1. **Inversión de Control (IoC) y D.I.: ¿Qué es la Inversión de Control y cómo la  implementa Spring a través de la Inyección de Dependencias? ¿Cuáles son las  ventajas de este patrón?**

**R/ -Inversión de Control (IoC)** es un principio donde el control del flujo de la aplicación se invierte: en lugar de que el código cree y gestione sus dependencias, un contenedor (como Spring) lo hace.

**-Inyección de Dependencias (DI)** es una implementación de IoC: Spring crea y suministra las dependencias (beans) automáticamente mediante constructor, setter o campo.

**Ventajas:** Menor acoplamiento, mayor reutilización de código, facilita pruebas unitarias, mejor mantenimiento.

1. **Spring vs. Spring Boot:** ¿Cuál es la principal diferencia entre el Framework de  Spring y Spring Boot? Explica el concepto de "auto-configuración" de Spring  Boot.

R/ **Spring Framework** es completo pero requiere configuración manual extensa (XML o JavaConfig).

**Spring Boot** es una extensión que simplifica la configuración y el despliegue de aplicaciones Spring.

**Auto-configuración**: Spring Boot configura automáticamente beans y dependencias según las librerías presentes en el classpath (por ejemplo, configura un DataSource si encuentra spring-boot-starter-data-jpa y H2/MySQL).

1. **Scopes de Beans:** ¿Cuál es el scope (alcance) por defecto de un bean en  Spring? Menciona y describe al menos otros dos scopes disponibles.

**R/ Scope por defecto**: singleton (un solo bean por contenedor de Spring).

Otros scopes comunes:

* **prototype:** se crea una nueva instancia cada vez que se solicita.
* **request:** una instancia por solicitud HTTP (en aplicaciones web).
* **session:** una instancia por sesión HTTP del usuario.

1. **Gestión de Transacciones:** ¿Qué hace la anotación @Transactional? Explica  brevemente qué significa una propagación de tipo REQUIRED vs. REQUIRES\_NEW.

**R/ @Transactional** gestiona transacciones de base de datos de forma declarativa.

**Propagación:**

* **REQUIRED (por defecto):** usa una transacción existente si existe; si no, crea una nueva.
* **REQUIRES\_NEW:** suspende cualquier transacción existente y crea una nueva transacción aislada.

1. **Pruebas en Spring: ¿Cuál es la diferencia entre usar @SpringBootTest y  @WebMvcTest para escribir pruebas? ¿Cuándo usarías cada una?**

**R/ @SpringBootTest:** carga todo el contexto Spring Boot. Ideal para pruebas de integración.

**@WebMvcTest:** carga solo el stack web (controladores, filtros, etc). Ideal para pruebas unitarias de controladores.

Usa **@WebMvcTest** para probar solo la lógica HTTP y Usa **@SpringBootTest** para pruebas completas (servicios, repositorios, etc.).

1. **Principios REST:** ¿Qué significa que una API REST sea "stateless" (sin estado)  y por qué es una característica importante en arquitecturas escalables?

**R/** Una API stateless no guarda estado entre solicitudes. Cada petición debe contener toda la información necesaria.

**Importancia:**

* Permite escalar fácilmente (cualquier servidor puede manejar cualquier petición).
* Mejora la resiliencia y el balanceo de carga.

1. **Manejo de Configuración:** ¿Cómo gestionarías diferentes configuraciones (ej.  conexión a base de datos) para los entornos de desarrollo, QA y producción en una aplicación Spring Boot?

R/ Puedes usar:

* **application-{profile}.properties** o **.yml** (por ejemplo: application-dev.yml, application-prod.yml).
* Define el perfil activo:

spring.profiles.active=dev

* También puedes configurar desde línea de comandos:

java -jar app.jar --spring.profiles.active=prod

1. **Spring Boot Actuator:** ¿Qué es Spring Boot Actuator y para qué sirve?  Menciona dos endpoints que consideres especialmente útiles.

**R/ Actuator** proporciona endpoints **REST** para monitorear y administrar la aplicación.

**Endpoints útiles:**

* **/health**: muestra el estado de la aplicación.
* **/metrics:** muestras métricas como uso de memoria, número de peticiones, etc.

1. **Controller vs. RestController:** ¿Cuál es la diferencia fundamental entre las  anotaciones @Controller y @RestController?

R**/ @Controller:** devuelve vistas (HTML). Usado con tecnologías como Thymeleaf.

**@RestController:** combina @Controller + @ResponseBody. Devuelve datos JSON directamente, ideal para APIs REST.

1. **Manejo de Errores:** Describe una estrategia robusta para manejar  excepciones y devolver respuestas de error consistentes y bien formateadas  en una API REST de Spring Boot.

**R/** Una estrategia robusta incluye:

1. **@RestControllerAdvice** para interceptar excepciones globalmente.
2. Definir **@ExceptionHandler** para diferentes tipos de error.
3. Devolver respuestas con estructura estándar (mensaje, código, timestamp, etc.).

**Ejemplo:**

@RestControllerAdvice

public class GlobalExceptionHandler {

@ExceptionHandler(EntityNotFoundException.class)

public ResponseEntity<?> handleNotFound(EntityNotFoundException ex) {

return ResponseEntity.status(HttpStatus.NOT\_FOUND)

.body(Map.of("error", ex.getMessage(), "timestamp", LocalDateTime.now()));

}

@ExceptionHandler(MethodArgumentNotValidException.class)

public ResponseEntity<?> handleValidation(MethodArgumentNotValidException ex) {

Map<String, String> errors = new HashMap<>();

ex.getBindingResult().getFieldErrors()

.forEach(err -> errors.put(err.getField(), err.getDefaultMessage()));

return ResponseEntity.badRequest().body(errors);

}

}