**SPRINGBOOT**

**1. What is Spring Boot?**

Spring Boot is built on top of the Spring framework to create stand-alone RESTful web applications with minimal configuration. It includes embedded servers like Tomcat and Jetty, eliminating the need for external servers.

* Spring Boot is independent.
* It creates executable Spring applications that are production-grade.

**2. What are the Features of Spring Boot?**

* Auto Configuration
* Embedded Servers (Tomcat, Jetty, etc.)
* Microservice Ready
* Production-Ready (Actuator)
* Minimal Configuration
* Rapid Development with Spring Boot Starters
* Wide Range of Tooling Support

**3. Advantages of Spring Boot**

* **Fast Development**
* **Easy to Deploy**
* **Scalable**
* **Minimal Configuration**
* **Production-Ready Features** (health checks, metrics)

**4. Define the Key Components of Spring Boot**

* **Spring Boot Starters**: Pre-configured dependencies for specific tasks (e.g., web, JPA).
* **Auto-Configuration**: Automatically configures Spring beans based on the classpath.
* **Spring Boot Actuator**: Provides production-ready features like health checks, metrics, etc.
* **Spring Boot CLI**: Command-line tool for rapid development.
* **Embedded Servers**: Tomcat, Jetty, or Undertow as embedded servers.

**5. Why Spring Boot Over Spring Framework?**

| **Feature** | **Spring Framework** | **Spring Boot** |
| --- | --- | --- |
| Ease of Use | Complex | Easy |
| Configuration | Manual | Auto |
| Deployment | External Server | Embedded Server |
| Development Speed | Slower | Faster |

**6. Explain the Internal Working of Spring Boot**

* **Create a Spring Boot Project**: Using Spring Initializr or manually.
* **Add Dependencies**: Required starters and dependencies.
* **Annotate Application**: Use annotations like @SpringBootApplication to bootstrap.
* **Run the Application**: Spring Boot takes care of setting up the application context, configuring the embedded server, and starting the app.

**7. What are the Spring Boot Starter Dependencies?**

Spring Boot provides several starter dependencies, such as:

* **Data JPA Starter**
* **Web Starter**
* **Security Starter**
* **Test Starter**
* **Thymeleaf Starter**

**9. What does the @SpringBootApplication annotation do internally?**

The @SpringBootApplication annotation combines three annotations:

* **@Configuration**: Marks the class as a source of bean definitions for the application context.
* **@EnableAutoConfiguration**: Automatically configures Spring application context based on the libraries in the classpath.
* **@ComponentScan**: Scans for Spring components like @Service, @Controller, and @Repository in the package of the annotated class.

**10. What is Spring Initializr?**

Spring Initializr is an online tool or IDE plugin that helps generate a Spring Boot project skeleton with essential files (Maven/Gradle) to start a Spring Boot application quickly.

**11. What are Spring Boot CLI and the most used CLI commands?**

Spring Boot CLI (Command Line Interface) is a command-line tool built on Groovy that helps you develop Spring Boot applications quickly.

* **Most used commands**:
  + run
  + test
  + jar
  + war
  + init
  + help

**12. What are the Basic Spring Boot Annotations?**

* **@SpringBootApplication**: Main annotation to bootstrap a Spring Boot application.
* **@Configuration**: Indicates that the class has Spring configuration.
* **@Component**: Marks a class as a Spring-managed bean.
* **@RestController**: Marks a class as a RESTful controller (combines @Controller and @ResponseBody).
* **@RequestMapping**: Maps HTTP requests to controller methods.

**13. What is Spring Boot Dependency Management?**

Spring Boot provides dependency management by ensuring that all necessary dependencies for the current version of Spring Boot are included and compatible.

**14. Is it possible to change the port of the embedded Tomcat server in Spring Boot?**

Yes, you can change the port of the embedded Tomcat server in Spring Boot by modifying the application.properties or application.yml

server.port=9090

**15. What is the Starter Dependency of the Spring Boot Module?**

Spring Boot Starters are collections of pre-configured Maven/Gradle dependencies for specific tasks (e.g., spring-boot-starter-web, spring-boot-starter-data-jpa). They make it easier to develop specific types of applications.

**16. What is the Default Port of Tomcat in Spring Boot?**

The default port for the embedded Tomcat server in Spring Boot is **8080**.

**17. Can We Disable the Default Web Server in Spring Boot?**

Yes, we can disable the default embedded web server by setting the following in application.properties:

server.port=-1

**20. Describe the Flow of HTTPS Requests through the Spring Boot Application**

1. **Client Request**: The client sends an HTTP request (GET, POST, etc.) to the server.
2. **Controller Layer**: The request is mapped and handled by the controller.
3. **Service Layer**: The business logic is processed.
4. **Repository Layer**: CRUD operations are handled using JPA repositories.
5. **Response**: A response is sent back to the client, typically as a view or JSON/XML data.

**21. Explain @RestController Annotation in Spring Boot**

@RestController is a convenience annotation that combines @Controller and @ResponseBody. It is used for RESTful web services. Methods in @RestController return data directly (like JSON, XML) rather than views.

* Defines endpoints for HTTP methods like GET, POST, PUT, DELETE.
* Automatically serializes return objects into HTTP responses.

**22. What are Profiles in Spring?**

Spring Profiles allow you to define different configurations for different environments (development, production, etc.).

* Use @Profile annotation to specify which beans should be used in a particular profile.
* Activate profiles through the spring.profiles.active property:properties

spring.profiles.active=dev

**23. What is Spring Boot Actuator?**

Spring Boot Actuator is a component of Spring Boot that provides production-ready features such as monitoring, health checks, metrics, and more. It helps manage and monitor Spring Boot applications.  
  
**24. How to enable Actuator in the Spring boot application?**

Below are the steps to enable actuator in Spring Boot Application:

* Add Actuator dependency.
* Enable endpoints in application.properties.
* Run your Spring Boot app.

**25. What is the purpose of using @ComponentScan in the class files?**

**@ComponentScan**annotation is used to tell Spring to scan a package and automatically detect Spring components, configurations, and services to configure. The @ComponentScan annotation can be used in the following ways:

* Without arguments
* With basePackageClasses
* With basePackages

**. What are the @RequestMapping and @RestController annotations in Spring Boot used for?**

**@RequestMapping:**@RequestMapping is used to map HTTP requests to handler methods in your controller classes. It can be used at the class level and method level. It supports mapping by:

* HTTP method – GET, POST, PUT, DELETE
* URL path
* URL parameters
* Request headers

**@RestController:**@RestController is a convenience annotation that combines **@Controller**and **@ResponseBody**. It indicates a controller where every method returns a domain object instead of a view.

**27. How to get the list of all the beans in your Spring boot application?**

* Using the **ApplicationContext**object in Spring Boot, we can retrieve a list of all the beans in our application.
* The ApplicationContext is responsible for managing the beans and their dependencies.

**28. Can we check the environment properties in your Spring boot application explain how?**

Yes, we can check the environment properties in our Spring Boot Application. The Environment object in a Spring Boot application can be used to check the environment’s properties.

Configuration settings for the application, includes:

* property files
* command-line arguments
* environment variables

**29. What is dependency Injection and its types?**

**Dependency Injection**(DI) is a design pattern that enables us to produce loosely coupled components. In DI, an object’s ability to complete a task depends on another object. There three types of dependency Injections.

* **Constructor injection:**This is the most common type of DI in Spring Boot. In constructor injection, the dependency object is injected into the dependent object’s constructor.
* **Setter injection:**In setter injection, the dependency object is injected into the dependent object’s setter method.
* **Field injection**: In field injection, the dependency object is injected into the dependent object’s field.

**30. What is an IOC container?**

An **IoC (Inversion of Control)**Container in Spring Boot is essentially a central manager for the application objects that controls the creation, configuration, and management of dependency injection of objects (often referred to as beans), also referred to as a DI (Dependency Injection) container.

**31. Mention the differences between WAR and embedded containers.**

**WAR (Web Application Archive)** files require an external server (like Tomcat) for deployment, with separate configuration files. **Embedded containers** package the server within the application itself, allowing it to run as a standalone executable (JAR). Embedded containers simplify deployment and configuration, while WAR files are more traditional and need an external server for execution.

**32 .What are the differences between @SpringBootApplication and @EnableAutoConfiguration annotation?**@SpringBootApplication = full setup (auto-config + scanning + config)

@EnableAutoConfiguration = only auto-config.

**33. Difference between @RequestMapping and @GetMapping**

* **@RequestMapping**: Used to map various HTTP methods (GET, POST, PUT, DELETE).
* **@GetMapping**: A shortcut specifically for handling **GET** requests only.  
  @GetMapping is more concise and readable for GET requests.

**34 .Difference between @Controller and @RestController**

* **@Controller**: Used for web applications; returns **views (HTML/JSP)**.
* **@RestController**: Returns **data (JSON/XML)** for REST APIs; combines @Controller + @ResponseBody.  
  Use @RestController when building RESTful web services.

**Hibernate**

**1. What is Hibernate?**

* Open-source ORM framework for Java.
* Maps Java objects to database tables.
* Simplifies CRUD without manual SQL queries.

**2. Advantages of Hibernate**

* Prevents data corruption by ensuring full changes.
* Uses Hibernate Query Language (HQL), similar to SQL.
* Automatically creates DB tables from Java classes.
* Improves performance with caching and optimized queries.

**3. What is ORM in Hibernate?**

* Object-Relational Mapping translates Java objects to relational DB tables.
* Reduces manual SQL coding

**5. Important Hibernate Interfaces**

* SessionFactory: Creates and manages Sessions.
* Session: Interface for DB operations, holds first-level cache.
* Transaction: Manages DB transactions.
* Query: Used to create/execute queries**.**
* SessionFactoryBuilder: Builds SessionFactory instances.

**6. Supported Databases**

MySQL, Oracle, PostgreSQL, H2, DB2, Microsoft SQL Server, Sybase, SQLite

**7. Java Persistence API (JPA)**

* ORM API for Java to persist data.
* Spring Data JPA simplifies CRUD and pagination.
* Spring Repository extends JPA functionality

**8. Hibernate Inheritance Mapping**

* Maps Java inheritance to DB tables.
* Strategies:
  + Single Table
  + Table Per Class
  + Joined Table

**9. What is HQL?**

* Hibernate Query Language, object-oriented SQL.
* Uses entities & properties, supports joins, aggregation, polymorphism.

**10. Creating HQL Queries**

* Create HQL string with entity/property.
* Use session.createQuery(hql, Class) to create Query object.
* Set parameters via setParameter().
* Execute with list() or uniqueResult().

**12. What is a Session?**

* Primary interface for DB operations.
* Manages transactions, caching, lazy loading, object persistence.

**13. What is SessionFactory?**

* Factory for Sessions.
* Reads configuration, manages connection, metadata, caching.
* Thread-safe and shared among threads.
* Sessions are not thread-safe.

**14. Is Session Thread-safe?**

* No, Session is NOT thread-safe.
* Each thread should have its own Session instance.

**17. Difference Between update() and merge()**

* update(): Works with persistent objects, returns void, throws exception if transient, copies all states including nulls, needs version property, might cause more SQL updates.
* merge(): Works with detached or transient objects, returns managed object, does not throw exception on transient, copies only non-null states, controls version conflicts, may generate fewer SQL updates.

**18. First-Level Cache vs Second-Level Cache**

* First-Level Cache: Session scoped, entity-level caching, memory-resident, no multi-session concurrency, auto-cleared after session ends.
* Second-Level Cache: SessionFactory scoped, supports queries & collections, configurable storage (memory/disk), supports multi-session concurrency, customizable strategies.

**19. getCurrentSession() vs openSession()**

* getCurrentSession(): Managed by Hibernate, tied to current transaction, auto-closed.
* openSession(): Manually managed, not transaction-scoped, needs explicit close.

**20. save() vs saveOrUpdate()**

* save(): For new (transient) entities only, generates new ID, throws exception if already exists.
* saveOrUpdate(): For transient or detached entities, inserts or updates accordingly, reconnects detached objects.

**21. Hibernate Object States**

* Transient: New object, not associated with session or DB.
* Persistent: Associated with session, changes tracked.
* Detached: Previously persistent, session closed.
* Removed: Marked for deletion, not associated anymore.

**22. Immutable Class in Hibernate**

* Use final class, private final fields, no setters, initialize via constructor/factory method, ensure referenced objects are immutable.

**23. Automatic Dirty Checking**

* Hibernate detects changes to persistent objects automatically and synchronizes with DB on transaction commit without explicit update calls.

**24. SQL Injection Protection**

* Hibernate prevents SQL injection by using parameter binding in HQL and Criteria API rather than concatenating raw SQL.

**25. Common Hibernate Annotations**

* @Entity, @Table, @Id, @GeneratedValue, @Column, @OneToMany, @ManyToOne, @ManyToMany.

**26. Hibernate Architecture Components**

* Application Layer → Hibernate API → Configuration → SessionFactory (one per app) → Session (per transaction) → Transaction → Mapping Metadata → ORM core.

**27. Criteria API**

* Type-safe, programmatic query builder in Java; easier dynamic queries with compile-time checks, avoids string-based HQL.

**28. session.lock() Method**

* Acquires a pessimistic lock on a persistent object to prevent concurrent modifications during transactions.

**29. Hibernate Caching**

* First-Level Cache: Session scoped, default, caches objects within a session.
* Second-Level Cache: SessionFactory scoped, shared across sessions, configurable providers, improves performance for repeated queries/data access.