# Spring

**Spring Boot Interview questions**:

<https://www.youtube.com/watch?v=-_tPeb3VE6w>

**Spring Boot Tutorial | Spring Data JPA | 2021**

<https://www.youtube.com/watch?v=8SGI_XS5OPw>

**Java Data Structures Tutorial Amigoscode**

<https://www.youtube.com/watch?v=8MmMm2-kJV8>

**Spring Boot Microservices Level 1 : Movie night project**

https://www.youtube.com/watch?v=bTFIduBgXuo&list=PLqq-6Pq4lTTZSKAFG6aCDVDP86Qx4lNas&index=8

**Elaborate**:

* Beans are the building blocks of a Spring application and represent various application components, such as services, controllers, data sources, and more.
* Difference between @Component, @Service, @Repository, or @Controller.
* These features include built-in support for monitoring, health checks, metrics, externalized configuration, and more. Example?
* Bean factory vs application context.
* What are REST endpoints. Hierarchy

List the JSR annotations provided by Java. When should we use them instead of Spring Annotation.

Beans and services are by default created as a Singleton. Singleton.

**Inversion of Control:**

In traditional programming, we create and manage objects directly in our code. In IoC design principle it flips the control of object creation and lifecycle management from application code to a container or framework. It helps in achieving loose coupling between components and promotes modularity which in turn helps in maintain and extend.

**Dependency Injection**:

The objects which are needed in the application code are created by framework or container supporting IoC. These objects are handed over to the application when needed. This is called Dependency Injection.

Dependency Injection can by done by constructor, methods, properties/setters.

A diagram on a white paper

Description automatically generated

**Spring is a modular framework**:

The term "modular framework" in the context of the Spring Framework means that Spring is organized into individual, self-contained modules, each of which provides specific sets of features and functionalities. These modules can be used independently or in combination, allowing developers to pick and choose the components they need for their applications.

Advantages: Granularity, Flexibility, Reusability, Testability, Customization, Reduced Dependencies, Evolvability.

**Bean**:

a Java object that is managed by the Spring IoC (Inversion of Control) container. Representing various application components, such as services, controllers, data sources, and more. Marked by @Component, @Service, @Repository, or @Controller.

These beans have scopes, singleton, prototype, request, session.

**Component**:

In Spring, a "component" is a more generic term that often refers to any Spring-managed bean. In other words, all Spring beans are components, but not all components are necessarily beans.

**Spring Container**:

The Spring IoC (Inversion of Control) container is a fundamental part of the Spring Framework that manages the instantiation, configuration, and lifecycle of Java objects, known as Spring beans. The IoC container implements the principle of Inversion of Control, which is central to the Spring framework's design.

**Spring IoC (Inversion of Control) container**:

Spring offers two main types of IoC containers: Bean Factory and Application Context.

Bean Factory: The Bean Factory is the basic form of the IoC container and provides essential features like object creation and dependency injection. It's suitable for resource-constrained environments because it initializes beans lazily (upon request).

Bean Factory is **deprecated** from Spring 3.0.

Application Context: The Application Context is an advanced version of the IoC container that builds on the Bean Factory's functionality. It offers additional features, such as event propagation, internationalization, and integration with other Spring modules. Application Context initializes beans eagerly at startup and is commonly used in most Spring applications.

**ApplicationContext**:

We can access the ApplicationContext by

1. @Autowire the member variable of class.

2. Implementing the ApplicationContextAware Interface and adding the methods.

3. Simply adding it as a parameter in the constructor. Spring Boot will automatically provide the it when it creates the bean.

A diagram of a computer program

Description automatically generated

|  |  |
| --- | --- |
| **Spring** | **Spring Boot** |
| **42** |  |
| involves writing XML or Java configuration files to define beans, data sources, and other application components. | It provides auto-configuration, which automatically configures common components based on the libraries and dependencies detected on the class path. Customization can be achieved through properties files, YAML, or Java configuration classes. |
| In a Spring application, you need to manage dependencies manually, typically using tools like Maven or Gradle. | Spring Boot simplifies dependency management by providing a set of curated, version-managed starter dependencies (opinionated defaults). These can be overridden. |
| you need to configure and manage an external web server, such as Apache Tomcat or Jetty, separately. | includes an embedded web server (e.g., Tomcat, Jetty, Undertow) that is configured automatically. |
| provides flexibility but can require a significant amount of configuration and customization | providing sensible defaults and reducing the need for boilerplate code. |
| doesn't include built-in features like health checks, metrics, and externalized configuration by default. | include built-in support for monitoring, health checks, metrics, externalized configuration, |

**Servlet Container or servlet engine or Web Container:**

It specifically focuses on executing Java Servlets and JSP(JavaServerPages). Its primary purpose is to manage the lifecycle of servlets, initializing servlets, handling HTTP requests & responses and managing threading for concurrent requests.

Example: Apache Tomcat, Jetty.

**Web server**:

Examples: Apache and Internet Information services (Microsoft IIS).

**Embedded Container**:

An embedded container is a type of servlet container, but it is designed to be embedded within an application. It allows you to package and run web applications as standalone applications with an embedded web server.

Embedded containers are commonly used in modern frameworks like Spring Boot and allow you to create self-contained executable JAR files.

Example: Tomcat embedded in Spring Boot, Jetty embedded in Dropwizard, and Undertow embedded in WildFly Swarm.

**Web Container**:

A web container is a broader term that encompasses the entire environment for running web applications, including components beyond servlets.

It includes the web server and servlet container.

A web container can serve static content, dynamic content (e.g., JSP pages), and handle different types of web application components.

A web server is software that handles HTTP requests and responses, serving web content to clients like web browsers.

Example: Apache Tomcat, Nginx.

STILL NOT CLEAR

**Resources:**

Resources can represent any information, such as data objects, documents, or services. Each resource is identified by a unique URI (Uniform Resource Identifier). Resources can have multiple representations (e.g., XML, JSON, HTML) to support different clients and scenarios.

Clients (browser or application) interact with resources through these representations. They request a particular representation of a resource by specifying the media type (e.g., application/json) in the request's Accept header.

Clients interact with resources by using standard HTTP methods (GET, POST, PUT, DELETE) to perform operations on the resource.

**Stateless principle**:

Each request from a client to a server must contain all the information needed to understand and process the request. The server does not store any client state between requests. This simplifies server implementation and enables scalability.

**Representational State Transfer (REST)**:

REST is not a protocol but a set of principles and constraints for creating scalable and maintainable networked systems. It uses standard HTTP methods to perform operations on resources.

1. REST APIs should model the application as a set of resources.
2. Use standard HTTP methods to perform operations on resources.
3. It should be a stateless architecture.
4. RESTful APIs should implement security mechanisms, such as authentication and authorization, to protect resources and ensure secure interactions.
5. RESTful APIs use standard HTTP status codes to indicate the success or failure of requests.

REST allows for a layered architecture, where intermediaries (e.g., load balancers, caches) can be placed between the client and server to improve scalability and security.