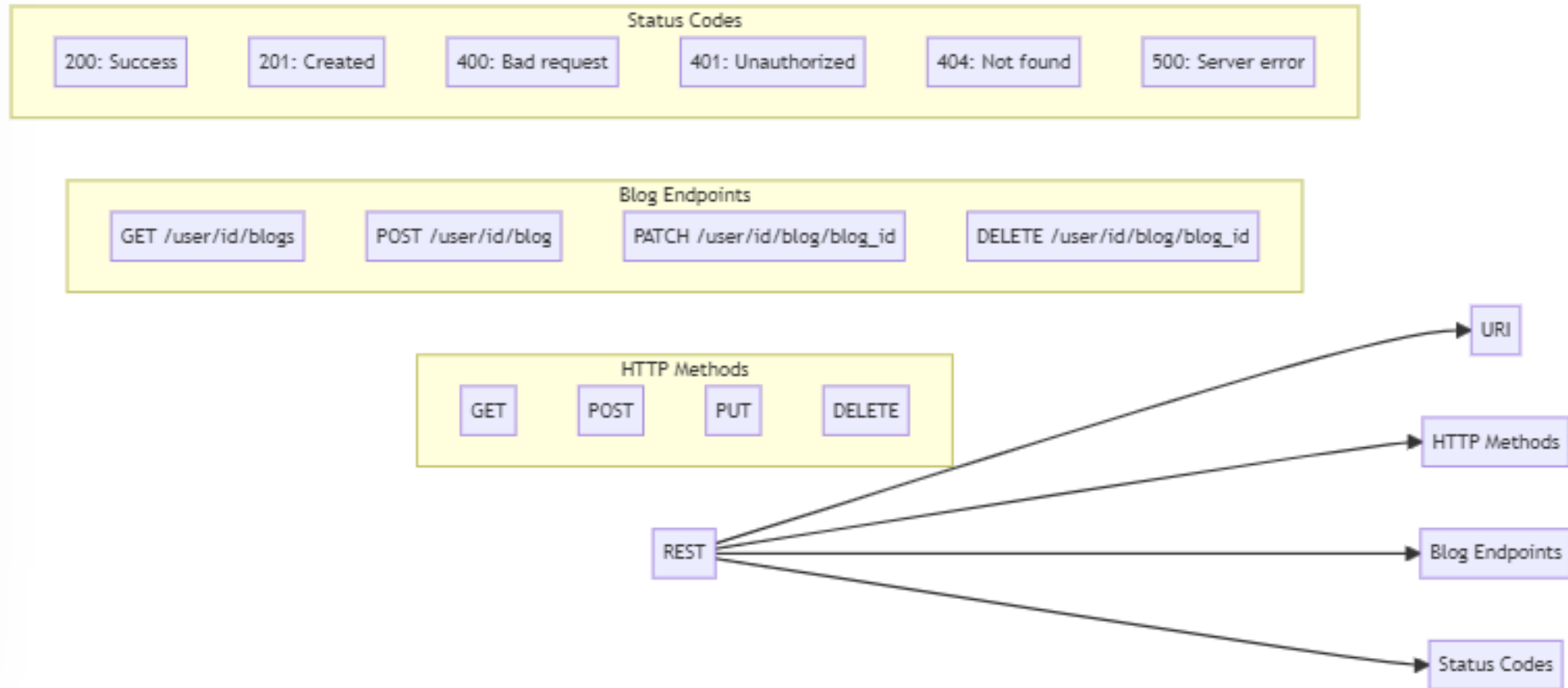
Spring Boot depends on the Spring Framework but does not require DAO and DAOimpl classes.

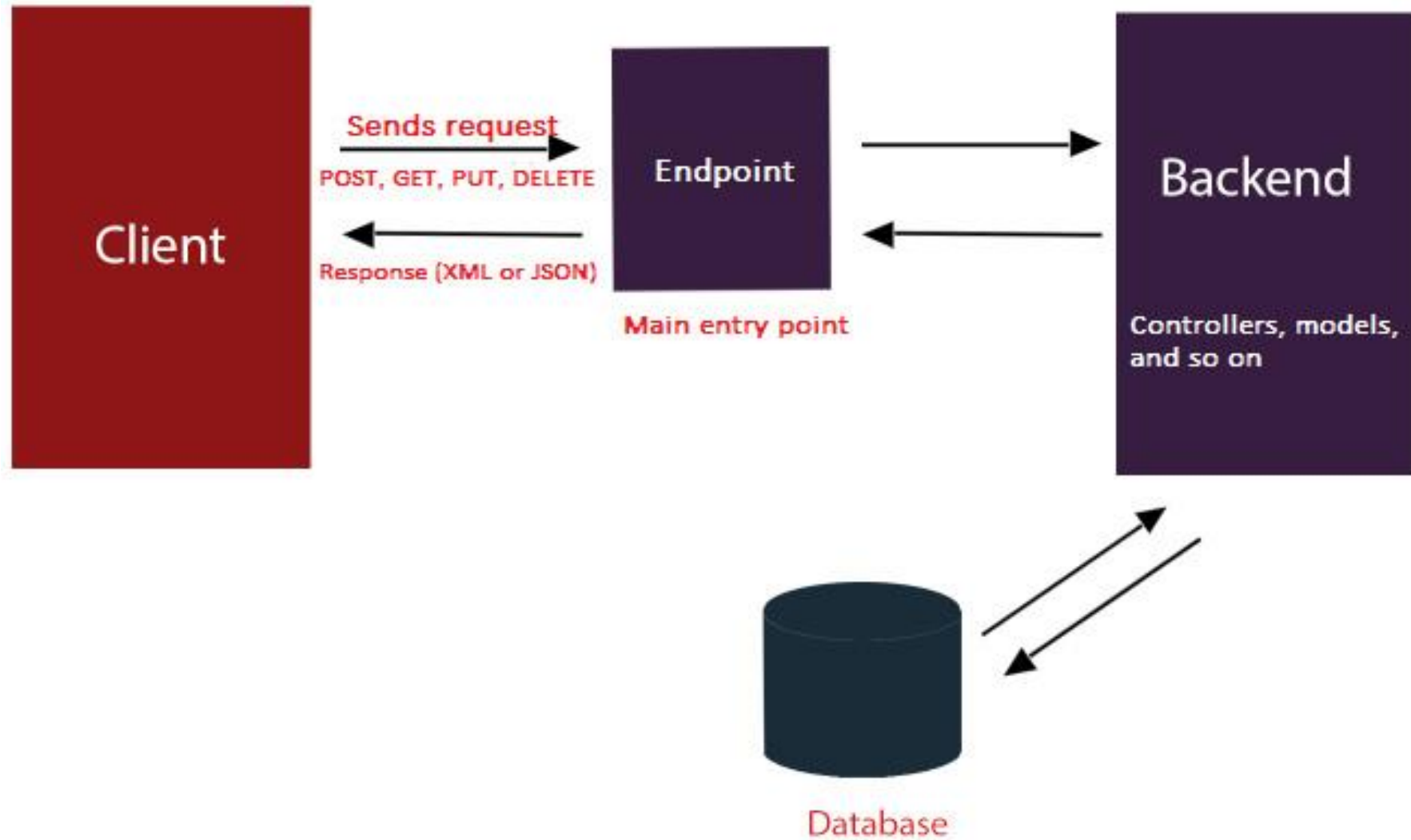


Spring Boot Flow



REST





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



The image shows the Spring Initializer web form. It is divided into several sections: Project, Language, Spring Boot, Project Metadata, Dependencies, and a footer with social media icons and action buttons.

Project

☐ Maven Project ☒ Gradle Project

Language

☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (RC1) ☐ 2.5.7 (SNAPSHOT) ☒ 2.5.6 ☐ 2.4.13 (SNAPSHOT) ☐ 2.4.12

Project Metadata

Group:

Artifact:

Name:

Description:

Package name:

Packaging: ☒ Jar ☐ War

Java: ☐ 17 ☒ 11 ☐ 8

Dependencies




No dependency selected

Footer

Social media icons: GitHub, Twitter

Theme toggle: ☒ Light ☐ Dark

Dependencies



Project
☒ Maven Project
☐ Gradle Project

Language
☒ Java ☐ Kotlin ☐ Groovy

Spring Boot
☒ 2.6.0 (SNAPSHOT) ☐ 2.6.0 (RC1) ☐ 2.5.7 (SNAPSHOT) ☐ 2.5.6
☐ 2.4.13 (SNAPSHOT) ☐ 2.4.12

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

Lombok **DEVELOPER TOOLS**
Java annotation library which helps to reduce boilerplate code.

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.

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

Spring Data Reactive Redis **NOSQL**
Access Redis key-value data stores in a reactive fashion with Spring Data Redis.

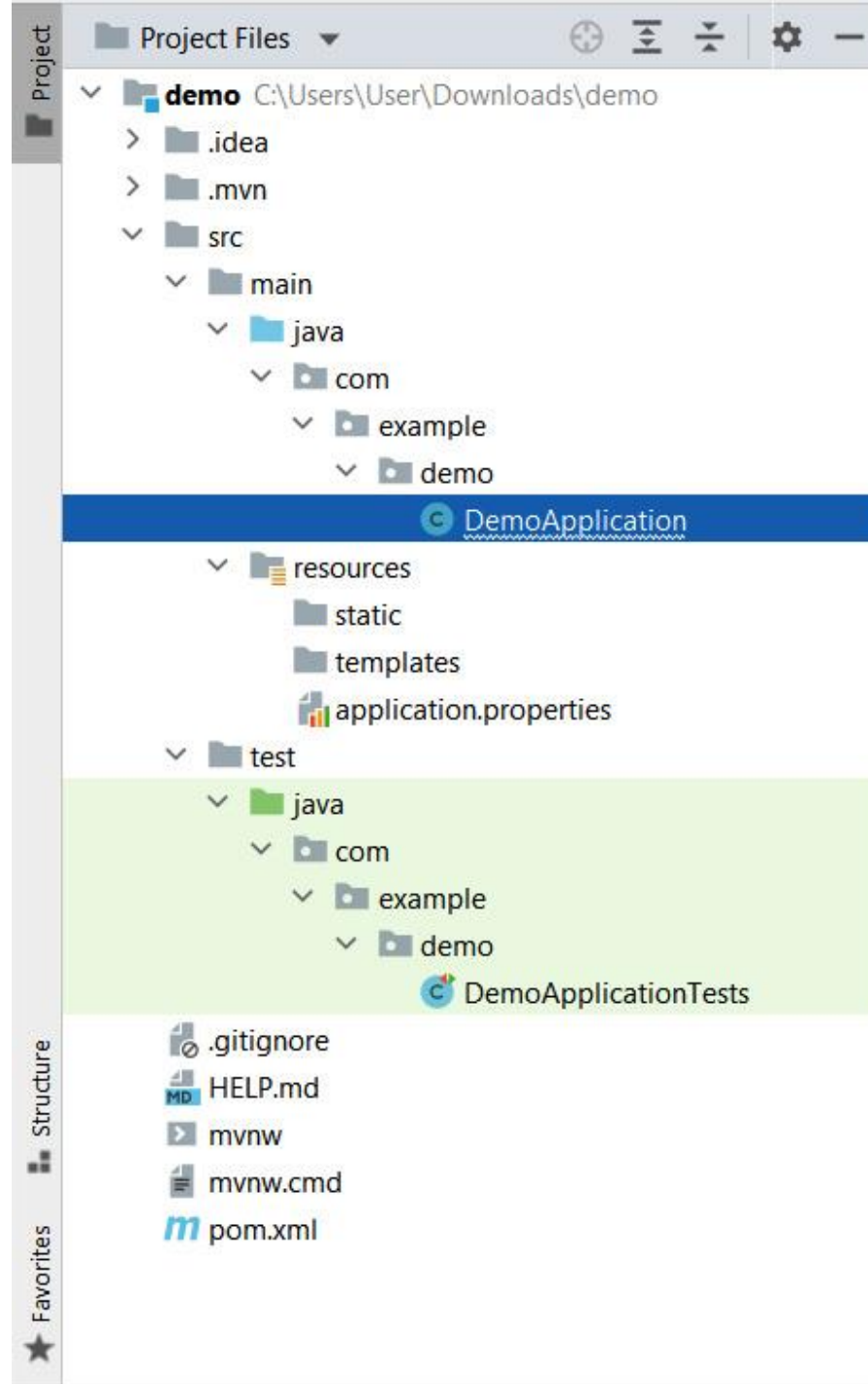
Spring Security **SECURITY**
Highly customizable authentication and access-control framework for Spring applications.

GENERATE CTRL + G

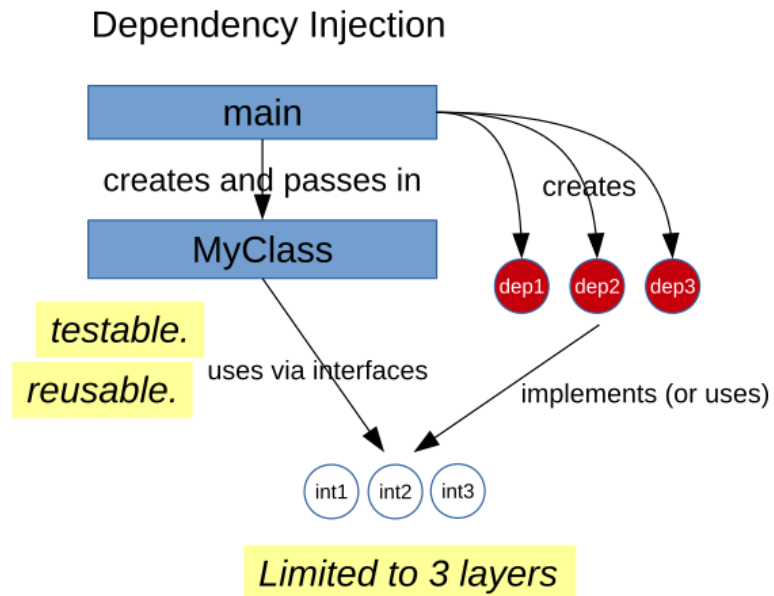
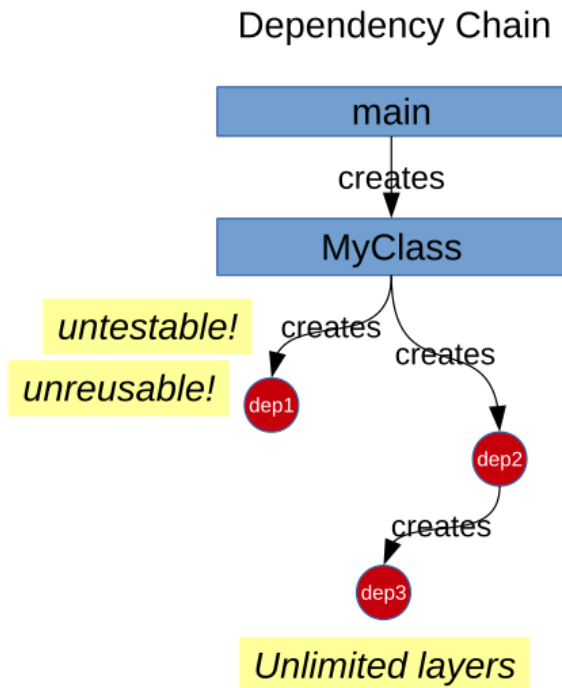
EXPLORE CTRL + SPACE

SHARE...





Dependency Injection



IoC : Inversion of Control

- **Definition:**

- A design pattern for object-oriented programming.
- Inverts the flow of a program for decoupling components.

- **Purpose:**

- Makes code reusable and modular.
- Allows injection of custom classes into other classes.

- **Benefits:**

- **Flexibility and Modularity:** Dependencies can be injected to change a class's behavior, flow, and performance as needed.
- **Control over Object Lifecycle:** Enables defining certain objects as singletons or creating new instances as required.
- **Maintainability:** Reduces overall code length due to reusable components.
- **Easier Testing:** Facilitates isolation of components and mocking of dependencies, allowing for more focused unit testing.

The Basic of Dependency Injection

- - Basics of Dependency Injection (DI)**
- **Overview:**
 - Extension of the IoC principle.
 - Enables objects or classes to receive external dependencies, enhancing flexibility and reusability.
- **Methods of Implementing Dependency Injection:**
 - **Constructor-based Dependency Injection:**
 - Involves injecting dependencies through a class's constructor.
 - The required dependencies are provided as arguments during object instantiation.


```
public class ExampleClass {  
    private DependencyClass dependency;  
  
    // Constructor accepting the dependency  
    public ExampleClass(DependencyClass dependency) {  
        this.dependency = dependency;  
    }  
}
```

```
package com.springexample;
/* Class for Student */
public class Student {
    private Grades grades;
    public Student(grades: Grades) {
        this.grades = grades;
    }
    public void retrieveGrades() {
        grades.getGrades();
    }
}
```

```
package com.springexample;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Main {
    public static void main(String[] args) {
        ApplicationContext context =
            new ClassPathXmlApplicationContext("Beans.xml");
        Student student =
            (Student) context.getBean("student");
        student.retrieveGrades();
    }
}
```

```
<?xml version = "1.0" encoding = "UTF-8"?>
<beans xmlns =
    "http://www.springframework.org/schema/beans"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation =
        "http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/
        spring-beans-3.0.xsd">
    <!-- Definition for student bean -->
    <bean id = "student"
        class = "com.springexample.Student">
        <constructor-arg ref = "grades"/>
    </bean>
    <!-- Definition for grades bean -->
    <bean id = "grades"
        class = "com.springexample.Grades"></bean>
</beans>
```

```
@Configuration
public class AppConfig
{
    @Bean
    public Student student() {
        return new Student(grades());
    }
    @Bean
    public Grades grades() {
        return new Grades();
    }
}
```

Setter Based Dependency Injection

- **Overview:**

- This approach involves injecting dependencies using setter methods rather than through constructors.
- Useful when an object's dependencies can be optional or changed after the object's instantiation.

- **How it Works:**

- An object is first created without the dependency.
- The dependency is then provided (injected) using a setter method.

```
public class ExampleClass {  
    private DependencyClass dependency;  
  
    // Setter method for injecting the dependency  
    public void setDependency(DependencyClass dependency) {  
        this.dependency = dependency;  
    }  
}
```

```
package com.springexample;

/* Class for Student */
public class Student {
    private Grades grades;
    public void setGrades(grades: Grades) {
        this.grades = grades;
    }
    public Grades getGrades() {
        return grades;
    }
    public void retrieveGrades() {
        grades.getGrades();
    }
}
```



```
package com.springexample;
import org.springframework.context.ApplicationContext;
import org.springframework.context.support.ClassPathXmlApplicationContext;
public class Main {
    public static void main(String[] args) {
        ApplicationContext context = new
            ClassPathXmlApplicationContext("Beans.xml");
        Student student =
            (Student) context.getBean("student");
        student.retrieveGrades();
    }
}
```

```
<?xml version = "1.0" encoding = "UTF-8"?>
<beans xmlns =
    "http://www.springframework.org/schema/beans"
    xmlns:xsi = "http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation =
        "http://www.springframework.org/schema/beans
        http://www.springframework.org/schema/beans/
        spring-beans-3.0.xsd">
<!-- Definition for student bean -->
<bean id = "student"
    class = "com.springexample.Student">
    <property name="grades" ref = "grades"/>
</bean>
<!-- Definition for grades bean -->
<bean id = "grades"
    class = "com.springexample.Grades"></bean>
</beans>
```

Field Based Dependency Injection

- **Overview:**

- This approach involves injecting dependencies using setter methods rather than through constructors.
- Useful when an object's dependencies can be optional or changed after the object's instantiation.

- **How it Works:**

- An object is first created without the dependency.
- The dependency is then provided (injected) using a setter method.

```
public class ExampleClass {  
    private DependencyClass dependency;  
  
    // Setter method for injecting the dependency  
    public void setDependency(DependencyClass dependency) {  
        this.dependency = dependency;  
    }  
}
```

```
package com.springexample;  
/* Class for Student */  
public class Student {  
    @Autowired  
    private Grades grades;  
}
```

Annotations and Beans

Spring

Annotations in Spring:

- **Definition:** Metadata tags that offer a way to configure the behavior of methods, classes, or variables.
- **Purpose:** Simplify configuration and setup, replacing traditional XML configurations.
- **Examples:**
 - **@Component:** Indicates a Spring-managed component.
 - **@Service:** Marks a class as a service in the business layer.
 - **@Repository:** Flags a class as a data repository.
 - **@Controller:** Denotes a class as a web controller.
 - **@Autowired:** Automates dependency injection

Core Annotations

- **@Component**: Basic annotation to denote a managed bean. Other component annotations (**@Service**, **@Repository**, **@Controller**) are specialized forms of **@Component**.
- **@Bean**: Used in configuration classes to define a new Spring bean.
- **@Autowired**: Performs dependency injection by type.
- **@Qualifier**: Used with **@Autowired** to specify which of multiple beans should be wired.
- **@Value**: Injects values into properties, can be used with SpEL (Spring Expression Language).
- **@Configuration**: Marks a class as a source of bean definitions.
- **@ComponentScan**: Configures which packages to scan for annotated components.
- **@Scope**: Defines the scope of a bean, e.g., singleton, prototype, request, session.
- **@PostConstruct**: Indicates a method that should be called right after bean initialization.
- **@PreDestroy**: Denotes a method that should be called right before a bean is removed from the container.


```
public class Car
{

    private String brand;

    @Required
    public void setBrand(String brand)
    {
        this.brand = brand;
    }

    public Integer getBrand()
    {
        return brand;
    }
}
```

