Deloitte.

Spring Boot

Deloitte Technology Academy (DTA)

Agenda

</>

Topics	Descriptions	Duration
Introducing Spring Boot	Introduction, Advantages, Features, Goals, Prerequisites, Spring vs. Spring Boot vs. Spring Model-View-Controller (MVC), and Spring Boot Architecture	X hours XX minutes
Spring Initializer and Spring Boot Example	Spring Initializer Overview	X hours XX minutes
Project Components: Annotations and Starters	Annotations, Dependency Management, Application Properties, Starters, Starter Parent, Starter Data Java Persistence API (JPA), and Starter Web	X hours XX minutes
Spring Boot–RESTful	Initializing Representational State Transfer (REST) Web Service, Auto Configuration, Approach: REST Application Programming Interface (API) Creation, and Example–REST API	X hours XX minutes
Actuator	Overview, Features, Enabling Spring Boot Actuator, Actuator Properties, and Executing different Actuator REST Endpoints	X hours XX minutes
Spring Boot Microservices–Create, Read/Retrieve, Update, and Delete (CRUD) Operations	Structured Query Language (SQL) vs. Hypertext Transfer Protocol (HTTP) Verbs vs. REST, Create, Read/Retrieve, Update and Delete (CRUD) Repository, and Java Persistence API (JPA)/Hibernate Repository	X hours XX minutes

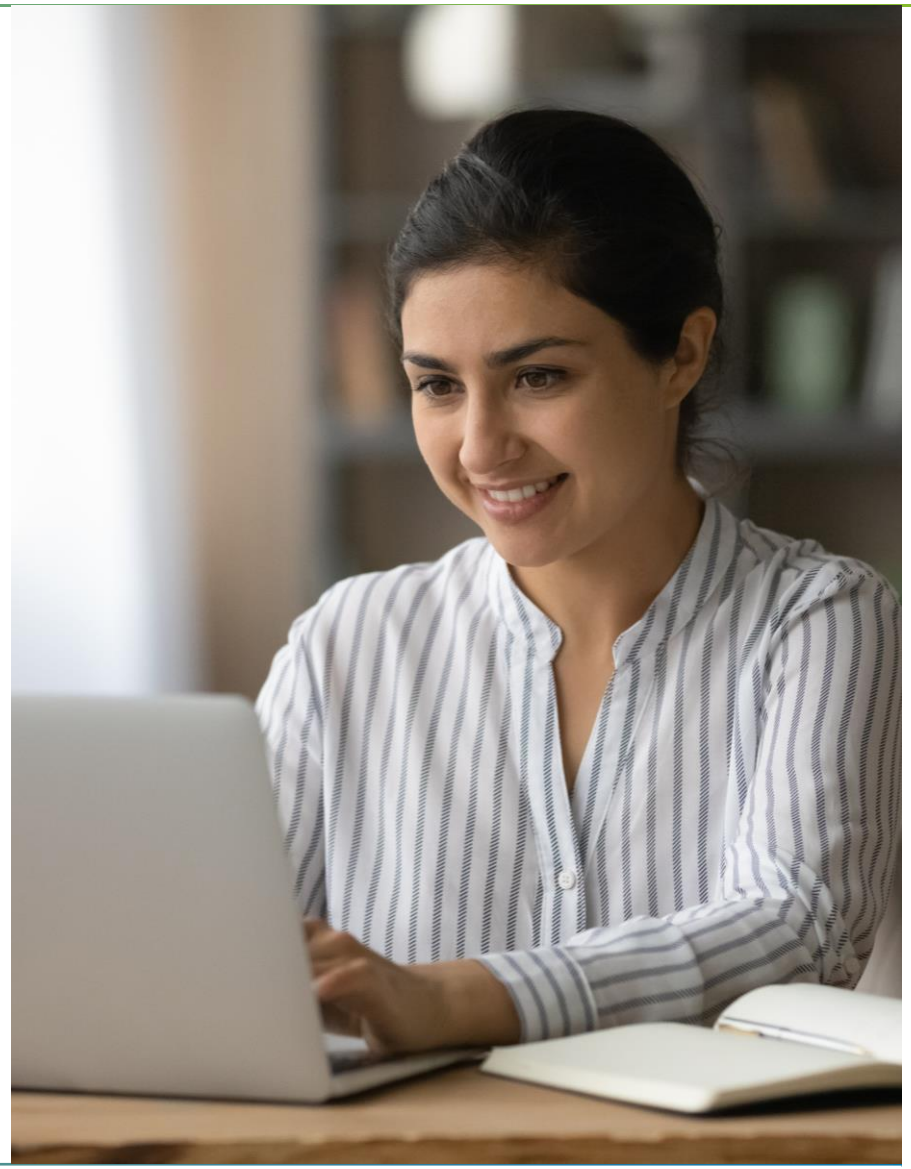
Learning Objectives

By the end of this session, you will be able to:

- Explain what is Spring Boot and why to use it
- Compare Spring Boot with Spring Model-View-Controller (MVC)
- Use Annotations and Starters
- Explain Java Persistence API (JPA) and its hierarchy
- Perform Create, Read/Retrieve, Update and Delete (CRUD) operations
- Create Actuator for monitoring and health checks

</ >

Introducing Spring Boot



What and Why?

< / >

What is Spring Boot?

- Built on the Spring Framework, Spring Boot is an open-source framework that aims to decrease overall development time and boost efficiency through a default setup for conducting unit and integration tests
- Focuses on shortening the code length and providing you with an easy way to run the Spring application
- Offers a very useful platform for Java developers to develop a standalone and production-grade spring application that you can just run

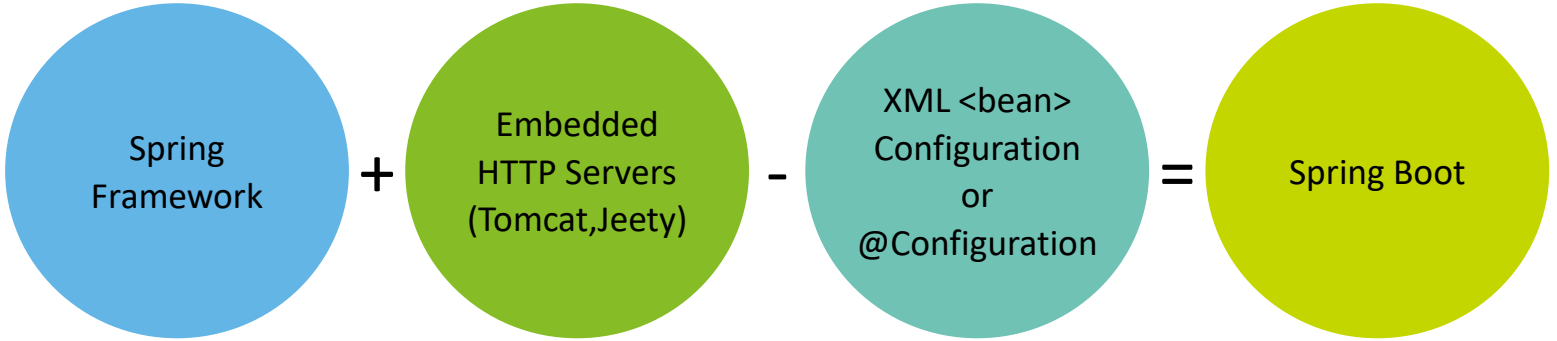
Why Spring Boot?

- Helps in getting enterprise-grade applications up and running quickly without having to worry about configuring your application correctly and safely
- Simplifies creating and testing Java-based applications for developers by autoconfiguring all components with embedded HTTP servers to test web applications
- Decreases development time and increases the overall productivity of the development team
- Compatible to easily connect with database and queue services

What is Spring Boot in a Nutshell?

</>

- Spring Boot is the combination of Spring Framework and Embedded Servers.
- In Spring Boot, there is no requirement for Extensible Markup Language (XML) configuration (deployment descriptor), thereby decreasing the developer's effort.



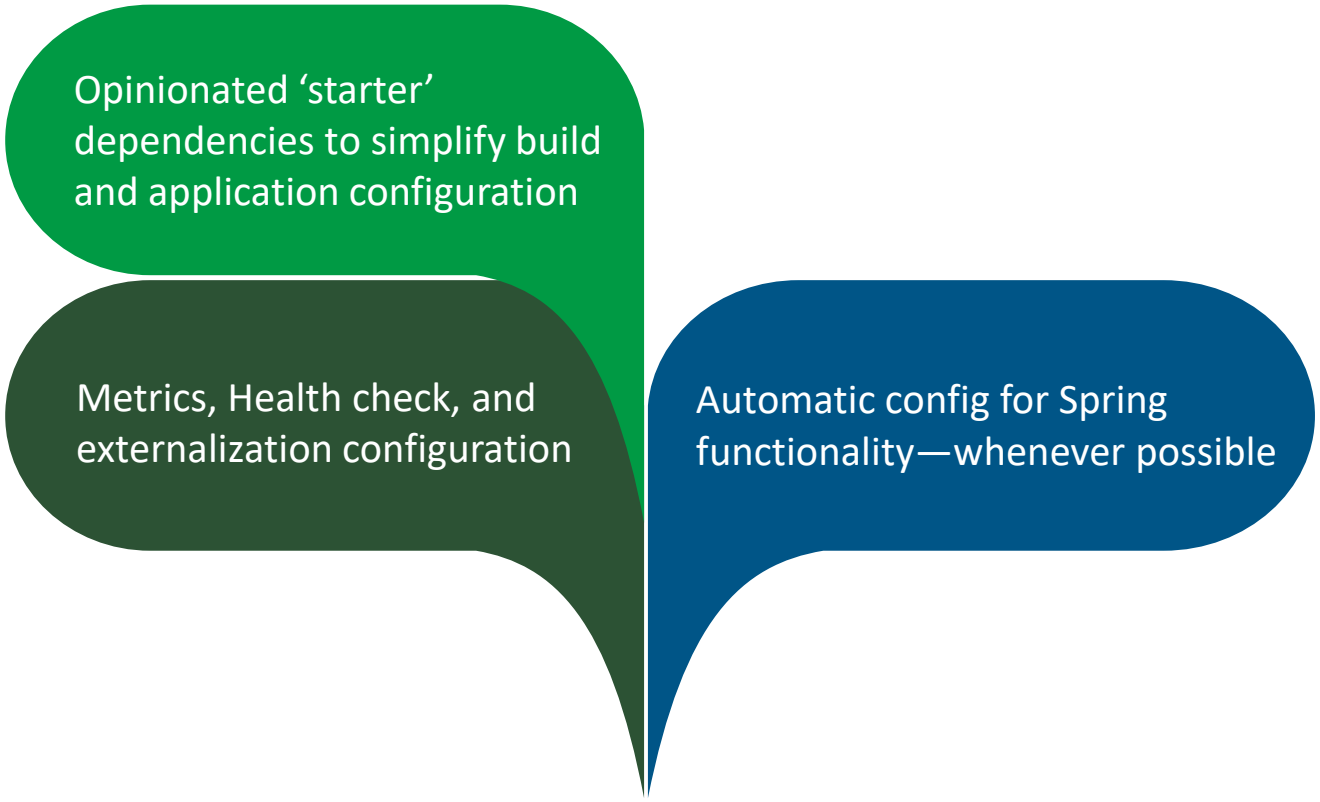

```
public static void main(String[] args)
{
TestSpringBoot.run(ClassName.class, args);
}
```

In the above example, `TestSpringBoot.java` provides a simple method to bootstrap a Spring application—just start it from the main method. We can call the application just by calling a static `run()` `method`.

Advantages

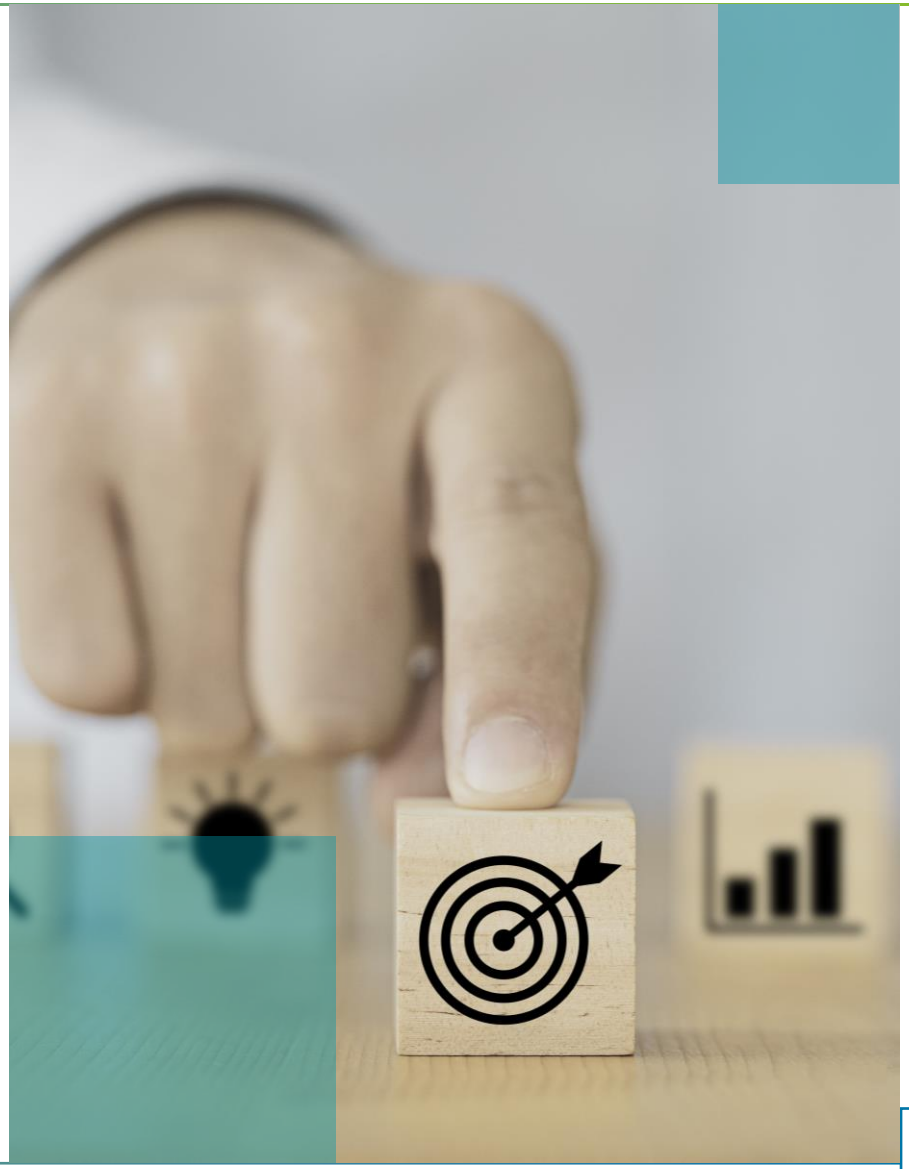
- Avoids writing lots of boilerplate code, annotations, and XML configuration
- Reduces complexity in deployment by embedding servers like Tomcat or Jetty
- Creates standalone Spring applications

Features



Goals

- To avoid complex XML configuration in Spring
- To develop production ready Spring applications in an easier way
- To reduce development time and run the application independently



Prerequisites

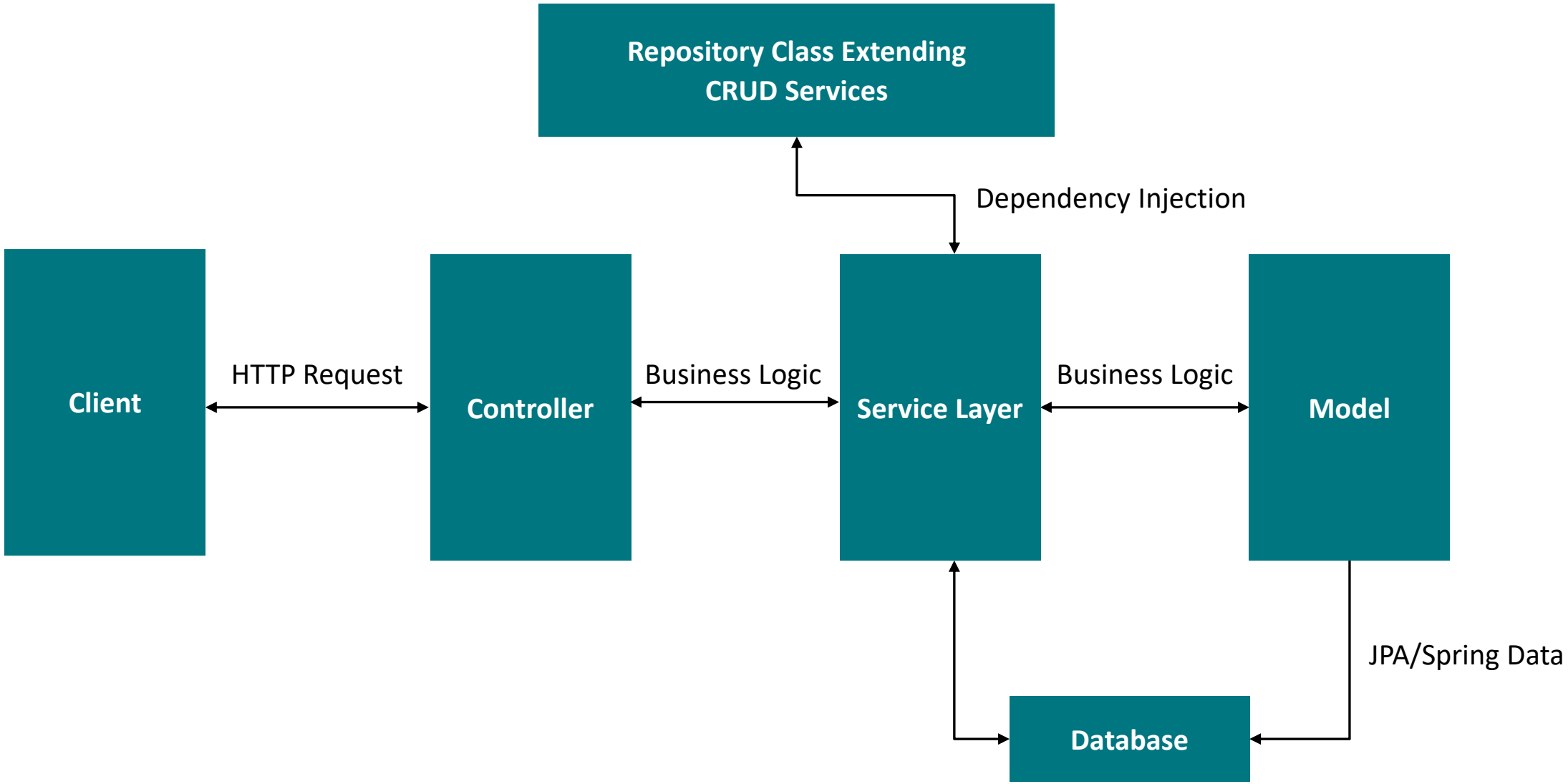
< / >

- Java 1.8
- Maven 3.0+
- Spring Framework 5.0.0.Build-snapshot
- An Integrated Development Environment (IDE) Spring Tool Suite is recommended

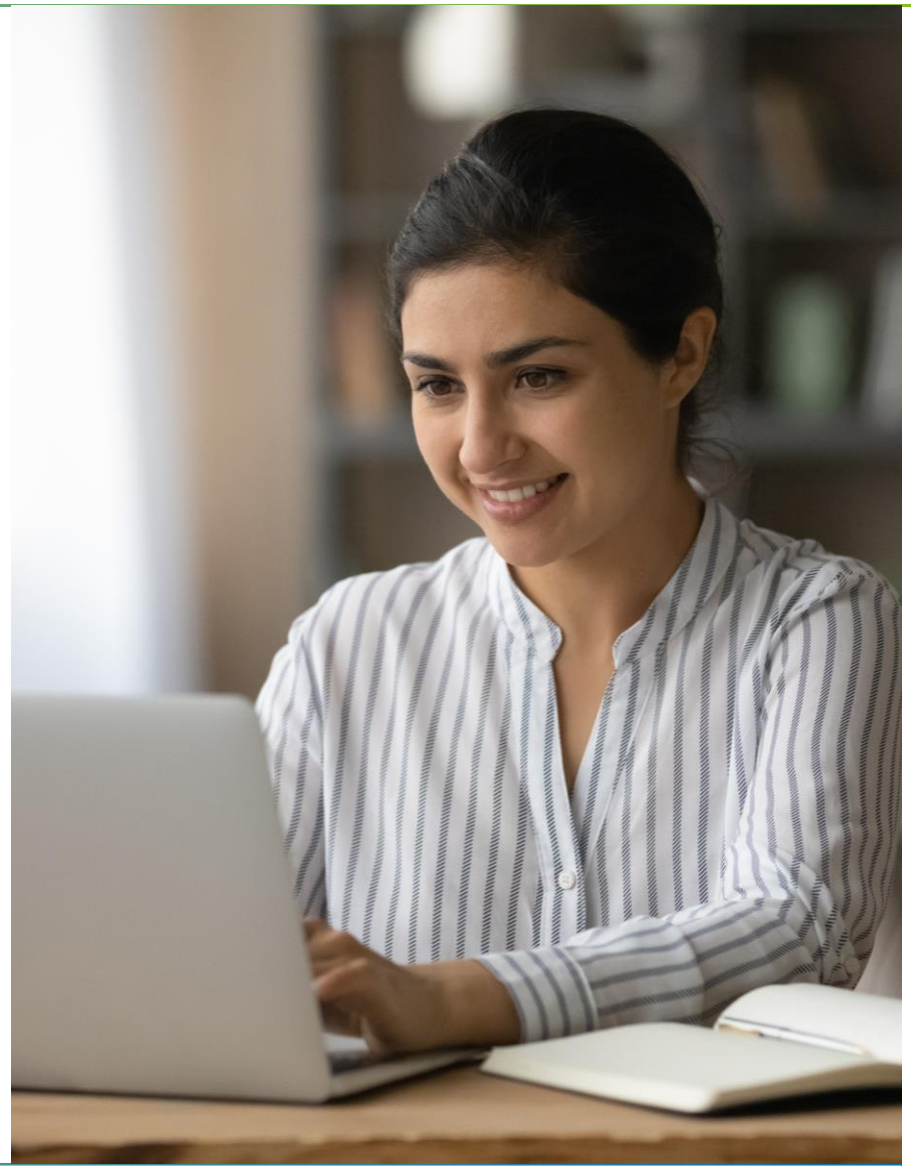
Spring vs. Spring MVC vs. Spring Boot

Spring	Spring MVC	Spring Boot
The developer writes a large piece of code (boilerplate code) to do simple task	Spring MVC is a Model-View-Controller-based web framework under the Spring framework.	It avoids boilerplate code and wraps dependencies together in a single unit.
Spring Framework is a widely used Java Enterprise Edition (EE) development for building applications.	It provides ready to use features for building a web application.	Spring Boot provides default configurations to build Spring-powered framework and to develop REST APIs.
The primary feature of Spring Framework is dependency injection.	It specifies each dependency separately.	The primary feature of Spring Boot is Auto Configuration.
To test Spring project, we need to set up the server explicitly.	A deployment descriptor is required.	Spring Boot offers embedded server such as Jetty, and Tomcat, etc.
The development time is more compared to Spring Boot and Spring MVC.	It takes more time to achieve the same.	It reduces development time and increases productivity.
It allows loose coupling and easy testability.	It requires building configuration manually. We need to configure ComponentScan, DispatcherServlet, a ViewResolver, and WebJava ARchives (WebJars) among other things.	Metrics, Health check, and externalized configuration (by using Yet Another Markup Language (YAML) file).

Spring Boot Architecture



Spring Initializer and Spring Boot Example



Spring Initializer Overview

< / >

- Provided by the Pivotal Web Service, Spring Initializr is a web-based tool that offers a quick way to gather all dependencies you need for an application and does a lot of the setup for you.
- Spring Initializr provides an extensible API to generate Java virtual machine (JVM) based projects and to inspect the metadata used to generate projects, for instance, to list the available dependencies and versions.

Follow these steps to create your first Spring Boot Application:

Step 1

Add Spring Boot Starter Parent

Step 2

Add Spring Boot Starter Web

Step 3

Configure Java Version 8

Step 4

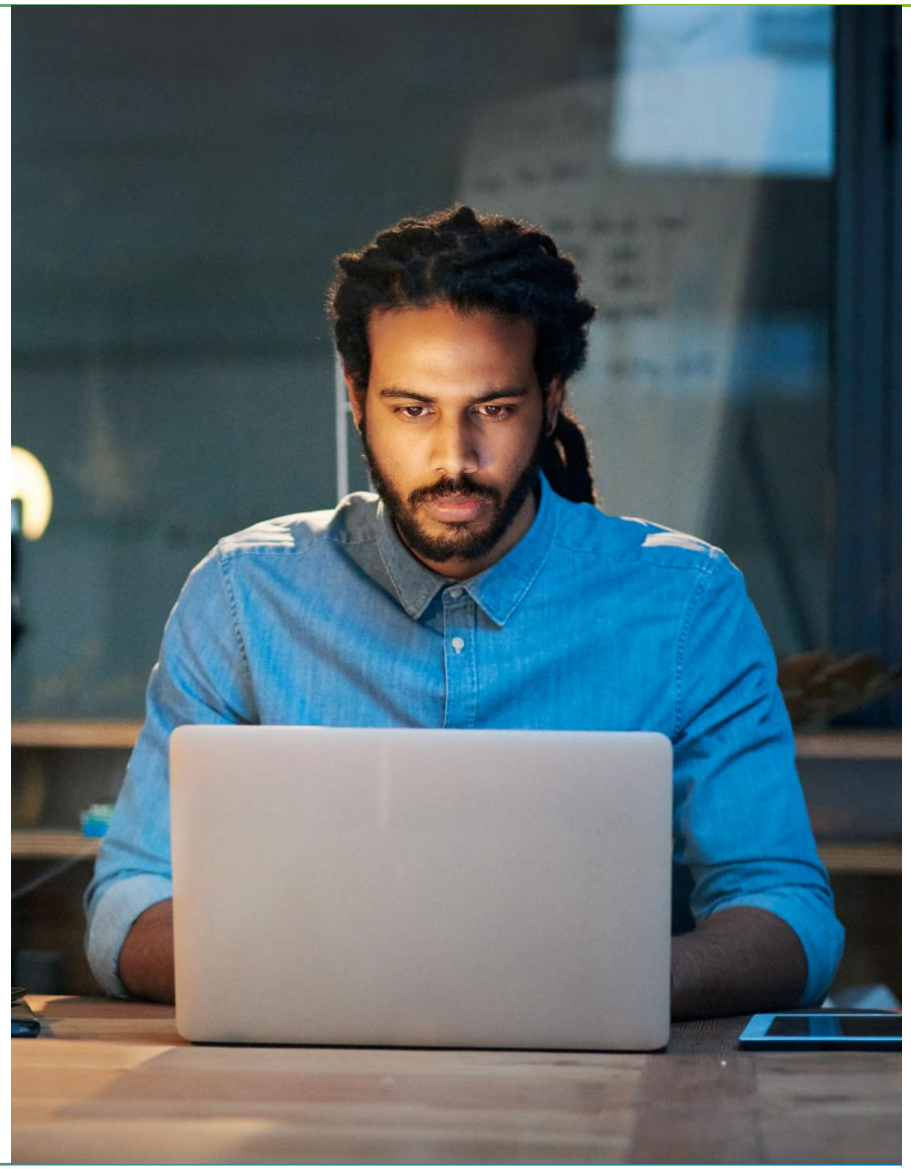
Add Spring Boot Plugin

Step 5

Create Spring Boot Application Launcher

</ />

Project Components: Annotations and Starters



Annotations

</>

Spring Boot Annotations: It is a form of meta data that provides data about a program.

@EnableAutoConfiguration

- Auto-configures the bean present in the class path, and configures it to run the methods
- There is little need to use this annotation in Spring Boot 1.2.0 release because developers provided an alternative to the annotation, i.e., `@SpringBootApplication`.

@SpringBootApplication

It is the combination of 3 annotations:

- `@EnableAutoConfiguration`
- `@ComponentScan`
- `@Configuration`

Annotations (Cont.)

</>

Spring Boot Annotations: It is a form of meta data that provides data about a program.

@ComponentScan	@Configuration
Enable `@ComponentScan` on the package where the application is located. We can also mark the base packages to scan for Spring Components.	Enables registering extra beans in the context or importing additional configuration classes. It is a class-level annotation.

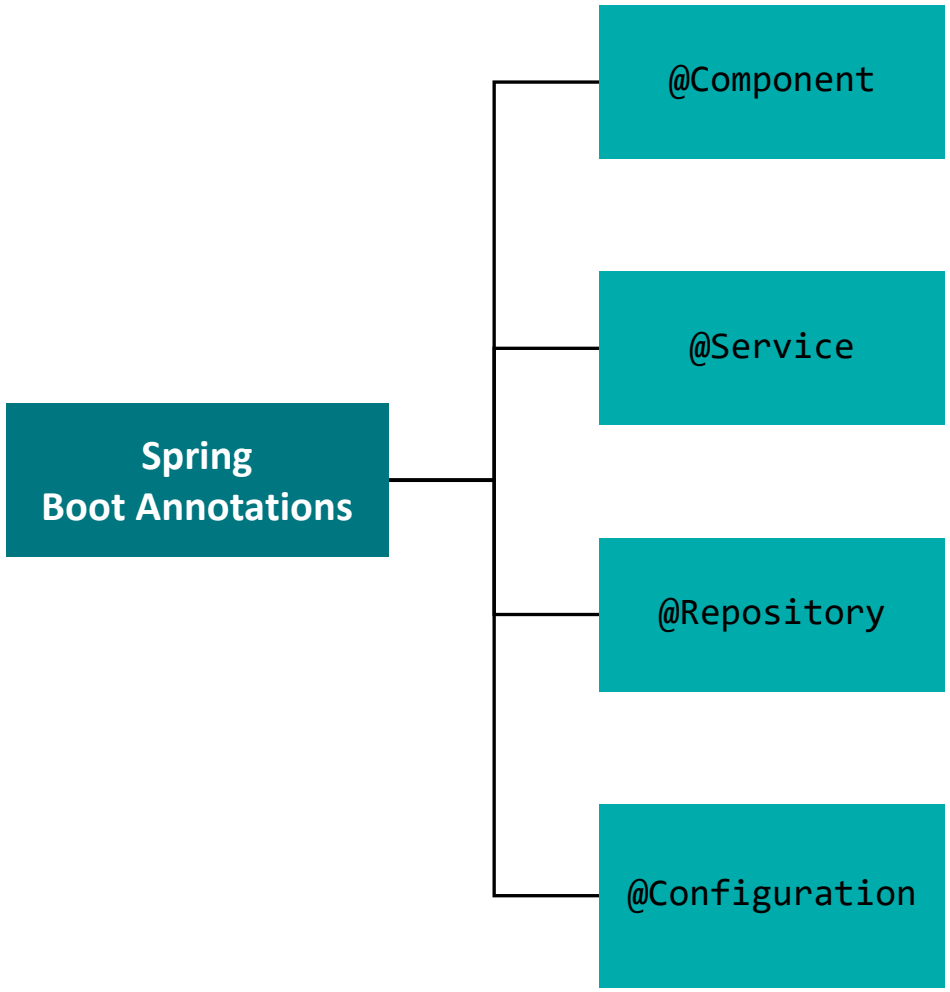
Annotations (Cont.)

</>

Example:

```
package com.example.myapplication;
import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
@SpringBootApplication // same as @Configuration @EnableAutoConfiguration @ComponentScan
public class Application {
 public static void main(String[] args) {  SpringApplication.run(Application.class, args);
 }
}
```

Annotations (Cont.)



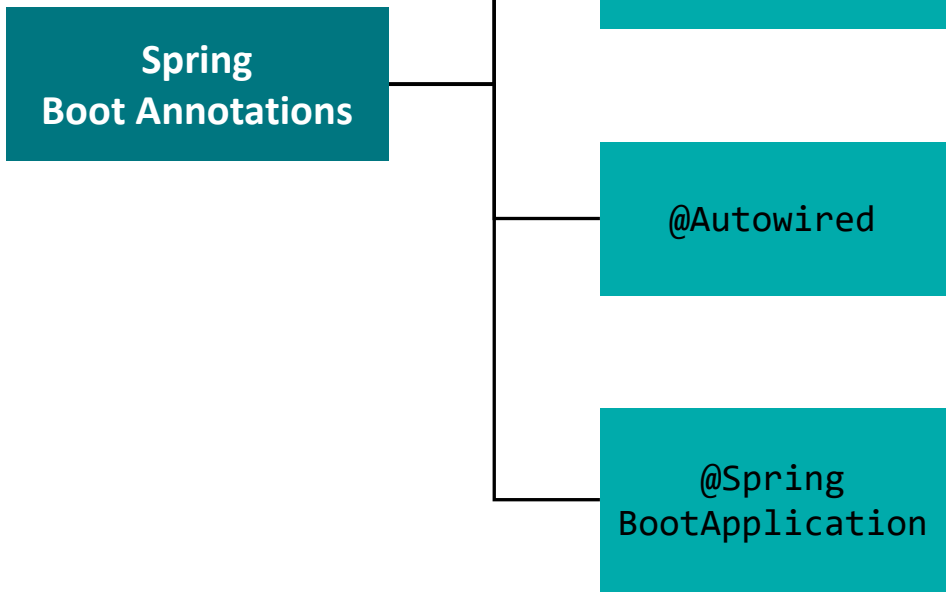
- A class-level annotation used to mark a Java class as a bean
- During the classpath, a Java class annotated with `@Component` is found. The Spring Framework picks it up and configures it in the application context as a Spring Bean.

Specifies that an annotated class is a service class. It is also used at a class level. It tells the Spring that the class contains the business logic.

Specifies that an annotated class is a repository, which is an abstraction of data access and storage. The repository is a Data Access Objects (DAOs) that access the database directly.

- A class-level annotation that specifies that a class is a configuration class that may contain bean definitions.

Annotations (Cont.)



- A class-level annotation that marks the class as a web controller, capable of handling the requests (as a web request handler to server web pages)
- By default, it outputs a string that specifies which route to redirect.
- It is mostly used with `@RequestMapping` annotation.

- Typically, used to map the web requests.
- Maps a HTTP request with a path to a controller method.

Marks a constructor, field, or setter method to be autowired by Spring dependency injection.

Enables Spring Boot autoconfiguration and component scanning.

Dependency Management

< / >

Features

- Spring Boot manages dependencies and configuration automatically.
- Each release of Spring Boot provides a list of dependencies that it supports.
- The list of dependencies is available as part of the Bills of Materials (spring-boot-dependencies) that can be used with Maven.
- Spring Boot manages all by itself which is why we need not specify the version of the dependencies in our configuration.
- Spring Boot upgrades all dependencies automatically in a consistent way when we update the Spring Boot version.
- Maven Dependency inherits a Dependency Section from the spring-boot-dependency-pom. It manages the version of common dependencies.

Advantages

- It provides the centralization of dependency information by specifying the Spring Boot version in one place.
- It helps when we switch from one version to another.
- It avoids mismatch of different versions of Spring Boot libraries.
- We only need to write a library name specifying the version. It is helpful in multi-module projects.
- When you upgrade Spring Boot, these dependencies are upgraded as well uniformly.
- The Maven project inherits the following features from spring-boot-starter-parent:
  - The default Java compiler version, UCS Transformation Format 8 (UTF-8) source encoding.
  - Sensible resource filtering, plugin configuration
  - Dependencies, inherited from the spring-boot-dependencies POM

Application Properties

</ />

Configuration

- The Spring Boot Framework is equipped with an in-built mechanism for application configuration using the file—application.properties.
- It is located inside the source/main/resources folder, as shown in the displayed example.
- Spring Boot allows us to define our own, custom property if needed.
- The application.properties file enables us to run applications in other environments

Example of application.properties:

```
#configuring application name
spring.application.name = demoApplication

#configuring port
server.port = 8081
```

Application Properties

</ />

Features

We can use the application.properties file to:

- Configure the Spring Boot framework.
- Define our application custom configuration properties.
- Spring Boot provides another file to configure the properties—yml file.
- All the database related configurations are mentioned in application properties.

The YAML file works, as the Snake YAML Java ARchive (JAR) is available in the classpath. Instead of using the application.properties file, we can use the application.yml file, but the .yml file should be available in the classpath.

Example of application.yml:

```
spring:
application:
name: demoApplication
server:
port: 8081
```

Starters

</ />

Spring Boot offers numerous starters that allow us to add JARs in the classpath. All the starters follow a naming pattern: `spring-boot-starter-*` where * denotes a particular type of application. There are many starters available. Here are just 5 of them to get a glimpse.

Starter	Transitive dependency	Dependency
`spring-boot-starter`	`spring-boot, spring-boot-autoconfigure, spring-boot-starter-logging, spring-core, snakeyaml`	It is used for Core starter, autoconfiguration support, logging, and YAML
`spring-boot-starter-data-jpa`	`Spring-boot-starter, spring-boot-starter-aop, spring-boot-starter-jdbc, hibernate-entitymanager, javax.transaction-api, spring-data-jpa, spring-aspects`	It is a starter for using spring data JPA with Hibernate
`spring-boot-starter-test`	`Junit, mockito-core, hamcrest-core, hamcrest-library, spring-core, spring-test`	It is a starter for testing using libraries Junit, Hamcrest, and Mockito
`spring-boot-starter-web`	`Spring-boot-starter, spring-boot-starter-tomcat, spring-boot-starter-validation, jackson-databind, spring-web, spring-webmvc`	It is a Starter for building Web application using Spring MVC, REST, and Tomcat as a default embedded container
`spring-boot-starter-data-mongodb`	`Spring-boot-starter, mongo-java-driver, spring-data-mongodb`	It is a starter for using MongoDB and Spring Data MongoDB

Starter Parent

</>

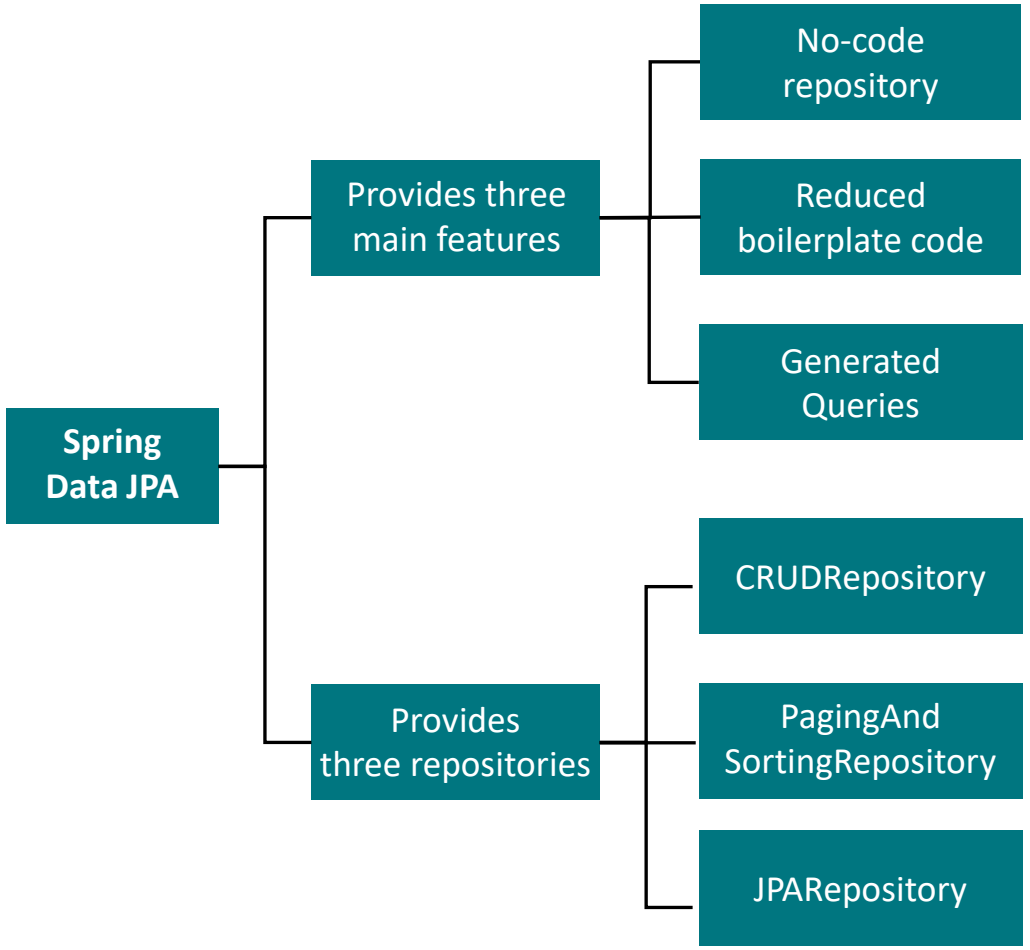
- The spring-boot-starter-parent is a project starter used internally by all dependencies.
- Offers default configurations for applications
- The spring-boot-starter-parent is used as a parent in the pom.xml file in all Spring Boot projects.
- Parent Poms enable us to manage multiple things for multiple child projects and modules, such as:
  - Configuration: It allows us to maintain consistency of Java Version and other related properties.
  - Dependency Management: It controls the versions of dependencies to avoid conflict.
  - Source encoding
  - Default Java Version
  - Resource filtering
  - It also controls the default plugin configuration
- The spring-boot-starter-parent inherits dependency management from spring-boot-dependencies.

Example:

```
<parent>
<groupId>org.springframework.boot</groupId
>
<artifactId>spring-boot-starter-
parent</artifactId>
<version>1.4.0.RELEASE</version>
</parent>
```

```
spring:
application:
name: demoApplication
server:
port: 8081
```

Starter Data JPA



- A widely used persistence-related pattern
- Allows implementation of business code on a higher abstraction level

- Offers default implementation for each method by its repository interfaces—potentially minimizing the need to implement read and write operations.
- Allows creation of database queries on the basis of the method name

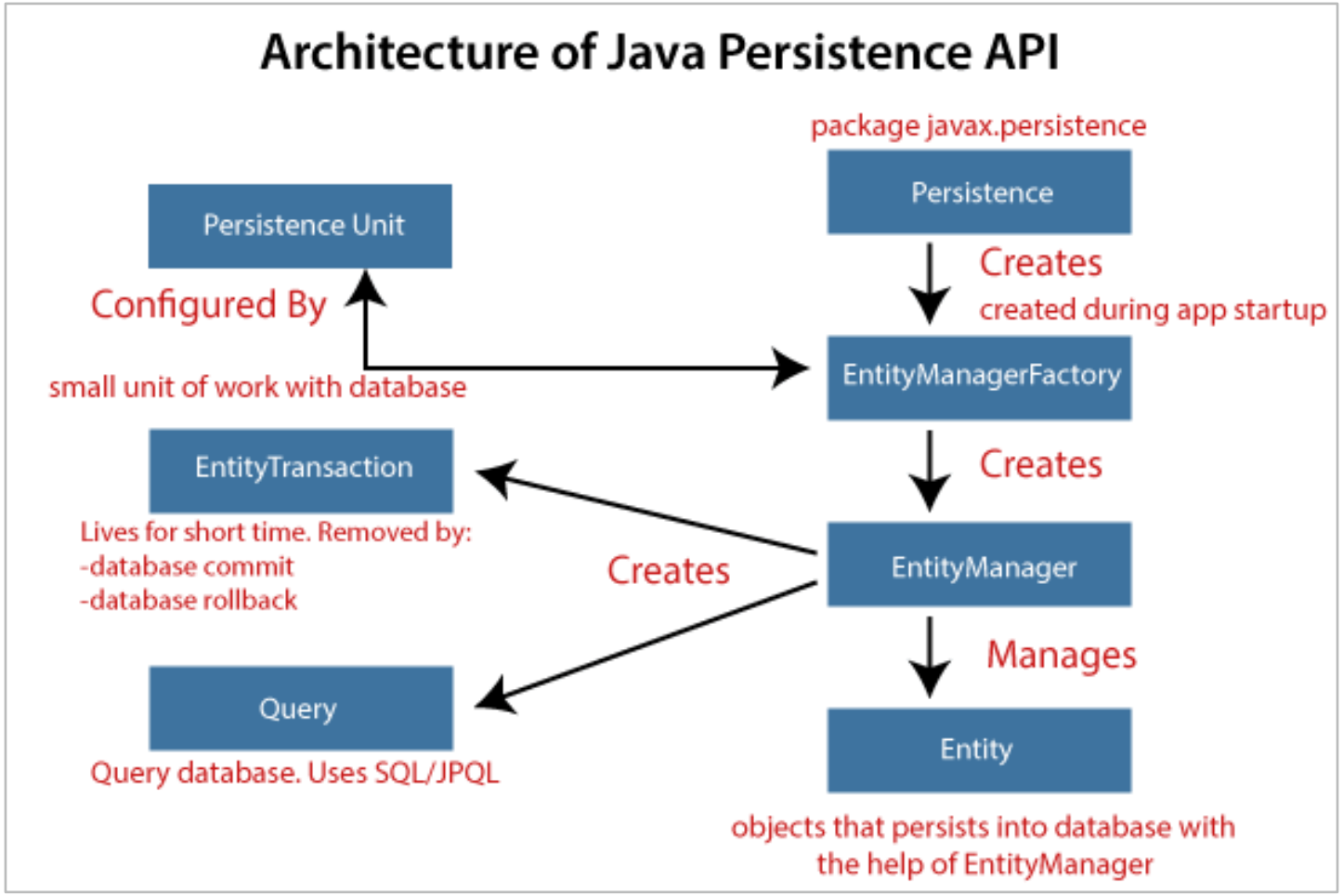
- Generation of database queries based on the method name.
- If the query is not too complex, a method with the name starting with `findBy` needs to be defined on our repository interface
- After defining the method, Spring parses the method name and creates a query for it.

- Offers standard create, read, update, and delete.
- Contains methods such as `findOne()`, `findAll()`, `save()`, and `delete()`, etc.

- Extends the CrudRepository and adds the `findAll` methods.
- Enables sorting and retrieving data in a paginated way.

- It is a JPA-specific repository that is defined in Spring Data JPA.
- It extends both CrudRepository and PagingAndSortingRepository.
- It adds the JPA-specific methods like `flush()` to trigger a flush on the persistence context.

Starter Data JPA (Cont.)



Starter Web

< / >

Features

- Compatible for web development
- Auto configuration
- Default embedded server
- Reduces build dependency count
- Tomcat server dependency by default

Spring Boot Embedded Web Server

- Embedded as part of deployable application
- Does not require pre-installed server
- Tomcat is the default embedded server
- Also supports another 2 embedded servers:
  - Jetty Server (It's an HTTP Server)
  - Undertow Server (HTTP Server)
  - e.g., We need to add the spring-boot-starter-jetty dependency in our pom.xml file.

Starter Web (Cont.)

</>

Auto configures the following that are required for web development:

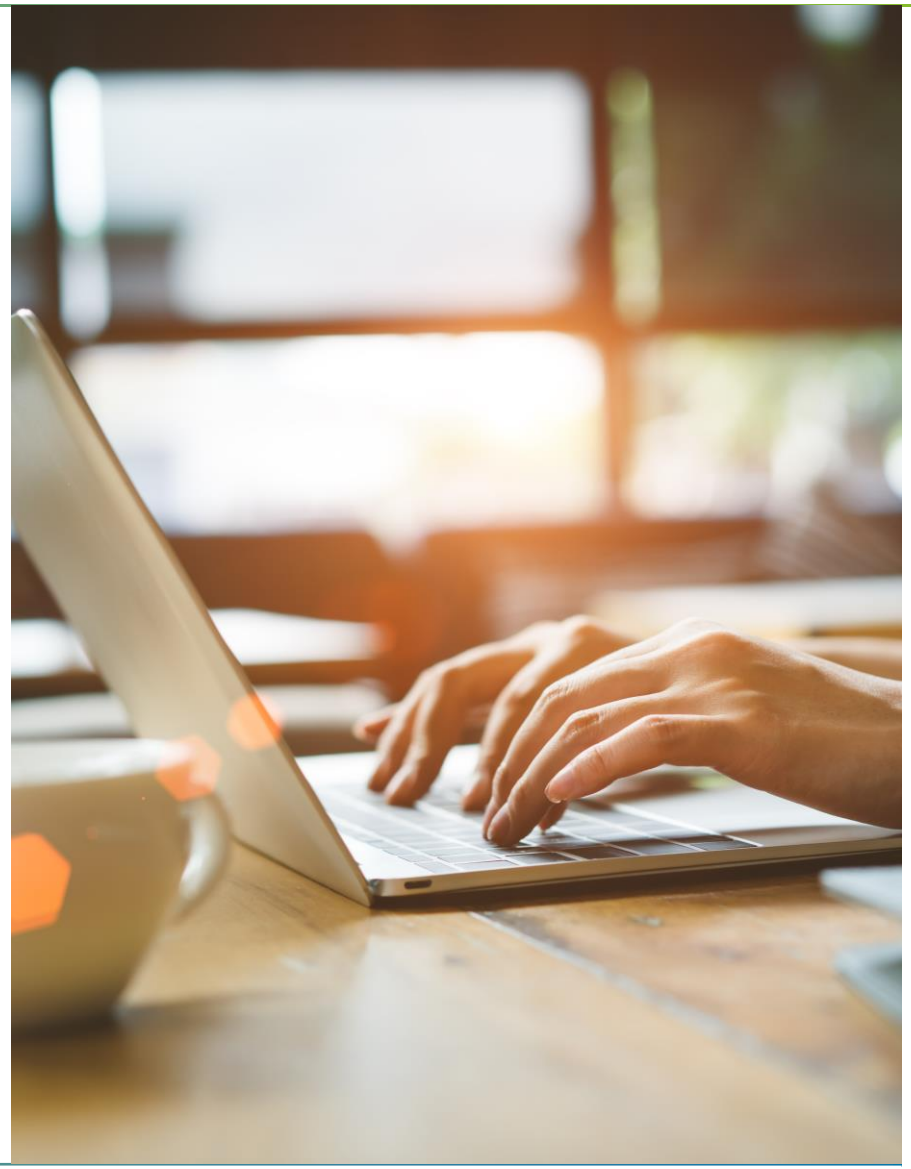
- DispatcherServlet
- Error Page
- WebJars
- Embedded servlet container

Example:

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
web</artifactId>
<exclusions>
 <exclusion>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
tomcat</artifactId>
 </exclusion>
</exclusions>
</dependency>
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
jetty</artifactId>
</dependency>
```

</ >

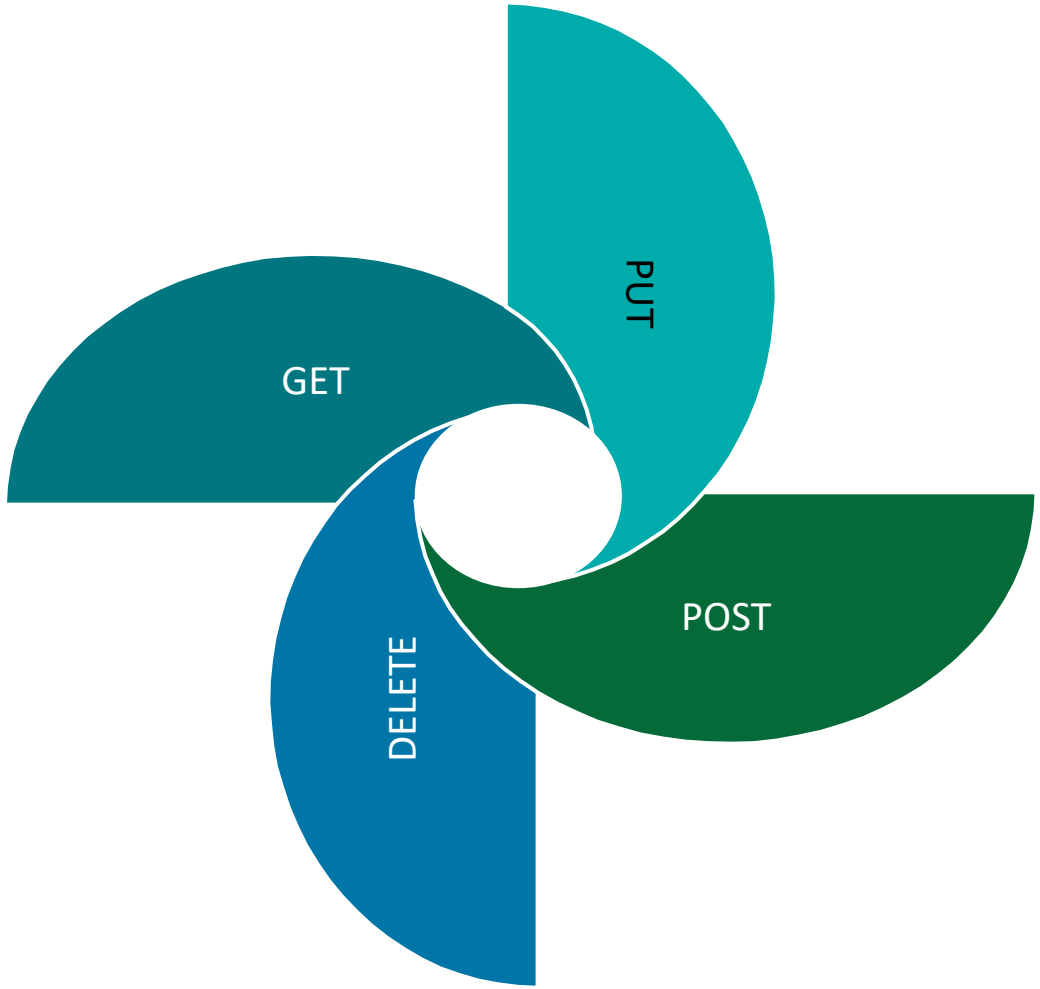
Spring Boot—RESTful



Initializing REST Web Service

</ />

- REST stands for Representational State Transfer. REST is an architectural approach, not a protocol.
- We can build REST services with both XML and JSON. JSON is the more popular format with REST.
  - GET: It reads a resource.
  - PUT: It updates an existing resource.
  - POST: It creates a new resource.
  - DELETE: It deletes the resource.



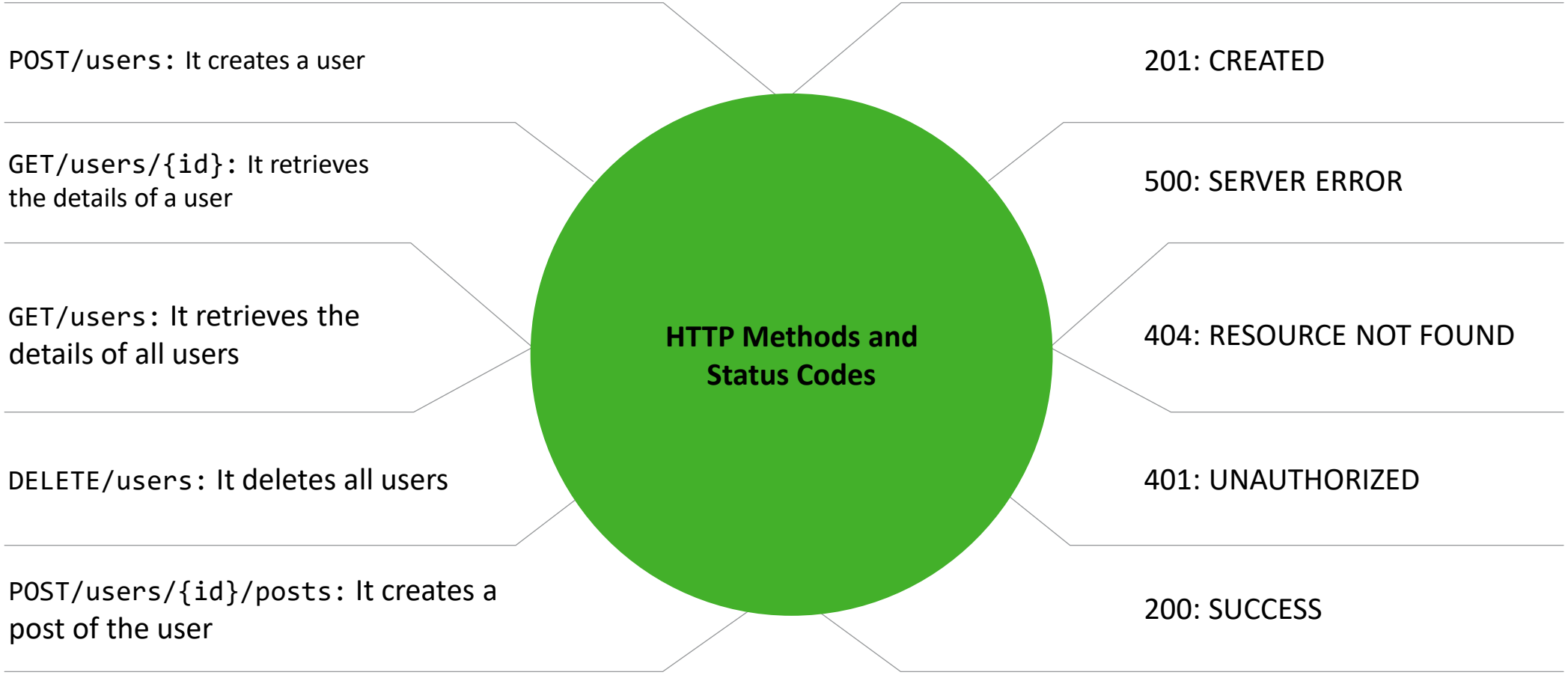
Initialize - Rest Web Service (Cont.)

< / >

- RESTful web services are platform-independent.
- It can be written in any programming language and can be executed on any platform.
- It provides different data formats like JavaScript Object Notation (JSON), text, HTML, and XML.
- It is fast in comparison to Simple Object Access Protocol (SOAP) because there is no strict specification like SOAP.
- These are reusable and language neutral.

Initialize - Rest Web Service (Cont.)

HTTP Methods And Standard Status Codes - These are the important methods in Uniform Resource Identifier (URI) implementation of REST call.



Initialize - Rest Web Service (Cont.)

</>

`@RestController`

- Used at the class level, this annotation specifies the class as a controller in which every method outputs a domain object instead of a view.
- By using this annotation, you don’t need to add @ResponseBody to all the RequestMapping methods anymore.
- In other words, you don’t need to use view-resolvers or send HTML in response. You just send the domain object as an HTTP response in the format that is understood by the consumers, like JSON.
- @RestController is a convenient annotation that combines @Controller and @ResponseBody.
- Spring RestController annotation is used to generate RESTful web services using Spring Boot.
- Spring RestController takes care of mapping request data to the defined request handler method.
- Once the response body is generated from the handler method, it converts it to JSON or XML response.

Example

```
@RestController
            @RequestMapping("books-rest")
            public class
SimpleBookRestController {
            @GetMapping("/{id}", produces
= "application/json")
            public Book
getBook(@PathVariable int id) {
            return findBookById(id);
            }
```

Auto Configuration

</ />

Features of Spring Boot Auto Configuration

1. Spring Boot automatically configures a spring application based on dependencies present or not present in the classpath as a JAR, beans, properties, etc.
2. It makes development easier and faster as there is no need to define certain beans that are included in the auto-configuration classes.
3. A typical MVC database driven Spring MVC application requires a lot of configuration such as a DispatcherServlet, a ViewResolver, Jackson, data source, and transaction manager, among many others.
4. Auto-configuration can be enabled by adding `@SpringBootApplication` or `@EnableAutoConfiguration` annotation in the startup class. It indicates that it is a spring context file.
5. It enables something called ComponentScan. It is the feature of Spring that will start automatically scanning classes in the package and subpackage for any bean file.
6. DispatcherServletAutoConfiguration, DataSourceAutoConfiguration, JacksonAutoConfiguration, ErrorMvcAutoConfiguration are some examples of auto configuration done by Spring Boot.

Approach: REST API Creation

Steps to understand the concept of REST API:

1. Create a Maven project using Spring Initializer.
2. Create Controller using annotation `@RestController`. This annotation informs the Spring Boot framework that the class contains methods that will be invoked through a web-based resource Uniform Resource Locator (URL).
3. Add `@RequestMapping` annotations to methods. These annotations define the HTTP method used along with the structure of the resource URLs that will be used to invoke them.
4. We need to generate a JSON-based response for the client.
5. There are excellent frameworks like Jackson and Google Gson (GSON), which you should use in larger projects. We can create simple Java string manipulation to generate JSON.
6. Run the Spring Boot Application class and invoke URLs using Postman.

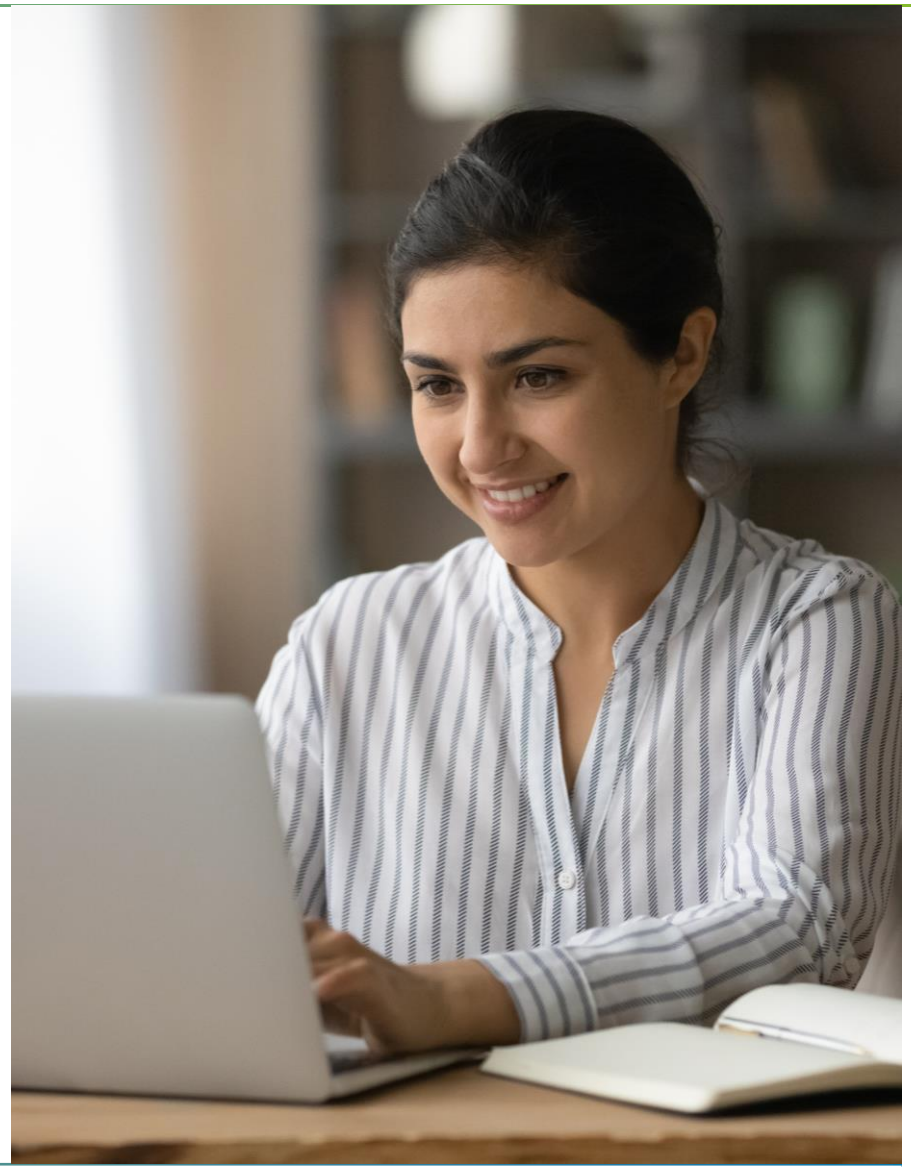
Example—REST API

</>

Code Example:

```
restTemplate.getForObject(REST_SERVICE_URI + "getDetails", Class.class)
restTemplate.put(REST_SERVICE_URI , object, params);
restTemplate.delete(REST_SERVICE_URI , params);
```

Actuator



Overview and Features

</ >

Endpoint

- The actuator endpoints allow us to monitor and interact with the application.
- Spring Boot provides a number of built-in endpoints. We can also create our own endpoint.
- We can enable and disable each endpoint individually.

Metrics

- Spring Boot Actuator provides dimensional metrics by integrating with the micrometer.
- The micrometer is integrated into Spring Boot.
- It is the instrumentation library powering the delivery of application metrics from Spring.

Audit

- Spring Boot provides a flexible audit framework that publishes events to an AuditEventRepository.
- It automatically publishes the authentication events if spring-security is in execution.

Enabling Spring Boot Actuator

</>

- We can enable the actuator by injecting the dependency spring-boot-starter-actuator in the pom.xml file.
- The actuator endpoints enable us to monitor and interact with our Spring Boot application.
- Spring Boot comes with numerous built-in endpoints. We can add custom endpoints in Spring Boot application as well.
- Some of the popular endpoints are Actuators, Health, Info, and metrics.

```
<dependency>
<groupId>org.springframework.boot</groupId>
<artifactId>spring-boot-starter-
actuator</artifactId>
<version>2.2.2.RELEASE</version>
</dependency>
```

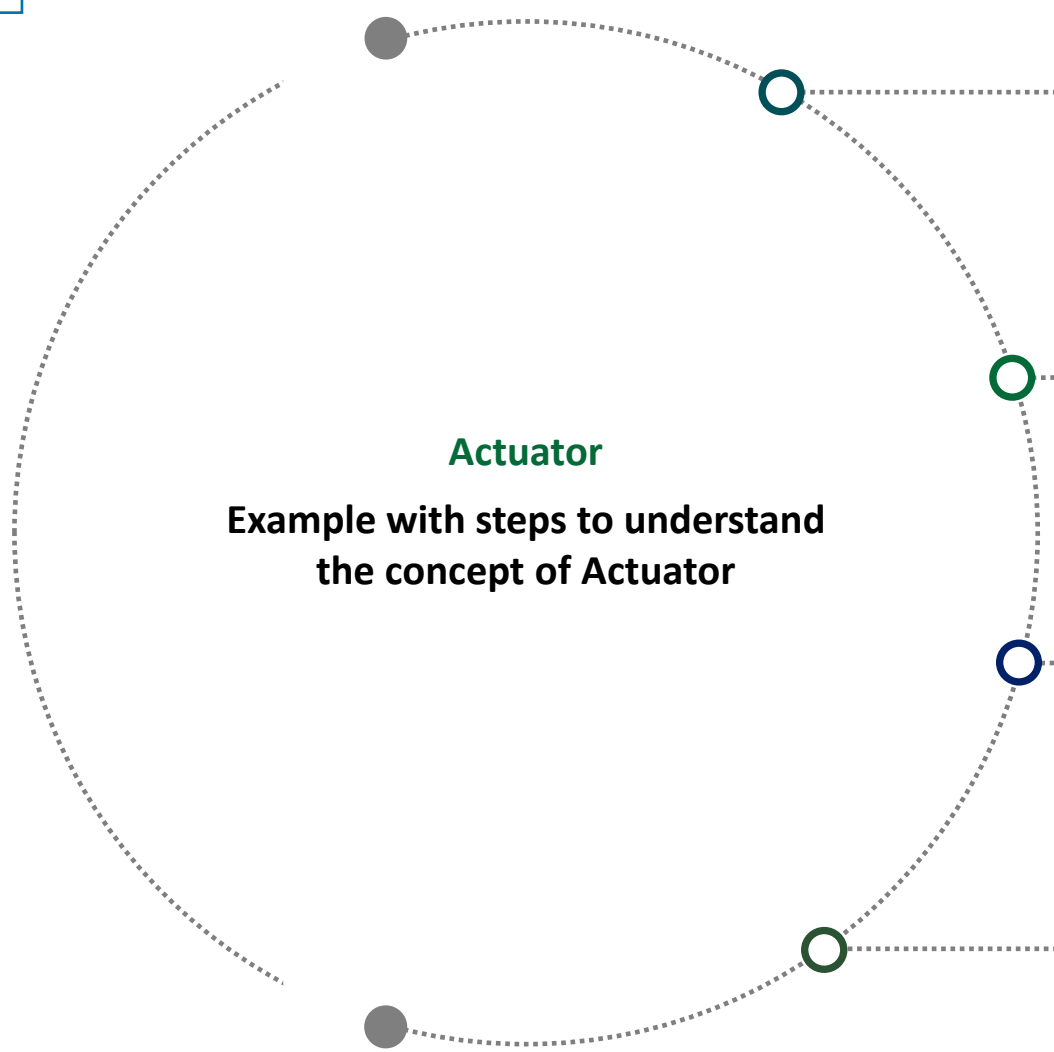
Actuator Properties

</>

- Spring Boot offers security for all actuator endpoints.
- Leverages form-based authentication—provides a user Id and a randomly generated password.
- We can use actuator-restricted endpoints by customizing basic auth security to the endpoints.
- We need to override this configuration by using the `management.security.roles` property.

```
management.security.enabled=true
management.security.roles=ADMIN
security.basic.enabled=true
security.user.name=admin
security.user.passowrd=admin
```

Steps to Create Spring Boot Actuator



Create Maven Project

- Open Spring Initializr https://start.spring.io/ and create a Maven project
- Provide the Group name

Add Dependencies

- Spring Web
- Spring Boot Starter Actuator
- Spring Data Rest HAL Browser

Add Security Feature

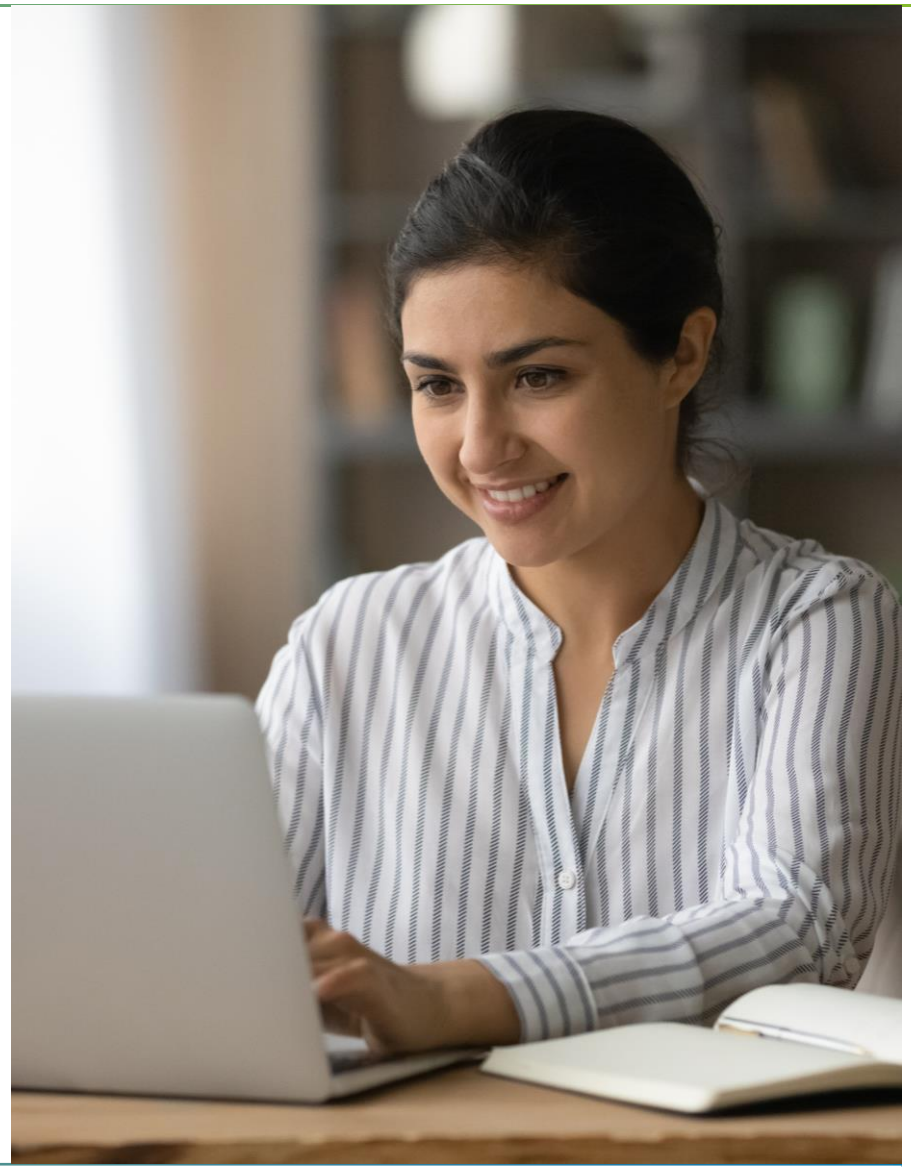
- Add "`management.security.enabled=false`" to disable the security feature of the actuator in application.properties

Invoke Actuator

- Open the browser and invoke the URL http://localhost:8080/actuator.
- The application runs on port 8080 by default.
- Once the actuator has started, we can see the list of all the endpoints exposed over HTTP.

</ / >

Spring Boot Microservices—CRUD Operations



Introduction

< / >

CRUD OPERATION

- CRUD is short for Create, Read/Retrieve, Update, and Delete—the four basic functions of the persistence storage.
- The CRUD operation are essentially user interface conventions that allow viewing, searching, and modifying information through computer-based forms and reports.
- CRUD is data-oriented and standardized the usage of HTTP action verbs.
- Within a database, each CRUD operation directly maps to a set of commands. However, their connection with a RESTful API is slightly complex.
- .

Introduction (Cont.)

</>

Standard CRUD Operation	Function
CREATE	Executes the insert statement to create a new record
READ	Reads table records based on the input condition
UPDATE	Updates the table based on the input parameter
DELETE	Deletes a particiular row in the table based on the input parameter

SQL vs. HTTP Verbs vs. REST

How CRUD Operations work?

- CRUD operations are at the foundation of the most dynamic websites. Therefore, we should differentiate CRUD from the HTTP action verbs.
- Suppose we want to create a new record; we should use HTTP action verb POST.
- To update a record, use the PUT verb. Similarly, to delete a record, use the DELETE verb.
- Through CRUD operations, users and administrators have the right to retrieve, create, edit, and delete records online.
- We have many options for executing CRUD operations. One of the most efficient choices is to create a set of stored procedures in SQL to execute operations.

SQL vs. HTTP Verbs vs. REST (Cont.)

</>

Operation	SQL	HTTP verbs	RESTful Web Service
Create	INSERT	PUT/POST	POST
Read	SELECT	GET	GET
Update	UPDATE	PUT/POST/PATCH	PUT
Delete	DELETE	DELETE	DELETE

Crud Repository vs. JPA Repository

</>

CrudRepository

- Spring Boot offers the CrudRepository interface that features methods for CRUD operations. It does not include any method for pagination and sorting.
- It extends the Spring Data Repository interface, and it is defined in the package org.springframework.data.repository.
- It provides CRUD function only. Eg: `findAll(),findById()` etc.
- It is used when we don't need the functions provided by JpaRepository and PagingAndSortingRepository.

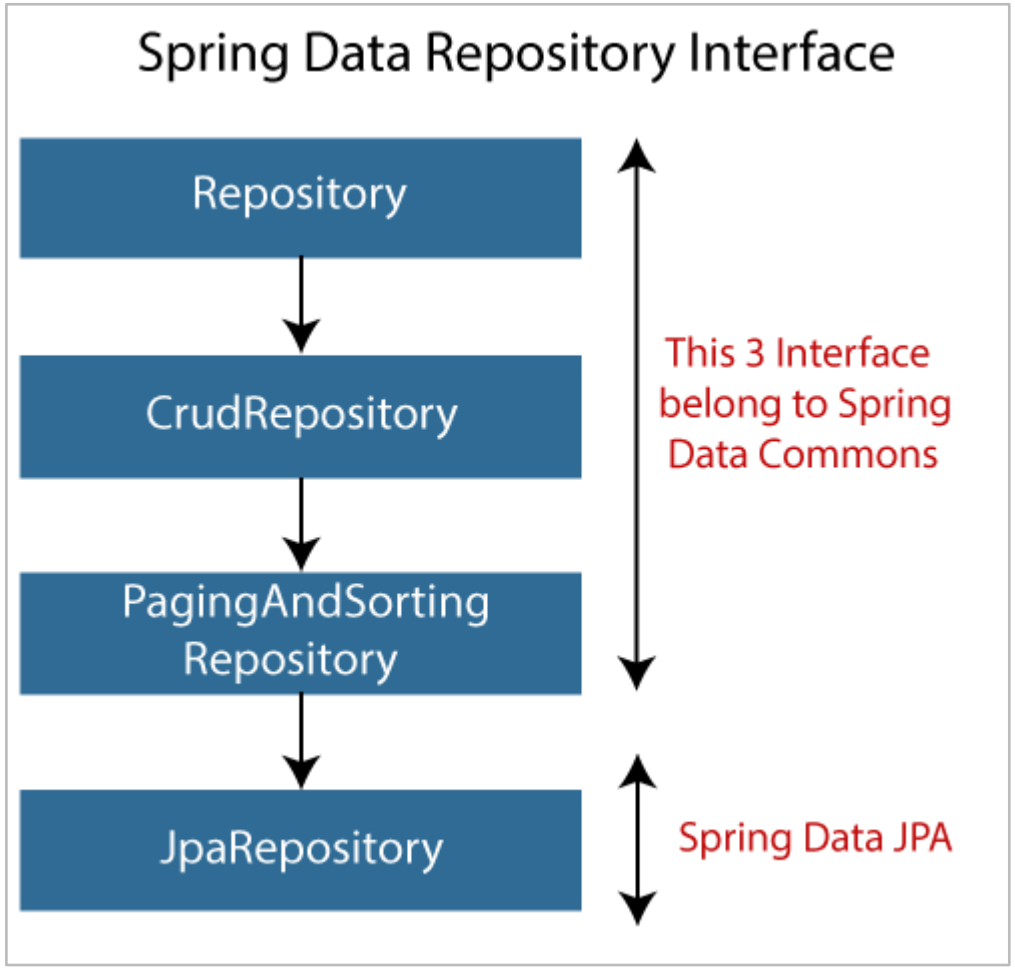
```
public interface CrudRepository<T,ID> extends
Repository<T,ID>
where
 T is the domain type that repository manages
ID is the type of the id of the entity that
repository manages
Eg: public interface EmployeeRepository
extends CrudRepository<Employee, Integer>  {
}
```

JPA Repository

- JpaRepository extends PagingAndSortingRepository and offers all the methods for implementing the pagination.
- JpaRepository extends both CrudRepository and PagingAndSortingRepository.
- It offers some extra methods along with the method of
- PagingAndSortingRepository and CrudRepository. E.g., `flush()`
- It can useful when we want to implement pagination and sorting functionality in an application.

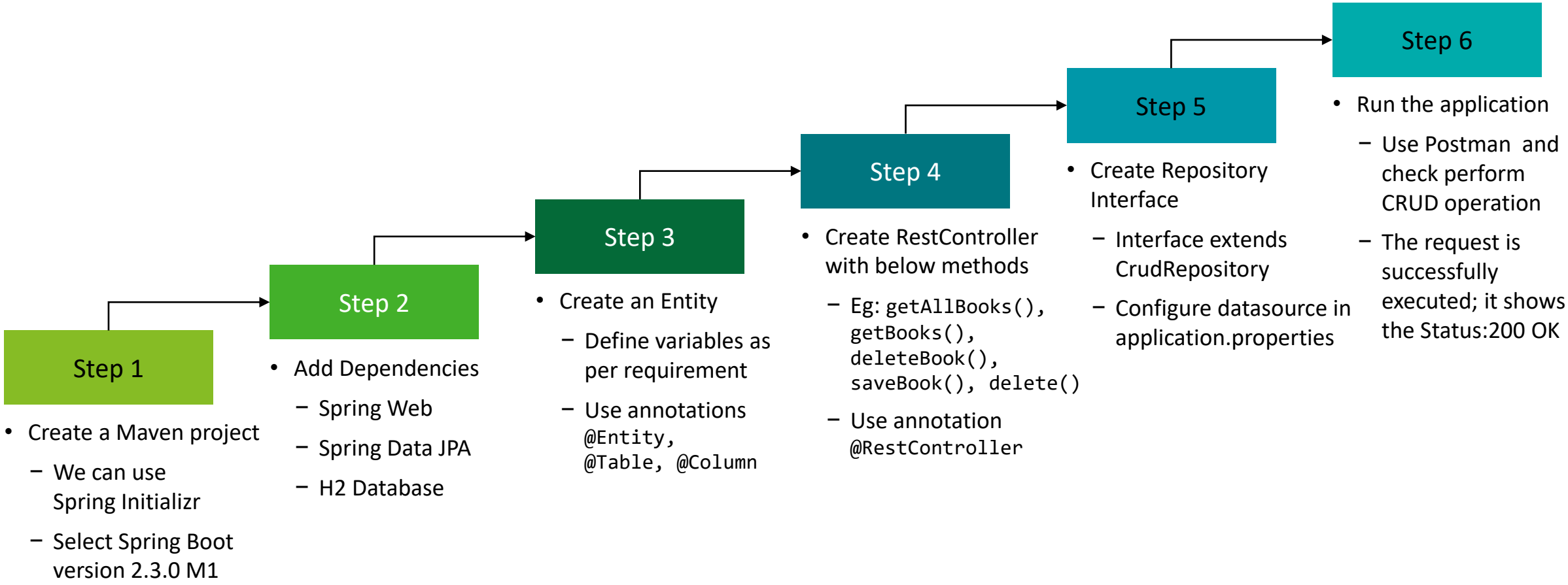
```
public interface CrudRepository<T,ID> extends
JpaRepository<T,ID>
Eg:
public interface BookDAO extends
JpaRepository
{
}
```

Spring Data Repository Interface



Example and Code Setup

Steps To Create Spring Boot CRUD Operation Example



Demo and Walkthrough/Code setup

</>

Model Class

```
import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.Id;
import javax.persistence.Table;
//mark class as an Entity
@Entity
//defining class name as Table name
@Table
public class Books
{
//Defining book id as primary key
@Id
@Column
private int bookid;
//  Getters and Setters
```

Demo and Walkthrough/Code setup (Cont.)

</>

Controller

```
@RestController
public class BooksController
{
//autowire the BooksService class
@Autowired
BooksService booksService;
//creating a get mapping that retrieves all the books detail from the
database
@GetMapping("/book")
private List<Books> getAllBooks()
{
return booksService.getAllBooks();
}
 // similarly try put,delete
```

Demo and Walkthrough/Code setup (Cont.)

</>

Service

```
@Service
public class BooksService
{
@Autowired
BooksRepository
//getting all books record by using the method findaAll() of CrudRepository
public List<Books> getAllBooks()
{
List<Books> books = new ArrayList<Books>();
booksRepository.findAll().forEach(books1 -> books.add(books1));
return books;
}
//similarly update,delete
```

Demo and Walkthrough/Code setup (Cont.)

< / >

Application.properties

```
spring.datasource.url=jdbc:h2:mem:books_data
spring.datasource.driverClassName=org.h2.Driver
spring.datasource.username=sa
spring.datasource.password=
spring.jpa.database-platform=org.hibernate.dialect.H2Dialect
#enabling the H2 console
spring.h2.console.enabled=true
```

Demo and Walkthrough/Code setup (Cont.)

</>

SpringBootCrudOperationApplication.java

```
@SpringBootApplication
public class SpringBootCrudOperationApplication
{
public static void main(String[] args)
{
SpringApplication.run(SpringBootCrudOperationApplication.class, args);
}
}
```

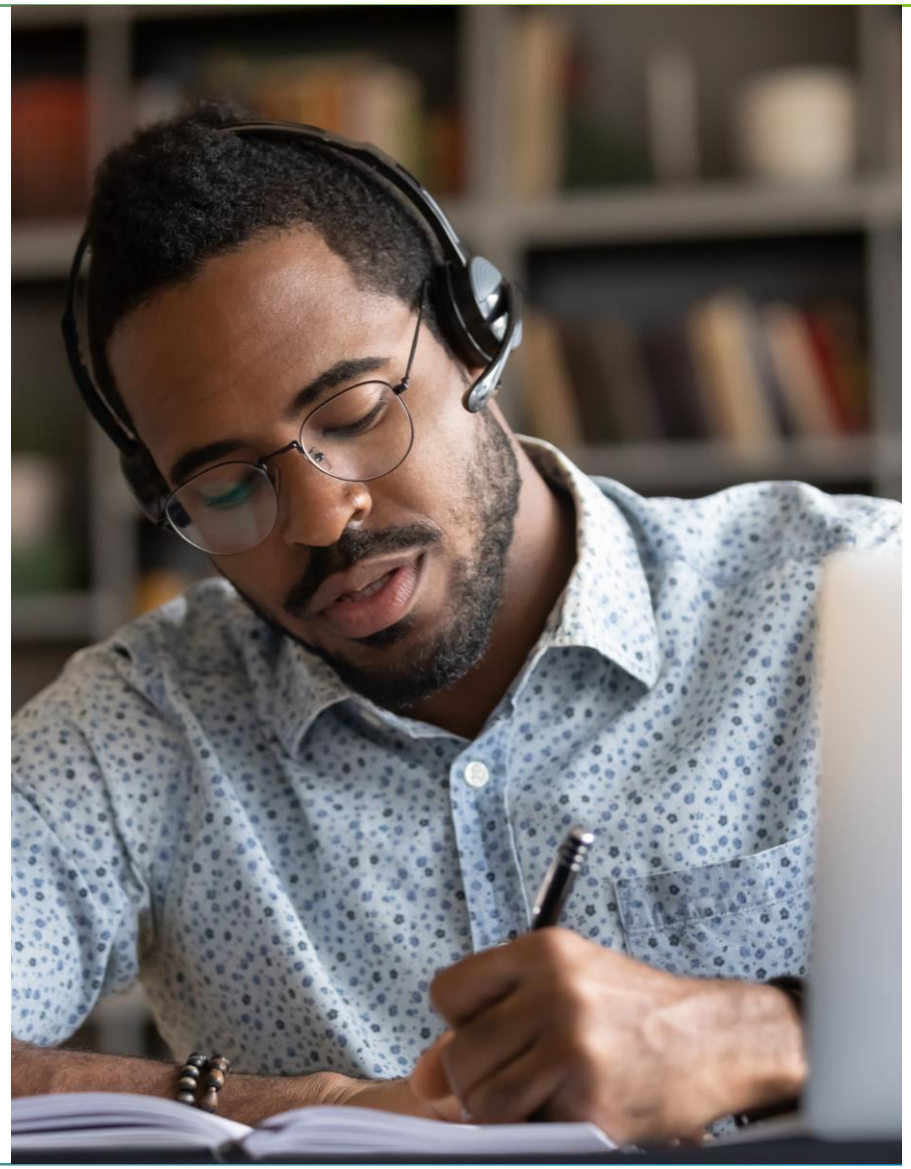
Summary

</ />

Here are the key learning points of the module.

- Spring Boot is an open-source framework that aims to decrease overall development time and boost efficiency by having a default setup for unit and integration tests.
- Using Spring Boot helps reduce development time and increases the overall productivity of the development team.
- While Spring MVC is a Model-View-Controller-based web framework under the Spring framework, Spring Boot avoids boilerplate code and wraps dependencies together in a single unit.
- Spring Boot Annotations is a kind of metadata that provides data about a program.
- CRUD operation refers to user interface conventions that allow viewing, searching, and modifying information through computer-based forms and reports.
- The actuator endpoints enable us to monitor and interact with the application.

Hands-On Labs



Hands-On Activity 1

</ >

Activity Details	
Problem Statement	Create a Spring Boot Maven project from Spring initializr website and add 3 dependencies (web, actuator, and devtools)
Sample Input	NA

Hands-On Activity 2

</ />

Activity Details	
Problem Statement	Create a Maven project using Spring Initializer. Add web dependency. Also create a Rest controller with HTTP methods like GET, and POST. Create a Model class called Employee with some fields such as firstname, lastname, and email. Create a DAO class with static list to store Employee data.
Sample Input	NA

Hands-On Activity 3

</>

Activity Details	
Problem Statement	• Create a Maven project using Spring Initializer • Add spring-boot-starter-parent, spring-boot-starter-web, and spring-boot-starter-actuator • Create REST API Controller with `@GetMapping` • Add following security related properties in application.properties
Sample Input	NA

Hands-On Activity 4

</ />

Activity Details	
Problem Statement	Perform a simple CRUD operation using Spring Boot project and JPA.
Sample Input	Provide HTTP methods like Get, Post, Put, and Delete.

Thank You

Deloitte.